Gracefully Degrading Gathering in Dynamic Rings
Marjorie Bournat, Swan Dubois, Franck Petit

To cite this version:
Marjorie Bournat, Swan Dubois, Franck Petit. Gracefully Degrading Gathering in Dynamic Rings.
[Research Report] LIP6, Sorbonne Université, CNRS, UMR 7606; DELYS; Inria. 2018. hal-01790554v2

HAL Id: hal-01790554
https://hal.archives-ouvertes.fr/hal-01790554v2
Submitted on 1 Aug 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
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 under which the original problem is impossible to solve. The underlying intuition is to solve the problem whenever possible but to provide some kind of quality of service if the dynamics become (unpredictably) higher.

In this paper, we apply for the first time this approach to robot networks. We focus on the fundamental problem of gathering a squad of autonomous robots on an unknown location of a dynamic ring. In this goal, we introduce a set of weaker variants of this problem. Motivated by a set of impossibility results related to the dynamics of the ring, we propose a gracefully degrading gathering algorithm.

1 Introduction
The classical approach in distributed computing consists in, first, fixing a set of assumptions that captures the properties of the studied system (atomicity, synchrony, faults, communication modalities, etc.) and, then, focusing on the impact of these assumptions in terms of calculability and/or of complexity on a given problem. When coming to dynamic systems, it is natural to adopt the same approach. Many recent works focus on defining pertinent assumptions for capturing the dynamics of those systems [7,16,21]. 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. A gracefully degrading algorithm guarantees that it will solve simultaneously the original problem under some assumption of dynamics and each of its variants under some other (hopefully weaker) assumptions. As an example, Biely et al. provide a consensus algorithm that gracefully degrades to $k$-set agreement when the dynamics of the system increase. The underlying idea is to solve the problem in its strongest variant when connectivity conditions are sufficient but also to provide (at the opposite of a classical algorithm) some minimal quality of service described by the weaker variants of the problem when those conditions degrade.

Note that, although being applied to dynamic systems by Biely et al. for the first time, this natural idea is not a new one. Indeed, indulgent algorithms [1, 9, 12] provide similar graceful degradation of the problem to satisfy with respect to synchrony (not with respect to dynamics). Speculation [8,15] is a related, but somewhat orthogonal, concept. A speculative algorithm solves the problem under some assumptions and moreover provides stronger properties (typically better complexities) whenever conditions are better.

The goal of this paper is to apply graceful degradation to robot networks where a cohort of autonomous robots have to coordinate their actions in order to solve a global task. We focus on gathering in a dynamic ring. In this problem, starting from any initial position, robots must meet on an arbitrary location in a bounded time. 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 [7,17,21]. In this paper, we adopt the evolving graph model [21] 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 us to compare the strength of these temporal connectivity properties based on the inclusion of classes between them. We are interested in the following classes: 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; RE (recurrent-edge graphs) where any edge that appears at least once does so recurrently; BRE (bounded-recurrent-edge graphs) where any edge that appears at least
We consider the model of Dynamic graphs.

2 Model

correctness of our gracefully degrading algorithm. Finally, Section 6 concludes the paper with some comments.

3 sums up impossibility results while Section 4 presents our gracefully degrading algorithm. Section 5 proves the

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

Dynamic graphs. We consider the model of evolving graphs [21]. 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 $G$ at time $i$. $G$ is the footprint of $G$. The underlying graph of $G$, denoted by $U_G$, is the static graph gathering all edges that are present at least once in $G$ (i.e. $U_G = (V,E_G)$ with $E_G = \bigcup_{i=0}^{\infty} E_i$). An eventual missing edge is an edge of $E$ such that there exists a time after which this edge is never present in $G$. A recurrent edge is an edge of $E$ that is not eventually missing. The eventual underlying graph of $G$, denoted $U^E_G$, is the static graph gathering all recurrent edges of $G$ (i.e. $U^E_G = (V,E^E_G)$ where $E^E_G$ is the set of recurrent edges of $G$). We only consider graphs whose footprints are anonymous and unoriented rings of size $n \geq 4$. The class $COT$ (connected-over-time graphs) contains all evolving graphs such that their eventual underlying graph is connected. The class $RE$ (recurrent-edges graphs) gathers all evolving graphs whose footprint contains only recurrent edges. The class $BRE$ (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 $\delta$ units of time. The class $AC$ (always-connected graphs) collects all evolving graphs where the graph $G_i$ is connected for any $i \in \mathbb{N}$. The class $ST$ (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 $R \geq 4$ autonomous mobile entities called robots moving in a discrete and dynamic environment modeled by an evolving graph $G = \{(V,E_0),(V,E_1)\ldots\}$, $V$ being a set of nodes representing the set of locations where robots may be, $E_i$ being the set of bidirectional edges through which robots may move from a location to another one at time $i$. Each robot knows $n$ and $R$. Each robot $r$ possesses a distinct (positive) integer identifier $id_r$ strictly greater than 0. Initially, a robot only knows the value of its own identifier. Robots have a persistent memory so they can store local variables.

Each robot $r$ is endowed with strong local multiplicity detection, meaning that it is able to count the exact number of robots that are co-located with it at any time $t$. When this number equals 1, the robot $r$ is isolated at time $t$. By opposition, we define a tower $T$ as a couple $(S,\theta)$, 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 $\theta$ and $S$ or $\theta$ is maximal for this property. We say that the robots of $S$ form the tower at time $t_s$ and that they are involved in the tower between time $t_s$ and $t_e$. Robots are able to communicate (by direct reading) the values of their variables to each others only when they are involved in the same tower.

Finally, all the robots have the same chirality, i.e. each robot is able to locally label the two ports of its current node with left and right consistently over the ring and time and all the robots agree on this labeling. Each robot $r$ has a variable $dir_r$, that stores the direction it currently considers (right, left or $\perp$).

Algorithms and execution. The state of a robot at time $t$ corresponds to the values of its local variables at time $t$. The configuration $\gamma_t$ 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 as $r$ at time $t$ if any, and of the following local functions: $\text{ExistsEdge(dir,round)}$, with $dir \in \{\text{right},\text{left}\}$ and $round \in \{\text{current,previous}\}$ 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; $\text{NodeMate()}$ which gives the set of all the robots co-located with $r$ (r is not included in this set); $\text{NodeMateIds()}$ which gives the set of all the identifiers of the robots co-located with $r$ (excluded the one of $r$) ; and $\text{HasMoved()}$ which indicates if $r$ has moved between time $t-1$ and $t$ (see below).

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

Given an evolving graph $G = \{G_0,G_1\ldots\}$ and an initial configuration $\gamma_0$, the execution $\sigma$ in $G$ starting from $\gamma_0$ of an algorithm is the maximal sequence $\langle\gamma_0,\gamma_1\rangle\langle\gamma_1,\gamma_2\rangle\langle\gamma_2,\gamma_3\rangle\ldots$ where, for any $i \geq 0$, the configuration $\gamma_{i+1}$ is the result of the execution of a synchronous round by all robots from $\gamma_i$ 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 its view. In the case the direction $dir_r$ of a robot $r$ is in $\{\text{right},\text{left}\}$, the Move phase consists of moving $r$ in the direction it considers if there exists an adjacent edge in that direction to its current node, otherwise (i.e. the adjacent edge is missing) $r$ is stuck and hence remains on its current node. In the case where the direction $dir_r$ 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 problem 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 \( G_E \) (the eventual variant of the gathering problem) is impossible to solve in \( \mathcal{AC} \). Note that Di Luna et al. [19] 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 though 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 counterexamples in the context of impossibility results. Indeed, it is sufficient to construct a sequence of evolving graphs with ever-growing common prefixes and prove that its 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 \( G_E \) 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 \( A \) that satisfies \( G_E \) 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 \( A \) 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 \( G_E \)). So, proving the existence of an execution of \( A \) 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 \( G_E \) 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 notation as follows. Given an evolving graph \( F \), an edge \( e \) of \( F \), and a time interval \( [t, t'] \), the evolving graph \( F \cap [t, t'] \) is the evolving graph \( F' \) defined by: \( e \in F' \) if and only if \( e = \bar{\bar{e}} \land e \in F \). Given two integers \( t_1, t_2 \) such that \( t_1 \leq t_2 \), we define \( F_{t_1,...,t_2} \) the subsequence \( \{F_{t_i},...,F_{t_j}\} \) of \( F \). Given two evolving graphs, \( F \) and \( H \), and an integer \( t \), the evolving graph \( F[0,...,t] \otimes H[t+1,...,+\infty] \) is the evolving graph \( F' \) defined by: \( e \in F' \) if and only if \( i \leq t \land e \in F_i \) or \( i > t \land e \in H_i \).

Let \( G = \{G_0, G_1, ...\} \) be a graph of \( \mathcal{AC} \) whose footprint \( G \) is a ring of size 4 or more such that for all \( v \in N, G_v = G \). Consider two nodes \( u \) and \( v \) of \( G \), such that the node \( v \) is the adjacent node of \( u \) in \( G \) to the right. We denote by \( e_{uv} \) the edge linking the nodes \( u \) and \( v \). Let \( G' = G[\{e_{uv}, N\}] \). Let \( e \) be the execution of the algorithm \( A \) in \( G' \) 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 \( G \) 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 \( (G_m)_{m \in \mathbb{N}} \) such that \( G_0 = G' \) and, for any \( i \geq 0, r_1 \) and \( r_2 \) are never co-located before time \( t_i \) in \( e_i \) (the execution of \( A \) in \( G_i \) starting from the same configuration as \( e_i \)), \( (l_m)_{m \in \mathbb{N}} \) being a strictly increasing sequence with \( t_0 = 0 \). First, we show in the next paragraph that, if some such \( G_i \) exists and moreover ensures the existence of a time \( t' \geq t_i \) where the two robots are still on different nodes in \( e_{vi} \), then we can construct \( G_{i+1} \) as shown on Figure 1. We then prove that our construction guarantees the existence of such a \( t' \), implying the well-definition of \( (G_m)_{m \in \mathbb{N}} \).

As \( r_1 \) and \( r_2 \) are not co-located at time \( t_i \) in \( e_i \), at least one of them must move in finite time in any execution starting from \( e \), otherwise, we obtain a contradiction with the liveness of \( G_E \). Let \( t' \geq t_i \) be the smallest such time in the execution where the topology of the graph does not evolve from time \( t_i \) to time \( t' \). In the following, we show how we construct the evolving graph \( G_{i+1} \), in function of \( t' \) and \( G_i \). As we assume that in \( G_i \), at time \( t' \), \( r_1 \) 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 \( t' \) + 1.

Let \( e \) be the edge between the respective locations of \( r_1 \) and \( r_2 \) at time \( t' \) + 1. We define \( G_{i+1} \) on the same footprint as \( G_i \) by \( G_{i+1} = G_i \otimes (G'[t'+1,...,+\infty] \setminus \{e, \{t' + 1, \ldots, +\infty\}\}) \).

**Case 2:** \( d(r_1, r_2) = 2 \) at time \( t' \) + 1.

Denote \( e \) and \( e' \) the two consecutive edges between the respective locations of \( r_1 \) and \( r_2 \) at time \( t' \) + 1.
Figure 1: Construction of $G_{i+1}$ from $G_i$.  

Legend:

- **Robots**
- Present edge
- Missing edge
- Present or missing edge
- Possibly other edges/nodes

$G_i$ at time $t_i' + 1$  $G_i'$ between time $t_i' + 1$ and $\alpha_i$  $G_i'$ at time $\alpha_i + 1$  

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

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

We define $G_{i+1}$ on the same footprint as $G_i$ by $G_{i+1} = G^0_i \cdots G^{i_i}_i \otimes G^{i_i}_{i+1} \cdots \infty$.

Note that $G_i$ and $G_{i+1}$ 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 $e_i$ and in $e_{i+1}$. By construction of $t'_i$, either $r_1$ or $r_2$ or both of the two robots move at time $t'_i$ in $e_{i+1}$. Moreover, by construction of $G_i$, 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 $G_i$ and $G_{i+1}$). Note that, by construction, $G_{i+1}$ 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 $G_{i+1}$ with the desired properties. Note that $t'_i$ and $G_0$ trivially satisfy all our assumptions. In other words, $(G_{m_i})_{m \in \mathbb{N}}$ is well-defined.

We can then define the evolving graph $G_\omega$ such that for all $i \in \mathbb{N}$, $G_{\omega}$ shares a common prefix with $G_i$ until time $t'_i$. As the sequence $(t_{m_i})_{m \in \mathbb{N}}$ is increasing by construction, this implies that the sequence $(G_{m_i})_{m \in \mathbb{N}}$ converges to $G_\omega$. Applying the theorem of Brand-Santoni et al. [5], we obtain that, until time $t'_i$, the execution of $A$ in $G_{\omega}$ is identical to the one in $G_i$. This implies that, executing $A$ in $G_{\omega}$ (whose footprint is a ring of size 4 or more), $r_1$ and $r_2$ are always on distinct nodes, contradicting the liveness of $G_E$ 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 $G_E$ is impossible under $\mathcal{COT}$ as well.

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

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

**Corollary 2.** There exists no deterministic algorithm that satisfies $G$ 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 $G$ under $\mathcal{RE}$ and of $G_W$ under $\mathcal{COT}$ and $\mathcal{RE}$) are obtained as follows. The definition of $\mathcal{COT}$ and $\mathcal{RE}$ does not exclude the ability for all edges of the graph to be missing initially and for any arbitrary long time, hence preventing the gathering of robots for any arbitrary long time if they are initially scattered. This observation is sufficient to prove a contradiction with the existence of an algorithm solving $G$ or $G_W$ in these classes.

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

## 4 Gracefully Degrading Gathering

This section presents $\mathcal{GDG}$, our gracefully degrading gathering algorithm, that aims to solve different variants of the gathering problem under various dynamics (refer to Table 1).

In Subsection 4.1, we informally describe our algorithm clarifying which variant of gathering is satisfied within which class of evolving graphs. Next, Subsection 4.2 presents formally the algorithm.

### 4.1 Overview

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

The overall scheme of the algorithm consists in first detecting $r_{\min}$, the robot having the minimum identifier so that the $R$ robots eventually gather on its node (i.e., satisfying specification $G_E$). Of course, depending on the particular evolving graph in which our algorithm is executed, $G_E$ may not achieved. In the weakest class (class $COT$) and the “worst” possible evolving graph, one can expect specification $G_{EW}$ only, i.e., at least $R-1$ 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 $G_E$ (or $G$) is achieved. By contrast, they can also never be completed in some weak settings (namely $\mathcal{A}C$ or $COT$), solving $G_{EW}$ (or $G_W$) only.

**Phase $M$.** This phase leads each robot to know whether it possesses the minimum identifier. Initially every robot $r$ considers the right direction. Then $r$ always moves to the right until it moves $4 \times n \times id_r$ 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 $4 \times n \times id_{r_{\min}}$ steps.

However, in that case at least $R-1$ 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 ($G_{EW}$) could be solved in that case. However,
the $R-1$ 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 $R-1$ robots that gathered. Further, it is possible for $R-1$ robots to gather (without $r_{\min}$) even when $r_{\min}$ succeeds in moving $4*n*id_{r_{\min}}$ steps to the right (i.e. even when $r_{\min}$ stops to move because it completed Phase $\Phi$). In that case, if the $R-1$ robots that gathered stop their execution, $G_E$ cannot be solved in $RE$, $BRE$ and $ST$ rings, as $GDG$ should do. Note that, it is also possible for $r_{\min}$ to be part of the $R-1$ robots that gathered.

Recall that robots can communicate when they are both located in the same node. So, the $R-1$ 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 $R-1$ robots try to visit all the nodes of the ring infinitely often in both directions by subtle round trips. Doing so, $r_{\min}$ eventually knows that it possesses the actual minimum identifier.

Phase $\kappa$. 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 $R-3$ other robots join it on its node so that its identifier is known by at least $R-3$ other robots. The obvious question arises: “Why waiting for $R-3$ extra robots only?” A basic idea to gather could be that once $r_{\min}$ is aware that it possesses the minimum identifier, it can just stop to move and just wait for the other robots to eventually reach its location, just by moving toward the right direction. Actually, depending on the particular evolving graph considered one missing edge $e$ may eventually appear, preventing robots from reaching $r_{\min}$ by moving toward the same direction only. That is why the gathering of the $R-2$ robots is eventually achieved by the same search phase as in Phase $\Phi$. 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 an eventually missing edge that will appear again later. This is why during Phase $\kappa$, $r_{\min}$ stops to move until $R-3$ other robots join it to form a tower of $R-2$ robots. In this way these $R-2$ robots start the third phase simultaneously.

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

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

Call $u$ the node where Tail is stuck on an eventual missing edge. In the two situations described even if Phase $\omega$ is not complete by both Head and Tail, either $G_E$ or $G_{EW}$ 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 $R-1$ robots including $r_{\min}$ is formed, $G_{EW}$ is solved. Then, the latter robot tries to reach the tower to eventually solve $G_E$ in favorable cases.

Phase $\tau$. The last phase starts once the robots of Head have completed Phase $\omega$. If it exists a time at which the robots of Tail complete Phase $\omega$, then Head and Tail form a tower of $R-2$ robots and stop moving. As explained in the previous phase, Phase $\omega$ ensures that at least one extra robot eventually joins the node where Head and Tail are located to form a tower of $R-1$ robots. Once a tower of $R-1$ robots including $r_{\min}$ is formed, $G_{EW}$ is solved. Then, the latter robot tries to reach the tower to eventually solve $G_E$ in favorable cases. In the case the robots of Tail never complete the phase $\omega$, 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 $\omega$ either $G_{EW}$ or $G_E$ is solved.
Algorithm 2 Functions used in GDG

Function StopMoving()
\[
\text{dir}_r := \bot
\]

Function MoveLeft()
\[
\text{dir}_r := \text{left}
\]

Function BecomeLeftWalker()
\[
\text{dir}_r := \text{left}
\]

Function Walk()
\[
\text{dir}_r := \begin{cases} 
\bot & \text{if } (\text{id}_r = \text{idHeadWalker}_r \land \text{walkerMate}_r \neq \text{NodeMateIds}() \lor 
\text{id}_r = \text{idHeadWalker}_r \land \text{idHeadWalker}_r \in \text{NodeMateIds}()) \\
\text{right} & \text{otherwise}
\end{cases}
\]
walkSteps\(_r := \) walkSteps\(_r + 1 \) if dir\(_r = \) right \&\& \text{ExistsEdge(right, current)}

Function InitiateWalk()
\[
\text{idHeadWalker}_r := \max\{\text{id}_r \cup \text{NodeMateIds}()\}
\]

walkerMate\(_r := \) NodeMateIds()

state\(_r := \begin{cases} 
\text{headWalker} & \text{if } \text{id}_r = \text{idHeadWalker}_r \\
\text{minTailWalker} & \text{if } \text{state}_r = \text{minWaitingWalker} \\
\text{tailWalker} & \text{otherwise}
\end{cases}
\]

Function BecomeWaitingWalker(r')
\[
\text{dir}_{r'} := \text{right}
\]

Function BecomeMinWaitingWalker()
\[
\text{dir}_{r'} := \text{right}
\]

Function BecomeTailWalker(r')
\[
\text{id}_{r'} := \text{idPotentialMin}_{r'}\land \text{idMin}_{r'} \lor \text{idPotentialMin}_{r'} \land \text{idMin}_{r'}
\]

Function Search()
\[
\text{idHeadWalker}_r := \max\{\text{id}_r \cup \text{NodeMateIds}()\}
\]

walkerMate\(_r := \) NodeMateIds()

state\(_r := \begin{cases} 
\text{headWalker} & \text{if } \text{id}_r = \text{idHeadWalker}_r \\
\text{minTailWalker} & \text{if } \text{state}_r = \text{minWaitingWalker} \\
\text{tailWalker} & \text{otherwise}
\end{cases}
\]

Function BecomeAwareSearcher(r')
\[
\text{dir}_{r'} := \text{right}
\]

Function BecomeTailWalker(r')
\[
\text{id}_{r'} := \text{idPotentialMin}_{r'}\land \text{idMin}_{r'} \lor \text{idPotentialMin}_{r'} \land \text{idMin}_{r'}
\]

Function MoveRight()
\[
\text{dir}_r := \text{right}
\]

rightSteps\(_r := \) rightSteps\(_r + 1 \) if \text{ExistsEdge(dir, current)}

Function InitiateSearch()
\[
\text{idPotentialMin}_{r'} := \min\{\text{id}_r \cup \text{NodeMateIds}()\}
\]

state\(_r := \begin{cases} 
\text{righter} & \text{if } \text{id}_r = \text{idPotentialMin}_{r'} \\
\text{dumbSearcher} & \text{otherwise}
\end{cases}
\]

rightSteps\(_r := \) rightSteps\(_r + 1 \) if \text{state}_r = \text{potentialMin} \land \text{ExistsEdge(dir, current)}

Function Search()
\[
\text{dir}_r := \begin{cases} 
\text{left} & \text{if } \text{NodeMateIds}() \geq 1 \land \text{id}_r = \max\{\text{id}_r \cup \text{NodeMateIds}()\} \\
\text{right} & \text{if } \text{NodeMateIds}() \geq 1 \land \text{id}_r \neq \max\{\text{id}_r \cup \text{NodeMateIds}()\} \\
\text{dir}_r & \text{otherwise}
\end{cases}
\]

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 \( R, n \) and \( \text{id}_r \) as constants.

In addition to the variable \( \text{dir}_r \) (initialized to \( \text{right} \) ), each robot \( r \) possesses seven variables described below. Variable \( \text{state}_r \) 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 \( \text{righter, dumbSearcher, awareSearcher, potentialMin, waiting Walker, minWaitingWalker, headWalker, tail Walker, minTailWalker and leftWalker} \). Initially, \( \text{state}_r \) is equal to \( \text{righter} \). Initially with \( 0 \), \( \text{rightSteps}_r \) counts the number of steps done by \( r \) in the \( \text{right} \) direction when \( \text{state}_r \in \{ \text{righter, potentialMin} \} \). The next variable is \( \text{idPotentialMin}_r \). Initially equals to \( -1 \), \( \text{idPotentialMin}_r \) contains the identifier of the robot that possibly possesses the minimum identifier (a positive integer) of the system. This variable is especially set when \( R = 1 \) \( \text{righter} \) are located on a same node. In this case, the variable \( \text{idPotentialMin}_r \) of each robot \( r \) that is involved in the tower of \( R - 1 \) robots is set to the value of the minimum identifier possessed by these robots. The variable \( \text{idMin}_r \) 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 \( \text{walkerMate}_r \) be the set of all the identifiers of the \( R - 2 \) robots that initiate the Phase \( \mathcal{W} \). Initially this variable is set to \( \emptyset \). The counter \( \text{walkSteps}_r \), initially \( 0 \), maintains the number of steps done in the right direction while \( r \) performs the Phase \( \mathcal{W} \). Finally, the variable \( \text{idHeadWalker}_r \), contains the identifier of the robot that plays the part of Head during the Phase \( \mathcal{W} \).

Moreover, we assume the existence of a specific instruction: \text{terminate}. By executing this instruction, a
Algorithm 3 $\mathcal{GDG}$

Rules for Termination

$\text{Term}_1 : \mathcal{G}_E() \rightarrow \text{terminate}$
$\text{Term}_2 : \mathcal{G}_W() \rightarrow \text{terminate}$

Rules for Phase $T$

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

Rules for Phase $W$

$W_1 : \text{HeadOrTailWalker()} \rightarrow \text{Walk()}$

Rules for Phase $K$

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

Rules for Phase $M$

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

robot stops executing the cycle Look-Compute-Move forever.

To ease the writing of our algorithm, we define a set of predicates (presented in Algorithm 1) and functions (presented in Algorithm 2), that are used in our gracefully degrading algorithm $\mathcal{GDG}$. Recall that, during the Compute phase, only the first rule whose guard is true in the view of an enabled robot is executed.

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

In this section, we first prove, in Subsection 5.1, that $\mathcal{GDG}$ solves $\mathcal{G}_W$ 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 $\text{Term}_1$ or Rule $\text{Term}_2$ 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 $\text{min}$ a robot such that its variable state is equal either to $\text{minWaitingWalker}$ or to $\text{minTailWalker}$.

5.1 $\mathcal{GDG}$ solves $\mathcal{G}_W$ in $\mathcal{COT}$ rings

In this subsection, we prove that $\mathcal{GDG}$ solves $\mathcal{G}_W$ 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 $\text{min}$. Therefore, in this section we have to prove that only
Case 1: \( r \) becomes \( \text{min} \), and that \( r_{\text{min}} \) effectively becomes \( \text{min} \) 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 \( \text{righter} \) or \( \text{potentialMin} \) cannot become a \( \text{righter} \) or a \( \text{potentialMin} \).

**Observation 2.** By the rules of \( \mathcal{GDG} \), a robot whose state is not \( \text{righter} \) cannot become a \( \text{righter} \) 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 \( \text{min} \) it stays \( \text{min} \) during the rest of the execution. We prove this statement in the following lemma.

**Lemma 1.** \( \text{min} \) is a closed state under \( \mathcal{GDG} \).

**Proof.** A robot is a \( \text{min} \) when its state is either equal to \( \text{minWaitingWalker} \) or to \( \text{minTailWalker} \). A \( \text{minTailWalker} \) robot can only execute the rules \( T_3 \) and \( W_4 \) that do not update the variable state. A \( \text{minWaitingWalker} \) robot can only execute the rules \( K_1 \) and \( K_2 \) that respectively makes it become a \( \text{minTailWalker} \) and does not change its state.

In the following lemma, we prove that \( \text{righter} \) and \( \text{potentialMin} \) are robots that always consider the right direction. This lemma helps us to prove the correctness of Phase \( \mathcal{M} \), as well as the correctness of Phase \( \mathcal{K} \).

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

**Proof.** Robots that are \( \text{righter} \) robots in a configuration \( \gamma_i \) at time \( i \) and that are still \( \text{righter} \) in the configuration \( \gamma_{i+1} \), consider the right direction during the move Phase of time \( i \) (Rule \( M_8 \)). Moreover, by Observation 2 and since initially all the robots are \( \text{righter} \) robots and consider the right direction, if a robot is a \( \text{righter} \) 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 \( \text{potentialMin} \) robots in a configuration \( \gamma_i \) at time \( i \) and that are still \( \text{potentialMin} \) in the configuration \( \gamma_{i+1} \), consider the right direction during the move Phase of time \( i \) (Rule \( M_9 \)). The only way for a robot to become a \( \text{potentialMin} \) is to be a \( \text{righter} \) and to execute Rule \( M_6 \). While executing Rule \( M_6 \), a \( \text{righter} \) that becomes \( \text{potentialMin} \) 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 \( \text{potentialMin} \) 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 \( r_{\text{min}} \) can be aware that it possesses the minimum identifier among all the robots of the system.

**Lemma 3.** Only \( r_{\text{min}} \) can become \( \text{min} \).

**Proof.** Assume that there exists a robot \( r \neq r_{\text{min}} \) that becomes \( \text{min} \). Assume also that \( r \) is the first robot different from \( r_{\text{min}} \) that becomes \( \text{min} \). By definition of \( r_{\text{min}} \), \( \text{id}_r > \text{id}_{r_{\text{min}}} \).

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

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

**Case 1:** \( r \) becomes \( \text{min} \) because the condition \( \text{state}_r = \text{potentialMin} \land \exists r' \in \text{NodeMate}(), (\text{state}_{r'} = \text{righter} \land \text{id}_r < \text{id}_{r'}) \) is true.

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

By \((*)\), \( r \) is not yet \( \text{min} \) at the time of its meeting with \( r' \). A robot \( r' \) can update its variable \( \text{idMin} \) with the identifier (other than \( \text{id}_r \)) of a robot that is not \( \text{min} \) only when it executes Rules \( M_5, M_7, M_9 \) or \( M_{10} \). Among these rules only the rules \( M_7 \) (in the case a \( \text{righter} \) is located with a \( \text{dumbSearcher} \)) and \( M_9 \) permit a robot to update its variable \( \text{idMin} \) with the identifier of a robot without copying the value of the variable \( \text{idMin} \) of another robot. Therefore at least one of these rules is necessarily executed at a time, since initially the variables \( \text{idMin} \) of the robots are equal to \(-1\). To execute Rule \( M_7 \) (in the case a \( \text{righter} \) is located with a \( \text{dumbSearcher} \)) or Rule \( M_9 \), a \( \text{dumbSearcher} \) robot must be present in the execution. Only the execution of Rule \( M_6 \) permits to have \( \text{dumbSearcher} \) robots in the execution. This rule is executed when \( \mathcal{R} - 1 \) \( \text{righter} \) robots are on a same node. The \( \mathcal{R} - 1 \) robots that execute this rule, set their variables \( \text{idPotentialMin} \) to the identifier of the robot that becomes \( \text{potentialMin} \) while executing this rule. Moreover if a robot is a \( \text{dumbSearcher} \) in a configuration \( \gamma_t \) at time \( t \) and is still a \( \text{dumbSearcher} \) in the configuration \( \gamma_{t+1} \) then it does not update its variable \( \text{idPotentialMin} \) during time \( t \) (since it executes Rule \( M_{14} \)).

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

Case 3: \( r \) becomes \( \text{min} \) because the condition \( \exists r' \in \text{NodeMate}(), (\text{state}_{r'} \in \{\text{dumbSearcher}, \text{potentialMin}\} \land \text{id}_{r'} < \text{idPotentialMin}_{r'}) \) is true.

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

Case 4: \( r \) becomes \( \text{min} \) because \( {\text{rightSteps}}_r = 4 \ast \text{id}_r \ast n \).

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

By the predicate \( \text{PotentialMinOrRighter}() \) of Rule \( M_1 \), only \( \text{righter} \) robots or \( \text{potentialMin} \) robots can become \( \text{min} \). By Lemma 2, if, at a time \( t \), a robot is a \( \text{righter} \) or a \( \text{potentialMin} \), then it considers the right direction from the beginning of the execution until the Look phase of time \( t \). Robots that are \( \text{righter} \) robots or \( \text{potentialMin} \) robots in a configuration \( \gamma_t \) at time \( t \) and that are either \( \text{righter} \) or \( \text{potentialMin} \) in the configuration \( \gamma_{t+1} \) increase from 1 their variables \( \text{rightSteps} \) each time an adjacent edge in the right direction to their positions is present (Rules \( M_6 \) and \( M_8 \)). Therefore, by the predicate \( \text{MinDiscovery}() \) of Rule \( M_1 \) a robot \( r_0 \) moves at most during \( 4 \ast \text{id}_r \ast n \) steps in the right direction before being \( \text{min} \).

By Lemma 1, from the time a robot becomes \( \text{min} \), it is either a \( \text{minWaitingWalker} \) or a \( \text{minTailWalker} \). Therefore it can only execute Rules \( \text{Term}_1, \text{Term}_2, K_1, K_2, W_1 \) and \( T_3 \). This implies that once a robot is \( \text{min} \), 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: \( K_2, K_1, W_1 \) and \( T_3 \)). Therefore by the previous paragraph, a \( \text{min} r \) 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 $4 \cdot \text{id}_r \cdot n + n$ 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 $\forall r_1, r_2 \in \mathbb{R}^2, \text{state}_{r_1}, \text{state}_{r_2} \in \{\text{righter, potentialMin}\}^2, |\text{rightSteps}_{r_1} - \text{rightSteps}_{r_2}| \leq n$.

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 $\text{id}_{r_{\text{min}}} < \text{id}_r$, therefore we have $4 \cdot \text{id}_{r_{\text{min}}} \cdot n + n + n < 4 \cdot \text{id}_r \cdot n$. Hence there exists a time at which $r$ meets $r_{\text{min}}$ while $r_{\text{min}}$ is min and $r$ is not yet min. At this time, by the rules of 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$_r \in \{\text{dumbSearcher, awareSearcher}\}$, then, during the Move phase of time $t - 1$, it does not consider the $\bot$ direction.

**Proof.** Consider a robot $r$ such that, at time $t$, state$_r \in \{\text{dumbSearcher, awareSearcher}\}$.

While executing 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 $M_5$, $M_6$ or $M_7$.

Consider first the case where $r$ executes Rule $M_5$ at time $t - 1$. Only righter robots can execute Rule $M_6$. While executing Rule $M_6$, $r$ becomes a dumbSearcher (since while executing this rule a righter can become either a dumbSearcher or a potentialMin). Moreover, while executing Rule $M_6$, 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 $M_5$ or $M_7$. 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 $M_5$ or $M_7$, 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 $\bot$ 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 $M_9$, $M_{10}$ or $M_{11}$. 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 $M_9$ and $M_{10}$, 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 $\bot$ during the Move phase of time $t - 1$, hence the lemma is proved.

It remains the case where $r$ executes Rule $M_{11}$ at time $t - 1$ while it is alone on its node. In this case, while executing Rule $M_{11}$, $r$ 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 $\bot$. 

13
Finally, we prove the other main lemma of this phase: we prove that \( r_{\text{min}} \) is aware, in finite time, that it possesses the minimum identifier among all the robots of the system.

**Lemma 5.** In finite time \( r_{\text{min}} \) becomes min.

*Proof.* Assume that \( r_{\text{min}} \) does not become min. By Lemma 3, only \( r_{\text{min}} \) 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 \( \gamma_t \) at a time \( t \) where there are only righter robots, it is not possible to have awareSearcher in the configuration \( \gamma_{t+1} \). A robot can become a dumbSearcher or a potentialMin only when Rule \( M_6 \) is executed. This rule is executed when \( R - 1 \) 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 \( M_6 \) is never executed.**

In this case all the robots are righter robots during the whole execution, and execute therefore Rule \( M_6 \) at each instant time. While executing Rule \( M_6 \), 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 \( r_{\text{min}} \) does not become min, then by Rule \( M_1 \) and predicate MinDiscovery(), \( r_{\text{min}} \) cannot succeed to have its variable rightSteps equals to \( 4 + id_{r_{\text{min}}} + n \), otherwise the lemma is true. Therefore it exists a time at which \( r_{\text{min}} \) 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 COTring, 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 \( M_8 \)), it exists a time at which \( R - 1 \) righter robots are on a same node, cases 2 and 3 are then considered.

**Case 2: Rule \( M_6 \) is executed but \( r_{\text{min}} \) is not among the \( R - 1 \) righter robots that execute it.**

While executing Rule \( M_6 \), among the \( R - 1 \) 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. By definition we have id > id 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 \( R - 1 \) righter are needed to execute Rule \( M_6 \), this rule can be executed only once. Note that if a robot is a dumbSearcher (resp. a potentialMin) in a configuration \( \gamma_t \) at time \( t \) and is still a dumbSearcher (resp. a potentialMin) in the configuration \( \gamma_{t+1} \) then it does not update its variable idPotentialMin during time \( t \) since it executes Rule \( M_{11} \) (resp. \( M_8 \)).

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

Similarly, \( r_{\text{min}} \) as a righter cannot meet a dumbSearcher \( r_d \). Indeed, only Rule \( M_6 \) permits a robot to become a dumbSearcher. Therefore, since idPotentialMin = id > id min, if \( r_{\text{min}} \) meets a dumbSearcher, then by Rule \( M_1 \) 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 \( \gamma_t \) at a time \( t \) where there are only righter robots, it is not possible to have awareSearcher in the configuration \( \gamma_{t+1} \). Therefore awareSearcher can be present in the execution only after the execution of Rule \( M_6 \). In the case where there is not yet awareSearcher, a robot can become an awareSearcher only if a righter meets a dumbSearcher (Rules \( M_9 \) and \( M_7 \)). However after the execution of Rule \( M_6 \), only \( r_{\text{min}} \) is a righter, and as explained in the previous paragraph, if \( r_{\text{min}} \) as a righter meets a dumbSearcher there is a contradiction.

Since there is no awareSearcher and since \( r_{\text{min}} \) as a righter cannot meet neither potentialMin nor dumbSearcher, this implies that \( r_{\text{min}} \) stays a righter during the whole execution and therefore executes Rule \( M_8 \) at each instant time. By the same arguments as the one used in case 1, necessarily it exists a time at which \( r_{\text{min}} \) 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 \( M_8 \) 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 \( r_{\text{min}} \), all the other edges are infinitely often present. Therefore, in finite time, the potentialMin is located on the same node as \( r_{\text{min}} \), which is a contradiction.
Case 3: Rule $M_6$ is executed and $r_{\text{min}}$ 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 $M_6$, $r_{\text{min}}$ becomes potentialMin, since $r_{\text{min}}$ possesses the minimum identifier among all the robots of the system.

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

Similarly, $r_{\text{min}}$ 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 $M_9$ and $M_7$). 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 $id_{\text{min}}$. 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 $M_{10}$. 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 $r_{\text{min}}$, as a potentialMin, meets an awareSearcher, by Rule $M_1$ and predicate MinDiscovery(), the lemma is true, which is a contradiction.

Therefore, as a potentialMin, $r_{\text{min}}$ executes Rule $M_8$ at each instant time. By the same arguments as the one used in case 1, necessarily it exists a time at which $r_{\text{min}}$ 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 $r_{\text{min}}$. At the time where Rule $M_6$ is executed there is a righter robot $r$ in the execution. In the case $r$ never meets a dumbSearcher robot it executes Rule $M_8$ 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 $r_{\text{min}}$, 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 $M_7$ (refer to the predicate RighterWithSearcher()) and all the dumbSearcher robots located with $r$ including $r'$ execute Rule $M_9$ (by the predicate DumbSearcherMinRevelation() and since $id_r > id_{r_{\text{min}}}$). 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 $M_7$ and $M_9$ are executed, there are at least two awareSearcher considering two opposite directions. Moreover once $M_7$ and $M_9$ are executed, except $r_{\text{min}}$ there are only dumbSearcher and awareSearcher robots in the execution. When a dumbSearcher robot meets an awareSearcher robot, it executes Rule $M_{10}$ and therefore becomes awareSearcher robot and executes the function SEARCH. An awareSearcher executes Rule $M_{11}$ 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 $\bot$ by Lemma 4). All this implies that in finite time an awareSearcher robot is located on the same node as $r_{\text{min}}$. Therefore by Rule $M_1$ and predicate MinDiscovery(), $r_{\text{min}}$ becomes min.

By Lemmas 3 and 5, we can deduce the following corollary which proves the correctness of Phase $\mathcal{R}$.

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

### 5.1.2 Proofs of Correctness of Phase $K$

Once $r_{\text{min}}$ completes Phase $\mathcal{R}$, 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 $r_{\text{min}}$ 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 $r_{\text{min}}$, as min, is waiting. In the previous subsection, we prove that, in finite time, only $r_{\text{min}}$ becomes min (Corollary 4) and that once $r_{\text{min}}$ 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 $K_1$). Moreover, by Rule $K_2$, $r_{\text{min}}$, as a minWaitingWalker, does not move until $\mathcal{R} - 3$ waitingWalker robots are on its node. Therefore, as minWaitingWalker, $r_{\text{min}}$ is, as expected, always on the same node. Let $u$ be the node on which $r_{\text{min}}$, as a minWaitingWalker, is located. Let $t_{\text{min}}$ be the time at which $r_{\text{min}}$ becomes a minWaitingWalker robot. In this subsection, we consider the execution from time $t_{\text{min}}$.

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 \( R - 3 \) 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 GDG, as long as there is no towerMin, \( r_{\text{min}} \) stays a minWaitingWalker robot.

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

Observation 3. There exists no rule in 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 \( R - 3 \) waitingWalker robots. Once a towerMin is formed, the \( R - 2 \) \((R - 2 \geq 2)\) robots involved in the towerMin execute Rule K1. While executing this rule the robot with the maximum identifier among the \( R - 2 \) 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 \( r_{\text{min}} \) 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 \( r_{\text{min}} \) when state\( r_{\text{min}} = \text{minWaitingWalker} \) and neither the waitingWalker robots nor \( r_{\text{min}} \), as a minWaitingWalker, move.

Proof. By the rules of GDG, as long as there is no towerMin, \( r_{\text{min}} \) is minWaitingWalker. While \( r_{\text{min}} \) is the minWaitingWalker, it executes Rule K2 at each instant time. While executing this rule, \( r_{\text{min}} \) considers the \( \perp \) direction and therefore does not move.

Only Rule K3 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 \( r_{\text{min}} \) can be minWaitingWalker. While executing Rule K3, \( r \) 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 \( r_{\text{min}} \), as a minWaitingWalker, is located.

While \( r \) is a waitingWalker robot, as long as there is no towerMin in the execution, it executes Rule K2 at each instant time. Therefore \( r \) does not move. As noted previously, the location where \( r \) stops moving is the location where \( r_{\text{min}} \), as the minWaitingWalker, is located.

Once a towerMin is present in the execution the waitingWalker robots and the minWaitingWalker composing this towerMin execute Rule K1. 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 \( r_{\text{min}} \) can be minWaitingWalker, all waitingWalker robots are located on the same node as \( r_{\text{min}} \) when state\( r_{\text{min}} = \text{minWaitingWalker} \) and neither the waitingWalker robots nor \( r_{\text{min}} \), 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 M6 permits a robot to become a potentialMin robot. Rule M6 is executed when \( R - 1 \) righter robots are located on a same node. When these \( R - 1 \) righter robots execute Rule M6, 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 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, \( r_{\text{min}} \) 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 K_1. The only way to be a minWaitingWalker is to execute Rule M_1. Only righter robots or potentialMin robots can execute Rule M_1 (refer to predicate PotentialMinOrRighter()). \hfill \Box

The 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 M_6 must be executed. This rule is executed when R \(-\) 1 righter are located on a same node. While executing Rule M_6, among the 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 R \(-\) 1 righter that execute Rule M_6, otherwise it cannot be a righter before being min. Similarly thanks to Observation 2, the R \(-\) 1 robots that execute Rule 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. \hfill \Box

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 R \(-\) 1 righter located on a same node after time t. However, in order to have a potentialMin in the execution, Rule M_6 must be executed. This rule is executed only if R \(-\) 1 righter are located on a same node. Therefore there is no potentialMin in the execution after time t. \hfill \Box

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 M_6 must be executed. This rule is executed when R \(-\) 1 righter are located on a same node. While executing Rule M_6, among the 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 R \(-\) 1 robots can become righter anymore after time t.

Consider then the two following cases.

Case 1: t \gt t_min.

By Observation 2, r_min cannot be a righter after time t_min. Therefore r_min is not among the R \(-\) 1 robots that execute Rule M_6, and hence, by (\*), after time t, there is no more righter in the execution.

Case 2: t \leq t_min.

By (\*), r_min cannot be among the R \(-\) 1 righter that execute Rule 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. \hfill \Box

The following lemma is an extension of Lemma 4. While Lemma 4 is true only when all the robots are executing Phase \(\mathcal{M}\), the following lemma is true whether the robots are executing Phase \(\mathcal{M}\) or Phase \(\mathcal{K}\).

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 \(\bot\) 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 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 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 $t-1$, $r$ 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 $K_4$, $M_5$, $M_6$ or $M_7$.

When a robot executes Rule $K_4$, 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 $t-1$, $r$ 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 $M_9$, $M_{10}$ or $M_{11}$. 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 $K_3$). 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 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 $M_{11}$, 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 $M_{11}$ 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 $M_{10}$ (if it is on the same node as an awareSearcher) and hence becomes an awareSearcher robot and executes the function SEARCH, Rule $M_9$ (if it is on the same node as a righter) and hence becomes an awareSearcher and executes the function SEARCH, or Rule $M_{11}$ 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 $M_5$ (if it is on the same node as an awareSearcher) and hence becomes an awareSearcher and executes the function Search, or Rule $M_4$ 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.

(iii) 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 $M_7$ and hence becomes an awareSearcher and executes the function Search.

(iii) 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.

Note that if there is a potentialMin while $r_{\text{min}}$ is minWaitingWalker, then it possesses the minimum identifier among all the robots not located on node $u$. Indeed, only Rule $M_6$ permits a robot to become a potentialMin. For this rule to be executed, $R - 1$ righter robots must be located on a same node. While executing this rule the robot with the minimum identifier among the $R - 1$ 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 $r_{\text{min}}$ possesses the minimum identifier among all the robots of the system, $r_{\text{min}}$ does not execute Rule $M_6$. Therefore, while $r_{\text{min}}$ is minWaitingWalker, the potentialMin possesses the minimum identifier among all the robots not located on node $u$. Thus, when a potentialMin executes Rule $M_4$ 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 $\text{state}_{r'} \in \{\text{potentialMin}, \text{dumbSearcher}, \text{awareSearcher}\}$ 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 $\{\text{potentialMin}, \text{dumbSearcher}, \text{awareSearcher}\}$ 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 $\text{state}_{r''} \in \{\text{potentialMin}, \text{dumbSearcher}, \text{awareSearcher}\}$ 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 $\text{Seg}(u, v) \cup \{v\}$ or on nodes in $\text{Seg}(w, u) \cup \{w\}$.

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

The edge $e$ is an eventual missing edge. It can exist only one eventual missing edge in $\text{COTring}$. Therefore all the edges between the nodes in $\{w\} \cup \text{Seg}(u, v) \cup \{v\}$ 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 $\text{state}_{r''} \in \{\text{potentialMin}, \text{dumbSearcher}, \text{awareSearcher}\}$, 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) ∪ {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,v ∈ {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 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, 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 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, r_min is a potentialMin.**

For a robot to become a potentialMin, Rule M_4 must be executed. This rule is executed when R − 1 righter are located on a same node. While executing this rule, among the R − 1 righter located on a same node, one becomes potentialMin while the other become dumbSearcher. By Observation 2, none of these 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 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 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 \( \text{state}_{r'}, \text{state}_{r''} \in \{\text{awareSearcher}, \text{dumbSearcher}\}^2 \) (all the other robots are \( \text{minWaitingWalker} \) and \( \text{waitingWalker} \) robots and are located on node \( u \)). Note that \( r' \) (resp. \( r'' \)) cannot be located on node \( u \), otherwise, by Rule \( K_3 \), a \( \text{towerMin} \) is formed. Therefore, \( r' \) and \( r'' \) always have their variable state in \( \{\text{dumbSearcher}, \text{awareSearcher}\} \).

When a righter robot is located on the same node as an \( \text{awareSearcher} \) or as a \( \text{dumbSearcher} \), it executes Rule \( M_7 \) and becomes an \( \text{awareSearcher} \). Similarly, if a righter is on the same node as a \( \text{minWaitingWalker} \) while the adjacent right edge to its position is present, then the righter robot executes Rule \( K_4 \) and becomes an \( \text{awareSearcher} \). Therefore, as highlighted previously, these situations cannot happen, otherwise a \( \text{towerMin} \) 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 \( M_8 \) and therefore stays a righter and considers the right direction. Therefore, since an isolated righter robot always executes Rule \( M_8 \), 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 \( \text{awareSearcher} \) or a \( \text{dumbSearcher} \) is isolated it executes Rule \( M_{11} \), hence executes the function \( \text{SEARCH} \), 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 \( \{\text{dumbSearcher}, \text{awareSearcher}\}^2 \), and since \( r' \) and \( r'' \) cannot be located on node \( u \) and cannot be located with \( r \), if a \( \text{dumbSearcher} \) is located on the same node as an \( \text{awareSearcher} \) or if an \( \text{awareSearcher} \) (resp. a \( \text{dumbSearcher} \)) is located on the same node as another \( \text{awareSearcher} \) (resp. \( \text{dumbSearcher} \)), 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 \( \text{dumbSearcher} \) is on the same node as an \( \text{awareSearcher} \) it executes Rule \( M_{10} \), hence it becomes an \( \text{awareSearcher} \) and executes the function \( \text{SEARCH} \). When an \( \text{awareSearcher} \) is on the same node as a \( \text{dumbSearcher} \) it executes Rule \( M_{11} \) and hence executes the function \( \text{SEARCH} \). Since \( r' \) and \( r'' \) have distinct identifiers, when an \( \text{awareSearcher} \) and a \( \text{dumbSearcher} \) are on a same node, they both execute the function \( \text{SEARCH} \), therefore one considers the right direction, while the other one considers the left direction. Similarly, if two \( \text{awareSearcher} \) (resp. \( \text{dumbSearcher} \)) robots are on the same node, they both execute Rule \( M_{11} \) and hence the function \( \text{SEARCH} \), 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 \( \text{Seg}(u,v) \) or in \( \text{Seg}(w,u) \), 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 \( \text{Seg}(u,v) \). 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 \( \text{towerMin} \) is formed.

Secondly, assume that \( r' \) and \( r'' \) are on nodes in \( \text{Seg}(u,v) \). 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 \( \text{towerMin} \) is formed.

Thirdly, assume that \( r \) and \( r'' \) are on nodes in \( \text{Seg}(v,u) \). 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 \( \text{Seg}(v,u) \), to \( r' \) and \( r'' \) when there are on nodes in \( \text{Seg}(u,v) \), we succeed to prove that in finite time at least one of them reaches node \( u \), making Rule \( \text{Term}_2 \) 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 $GDG$ 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 $M_6$. 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 $t_{\text{min}}$, by the rules of $GDG$, there are only righter in the execution. Indeed, by Corollary 4, only $r_{\text{min}}$ can be minWaitingWalker and it becomes minWaitingWalker at time $t_{\text{min}}$. Moreover, the only way for a robot to become waitingWalker is to execute Rule $K_4$. In the case where there is no potentialMin in the execution, only an awareSearcher located with $r_{\text{min}}$, 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 $M_7$), or to be a righter and to be located with $r_{\text{min}}$, as a minWaitingWalker, while an adjacent right edge to their location is present (refer to Rule $K_4$). Since initially all the robots are righter, the first awareSearcher of the execution can be present only thanks to the execution of Rule $K_4$.

All this implies that, even after time $t_{\text{min}}$, as long as no righter robot is on node $u$ with $r_{\text{min}}$, as a min-WaitingWalker, 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 $R - 1$ righter. Moreover, this implies that as long as the situation described has not happened, all the righter robots only execute Rule $M_8$, hence always consider the right direction.

Consider the execution just after time $t_{\text{min}}$. In this context, necessarily, in finite time, there exists a righter robot $r$ that succeeds to reach $u$ (while $r_{\text{min}}$ 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 $R - 1$ righter robots reach in finite time the same extremity of $e$. Thus, Rule $M_6$ 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 Term$_1$ is executed, which leads to a contradiction.

Therefore there exists a time $t'$ at which $r$ executes Rule $K_4$. 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 $r_{\text{min}}$, as a minWaitingWalker, it executes Rule $K_3$ 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 $M_{11}$, 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 $r_{\text{min}}$, as a minWaitingWalker, such that there is no adjacent right edge to $u$, it executes Rule $M_8$, hence it stays a righter and considers the right direction.

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

(vi) If a righter robot is on a node different from node $u$ with an awareSearcher, it executes Rule $M_7$ 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 \((\ast)\).

\((\ast\ast)\) 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 \( S_r \) 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 \( S_r \) can be located on the same node as an awareSearcher, otherwise, by \((vi)\), they become awareSearcher. Therefore, all the robots of \( S_r \) 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 \( S_r \), 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 \( S_r \), 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 \( S_r \) are awareSearcher robots. Hence, by \((\ast\ast)\), 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 \( S_r \) can be located on the same node as an awareSearcher, otherwise, by \((vi)\), they become awareSearcher. Therefore, all the robots of \( S_r \) 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 \( S_r \) succeed to reach node \( u \), and stay on node \( u \) (refer to \((iv)\)). Necessarily, \(|S_r| < R - 2\), otherwise Rule Term2 is executed. At the time at which all the robots of \( S_r \) are on node \( u \), by \((\ast)\), we know that at least one awareSearcher, on a node \( v \), considers the right direction. By \((vi)\), none of the righter of \( S_r \) 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 \( S_r \) become awareSearcher, hence by \((\ast\ast)\), the lemma is proved.

**Case 2.2: \( x \neq u \).**

None of the robots of \( S_r \) can be located on the same node as an awareSearcher, otherwise, by \((vi)\), they become awareSearcher. Therefore, none of the robots of \( S_r \) can be located on \( \text{Seg}(u, x) \cup \{x\} \), otherwise, in finite time, by \((iv)\) they are located on node \( x \). However, once all the robots of \( S_r \) are on node \( x \), by \((\ast)\), 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 \( S_r \) are on nodes in \( \text{Seg}(x, u) \). 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 \( S_r \), 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 \( S_r \) are awareSearcher robots. Hence, by \((\ast\ast)\), 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 one towerMin in the whole execution. Necessarily, as explained above when there is no potential-Min in the execution, in order to have a towerMin, a righter must become an awareSearcher while executing Rule K4. The property \((\ast)\) 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 \((\ast)\), this implies that there is an awareSearcher at time \( t_{tower} \). However, by definition, a towerMin is composed of one minWaitingWalker and \( R - 3 \) waitingWalker, therefore, since there are \( R \) 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 $\mathcal{W}$ and $\mathcal{T}$

The combination of Phases $\mathcal{W}$ and $\mathcal{T}$ of $\mathcal{GDG}$ permit to solve $\mathcal{GEW}$ in $\mathcal{COT}$ rings. Since $\mathcal{GEW}$ is divided into a safety and a liveness property, to prove the correctness of Phases $\mathcal{W}$ and $\mathcal{T}$, 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 $\mathcal{GEW}$, 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 $\mathcal{GEW}$ 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 $t_{\text{tower}}$ 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 $t_{\text{tower}}$, 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 $\text{state}$ at time $t_{\text{tower}}$ of each of these robots.

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

Proof. Until the Look phase of time $t_{\text{tower}}$, 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 $t_{\text{tower}}$.

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

By Corollary 5, at time $t_{\text{tower}}$, 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 $t_{\text{tower}}$. In this case by Corollary 5, $r_2$ cannot be a righter at time $t_{\text{tower}}$. Moreover, in this case, by Lemma 10, $r_2$ cannot be a potentialMin at time $t_{\text{tower}}$.

Now assume without lost of generality that $r_1$ is a potentialMin robot at time $t_{\text{tower}}$. By Lemmas 8 and 10, $r_2$ cannot be a potentialMin nor a righter at time $t_{\text{tower}}$.

This prove the lemma.

In the following lemma, we prove a property on Rules Term$\mathcal{1}$ and Term$\mathcal{2}$ that helps us to prove that $\mathcal{GDG}$ solves $\mathcal{GEW}$ in $\mathcal{COT}$ rings.

Lemma 18. If a robot $r$, on a node $x$, at a time $t$, executes Rule Term$\mathcal{1}$ (resp. Term$\mathcal{2}$), then there are $\mathcal{R}$ (resp. $\mathcal{R} - 1$) robots on node $x$ at time $t$ and they all execute Rule Term$\mathcal{1}$ (resp. Term$\mathcal{2}$) at time $t$ (if they are not already terminated).

Proof. If a robot $r$, on a node $x$, executes Rule Term$\mathcal{1}$ (resp. Term$\mathcal{2}$) at a time $t$, by the predicate $\mathcal{GEW}(r)$ (resp. $\mathcal{GEW}(x)$), there are $\mathcal{R}$ (resp. $\mathcal{R} - 1$) robots on $x$ at time $t$. Moreover, if the predicate $\mathcal{GEW}(r)$ (resp. $\mathcal{GEW}(x)$) 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 Term$\mathcal{1}$ (resp. Term$\mathcal{2}$) 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 Term$\mathcal{1}$ or Rule Term$\mathcal{2}$.

By Lemma 18, if a robot $r$, on a node $x$, at a time $t$, executes Rule Term$\mathcal{1}$, then there are $\mathcal{R}$ robots on node $x$ at time $t$ and they all execute Rule Term$\mathcal{1}$ at time $t$ (if they are not already terminated). Therefore, in the case where $r$ executes Rule Term$\mathcal{1}$ 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 Term$\mathcal{2}$, then there are $\mathcal{R} - 1$ robots on node $x$ at time $t$ and they all execute Rule Term$\mathcal{2}$ at time $t$ (if they are not already terminated). Therefore,
in the case where $r$ executes Rule $\text{Term}_2$ 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 $\text{Term}_1$ or Rule $\text{Term}_2$. 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 $\text{Term}_2$ at time $t$ on node $x$. Indeed, to execute Rule $\text{Term}_2$ 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 $\mathcal{K}$ or Phase $\mathcal{K}$, 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 $\text{state}_r \in \{\text{dumbSearcher, awareSearcher}\}$, 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 $t$, $\text{state}_r \in \{\text{dumbSearcher, awareSearcher}\}$.

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 $\text{K}_4$, $\text{M}_2$, $\text{M}_3$, $\text{M}_5$, $\text{M}_6$ or $\text{M}_7$.

When a robot executes Rule $\text{M}_2$, it calls the function $\text{BECOMEAWARESEARCHER}$. When a robot executes the function $\text{BECOMEAWARESEARCHER}$, it sets its direction to the right direction, therefore the lemma is true in this case.

A robot executes Rule $\text{M}_3$ 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 $\text{M}_2$. By the rules of $\mathcal{GDG}$, a headWalker only considers the $\perp$ direction or the right direction. Indeed, a headWalker can only execute Rules $\text{T}_2$, $\text{T}_3$ and $\text{W}_1$. While executing Rule $\text{T}_2$, a headWalker becomes a leftWalker and considers the $\perp$ direction. While executing Rule $\text{T}_3$, a headWalker considers the $\perp$ direction. Finally, while executing Rule $\text{W}_1$, a headWalker considers either the right direction or the $\perp$ direction. Therefore, even if, after the execution of Rule $\text{M}_3$, $r$ 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 $\text{M}_2$, $\text{M}_3$, $\text{M}_9$, $\text{M}_{10}$ or $\text{M}_{11}$.

We can use the arguments of Case 1 to prove that while executing Rule $\text{M}_2$ or $\text{M}_3$, 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 $\mathcal{G}_{EW}$ in $\text{COT}$ 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 $\mathcal{G}_{EW}$ in $\text{COT}$ rings.
Proof. By contradiction, assume that GDG does not solve the liveness of $G_{EW}$ in COT rings. Since the execution of Rules Term$_1$ and Term$_2$ permits a robot to terminate its execution, by Lemma 18, this implies that there exists a COT ring such that, during the execution of GDG, neither Rule Term$_1$ nor Rule Term$_2$ is executed. Consider the execution of GDG on that ring.

By Corollary 5, there exists a time $t$ at which a towerMin is formed. Note that $R-2 \geq 2$ robots are involved in a towerMin. Once a towerMin is formed the $R-3$ waitingWalker and the minWaitingWalker involved in this towerMin execute Rule $K_1$. While executing this rule, the robot $r$ with the maximum identifier among the $R-2$ 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 GDG, only $r$ can be headWalker and only $r_{min}$ can be minTailWalker during the execution.

There is no rule in GDG permitting a tailWalker or a minTailWalker robot to become another kind of robot. A tailWalker and a minTailWalker can only execute Rules $T_3$ and $W_1$. By the rules of 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 $G_{EW}$ cannot be solved, then it is not possible for $r$ to become a leftWalker. Indeed, a headWalker can only execute Rules $T_3$, $T_3$ and $W_1$. Note that, by the rules of GDG, after the execution of Rule $K_1$, the headWalker and the tail both execute Rule $W_1$. Therefore, since the headWalker and the tail start the execution of Rule $W_1$ on the same node at the same time, by the rules of GDG, while the headWalker is executing Rule $T_3$ or Rule $W_1$, if the tail is not on the same node as the headWalker, it is either executing Rule $W_1$ 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 $T_3$, and therefore becomes a leftWalker, then this implies that during time $t' - 1$ it is executing either Rule $T_3$ or Rule $W_1$ 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 $W_1$). Since only Rules Term$_1$ and Term$_2$ permit a robot to terminate its execution, by Lemma 18, this implies that the tail has executed Rule Term$_2$, which leads to a contradiction with the fact that GDG does not solve the liveness of $G_{EW}$.

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

Call $r_1$ and $r_2$ the two robots that are not involved in the towerMin at time $t$. Since, by contradiction, neither Rule Term$_1$ nor Rule 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 $M_8$).

If a potentialMin is located only with an awareSearcher, it becomes an awareSearcher and it executes the function SEARCH (Rule $M_5$). An isolated righter stays a righter and considers the right direction (Rule $M_8$).

If a righter is located only with a dumbSearcher (resp. an awareSearcher), it becomes an awareSearcher and executes the function SEARCH (Rule $M_5$).

If a dumbSearcher is located only with a righter, it becomes an awareSearcher and executes the function SEARCH (Rule $M_5$).

If a dumbSearcher is located only with a potentialMin it stays a dumbSearcher and executes the function SEARCH (Rule $M_{11}$). 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 $M_6$ permits a robot to become potentialMin or dumbSearcher. This rule is executed when $R-1$ righter are located on a same node. While executing Rule $M_6$, among the $R-1$ righter, the one with the minimum identifier becomes potentialMin while the others become dumbSearcher. By Observation 2, Rule $M_6$ 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 $M_{11}$).
If a dumbSearcher is located only with an awareSearcher, it becomes an awareSearcher and it executes the function SEARCH (Rule 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 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 \( \text{state}_{r_1} \in \{ \text{righter}, \text{potentialMin}, \text{dumbSearcher}, \text{awareSearcher} \} \) and \( \text{state}_{r_2} \in \{ \text{dumbSearcher}, \text{awareSearcher} \} \).

Now, consider the possible interactions between the headWalker and \( r_1 \) and/or \( r_2 \) when \( \text{state}_{r_1} \in \{ \text{righter}, \text{potentialMin}, \text{dumbSearcher}, \text{awareSearcher} \} \) and \( \text{state}_{r_2} \in \{ \text{dumbSearcher}, \text{awareSearcher} \} \).

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 M_{3}, hence it becomes an awareSearcher and stops to move.

\((\ast)\) 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 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 \( \text{state}_{r_1} \in \{ \text{righter}, \text{potentialMin}, \text{dumbSearcher}, \text{awareSearcher} \} \) and \( \text{state}_{r_2} \in \{ \text{dumbSearcher}, \text{awareSearcher} \} \).

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 M_{4} and becomes a tailWalker. From this time, by the function BECOMETAILWALKER and the rules of G_DG, the robot belongs to the tail.

We assume that there exists an eventual missing edge. Call \( t^* \) the time after the execution of Rule K_{1} and after the time when the eventual missing edge is missing forever. Consider the execution from \( t^* \). Since Rule 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 W_{1} and 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 W_{1} or Rule T_{3} at time \( t^* \), and the tail executes Rule 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 W_{1}), or because it executes the function STOPMOVING (Rule T_{3}). Similarly, after time \( t^* \), the tail does not move, since it tries to join the headWalker considering the right direction (Rule W_{1}), but the edge is missing forever.

Since there is at most one eventual missing edge in a 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 \( G_{EW} \) 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 T_{3}, or because they are executing Rule 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 \( G_{EW} \) 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 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 extremities 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 $G_{EW}$ solved. Similarly, if the headWalker and the tail are located, at time $t_n$, on one of the extremities 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 $G_{EW}$ 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 extremities 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 $W_1$ 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 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 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 $T_3$, therefore they succeed to execute Rule $W_1$ 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 $W_1$ 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 $t_i$ and $t_n$. 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 $t_i$ 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 $t_i$, since the two robots are located on a same node, by the movements of the robots described, during the Move phase of time $t_i$ one considers the right direction while the other one considers the left direction. By assumption, between times $t_i + 1$ and $t_n$, $r_1$ and $r_2$ are not located on a same node. Moreover, as explained previously, between times $t_i + 1$ and $t_n$, neither $r_1$ nor $r_2$ can be located with the tail, otherwise Rule Term$_2$ is eventually executed. Besides, between times $t_i + 1$ and $t_n$ the robot that considers the left direction during the Move phase of time $t_i$ 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 $t_i$ to be located with the headWalker between times $t_i + 1$ 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 $G_{EW}$ 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 $t_i$ is located with the headWalker. Therefore, from time $t_i + 1$ 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 $t_i$ 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 $t_i$ 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 \( G_{EW} \) solved.

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

**Theorem 2.** \( GDG \) solves \( G_{EW} \) in \( COT \) rings.

### 5.2 What about \( GDG \) executed in \( AC \), \( RE \), \( BR \) and \( ST \) rings?

In the previous subsection we prove that \( GDG \) solves \( G_{EW} \) in \( COT \) rings. In this subsection, we consider \( AC \), \( RE \), \( BR \) and \( ST \) rings. For each of these classes of dynamic rings, we give the version of gathering \( GDG \) solves in it.

First, we consider the case of \( AC \) rings. In the following theorem, we prove that \( GDG \) solves \( G_{W} \) in \( AC \) rings.

**Theorem 3.** \( GDG \) solves \( G_{W} \) in \( AC \) rings in \( O(id_{\min} \cdot n^2 + R \cdot n) \) rounds.

**Proof.** By Corollary 2, \( GDG \) solves \( G_{EW} \) in \( COT \) rings, since \( AC \subset COT \), this implies that \( GDG \) also solves \( G_{EW} \) in \( AC \) rings. Therefore, to prove that \( GDG \) solves \( G_{W} \) in \( AC \) rings, it stays to prove that each phase of \( GDG \) is bounded.

**Phase 9:** By Corollary 4, only \( r_{\min} \) becomes \( min \) in finite time. By the rules of \( GDG \), when \( r_{\min} \) becomes \( min \), it is first \( minWaitingWalker \) before being \( minTailWalker \) (since only a \( minWaitingWalker \) can become a \( minTailWalker \) while executing Rule \( K_4 \)). Therefore, since only Rule \( M_1 \) permits a robot to become \( minWaitingWalker \), by the predicate \( MinDiscovery() \) of this rule, \( r_{\min} \) becomes \( min \) either because it moves during \( 4 \cdot n \cdot id_{\min} \) 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 \(-1 \). Therefore, in this last case, either \( r_{\min} \) possesses a variable \( idPotentialMin \) different from \(-1 \), or \( r_{\min} \) meets a robot \( r \) such that \( idPotentialMin_r \) is different from \(-1 \) (since a \( potentialMin \) and a \( dumbSearcher \) have their variable \( idPotentialMin \) different from \(-1 \) (Rule \( M_6 \)) and since, while executing \( GDG \), each time the variable \( idMin \) of a robot is set with a variable different from \(-1 \), 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 \( M_6 \) is never executed.**

By the rules of \( GDG \), 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 \( 4 \cdot n \cdot id_{\min} \) steps in the right direction (since \( righter \) robots have their variables \( idPotentialMin \) equal to \(-1 \)). Second, in this context, as long as \( r_{\min} \) is not \( min \), all the \( righter \) always consider the right direction (Rule \( M_5 \)). 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 \( AC \) ring at most one edge can be missing at each instant time, Rule \( M_6 \) is executed. Therefore in case 1.1 \( r_{\min} \) becomes \( min \) in at most \( 4 \cdot id_{\min} \cdot n \cdot n \) rounds.

Now let consider the case where Rule \( M_6 \) 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 \( 4 \cdot n \) 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 \( M_6 \) is executed but \( r_{\min} \) is not among the \( R - 1 \) righter robots that execute it.**

Taking back the arguments of the proof of Lemma 5, we know that Rule \( M_6 \) 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 \( M_6 \), since, by Corollary 4, \( r_{\min} \) necessarily becomes \( min \), since, by Lemma 9, only \( righter \) and \( potentialMin \) can be \( min \), and since only Rule \( M_6 \) 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 \( GDG \), as long as \( r_{\min} \) is not \( min \), there are only one \( righter \), one \( potentialMin \) and \( R - 2 \) dumbSearcher in the execution. Therefore, by the rules of \( GDG \), the \( potentialMin \) is \( potentialMin \) at least until \( r_{\min} \) becomes \( min \). Hence, the
potentialMin executes Rule $M_8$ and thus considers the right direction at least until $r_{\text{min}}$ becomes $\text{min}$. We have assumed that, while it is not yet $\text{min}$, $r_{\text{min}}$ is stuck more than $4*n$ consecutive rounds on a same node. Since $r_{\text{min}}$ is a righter before being $\text{min}$, it is stuck because the adjacent right edge to its position is missing (Rule $M_9$). Therefore, since in an $AC$ ring of size $n$ at least $n - 1$ edges are present at each instant time, either the potentialMin (or a dumbSearcher) meets $r_{\text{min}}$ in at most $n$ rounds. When $r_{\text{min}}$ meets a potentialMin (or a dumbSearcher), it becomes $\text{min}$ by definition of the predicate $\text{MinDiscovery()}$ in Rule $M_1$. Therefore, if it is stuck more than $4*n$ rounds, $r_{\text{min}}$ becomes $\text{min}$ in at most $n$ rounds.

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

In this case, by Rule $M_6$, $r_{\text{min}}$ becomes potentialMin. By Observations 2 and 1, by Corollary 4 and by Lemma 9 $r_{\text{min}}$ is potentialMin until it becomes $\text{min}$. Therefore, $r_{\text{min}}$, while it is not yet $\text{min}$, can be stuck only because the adjacent right edge to its position is missing (Rule $M_9$).

First, consider that at the time when $r_{\text{min}}$, 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 $M_6$ is executed, the robot $r$ that is not among the robots that execute this rule is a righter. Therefore, necessarily, as a righter, meets at least one dumbSearcher at a time $t'$. Indeed, it cannot meet the potentialMin, otherwise $r_{\text{min}}$ is $\text{min}$ (Rule $M_4$), 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 $M_9$). Hence, at time $t'$, $r$ becomes an awareSearcher (Rule $M_7$). Consider an awareSearcher $r_a$ of the execution. By Lemma 4, $r_a$ cannot consider the $\perp$ direction. Moreover, by the rules of $GDG$, as long as there is no $\text{min}$, an awareSearcher executes the function $\text{SEARCH}$ (rule $M_{11}$). Besides, by the proof of Lemma 5 if a robot is not isolated and executes the function $\text{SEARCH}$, then all the robots of its node are or become awareSearcher and execute the function $\text{SEARCH}$. While executing the function $\text{SEARCH}$, an isolated robot does not change its direction. When a robot executes the function $\text{SEARCH}$ 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 $AC$ 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 $r_{\text{min}}$, 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 $r_{\text{min}}$, as a potentialMin, is stuck. Indeed, at most $n$ rounds are needed for an awareSearcher to reach the extremity of the missing edge where $r_{\text{min}}$ is not located. Then, at most $2*n$ other rounds are needed for a dumbSearcher (execution of the function $\text{SEARCH}$, rule $M_{11}$) 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 $r_{\text{min}}$ 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 $r_{\text{min}}$, as a potentialMin, is stuck. Note that $R > 4$, and there are $R - 1$ dumbSearcher/awareSearcher in the execution as long as $r_{\text{min}}$ is not $\text{min}$. Therefore, the previous scenario can effectively happen. When $r_{\text{min}}$ meets an awareSearcher, it becomes $\text{min}$ by definition of the predicate $\text{MinDiscovery()}$ of rule $M_1$. Therefore, $r_{\text{min}}$ becomes $\text{min}$ in at most $4*n$ rounds if it is stuck more than $4*n$ rounds.

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

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

Moreover, by Case 1.1 and the previous paragraph, we can conclude that Phase $\mathcal{M}$ is in $O(id_{\text{min}} * n^2)$ rounds.

Now we consider Phase $\mathcal{K}$ of $\mathcal{GDG}$. In this phase $r_{\text{min}}$ is $\text{min}$ and waits for a $\text{towerMin}$ 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 $\mathcal{K}$: 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 $\text{min}$, $r_{\text{min}}$ is a $\text{righter}$, 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 $M_{\text{s}}$, or it becomes an awareSearcher (Rule $M_{\text{s}}$). While executing Rule $M_{\text{s}}$, 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 $\mathcal{M}$ case 1.3 of this proof) and by Lemma 13, at most $3 * n$ rounds are needed (in $AC$ rings) for a robot $r$ such that $\text{state}_r \in \{\text{potentialMin, dumbSearcher, awareSearcher}\}$ 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 $K_3$, this implies that at most $3 * n$ rounds are necessary for a supplementary waitingWalker to be located on node $u$. Therefore, at most $(R - 3) * 3 * n$ rounds are needed for a towerMin to be formed.

Now consider the case where before being $\text{min}$, $r_{\text{min}}$ 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 $(R - 4) * 3 * n$ rounds are needed for $R - 4$ 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 $M_7$, 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 $u$. Therefore, for all the cases of Case 2.1 of the proof of Lemma 15, at most $(R - 4) * 3 * n + 3 * n$ rounds are needed for a towerMin to be formed.

If we consider Case 2.2 of the proof of Lemma 15, similarly as in the previous case, at most $(R - 4) * 3 * n + 3 * n$ rounds are needed for Rule $\text{Term}_2$ 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 $r_{\text{min}}$ becomes $\text{min}$, it takes at most $n * n$ rounds for a robot $r$ to join the node where $r_{\text{min}}$ is located. Indeed, as long as no robot is on node $u$ with $r_{\text{min}}$, as a minWaitingWalker, all the robots except $r_{\text{min}}$ are righter. By the same arguments as the one used in Phase $\mathcal{M}$ Case 1.1 of this proof, a righter cannot be stuck more than $n$ rounds on the same node, otherwise Rule $M_6$ 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 $\text{Term}_1$ 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 $\mathcal{R} - 3$ \texttt{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 \textbf{Term}_2 to be executed.

Therefore, in this case, at most $n + n + 2 + n + (\mathcal{R} - 3) * 3 + n$ rounds are needed for either Phase $K$ to be achieved or Rule \textbf{Term}_2 to be executed.

Therefore Phase $K$ is bounded. Moreover, by the two previous cases, we can conclude that Phase $K$ is in $O(\mathcal{R} * n + n^2)$ rounds.

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

\textbf{Phase $\mathcal{W}$:} Here we consider the worst execution in terms of times. Therefore, we consider that Rules $\textbf{Term}_1$ and $\textbf{Term}_2$ are executing at the very last moment. The robots $r_1$ and $r_2$ that are not involved in $T$ at time $t_{\text{tower}}$ are such that $\text{state}_{r_1} \in \{\text{righter, potentialMin, dumbSearcher, awareSearcher}\}$ and $\text{state}_{r_2} \in \{\text{dumbSearcher, awareSearcher}\}$. Therefore, as explained previously, each time the headWalker, or the $\text{minTailWalker} / \text{tailWalker}$ robots move from one steps in the right direction, they can be stuck at most during $3 + n$ rounds, otherwise either Rule $\textbf{Term}_1$ or Rule $\textbf{Term}_2$ is executed. Indeed, this is especially the case when the headWalker and the $\text{minTailWalker} / \text{tailWalker}$ 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 $\text{minTailWalker} / \text{tailWalker}$ 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 $\text{minTailWalker} / \text{tailWalker}$ are stuck (and thus for Rule $\textbf{Term}_2$ to be executed). Therefore, Phase $\mathcal{W}$ is achieved in at most $2 + n + (3 + n)$ rounds since the headWalker and the $\text{minTailWalker} / \text{tailWalker}$ robots have to move alternatively during $n$ steps to complete Phase $\mathcal{W}$. In other words, Phase $\mathcal{W}$ is bounded and is in $O(n^2)$ rounds.

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

\textbf{Phase $T$:} Using similar arguments as the one used in Phase $\mathcal{W}$, once the headWalker and the $\text{minTailWalker} / \text{tailWalker}$ stop to move forever, if they are located on a same node, at most $3 + n$ rounds are necessary for Rule $\textbf{Term}_2$ to be executed. In the case where the headWalker and the $\text{minTailWalker} / \text{tailWalker}$ stop to move forever, if they are located on different nodes, at most $2 + n$ rounds are necessary for Rule $\textbf{Term}_2$ 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 $t_{\text{tower}}$ to be located on the node where the $\text{minTailWalker} / \text{tailWalker}$ 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 $\text{minTailWalker} / \text{tailWalker}$ are located). Therefore, Phase $T$ is bounded and is in $O(n)$ rounds.

In conclusion each of the four phases of algorithm $\mathcal{GDG}$ are bounded when executed in an $\mathcal{AC}$ ring, therefore $\mathcal{GDG}$ solves $G_W$ in $\mathcal{AC}$ rings. Moreover, $\mathcal{GDG}$ solves $G_W$ in $\mathcal{AC}$ rings in $O(id_{\min} * n^2 + \mathcal{R} * n)$ rounds. \hfill $\square$

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

\textbf{Theorem 4.} $\mathcal{GDG}$ solves $G_E$ in $\mathcal{RE}$ rings.

\textbf{Proof.} By Corollary 2, $\mathcal{GDG}$ solves $G_{EW}$ in COT rings, therefore it solves the safety and the liveness of $G_{EW}$ in COT rings. Since $\mathcal{RE} \subset COT$, $\mathcal{GDG}$ also solves the safety and the liveness of $G_{EW}$ 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 $G_E$ in $\mathcal{RE}$ rings, this implies that it exists a robot $r$ that never terminates its execution.

Call \textit{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 $\textbf{Term}_1$ or Rule $\textbf{Term}_2$. By Lemma 18, for a \textit{towerTermination} to be formed at time $t$, Rule $\textbf{Term}_2$ has to be executed at this time.
By the predicate of Rule $\text{Tern}_2$, $r_{\text{min}}$ belongs to the $\text{towerTermination}$. By Lemma 18, all the robots that are located on the same node as $r_{\text{min}}$ at time $t$ belong to the $\text{towerTermination}$.

Call $w$ the node where the $\text{towerTermination}$ is located at time $t$.

Note that $r$ cannot be located on node $w$ after time $t$ included, otherwise it executes Rule $\text{Tern}_3$ and the lemma is proved.

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

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

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

Therefore, at time $t$, $r$ can be either a $\text{righter}$, a $\text{potentialMin}$, a $\text{dumbSearcher}$, an $\text{awareSearcher}$, a $\text{headWalker}$ or a $\text{leftWalker}$ robot.

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

An isolated $\text{righter}$ or an isolated $\text{potentialMin}$ only executes Rule $M_8$. While executing this rule, a robot considers the right direction and stays a $\text{righter}$ or a $\text{potentialMin}$. 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 $\text{dumbSearcher}$ or an isolated $\text{awareSearcher}$ only executes Rule $M_1$. While executing this rule, an isolated robot stays a $\text{dumbSearcher}$ or an $\text{awareSearcher}$, and considers the direction it considers during the previous Move phase. By Lemma 20, this direction cannot be equal to $\perp$. Therefore, an isolated $\text{dumbSearcher}$ or an isolated $\text{awareSearcher}$ 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 $\text{leftWalker}$. A $\text{leftWalker}$ only executes Rule $T_1$. While executing this rule, a robot considers the left direction and stays a $\text{leftWalker}$. 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 $\text{headWalker}$. A $\text{headWalker}$ can execute either Rule $T_2$ or Rule $T_3$ or Rule $W_1$. While executing Rule $T_2$, a $\text{headWalker}$ becomes a $\text{leftWalker}$, 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 $T_3$ or Rule $W_1$. In these cases, after time $t$, it necessarily exists a time at which $r$ executes Rule $T_2$. Assume, by contradiction, that this is not true. The only way for a robot to become a $\text{headWalker}$ is to execute Rule $K_1$. Rule $K_1$ is executed when $R - 2$ robots are located on a same node. While executing this rule, a robot sets its variable $\text{walkerMate}$ with the identifiers of the robots that are located on its node. Only Rule $K_1$ permits a robot to update its variable $\text{walkerMate}$. Note that, since $R - 2 \geq 2$, the variable $\text{walkerMate}$ 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 $\text{towerTermination}$, 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 $T_3$ at time $t$, $r$ considers the $\perp$ direction and does not update its other variables. (3) Hence, by the rules of $\mathcal{GD}$, since $r$ cannot execute Rule $T_2$, after time $t$, $r$ can only execute Rule $T_3$, 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 $\text{HeadWalkerWithoutWalkerMate}()$ is true, thus at this time Rule $T_2$ is executed. Similarly, if at time $t$, $r$ executes Rule $W_1$, since $r$ can never be located on the same node as $i$, while executing Rule $W_1$, it considers the $\perp$ direction and does not update its other variables. (4) Hence, by the rules of $\mathcal{GD}$, since $r$ cannot execute Rule $T_2$, after time $t$, $r$ can only execute Rule $W_1$, 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 $\text{HeadWalkerWithoutWalkerMate}()$ is true, hence at this time Rule $T_2$ is executed. Therefore, even in the cases where, at time $t$, $r$ executes either Rule $T_3$ or Rule $W_1$, it exists a time greater than $t$ at which $r$ becomes a $\text{leftWalker}$ 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 Term1 making \( G_E \) solved.

Now, we consider the case of \( BRE \) rings. We prove, in Theorem 5, that \( GDG \) solves \( G \) in \( BRE \) rings. To prove this, we first need to prove the following lemma that is useful to bound Phase \( K \) of \( GDG \) in \( BRE \) rings.

**Lemma 22.** If the ring is a \( BRE \) 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 \( t + n \ast \delta \) 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 \( BRE \) ring each edge appears at least once every \( \delta \) units of time.

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

**Proof.** By Lemma 4, \( GDG \) solves \( G_E \) in \( RE \) rings. Therefore, since \( BRE \subset RE \), then \( GDG \) also solves \( G_E \) in \( BRE \) rings. We want to prove that \( GDG \) solves \( G \) in \( BRE \) rings. Therefore, we have to prove that each phase of the algorithm is bounded.

**Phase 8:** 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 \( GDG \), only righter can become potentialMin (refer to Rule \( M_6 \)), 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 \( M_8 \) and \( M_6 \)). Therefore, since each edge of the footprint of a \( BRE \) ring is present at least once every \( \delta \) units of time, by definition of \( min \) and of the predicate MinDiscovery() of Rule \( M_1 \), \( r_{min} \) becomes \( min \) in at most \( 4 \ast n \ast id_{r_{min}} \ast \delta \) rounds. Hence, Phase \( K \) is bounded and is in \( O(id_{r_{min}} \ast n \ast \delta) \).

**Phase \( K \):** Now, consider the execution when \( r_{min} \) just becomes \( min \). Therefore, we consider the execution once \( r_{min} \) is \( min \) in finite time. By Lemma 6, there is only one towerMin in the whole execution. Therefore, before a towerMin is formed, by the rules of \( GDG \) 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 \( K_3 \), 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 \( n \ast \delta \ast (R - 3) \) 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 \( K_4 \), the righter becomes an awareSearcher and moves on the right. If a righter is located only with \( R - 2 \) other righter, they all execute Rule \( M_6 \), 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 \( M_7 \)). 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 \( \lvert NodeMate() \rvert < R - 2 \), 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 \( M_8 \)). Therefore, by Observation 2 and since each edge of the footprint of a \( BRE \) ring is present at least once every \( \delta \) units of time, it takes at most \( n \ast \delta \) rounds in order to stop having righter robots in the execution. Indeed, even if a righter does not execute Rule \( M_7 \) or Rule \( M_6 \), at most \( n \ast \delta \) 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 and is in \( O(n \ast R \ast \delta) \).

Once a towerMin is present in the execution, the robots forming this towerMin execute Rule \( K_1 \). 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

34
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 $GDG$, only $r$ can be $headWalker$ during the execution.

There is no rule in $GDG$ 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 $T_2$, $T_3$ and $W_1$. Since $r$ does not become a $leftWalker$, it cannot execute Rule $T_2$. Moreover, since we consider the worst case execution in terms of time, this implies that $r$ is able to execute Rule $W_1$ entirely. This means that $r$ is able to execute Rule $W_1$ until its variable $walkSteps$ reaches the value $n$. In other words, $r$ is able to execute Rule $W_1$ until it executes Rule $T_3$.

In this case, the $tailWalker$ and $minTailWalker$ are also able to execute Rule $W_1$ entirely. Indeed, if, at a time $t'$, while executing Rule $W_1$ or Rule $T_3$, 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 $W_1$ they would have join it. Moreover, if such an event happens, $r$ executes Rule $T_1$ and therefore becomes a $leftWalker$, which leads to a contradiction.

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

**Phase $T$:** Call $t_0$, the time at which the $headWalker$ and $minTailWalker/tailWalker$ robots finish to execute Rule $W_1$ entirely. Since the $headWalker$ and $minTailWalker/tailWalker$ start the execution of Rule $W_1$ on the same node, at time $t_0$, 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 $t_0$.

If at time $t_0$, $r_1$ and $r_2$ are on node $v$, then Rule $Term_1$ is executed at time $t_0$. In this case, by Lemma 18, Phase $T$ last 0 round, hence it is bounded.

If at time $t_0$, only one robot among $r_1$ and $r_2$ is located on node $v$, then Rule $Term_2$ is executed at time $t_0$. Hence, by Lemma 18, $R - 2$ robots are terminated on node $v$ at time $t_0$. 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 BRE ring is present at least once every $\delta$ units of time, it takes at most $n \times \delta$ rounds for the last robot to reach node $v$. Therefore, it takes at most $n \times \delta$ rounds for Rule $Term_1$ 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 $n \times \delta$ rounds, therefore it is bounded.

Now, consider that at time $t_0$, neither $r_1$ nor $r_2$ is located on node $v$. In this case, at time $t_0$, the $headWalker$ and $minTailWalker/tailWalker$ execute Rule $T_3$. While executing Rule $T_3$, the $headWalker$ (resp. $minTailWalker/tailWalker$) stays a $headWalker$ (resp. $minTailWalker/tailWalker$) and considers the $\perp$ direction. Then, by the rules of $GDG$, they can only execute Rule $T_3$ until they terminate. Therefore, they remain on node $v$ from time $t_0$ 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 BRE ring is present at least once every $\delta$ units of time, it takes at most $2 \times n \times \delta$ 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 $n \times \delta$ rounds of movements in the same direction). In the case the two robots reach node $v$ at the same time, then Rule $Term_1$ 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 $Term_2$ (hence, by Lemma 18, permits the termination of $R - 2$ robots on node $v$) and the second that reaches $v$ permits the execution of Rule $Term_2$. Hence, Phase $T$ last at most $2 \times n \times \delta$ 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 $2 \times n \times \delta$ rounds are needed for $r$ to become $leftWalker$ and for the $R - 2$ other robots to terminate their execution on a node $v$.  

35
By the rules of GDG, a leftWalker only executes Rule $T_1$. While executing this rule, a robot considers the left direction and stays a leftWalker. Since each edge of the footprint of a BRE ring is present at least once every $\delta$ units of time, such a robot reaches the node $v$ in at most $n \cdot \delta$ rounds. Hence, in this case, Phases $\mathcal{W}$ and $\mathcal{T}$ take at most $3 \cdot n \cdot \delta$ rounds, hence they are bounded.

By the two previous cases, phase $\mathcal{W}$ and Phase $\mathcal{T}$ take $O(n \cdot \delta)$ rounds.

Whatever the BRE ring considered, each phase of GDG is bounded, therefore, GDG solves $G$ in BRE rings. Moreover, GDG solves $G$ in BRE rings in $O(n \cdot \delta \cdot (id_{r_{\min}} + R))$ rounds.

Now, we consider the case of ST rings. We know that ST rings are BRE rings such that $\delta = 1$, hence, by Lemma 5, we can deduce the following corollary.

**Corollary 6.** GDG solves $G$ in ST rings in $O(n \cdot (id_{r_{\min}} + R))$ rounds.

### 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 focusing on the gathering problem in COT, the class of dynamic graphs that exhibits the weakest recurrent connectivity.

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


