UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
POSGRADO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

WAIT-FREE GATHERING PROBLEMS ON GRAPHS

T E S I S

QUE PARA OPTAR POR EL GRADO DE:

Doctor en Ciencias (Computación)

PRESENTA:
Manuel Alcántara Juárez

TUTORES:
Dr. Sergio Rajsbaum Gorodezky
Instituto de Matemáticas, UNAM
Dr. José David Flóres Peñaloza
Facultad de Ciencias, UNAM

COMITÉ TUTORAL
Dr. Armando Casteñada Rojano
Instituto de Matemáticas, UNAM

Ciudad Universitaria, Ciudad de México, Enero de 2019

JURADO ASIGNADO:

Presidente:	Dr. José David Flores Peñaloza
Vocal:	Dr. Sergio Rajsbaum Gorodezky
Secretario:	Dr. Armando Castañeda Rojano
1er. Suplente:	Dr. José de Jesús Galaviz Casas
2o. Suplente:	Dr. Ricardo Marcelín Jiménez

La tesis se realizó en el Posgrado en Ciencia e Ingeniería de la Computación, IIMAS, UNAM.

TUTOR DE TESIS:

Dr. Sergio Rajsbaum Gorodezky

_____

Agradecimientos

Sin lugar a dudas, la culminación del presente trabajo no hubiera sido posible sin el apoyo de todas aquellas personas que directa o indirectamente contribuyeron durante el desarrollo del mismo. En especial, deseo agradecer de todo corazón:

A mi madre Teresa Juárez y a mi hermano Daniel Alcántara, por su apoyo incondicional, por las palabras de aliento cuando las necesitaba y por haberme siempre impulsado a seguir adelante hasta conseguir este gran logro.

A mi novia Norma Verónica Trinidad Hernández, por haber soportado de manera titánica todas mis caidas, por los sacrificios que en ocasiones tuvimos que hacer, pero sobre todo por su gran amor y compresión que aún en los peores momentos siempre lograban sacar lo mejor de mí.

A los miembros de mi comité tutoral el Dr. Sergio Rajsbaum Gorodezky, el Dr. David Flores Peñaloza y el Dr. Armando Castañeda Rojano, por haberme guiado durante toda esta travesía, compartiendo conmigo su experiencia, conocimiento y sus invaluables consejos.

A Luis Chacón Ochoa, por ayudarme a dirigir sin condición alguna el club PU++ durante mis días de ausencia, por ser mi amigo y compañero en la realización de los concursos de programación y por haber compartido conmigo todas aquellas tardes de juegos memorables.

A mis amigos, Renato Zamudio, Armando Ballinas y Karla Rocío, por escucharme y apoyarme siempre que lo necesitaba, y que, aunque cada vez sea más difícil coincidir para vernos, siempre tienen un par de minutos para hacerme reír.

A CONACYT por haberme brindado la oportunidad de realizar todos mis estudios de posgrado con ayuda de una beca; así mismo, al proyecto UNAM-PAPIIT IN109917 por el apoyo económico para asistir a diversos encuentros y congresos.

Abstract

The Gathering problem in the Look-Compute-Move model for a set of autonomous robots has been thoroughly studied for over two decades. If the space is a graph G, it is required that all robots gather in *finite time* on the same vertex of G not fixed in advance. Each robot works as follows: repeatedly Looks at its surroundings to obtains a snapshot containing the vertices of G, where all robots are located; based on this information, the robot Computes a destination position (adjacent to its current position) and then Moves to it.

What would be the effect of requiring termination after a bounded number of cycles? Namely, we are looking for solutions where each robot irrevocably decides a vertex of the graph without waiting for the actions of other robots or waiting that some condition happens. This research studies the hardness of termination in Gathering type problems.

It first shows that the classic Gathering problem with the termination requirement is unsolvable by any non-trivial algorithm in the standard Asynchronous Luminous Robots model (ALR). The impossibility of Gathering exposes a relationship between the ALR model and the asynchronous crash-prone wait-free multiprocess shared memory (WFSM) model.

Then, we extend the ALR model to encompass crash failures and asynchronous appearing times. We called this model EALR, and we studied on it two weaker variants of Gathering with termination: Edge-Gathering and $\mathcal{K}$-Gathering. We fully characterize the solvability of Edge-Gathering: it is solvable for three or more robots if and only if the base graph is a tree. Particular solutions for $\mathcal{K}$-Gathering in some graph families are also provided.

On the computability side, we show that the EALR and WFSM models are equally powerful to solve general robot tasks on graphs. Using this equivalence, we derive a full combinatorial topology characterization of the solvability of general robot tasks in EALR. These results bring together, for the first time, two areas that have been independently studied for a long time into a common theoretical foundation.

Contents

List of Figures

Chapter 1

Introduction

Problems of coordination and collaboration among a group of independent individuals are undoubtedly the object of study in several areas of knowledge. From the distributed computing perspective, in the last years, several theoretical models have been raised based on ***mobile robots*** [2, 3]. The main challenge is to propose distributed algorithms capable of coordinating a set of autonomous process units (commonly called robots), which move over a specific defined space where they communicate and cooperate with each other to achieve a common goal.

The current research trend in the mobile robots area tries to figure out what are the minimum robots' capabilities to solve a certain group of problems. Usually, models assume a system composed by a set of mobile robots that execute the same algorithm, and the only way that each of them has to communicate with others is based on the observation of its surroundings and the movement decision that they take as a consequence. So, in this type of models, robots are *silent*, meaning that they do not have a direct way to exchange information, and hence, the algorithms design process is commonly based on the execution of Look-Compute-Move (LCM) cycles [4]. Where in each of them, a robot obtains information of its surroundings (Look), then it analyzes the data to compute an adjacent position to its current location, possibly the same (Compute), and finally moves there (Move).

In some settings, the mere observation of the environment does not always provide enough information to the robots to solve problems, so that is why they are endowed with certain capabilities to help them achieve its goal. For this reason, throughout literature, it is possible to find a wide range of models that define its own characteristics. Some examples could be

orientation: robots share the same coordinate system, or they know where the cardinal points are; *visibility:* how much information can be obtained from the environment (complete or partial); and *memory*: how much information robots could remember. Also, recent papers (e.g. [5, 6]) have begun to use external lights, which allows robots to transmit additional information to their current position. And although in principle one could conceive any capability, what it is sought is that robots be as simple as possible, to be consistent with any possible real application, where the energy consumption and the cost per unit are decisive factors.

Among the problems that have received considerable attention in its study, because it constitutes one of the fundamental primitives principles for the control and coordination of autonomous mobile robots, is the ***Gathering*** problem [7] (Gathering for two robots is often called *rendezvous*). In which, given an initial configuration of $N \geq 2$ robots, all of them should occupy a single vertex in finite time.

Some Gathering solutions have been proposed, in which either the robots start on a specific initial configuration, or they have more powerful capabilities (e.g., [7, 8, 9]). However, some of these algorithms do not guarantee *termination*, i.e., each robot never decides a vertex; while others only guarantee a *weak* type of termination, in which a robot decides a vertex only if it detects a particular configuration, for example, the position of other robots satisfy a given property. On the other hand, few models consider the presence of faults (e.g., [10, 11]), and in all of them the assumption that robots move on the plane or their executions are synchronous/semi-synchronous[1] is always present.

Therefore, this work seeks to extend the results on the Gathering problem considering a *strong* termination property, in which each robot terminates in finite LCM cycles, regardless of the actions of others. We first analyze the hardness of termination using the standard ALR (Asynchronous Luminous Robots) model [4], and later we consider weaker variants of Gathering with termination in a more challenging model that we have called EALR, where robots can appear asynchronously and might also present a stopping failure at any moment during the execution. Preliminary results of this work were presented in [12].

[1] In semi-synchronous executions, the scheduler repeatedly chooses an arbitrary non-empty subset of robots which execute their Move-Look steps concurrently.

1.1 Related Work

The Gathering problem has been extensively studied from the distributed computing perspective under the LOOK-COMPUTE-MOVE model; mostly in the plane, assuming synchronous, semi-synchronous and asynchronous models, and considering a vast range of capabilities such as visibility, anonymity, shape, memory or orientation. Some surveys can be found at [2, 13, 14, 15, 16].

It has been shown that Gathering is unsolvable for $N \geq 2$ robots in a very restricted model, where robots move on the plane, are anonymous, asynchronous, disoriented, unable to remember events in the past, without multiplicity detention and starting in arbitrary positions [17]. The main reason is that robots do not have any way to distinguish between similar situations, so it is possible to create scenarios where, repeatedly, robots detect each other, exchange positions and leave the state of the system in a symmetrical configuration, this without finishing in the same place.

On the other hand, favorable results have been reported if some assumptions about the robot's capabilities are made. For example, surprisingly, the Gathering problem is solvable for $N > 2$ robots if they are allowed to detect multiplicities, i.e., they can identify the number of robots located in a specific location [7]. Also, it is possible to solve Gathering if robots are allowed to make decisions based on past events [9]. Likewise, the use of a compass also let them gather in the same place, even if robots have limited visibility of their environment [8].

When working on graphs, the Gathering problem has been mostly studied in rings, where several papers have analyzed all the initial asymmetric configurations, detecting the scenarios where it is possible to find a solution [18, 19, 20]. Additional results working on grids [21, 22] or in the case with only two robots (Rendezvous) [4, 23, 24, 25] can also be found.

On the other hand, the case of study where robots might fail has already been taken into consideration [10, 11, 26, 27], investigating mainly two possibilities: *stop failures* and *Byzantine failures*. In the first one, robots stop their activity entirely and terminate their execution prematurely, remaining immobile in the place where the crash occurred. While in the second, robots perform malicious behaviors that could influence the decisions of others. In both cases, the solutions demand that the robots that do not fail

must succeed in solving the problem, regardless of the behavior of those with a malfunction. For example, a fault-tolerant algorithm for *converge* (robots never reach the same point) is shown in [28] under a configuration without termination, using the center of gravity, or also called the center of mass, of the group of robots.

Recently, a small extension has started to be considered, in which robots use lights to communicate additional information of their current position [5]. Under this variant, it is known that two robots can solve Gathering in the plane, using only two lights and starting in arbitrary positions [24]; however, the algorithms are not fault-tolerant.

A similar model where Gathering has been studied is the one based on *agents*. In this setting, there are N programs that live inside of a network. Different from the robot models, the agents cannot perform a Look operation to perceive all the environment. Usually, each agent explores the graph to save its topology locally, and each one has a unique label that shares with other agents when they meet on the same node or by writing it on a persistence storage inside each node. For more references, see for example [29, 30, 31, 32, 33], where Gathering and similar problems are studied from the agents perspective, considering crash or Byzantine failures, or even asynchronous starting/appearing times.

1.2 Contributions

The main goal of this research is to study the hardness of achieving *strong* termination on the Gathering problem for the standard ALR model, where robots are fully asynchronous, anonymous, oblivious, disoriented, without detecting multiplicities, moving on the vertices of a connected simple graph, running the same deterministic algorithm (uniform) and empowered with lights to communicate with each other.

Since robots are asynchronous in ALR (thus delays are unpredictable and arbitrarily long), the *strong* termination property guarantee that each robot terminates in a fixed number of LCM cycles, regardless of the delays of other robots. That means, a robot cannot wait for others' actions to terminate (see for example [34], where a robot ends its algorithm only when the light of the other robots shows a specific color). Observe that the definition of Gathering does not require that robots irrevocably make a decision

when they gather, it is only expected that eventually, they arrive at the same vertex in *finite time*.

Hence, our first contribution shows that Gathering with strong termination is unsolvable by any non-trivial algorithm (namely, one in which robots gather on a non-fixed vertex) in ALR, even if robots have strong capabilities, like having an unbounded number of lights, share the same labeling of the graph, being non-anonymous or non-oblivious. Interestingly, this result is obtained by reducing the fundamental Binary Consensus problem in crash-prone asynchronous multiprocess shared memory systems into the Gathering problem in ALR. Then, the well-known impossibility of Binary Consensus implies the impossibility of Gathering with strong termination. This reduction shows an interesting connection between asynchronous mobile robot models and the asynchronous multiprocess shared memory model.

Later, we study a novel and challenging extension of the ALR model that we called *EALR* (Extended Asynchronous Luminous Robots), where robots might also present a crash failure, and we consider a new property in which it is not necessary that all the robots start its execution at the same time. That means that robots might appear unexpectedly in any vertex of the graph or disappear in the case of failure without interfering in the general solution of the problem. More in detail, in the EALR model a subset of robots asynchronously join the computation, and moreover, some of them may never appear due to crash failures. Remarkably, there are executions in EALR that cannot happen in ALR, even if this is extended to include crash failures. For example, due to the combination of the strong termination property and the asynchronous appearing times, it is possible that a "fast" robot runs several (even all) of its LCM cycles alone, without seeing any other robot. To the best of my knowledge, the study of a model that links these three particularities (strong termination, failures and asynchronous appearing times) is explored for the first time in this work.

Since Gathering with termination is unsolvable by any non-trivial algorithm in the ALR model, it is also unsolvable in the more challenging EALR model either. Following in the same direction, we then relax the requirement that robots should gather exactly in the same vertex to consider two weakened Gathering type problems with strong termination in EALR: *Edge-Gathering* and $\mathcal{K}$*-Gathering*. In the Edge-Gathering problem robots are required to move to vertices of the same edge; while in the $\mathcal{K}$-Gathering problem robots are required to move to a subset of vertices of the same complete subgraph. We considered these problems because it is natural to

think that although robots are not in the same vertex, they are close enough between them.

We first show that lights are needed in order to solve Edge-Gathering for $N \geq 3$ robots whenever $diam(G) \geq 3$. With the help of the lights, we present an algorithm that solves Edge-Gathering on any tree. Furthermore, if the graph G has at least one cycle, then Edge-Gathering becomes unsolvable, even if robots have stronger capabilities. With these results we can fully characterize Edge-Gathering: it is solvable on a graph G if and only if G is a tree. Then, we proved that $\mathcal{K}$-Gathering is solvable using lights on any graph whose related clique graph is a tree, and it is also solvable without lights on any graph with at least one dominating vertex. Observe that the last algorithm shows a separation between Edge-Gathering and $\mathcal{K}$-Gathering. Because, while the first one is unsolvable without lights for any non-trivial graph (i.e., $diam(G) \geq 3$), the second is solvable without lights in some graphs.

Our impossibility result of Edge-Gathering in presence of cycles, exposes a strong connection between EALR and the standard asynchronous multiprocess crash-prone Wait-Free Shared Memory model (WFSM): if robots can use an unbounded number of lights then the two models are equally powerful to solve general robot tasks on graphs (which include Gathering, Edge-Gathering, and $\mathcal{K}$-Gathering with strong termination). This equivalence between EALR and WFSM uncovers an intimate connection between EALR and topology, where powerful topological techniques have been used in the past two decades in the study of fault-tolerant distributed computing, and in particular in WFSM (see textbook [1]). Using this connection, we derive a full combinatorial topology characterization of the solvability of general robot tasks in EALR.

Finally, while our algorithms work in the fully asynchronous model, our impossibility results remain even if the model is *semi-synchronous* (where the scheduler repeatedly chooses an arbitrary non-empty subset of robots which execute their Move-Look steps concurrently, e.g. [35]). We uncover a direct link between the semi-synchronous robot model and the *immediate-snapshot* model, that is significant in shared-memory computing [1]. Our results show that the asynchronous and semi-synchronous models are equally powerful when robots might appear asynchronously.

Problem in EALR	Robots	Graph	Solvability
Gathering	$N \geq 2$	Connected	**Unsolvable:** In ALR and EALR, even with stronger capabilities. Theorem 2.4.1.
Edge-Gathering	$N \geq 2$	Tree	**Solvable:** $diam(T) - 1$ rounds and light colors. Theorem 3.2.1.
	$N \geq 3$	Connected with $diam(G) \geq 3$	**Unsolvable**: Without lights. Theorem 4.3.2.
	$N \geq 3$	With a cycle	**Unsolvable:** Even with unbounded lights. Theorem 4.3.4.
$\mathcal{K}$-Gathering	$N \geq 2$	Tree	**Solvable:** $diam(T) - 1$ light colors and rounds. Corollary 3.3.1.
	$N \geq 2$	With a dominating vertex	**Solvable:** Without lights and 1 round. Theorem 3.3.1.
	$N \geq 2$	The clique graph of G is a tree	**Solvable:** $4 \cdot diam(K(G)) \cdot \|V(G)\| \cdot \|V(K(G))\|\|$ lights and $2 \cdot diam(K(G)) + diam(G)$ rounds. Theorem 3.3.2.
	$N \geq 3$	$diam(G) \geq 3$ and no triangles	**Unsolvable:** Without lights. Corollary 4.3.1.
	$N \geq 3$	With cycles and no triangles	**Unsolvable:** Even with unbounded lights. Theorem 4.3.5.

Table 1.1: Summary of positive and impossibility results presented in this research.

1.3 Organization

The organization of this work is divided into 5 chapters. In Chapter 2, we prove that the Gathering problem with strong termination is unsolvable on the ALR model, even if robots have stronger capabilities. We first introduce the standard model for asynchronous robots with lights (ALR) and then we study the relation of Gathering with strong termination from the wait-free read/write shared memory model perspective, reducing it to the Binary Consensus problem. Next, Chapter 3 extends the ALR model to consider crash

failures and asynchronous appearing times, calling this new model EALR. Also, we present weakened Gathering types problems: Edge-Gathering and $\mathcal{K}$-Gathering. And later, we introduce some algorithms for the solvable cases. Chapter 4 shows our impossibility results and presents a solvability characterization for general robot tasks. It also shows that the EALR and its semi-synchronous variant are equivalent. Finally, Chapter 5 concludes this thesis with final remarks and some future work.

Chapter 2

The Hardness of Strong Termination

In this chapter, we will show that Gathering with strong termination is unsolvable by any non-trivial algorithm in the ALR model, even if robots have stronger capabilities, like an unlimited amount of local memory, unbounded number of light colors or sharing a common labeling of G. Since robots are asynchronous in ALR (thus, delays are unpredictable and arbitrarily long), the *strong* termination property guarantees that each robot terminates in a bounded number of its LCM cycles, regardless of the delays of other robots. That means, a robot cannot wait that some condition happens in order to terminate. Weaker termination properties have been studied in literature, where a robot ends its execution when it detects a "good" configuration (e.g., [36]); however, for that to happen, a robot has to wait for others to do something.

Generally speaking, the impossibility result comes from a reduction from the well-known Binary Consensus problem in the multiprocess asynchronous shared memory model to the Gathering problem in ALR. This reduction highlights one of the main contributions of this research: there is a connection between asynchronous robot models and the standard *asynchronous wait-free shared memory* (WFSM) model with crash failures. As we will see, this connection opens the possibility to import all the results developed for decades in the WFSM model to use them in ALR. Moreover, on chapter 3 we will extend the ALR model to make it equivalent to WFSM, proving that one can go back and forth between models.

2.1 Asynchronous Luminous Robots

We consider the usual Look-Compute-Move (LCM) model of asynchronous robots (in literature is usually found as $\mathcal{ASYNC}$ [37]), moving on the vertices of a simple connected graph G (e.g. [38]), and extended with the possibility of robots communicating using lights [34]. We called this model *Asynchronous Luminous Robots* (ALR).

More in detail, the ALR model is a distributed system formed by a group of $N \geq 2$ autonomous computational entities called robots, denoted by $p_1, p_2, \ldots, p_N$. Each robot is a simple state machine that executes a deterministic algorithm $\mathcal{A}$ in *asynchronous* way, so the actions of each robot happen at unpredictable moments of time. Hence, there is no concept either of a global clock or a central control unit.

Initially, all the robots reside over the graph G in an *inactive* state, and at arbitrary times (due to the asynchrony), the robots start executing the same algorithm $\mathcal{A}$, which receives for each robot p_i the graph $G = (V, E)$ and a vertex v_i; that represent the definition of the space where the robots are allowed to move and the initial vertex where p_i will start the execution of $\mathcal{A}$. Although all the robots know the graph G, it is assumed that they do not share a common labeling, either on the vertices or edges of G (*disoriented*).

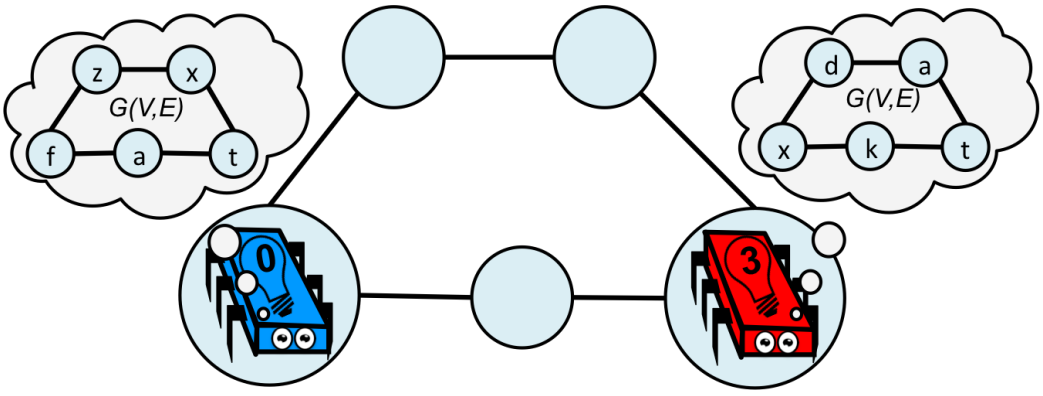

Figure 2.1: All robots know the input graph G, but they do not share the same labeling, either on the vertices or edges of G.

The robots are modeled as points without volume, so it is possible that more than one occupies the same vertex at the same time without this coming to represent some blockage or interference between them. Also, robots are *anonymous* an identical, so, there are not unique identifiers in the system, and

it is impossible to distinguish two different robots only by their appearance.

Each robot has the following theoretical devices to interact with others and with the environment:

- A motor: to move over the vertices of the graph.
- An external visible light: which can show a *constant* number of colors associated with an integer value. Each light is visible to all the robots, even in the scenarios where the positions of three or more of them are collinear. The use of this external light gives to each robot the ability to transmit more information that only its actual position [5].
- A set of cameras: able to detect the positions and the light color of all the active robots in the system.

Definition 2.1.1. The ***state*** of a robot is a tuple made of the state of its local state machine $\mathcal{A}$ (which encodes its local variables, current vertex, etc.) and an integer value associated to its light.

Definition 2.1.2. The ***initial state*** of a robot is composed by the initial state of $\mathcal{A}$ and its light equal to a predefined value.

The way in which robots exchange information is *silent*, meaning that there is no direct communication between them, i.e., the communication occurs implicitly only by looking at other robots' positions and its lights values. The actions that each robot can perform are based on the following interface of operations:

- LOOK(G): Examines atomically (in a single instant of time) every vertex in the graph G to obtain a set of pairs (v, q) that denotes the existence of at least one active robot on the vertex v, whose light is showing color q. Also, due to the possibility that more than one robot could be in the same vertex v showing in their light the same value q, the LOOK operation will contemplate in its result only a single pair (v, q). That means, in ALR is not possible to detect *multiplicities*, so, a robot can only detect the existence of other with state (v, r), but it could not know exactly how many of them are in the same situation. See figure 2.2 for clarity.
- COMPUTE(*view*): According to the specification of a deterministic algorithm and based on the information returned by the LOOK operation, each robot p_i performs a local computation to decide a pair (v, r), where v is an adjacent vertex to its current position (possible the same) and r

is the value to show in its light; p_i changes to this new local state when it executes its next MOVE operation. Although the computational complexity of this process has not been established, it is assumed that always produces a result in a bounded period of time.

- MOVE(u, r): The robot that invokes this operation standing on vertex x moves immediately to vertex u and sets its light to value r, where u must be an adjacent vertex to x (or equal to x) and r is a non-negative value. If none of this conditions holds, then the operation is not performed, and the robot remains in its previous state. This operation moves the robot atomically, independently of the distance between adjacent vertices.

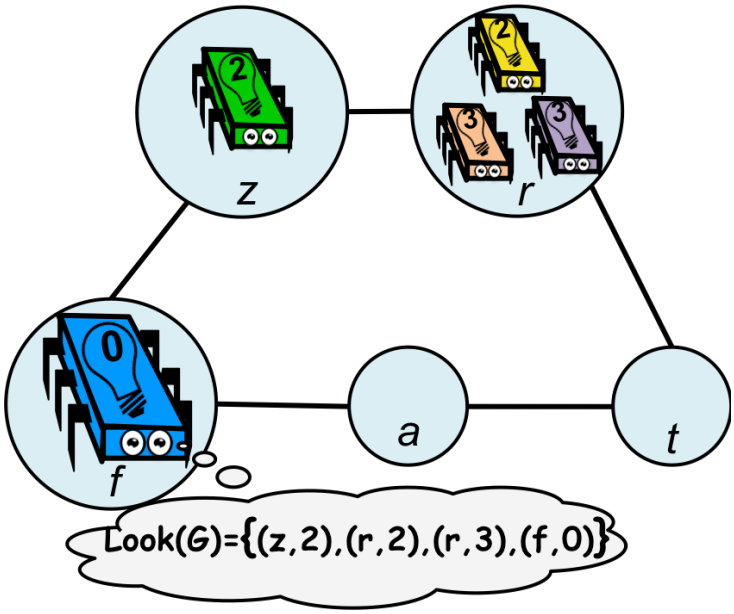

Figure 2.2: Although there are six robots in the system, the LOOK operation invoked by the blue robot returns a set of only four pairs. Observe that there are two robots with the same state on vertex r.

Robots move on G from vertex to vertex operating in LOOK-COMPUTE-MOVE (LCM) cycles, where in each of them, a robot p_i LOOKS at its surroundings to collect information from the environment (such as the positions of other robots). Then, based on this information, p_i COMPUTES a destination vertex (adjacent to its current position) and then MOVES to it. The model is asynchronous, so, robots do not have a common notion of time, and each one runs at its own speed. Namely, a scheduler decides, at any moment, the robot that will execute its next operation, which depending on the state

of the robot, it will be either a LOOK, COMPUTE or a MOVE operation. Also, robots are unable to remember events made in the past; hence, in each cycle, the COMPUTE operation will only depend on the information obtained by the LOOK operation, namely, robots are *oblivious*.

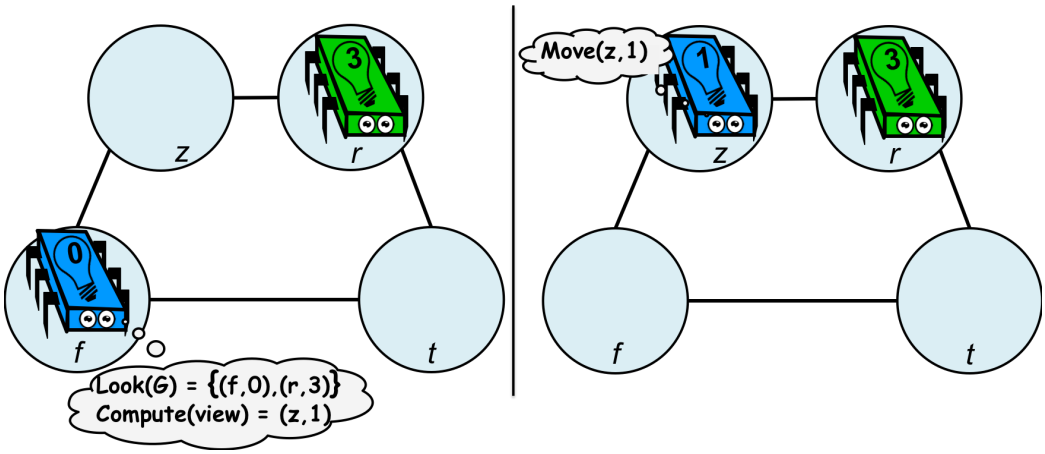

Figure 2.3: Example of a LCM cycle: the blue robot detects the other one, computes the pair $(z, 1)$ and finally moves there, setting its light to value one.

Definition 2.1.3. A ***configuration*** of the system is an N-vector containing the state of robot p_i in its i-th entry.

Definition 2.1.4. An ***initial configuration*** is a configuration in which all the robots are in their initial states.

Definition 2.1.5. A robot performs a ***step*** when it executes one of its available operations, either MOVE or LOOK, and then changes its local state.

Definition 2.1.6. An ***execution*** E is an infinite alternating sequence of configurations and steps $E = C_0\ s_0\ C_1\ \ldots$, where C_0 is an initial configuration and C_{k+1} is the configuration obtained by applying step s_k to configuration C_k. To represent an execution, we will use an infinite time-line containing all the events made by the robots.

Definition 2.1.7. The state machine of a robot p_i models a ***local algorithm*** $\mathcal{A}_i$ that determines the next step of p_i.

Definition 2.1.8. A ***robot algorithm*** is a collection $\mathcal{A}$ of local algorithms $\mathcal{A}_1, \ldots, \mathcal{A}_n$. As said above, all $\mathcal{A}_i$ are identical.

Algorithm 1 illustrates a general algorithm in the ALR model. Bear in mind that, due to the asynchronous nature of the robots, it is possible

that between the LOOK and COMPUTE steps (lines 5 and 7) there could be concurrent operations of other robots. This implies that a robot p_i could compute its next state with information that does not necessarily correspond to the current state of the system. This uncertainty is the one that sometimes makes difficult to prove that any given algorithm is correct.

In summary, in the ALR model robots are: uniform, silent, anonymous, identical, asynchronous, oblivious, rigid, disoriented, without detecting multiplicities and with a constant number of light colors.

Algorithm 1 General algorithm in the ALR model. Code for robot p_i, where $G = (V, E)$ is an arbitrary graph and $v_i \in V$.

```
1: procedure ALGORITHM(G, v_i)
2:     view_i ← {}
3:     while true do
4:         // An oblivious algorithm does not store previous events
5:         view_i ←LOOK(G)
6:         // p_i computes its next state
7:         (v_i, r_i) ← COMPUTE(view_i)
8:         // p_i moves to vertex v_i and sets its light to r_i
9:         MOVE(v_i, r_i)
10:    end while
11: end procedure
```

Finally, we will be interested in algorithms in which each robot irrevocably decides a vertex on the graph. Formally, in a *robot algorithm with strong termination*, in every execution, every robot reaches a *decision* state; such that, once the robot enters to that state it never moves to a different vertex or change its light value (the robot still executes LCM cycles afterward, but it moves to the same vertex and sets its light to the same value). We say that a robot *decides* the vertex in which is standing on when it reaches a decision state. Typically, *return* instructions in the pseudo-code of an algorithm represent decision states. Moreover, decision states are well identified in the local state machine that every robot follows, thus, given an algorithm with termination, it is possible to determine when a robot has made a decision.

2.2 The Gathering problem

In the well-known *Gathering* problem (without strong termination), robots start at any arbitrary position, and it is required that *in finite time*, all of them meet at the same location not fixed in advance. This problem for two robots is also known as *rendezvous*.

In the case where the robots move on the plane and no assumptions are made on its capabilities, i.e., robots are oblivious, disoriented, asynchronous, anonymous, without multiplicity detection, then Gathering is impossible for $N \geq 2$ robots [17]. However, the problem starts becoming solvable if some additional capabilities are added to the robots. For example, Gathering is solvable if robots can detect multiplicities [7], or have a compass [8], or can store previous events (non-oblivious) [9].

As a mere exercise, we can trivially solve Gathering, even without lights, if we allow robots to share the same vertex labeling. For instance, Algorithm 2 shows one possible solution that works as follows: in each round, each robot obtains the set of positions where all robots are standing, and then, it moves in the direction towards the vertex with the minimum label in the view. In this way, the sequence of local minimums that are considered during an execution is monotonically decreasing. Therefore, since the vertices in the input graph are finite, at some point, a robot will reach the vertex with the global minimum of all the views, and this is going to be the vertex where all the robots meet.

Algorithm 2 Converge for $N \geq 2$ robots on any connected graph $G = (V, E)$. Code for robot p_i.

Function Convergence(v_i, G)

1: **loop**
2: $view_i \leftarrow$ **Look**(G)
3: // p_i moves in the direction of the vertex with the minimum label in the view
4: $localMinimum_i \leftarrow$ the vertex with minimum label in $view_i$.
5: $v_i \leftarrow$ some closest vertex to $localMinimum_i$.
6: $Move(v_i)$
7: **end loop**

Take into consideration that in the previous algorithm, robots cannot just meet on the vertex with the minimum label, because by definition, the gathering vertex cannot be fixed in advance. Also, observe that although robots gather on the same vertex in a finite time, individually, they do not "know" when that happens, because it is always possible that a robot discovers a new vertex with a lower label, which would be the new meeting point.

As we can see, if robots share the same vertex labeling, it is relatively easy to solve Gathering, mainly because the definition itself does not require that each robot irrevocably decides a vertex at some point during the execution, so robots could change its decision at any time. But, in a real scenario, where robots perform a sequence of tasks and Gathering is only one of them, it is not enough to say that robots will gather in finite time, it would be desirable to know in advance how many LCM cycles each robot will execute.

Our definition of Gathering with *strong* termination is the following:

Definition 2.2.1 (Gathering with strong termination)**.** In the Gathering problem with termination on a simple graph G, each robot starts on a vertex of G and executes its code, so that the following three properties are satisfied:

- TERMINATION: Every robot decides a vertex of G in a bounded number of its LCM cycles, regardless of the delays of other robots.
- VALIDITY: The decided vertex of a robot cannot be fixed in advance.
- AGREEMENT: All decided vertices are the same.

What would the effect of modifying the Gathering problem to require strong termination be? Can Algorithm 2 or any of the known algorithms be changed so that strong termination is guaranteed? As we will see in the next sections the answer to this question is no, even if robots have powerful capabilities, such as the use of IDs or the possibility to detect multiplicities.

2.3 Wait-Free Shared Memory Model

The *Wait-Free Shared Memory* (WFSM) models a multiprocess architecture where asynchronous crash-prone processes communicate through an array M of shared memory registers by applying atomic read/write operations. In more detail, the WFSM system consists of $N \geq 2$ asynchronous *processes* (state machines), denoted by $q_1, q_2, \ldots, q_N$, where at most $N-1$ of them may fail by crashing (when a process *crashes* it stops executing steps). Each register $M[i]$ has associated with process q_i, where only this one can deposit a value executing a $\text{WRITE}_i(x)$ operation, but it can read the entire array M atomically using a SNAPSHOT [1].

[1] We could have instead assumed that a process can only atomically read individually shared registers because it is known that it is possible to implement an atomic SNAPSHOT in a wait-free manner from read/write operations (see for example [39, 40]).

Since there are at most $N-1$ failures and processes are asynchronous, they run a ***wait-free*** algorithm, intuitively, the code of a process cannot include instructions that wait for a read/write operation of other process. We refer the reader to standard books (e.g. [39, 40]) for a full description of the model.

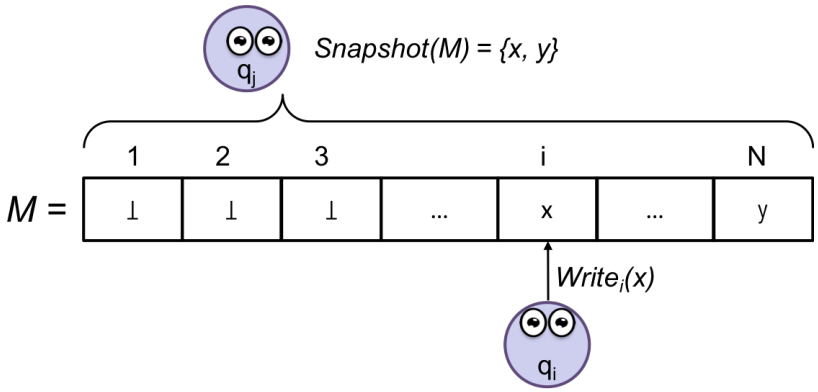

Figure 2.4: In the WFSM model, $N \geq 2$ processes communicate through an array M of shared memory registers. Each process q_i has associated with the register $M[i]$, where only q_i can deposit a value, but it can read the entirely array M atomically with a SNAPSHOT operation.

2.4 Unsolvability of Gathering with strong termination

The next simple lemma exhibits a link between ALR and WFSM.

Lemma 2.4.1. *Let $\mathcal{A}$ be any algorithm in the ALR model. Then, $\mathcal{A}$ can be simulated in the WFSM model in a wait-free manner (i.e., tolerating up to $N-1$ crash failures), and the simulation can start at any configuration of $\mathcal{A}$.*

Proof. To simulate $\mathcal{A}$ from a configuration C in the WFSM model, each process q_i simulates step by step the code of algorithm $\mathcal{A}$ for robot p_i, and decides whatever p_i decides in the resulting simulated execution (if $\mathcal{A}$ satisfies termination).

To do that, first each entry $M[i]$ of the WFSM model is initialized with the tuple (v_i, r_i) that corresponds to the state of p_i in C, namely, its position

and light value. Intuitively, at any time, the i-th entry of the shared array M will contain the visible state of robot p_i. Next, we need to specify how processes can simulate the atomic MOVE and LOOK operations of the ALR model as WRITE and SNAPSHOT operations of the WFSM model.

To simulate a MOVE(v_i, r_i) operation, process q_i simply executes WRITE$_i((v_i, r_i))$; namely, it atomically writes the pair (v_i, r_i) into $M[i]$. On the other hand, to simulate a LOOK(G) operation, process q_i takes an atomic snapshot of M, i.e., it executes the SNAPSHOT(M) operation and returns the set of pairs (v_j, r_j) that it detected in the array M.

The simulation is correct because as already argued, processes in the WFSM model can atomically simulate the MOVE and LOOK operations in ALR. □

Once established a connection between models, we can now show a relationship between Gathering with strong termination and the well-known Binary Consensus problem, which is one of the fundamental problems in the theory of concurrent computing.

Definition 2.4.1 (Binary Consensus)**.** In the Binary Consensus problem [41], each of $N \geq 2$ processes starts with a private input $w \in \{ 1, 2 \}$ and decides a value, such that the following properties are satisfied:

- TERMINATION: Every correct process decides a value.
- VALIDITY: Every correct process decides a value that is proposed by at least one process.
- CONSENSUS-AGREEMENT: All decided values are the same.

Given an algorithm with strong termination in the ALR model, we say that an execution starting at a configuration C (possibly a non-initial configuration) is p_i***-solo*** if robot p_i is the only one taking steps from the beginning of the execution (starting at C) until it decides, and then the other robots take steps. Since robots are deterministic, in every p_i-solo execution starting at C, p_i must decide the same vertex and do so after taking the same number of steps.

Definition 2.4.2. We say that a Gathering algorithm with strong termination has the ***solo-trivial*** property, if for every configuration C there is a vertex $\hat{u}$, such that for every robot p_i, it decides $\hat{u}$ in a p_i-solo execution starting at C.

Observe that in an algorithm with the solo-trivial property, for every initial configuration and every execution of the algorithm starting at that configuration, the robots decide the same vertex. Thus, in that sense, such an algorithm is *trivial*. An example of algorithm with the solo-trivial property could be the one that works on graphs with only one dominating vertex $\hat{v}$, where all the robots (without even communication) move directly to that vertex.

Lemma 2.4.2. *Let $\mathcal{A}$ be an algorithm for $N \geq 2$ robots in the ALR model. Suppose that $\mathcal{A}$ satisfies the* Termination *and* Agreement *properties of the Gathering problem with strong termination [2.2.1]. If $\mathcal{A}$ does not have the solo-trivial property, then the Binary Consensus problem is solvable by two processes in the WFSM model.*

Proof. Since $\mathcal{A}$ does not have the solo-trivial property, there exists at least one configuration C, such that, there is a pair of distinct robots r_1 and r_2, and two distinct vertices x_1 and x_2 of G, satisfying that for each $i \in \{1, 2\}$, in a r_i-solo execution starting at C, r_i decides x_i. Observe that C is not necessarily an initial configuration of $\mathcal{A}$.

Now, using $\mathcal{A}$, Algorithm 3 below solves the Binary Consensus problem for two processes in WFSM. The basis of the solution is that processes simulate $\mathcal{A}$ starting from C using the result of Lemma 2.4.1. Process q_i simulates robot r_i with $i \in \{1, 2\}$. Note that there might be more than two robots in C (i.e., $\mathcal{A}$ might be designed for more than two robots); however, the simulation only simulates two of them, r_1 and r_2. Thus, the executions of $\mathcal{A}$ starting at C are simulating, in which only r_1 and r_2 take steps until they decide while all other robots remain static. It is guaranteed that at some point on the execution r_1 and r_2 will decide a vertex, because $\mathcal{A}$ satisfies the Termination property.

Claim: *Algorithm 3 solves the Binary Consensus problem for 2 processes.* To prove it, we need to show that the algorithm satisfied each property of the Binary Consensus problem:

- Termination: Clearly, each correct process terminates because there are no loops in the algorithm and every correct process simulating a robot in $\mathcal{A}$ eventually decides a vertex in the simulation, due to the Termination property.
- Validity: By contradiction. Suppose first that process q_1 returns a value that has not been proposed. Thus, q_1 must read the value in $M[2]$ in line 9 before q_2 executes line 2. Now, observe that the only

Algorithm 3 Binary Consensus for $N = 2$ processes tolerating one failure. Code for process q_i.

Function BinaryConsensus(i, w)

```
1: // Variable i corresponds to the ID of the respective process in the WFSM model.
2: M[i] ← w
3: // Process q_i simulates robot r_i
4: v ← Decision of r_i in the simulation of A starting from C
5: // x_1 is the decided vertex in the r_1-solo execution starting at C
6: if v = x_1 then
7:     return M[1]
8: else
9:     return M[2]
10: end if
```

way in which q_1 could return $M[2]$ is if it gets a vertex $v \neq x_1$ after simulating $\mathcal{A}$. But that implies that the process did not simulate a r_1-solo execution of $\mathcal{A}$ starting from C. Thus, both processes q_1 and q_2 participated concurrently in the simulation, and then q_2 executed line 2 before q_1 executed line 9, which is a contradiction.

The other case when process q_2 returns a value that has not been proposed is analogous.

- Consensus-Agreement: Since $\mathcal{A}$ satisfies the Agreement property of Gathering with strong termination, in every execution all correct processes get the same vertex from it. Therefore, all of them decide the same value.

□

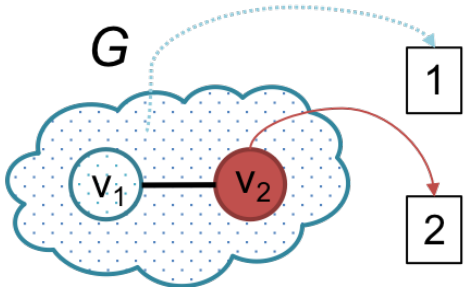

Figure 2.5: If we suppose that there exists an algorithm $\mathcal{A}$ that solves the Gathering problem with strong termination, then, we can use it to solve the Binary Consensus problem in the WFSM model, simulating two robots and mapping vertex v_1 to value 1 and the rest of the vertices to value 2.

Theorem 2.4.1. *The Gathering problem with strong termination is unsolvable by any algorithm in the ALR model, even if robots have powerful capabilities, such as, unbounded number of light colors, non-oblivious or sharing the same labeling of G.*

Proof. The VALIDITY property of the Gathering problem with termination guarantee that robots cannot fixed in advance the vertex where they gather, so, that implies that any algorithm solving the problem does not have the solo-trivial property. Thus, if there is such an algorithm in ALR, then by Lemma 2.4.2, the Binary Consensus for two processes is solvable in WFSM. That is a contradiction, since Binary Consensus is impossible in that model [39]. □

It is worth to observe that the impossibility of Gathering with strong termination does not come from the capacities of robots (number of lights or amount of local memory), the common information they have (orientation, labeling, etc.), or from the fact that we are working on graphs, as essentially the same definitions can be stated for Gathering algorithms with termination on the plane, and the same result holds. The impossibility comes from the validity condition in the statement of the problem. As an example, consider the following version of Gathering.

Definition 2.4.3 (Gathering with same-vertex-validity)**.** In the Gathering problem with the ***same-vertex-validity*** on a graph G, each robot starts on a vertex of G and executes its code, so that the following three properties are satisfied:

- TERMINATION: Every robot decides a vertex of G in a bounded number of its own LCM cycles, regardless of delays of other robots.
- VALIDITY: If all robots start on the same vertex v, then every decided vertex is v.
- AGREEMENT: All decided vertices are the same.

Observe that the only thing that we have modified is the VALIDITY property, and with that change the very definition of the problem implies that any algorithm solving it does not have the solo-trivial property, and thus the reduction in Lemma 2.4.2 implies that Gathering with the same-vertex-validity is impossible on any non-trivial graph.

Corollary 2.4.1. The Gathering problem with the same-vertex-validity is unsolvable in the ALR model on any non-trivial graph, even if robots are

non-oblivious, share the same labeling of the graph, are able to detect multiplicities and possess an unbounded number of light colors.

Hence, although it is possible to solve the Gathering problem without termination using more powerful capabilities or restricting the initial configurations, the simple fact of demanding strong termination makes the problem unsolvable as long as the solo-trivial property does not hold, and the reason is that any solution to Gathering implies also a solution for the Binary Consensus problem.

In the light of this result, a natural question that arises is if there exists weakened Gathering type problems for which it is possible to achieve strong termination. In the rest of the paper, we will study two weaker variants of Gathering with strong termination that are not only solvable in ALR; they are solvable by a more challenging and novel model in which robots might crash and appear asynchronously at any time on the execution. In fact, this model turns out to be equivalent to WFSM, and the impossibility of the problems on some graphs will be obtained by using fundamental impossibility results in that model.

Chapter 3

The Extended ALR Model (EALR)

In this chapter, we extend the ALR model to encompass crash failures and asynchronous appearing times, calling this new model EALR. Later, we show that the weakened Gathering problems, Edge-Gathering and $\mathcal{K}$-Gathering, with strong termination can be solved not only in ALR but also in the more challenging EALR model.

More in detail, in the extended version of ALR model (EALR) we consider three additional assumptions:

1. Robots might present a stopping failure at any time during the execution (same as in [11, 42, 43]), from which it never recovers, causing the robot to stop its movement, but leaving visible at all time the last value set on its light. Therefore, it is not possible for a robot to distinguish between one that is very slow of another that has presented a failure.
2. Robots may use an unbounded number of light colors. Namely, a robot can show any positive integer on its light.
3. We lift an assumption made in almost all previous work: we do not require that all robots must be present initially. Thus, robots may appear in (not necessarily distinct) vertices of G at any time. Some robots may never appear due to the crash failures (hence, never seen by other robots), and once a robot is placed on a vertex, it starts running its local algorithm asynchronously together with the other robots. We call this property ***Asynchronous Appearing Times***.

One point that we want to highlight is that although in EALR the light of a crashed robot is still visible for all active robots, our algorithms can tolerate robots disappearing after crashing, i.e., the scheduler decides that the last step of a faulty robot is a Move operation setting its light to

a negative value.

It is considered that initially all robots are *inactive*, and none of them is present in the graph. To model the asynchronous appearing times of each robot, we use the lights in the following way: the initial light value is set to -1, symbolizing the fact that its participation in the execution has not yet begun, and it remains invisible to other robots. At the moment that p_i sets its light to a non-negative value, then it becomes visible to the others, and we say that p_i is an *active* robot.

Definition 3.0.1. A robot is ***inactive*** if the value set in its light is negative; otherwise, it is said that the robot is ***active***.

Observe that initially all robots are inactive, thus, the first action that each of them must perform is a MOVE operation (since it is the only instruction that modifies the state of the light) with its initial vertex and its light set to a non-negative value. Also, it should be remarked that the LOOK operation is modified to discards the state of any robot whose light is showing a negative value (those that are inactive).

If the lights are unnecessary to solve a problem (we allow robots to solve problems without the help of lights), we will consider a single *fictitious* light that merely models when a robot starts executing its algorithm and becomes visible, whose value will always be 0 once the robot is active. Note that with the previous assumption, it is straightforward to simulate models where robots have no lights, or they are visible since the begging of the execution (with synchronous appearing times), only be initializing the light of every robot to 0 instead of -1.

Same as in the WFSM model, EALR considers that the number of robots that may fail in every execution is at most $N - 1$; thus, robots run a *wait-free* algorithm. Take into consideration that as $N - 1$ robots might crash without becoming active, the *solo executions* are possible, where only one robot runs its algorithm without detecting any other until it decides.

Definition 3.0.2. We will say that a robot is ***correct*** if by extending its execution indefinitely it performs an infinite number of steps. Otherwise, it is a ***crashed*** robot.

3.1 Weakened Gathering type problems

In this section, we will analyze two weaker Gathering type problems with termination: Edge-Gathering and $\mathcal{K}$-Gathering. In which the VALIDITY property prioritizes the idea of closeness: if robots start on a "small region", they should stay there. In the first problem, robots decide vertices that either are the same or belong to the same edge. While in second, robots are required to decide vertices that are at a distance of at most one (hence, they decide vertices of a complete subgraph).

3.1.1 Edge-Gathering

In the Edge-Gathering problem, we allow robots to swarm at the same vertex or vertices belonging to the same edge. Formally:

Definition 3.1.1. In the ***Edge-Gathering*** problem on a graph G, each robot starts with a vertex of G and executes its code, so that the following three properties are satisfied:

1. TERMINATION: Every correct robot decides a vertex of G.
2. VALIDITY: If all active robots start on the same vertex v, then every decided vertex is v. If the initial vertices of the active robots cover an edge e, then every decided vertex is a vertex of e.
3. EDGE-AGREEMENT: All decided vertices belong to the same edge.

Unexpectedly, even if we allow robots to end up close enough to each other, the Edge-Gathering problem remains unsolvable in EALR if the base graph has a cycle (Theorem 4.3.4). However, in the next sections, we present an algorithm to solve it on any tree using $diam(G) - 1$ light colors and LCM cycles. Also, we prove that Edge-Gathering is unsolvable unless lights are available for communication (Theorem 4.3.2).

3.1.2 $\mathcal{K}$-Gathering

An interesting observation in the Edge-Gathering problem that must be highlighted is that if all pairs of robots decide vertices at a distance at most one,

then the EDGE-AGREEMENT property is not necessarily satisfied. Let's take for example an execution with three robots on a graph G that contains K_3 (complete graph with three vertex), where each robot decides different vertices of K_3. See figure 3.1.

Although it is clear that the EDGE-AGREEMENT property is not fulfilled in the previous scenario, the result does not seem negligible, because although the robots are not on the same vertex or the same edge, they are very close among them. So, with this in mind, we want to present an even weaker version of Gathering, which generalizes the VALIDITY property: if robots start in a "small enough region", they should stay there. Observe that Edge-Gathering and $\mathcal{K}$-Gathering are the same problems for two robots. Also, note that every solution to Edge-Gathering also solves $\mathcal{K}$-Gathering, while the opposite might not be true.

Definition 3.1.2. In the ***$\mathcal{K}$-Gathering*** problem on a graph G, each robot starts with a vertex of G and executes its code, so that the following three properties are satisfied:

- TERMINATION: Every correct robot decides a vertex of G.
- VALIDITY: If all of active robots start on a set of vertices of a complete subgraph, then every decided vertex must belongs to that set.
- $\mathcal{K}$-AGREEMENT: All decided vertices belong to the same complete subgraph.

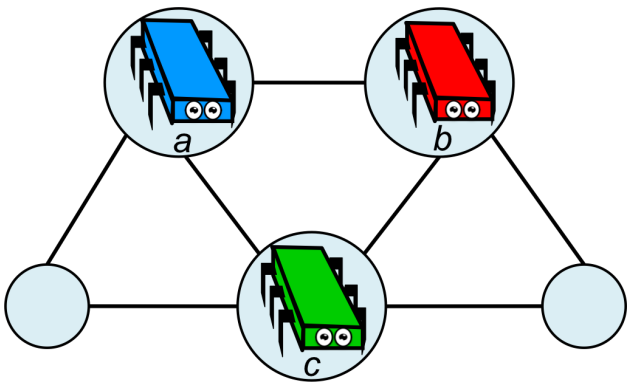

Figure 3.1: If the figure represents the initial positions of the robots and we want to solve the $\mathcal{K}$-Gathering problem, then any subset of $\{ a, b, c \}$ is a valid final positions of the robots.

3.2 Algorithm for Edge-Gathering on trees

Algorithm 4 solves Edge-Gathering for $N \geq 2$ robots on any tree T. The general idea is that every robot goes through a sequence of $diam(T) - 1$ rounds. In every round, a robot p_i uses its light to announce its round number and atomically collects (in $view_i$ using a LOOK operation) the position and the state of the light of all active robots in the system. Then, p_i considers the set S_i containing its current vertex v_i and the positions of robots executing the maximal round (max_round_i), these vertices define a subtree T_i of T. Later, if v_i is a leaf of T_i then p_i moves one step inside T_i, otherwise it does not move. Intuitively, robots in maximal rounds are the leaders, and indicate to the rest where to gather.

Algorithm 4 Edge-Gathering algorithm for $N \geq 2$ robots on any tree $T = (V, E)$. Code for robot p_i.

Function EdgeGatheringTree(v_i, T)
1: // p_i becomes visible to the others
2: **Move**$(v_i, 0)$
3: **for** $r_i \leftarrow 1$ **to** $diam(T) - 1$ **do**
4: // position and light color of all active robots
5: $view_i \leftarrow$ **Look**(T)
6: $max\_round_i \leftarrow \max\{r_j : (*, r_j) \in view_i\}$
7: // positions of robots in maximum round and p_i's position
8: $S_i \leftarrow \{v_j : (v_j, max\_round_i) \in view_i \vee v_j = v_i\}$
9: // subtree induced by positions in S_i
10: $T_i \leftarrow$ smallest subtree of T spanning all vertices in S_i
11: **if** v_i is a leaf of $T_i \wedge diam(T_i) > 0$ **then**
12: $v_i \leftarrow$ vertex of T_i that is adjacent to v_i
13: **end if**
14: // p_i makes visible its new position and updates its color light
15: **Move**(v_i, r_i)
16: **end for**
17: **return** v_i

Although Algorithm 4 is simple its behavior is not, due to the combination of asynchrony, distinct appearing times and failures. For example, in a given execution, due to asynchrony, two robots could compute their next vertices using the same maximal round, but with very different sets of positions; moreover, a robot could compute its next position considering only crashed robots. Such difficulties make the correctness of the algorithm far from trivial. Notice that in Algorithm 4 robots do not share any labeling or orientation of T.

Now, we need to prove that each property (TERMINATION, VALIDITY and EDGE-AGREEMENT) of the Edge-Gathering problem holds.

Lemma 3.2.1. *Algorithm 4 satisfies the* TERMINATION *property of the Edge-Gathering problem.*

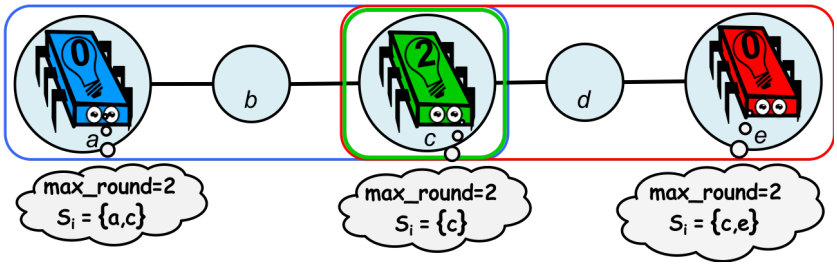

Figure 3.2: Scenario where different robots compute the same maximum round but with very different sets S_i. The nodes within the blue frame correspond to the subtree T_i of the blue robot; analogous to the others.

Proof. Algorithm 4 contains only one loop controlled by variable r_i, that it is never altered inside the loop, so, every robot runs exactly $diam(T) - 1$ rounds and then it decides a vertex. Therefore, the TERMINATION property is satisfied. □

The proof of Lemma 3.2.2 shows that Algorithm 4 satisfies the VALIDITY property of Edge-Gathering. Actually, proves a stronger property: the decision of a robot belongs to the subtree spanning the initial positions of all the active robots. In other words, if T^0 is the minimum subtree of T generated by the initial vertices of all the active robots, then the final vertices of the correct robots belong to T^0. Note that T^0 is an abstract concept, since it may not exist during the execution of the algorithm. Subsequent lemmas use the following notation.

Definition 3.2.1. Let α' be a prefix of an execution α. The ***real position*** of a robot p_i in α' is the vertex belonging to the last MOVE operation made by p_i in α'.

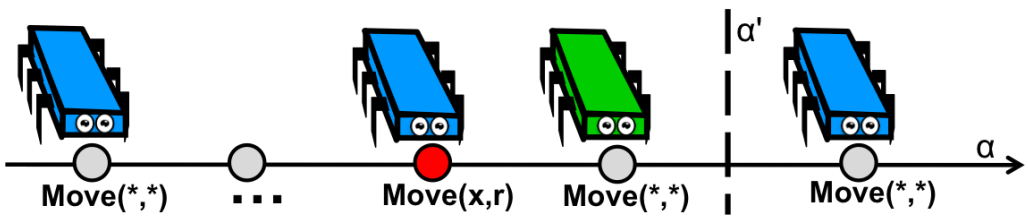

Figure 3.3: The execution α contains all the steps made by the robots. The vertex x in the red step corresponds to the real position of the blue robot under the prefix α'.

Definition 3.2.2. For every local variable x_i of p_i, $x^r_{i,\alpha'}$ will denote the value of x_i in round r in a prefix α'. With this notation, $v^0_{i,\alpha'}$ is the initial vertex of p_i. When α' is understood it is omitted from the subscript.

Lemma 3.2.2. *Let α be any execution of Algorithm 4. We claim that for every prefix α' of α and for every active robot p_i in α', T^0 contains the real position of p_i in α'.*

Proof. We prove the claim by strong induction over the steps in α'.

Base Case: $|\alpha'| = 1$. Hence, α' contains only one step, that must be $\textsc{Move}(v_i, 0)$ for some robot p_i. The claim holds as the current value of v_i corresponds to the initial vertex of p_i.

Inductive Hypothesis: Suppose that the claim holds for the prefix α'.

Inductive Step: We will show that the claim holds when one more step is executed, namely, $\alpha' \cdot e$.

Case 1: $e = \textsc{Look}$ or $\textsc{Compute}$
Then, all the real positions are the same, both in α' and $\alpha' \cdot e$. Hence, the claim holds by I.H.

Case 2: $e = \textsc{Move}(v_i, r_i)$

> Sub-case 2.1: $r_i = 0$
> The claim holds, because the current value of v_i corresponds to the initial vertex of p_i, and the claim holds for every prefix of α' by I.H.
>
> Sub-case 2.2: $r_i \geq 1$.
> By I.H. we know that all the real positions contained in α' belong to T^0, so we just need to prove that $v_i \in T^0$. Now, to compute v_i, p_i uses a set of vertices $S = \{u_1, \ldots, u_x\}$ which is a subset of the vertices that it collects in its r-th round. Observe that for every $u \in S$ there is a prefix α'' of α' such that u is the real position of a robot p_j, and then, by I.H. $u \in T^0$. Hence, if T_S represents the smallest subtree of T spanning the vertices in S, we have that $T_S \subseteq T^0$, because if we suppose that there exists a vertex of T_S that is not contained in T^0, then there would be a cycle in T, contradicting the hypothesis that T is a tree (See figure 3.4). Finally, following the pseudo-code of the Algorithm 4, or p_i does not move or it moves one step forward to the center of T_S. In both cases, $v_i \in T^0$.

□

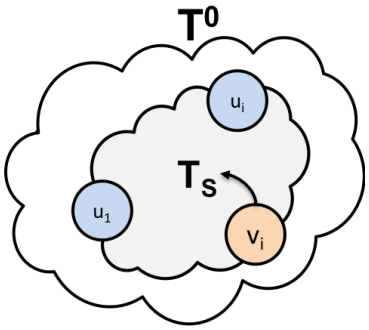

Figure 3.4: T_S represents the smallest subtree of T spanning the vertices in S. We have that $T_S \subseteq T^0$.

Lemma 3.2.3. *Algorithm 4 satisfies the* VALIDITY *property of Edge-Gathering.*

Proof. If the initial vertices of all active robots are the same or they cover an edge, then by lemma 3.2.2 it is known that every decided vertex belongs to T^0, which implies that the VALIDITY property of Edge-Gathering is satisfied. □

The hardest part of the correctness proof of Algorithm 4 is showing the EDGE-AGREEMENT property, that it is based on Lemma 3.2.4 below. Which roughly speaking, shows that as rounds go by, positions of robots are not far away from each other.

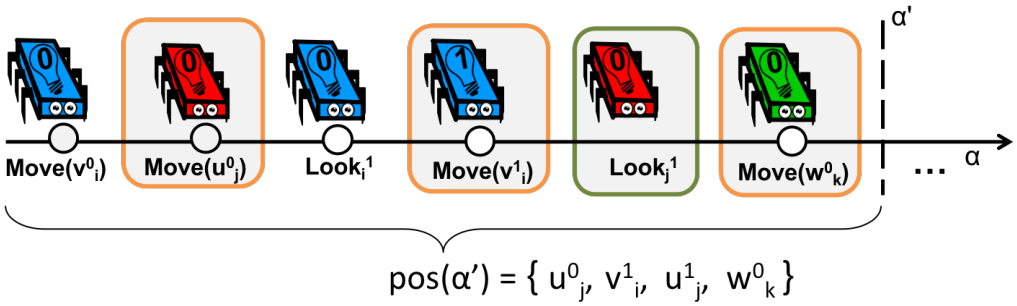

Figure 3.5: The last event of the red robot in α' is a LOOK operation, which implies that it has two positions, a real one highlighted in an orange frame and a computed one in the green frame. The other robots only have a real position.

Due to the LOOK and COMPUTE operations only modify the internal state of a robot, we are going to assume without loss of generality that both

instructions are performed atomically, since from that moment the following state of p_i is completely defined. So, if the last operation of a robot p_i in a prefix α' is its LOOK operation of the r-th round, we will say that p_i has a real position in vertex v_i^{r-1} and a ***computed*** one in v_i^r (See figure 3.2.3). Otherwise, p_i will have only one real position. It should be noted that the computed position of p_i only exists in its local variables and therefore is unknown to all other robots.

Definition 3.2.3. Given a prefix α' of an execution α, $pos(\alpha')$ denotes the set of both real and computed positions of all the active robots in α'.

Lemma 3.2.4. *Let α be any execution of Algorithm 4. We claim that for any prefix α' of α and for any pair of position $v_i^r, v_j^m \in pos(\alpha')$ it holds that:*

$$dist(v_i^r, v_j^m) \leq diam(T) - min(r, m)$$

Proof. By strong induction on the steps included in α'.

Base Case: $|\alpha'| = 1$. Hence, α' contains a single step, which must be MOVE$(v_i, 0)$ for some robot p_i. In this case, there is only one active robot in α', so $pos(\alpha') = \{ v_i^0 \}$. Thus, the claim of the lemma clearly holds for α'.

Inductive Hypothesis: Suppose that the claim holds for prefix α'.

Inductive Step: We need to prove that the claim still holds when one more step is performed, namely, $\alpha' \cdot e$.

Case 1: $e =$ MOVE(v_i, r)

> Sub-case 1.1: $r = 0$
> The positions of $\alpha' \cdot e$ are all the positions of α' and v_i^0 (which is the real position of p_i), i.e., $pos(\alpha' \cdot e) = pos(\alpha') \cup \{ v_i^0 \}$. For any two positions in $pos(\alpha')$ the claim holds by I.H, and for v_i^0 and any position $v_j^m \in pos(\alpha')$ we have that $dist(v_i^0, v_j^m) \leq diam(T) \leq diam(T) - min(0, m)$ because both v_i^0 and v_j^m are vertices of T and $0 \leq m \leq diam(T) - 1$ because m is a valid round number.
>
> Sub-case 1.2: $r \geq 1$
> Then the last step of p_i in α' is the LOOK operation of the r-th round. Hence, in α', v_i^r is the computed position of p_i and v_i^{r-1} is the real one. Then, in $\alpha' \cdot e$ the computed position simply becomes real replacing v_i^{r-1}, i.e., $pos(\alpha' \cdot e) = pos(\alpha') \setminus \{ v_i^{r-1} \}$. Thus, by I.H. the claim holds for $\alpha' \cdot e$.

Case 2: $e = \text{Look}_i^{r+1}(T)$
Observe that the last step of p_i in α' is the Move operation of the r-th round, hence, v_i^r is its real position in α'. Thus, the positions in $\alpha' \cdot e$ are all the position of α' together with v_i^{r+1}, which is the computed position of p_i, i.e., $pos(\alpha' \cdot e) = pos(\alpha') \cup \{ v_i^{r+1} \}$. By I.H. the claim holds for any two positions in $pos(\alpha')$. Thus, we only need to show that $dist(v_i^{r+1}, v_j^m) \leq diam(T) - min(r+1, m)$ for any (real or computed) position $v_j^m \in pos(\alpha')$.

Sub-case 2.1: $dist(v_i^{r+1}, v_j^m) < dist(v_i^r, v_j^m)$
(2) $dist(v_i^r, v_j^m) \leq diam(T) - min(r, m)$ — I.H.
(3) $dist(v_i^{r+1}, v_j^m) < diam(T) - min(r, m)$ — (2.1) and (2)

Sub-case 2.1.1: (4) $r \geq m$
(5) $min(r, m) = m$ — From (4)
(6) $min(r+1, m) = m$ — From (4)
(7) $dist(v_i^{r+1}, v_j^m) < diam(T) - m$ — (3) and (5)
(8) $dist(v_i^{r+1}, v_j^m) \leq diam(T) - min(r+1, m)$ — (7) and (6)

Sub-case 2.1.2: (4) $r < m$
(5) $min(r, m) = r$ — From (4)
(6) $min(r+1, m) = r+1$ — From (4)
(7) $dist(v_i^{r+1}, v_j^m) < diam(T) - r$ — (3) and (5)
(8) $dist(v_i^{r+1}, v_j^m) \leq diam(T) - r - 1$ — (7) and Prop. $<$
(9) $dist(v_i^{r+1}, v_j^m) \leq diam(T) - min(r+1, m)$ — (8) and (6)

Sub-case 2.2: $dist(v_i^{r+1}, v_j^m) > dist(v_i^r, v_j^m)$
Then p_i moved to an adjacent vertex with $v_i^{r+1} \neq v_i^r$ and $(v_i^r, v_i^{r+1}) \in E$. Thus, the conditions in line 11 of algorithm 4 was satisfied. Therefore, $diam(T_i^{r+1}) > 0$ and v_i^r is a leaf in T_i^{r+1}.

Sub-case 2.2.1: $v_j^m \in T_i^{r+1}$
As v_i^r is a leaf of T_i^{r+1} and $v_j^m \in T_i^{r+1}$, the only situation that leads to have (2.2) is that $v_i^r = v_j^m$. Thus, $dist(v_i^r, v_j^m) = 0$ but as $(v_i^r, v_i^{r+1}) \in E$ then $dist(v_i^{r+1}, v_j^m) = 1 \leq diam(T) - min(r+1, m)$, because $0 \leq m, (r+1) \leq diam(T) - 1$ due to m and $(r+1)$ are valid round numbers.

Sub-case 2.2.2: $v_j^m \notin T_i^{r+1}$
So that the position of p_j is not found in T_i^{r+1}, it means that $v_j^m \notin S_i^{r+1}$, which indicates that p_i had to see at least another robot running a round equal to or greater than m, depending on whether v_j^m is a real or computed position. Therefore, $m \leq max\_round_i^{r+1} = \ell$. On the other hand, as $diam(T_i^{r+1}) > 0$ and

$dist(v_i^{r+1}, v_j^m) > dist(v_i^r, v_j^m)$, then there exits at least one leaf $v_k^\ell \in T_i^{r+1}$ such that $v_i^{r+1} \in Path(v_k^\ell, v_j^m)$, which implies that (1) $dist(v_i^{r+1}, v_j^m) \leq dist(v_k^\ell, v_j^m)$ (See figure 3.6). Observe that $v_k^\ell \in pos(\alpha')$.

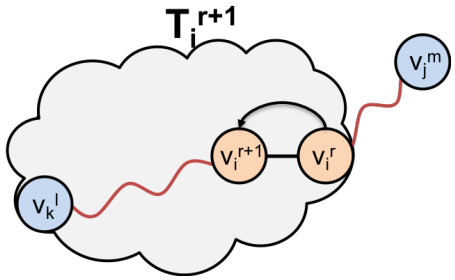

Figure 3.6: As $v_j^m \notin T_i^{r+1}$ then there exits at least one leaf $v_k^\ell \in T_i^{r+1}$ such that $v_i^{r+1} \in Path(v_k^\ell, v_j^m)$

Sub-case 2.2.2.1: (2) $r \geq m$

(3) $min(r+1, m) = m$	From (2)
(4) $dist(v_i^{r+1}, v_j^m) \leq dist(v_k^\ell, v_j^m)$	From (1)
(5) $dist(v_k^\ell, v_j^m) \leq diam(T) - min(\ell, m)$	I.H.
(6) $dist(v_k^\ell, v_j^m) \leq diam(T) - m$	(5) and $\ell \geq m$
(7) $dist(v_k^\ell, v_j^m) \leq diam(T) - min(r+1, m)$	(6) y (3)
(8) $dist(v_i^{r+1}, v_j^m) \leq diam(T) - min(r+1, m)$	(4) and (7)

Sub-case 2.2.2.2: (2) $r < m$

(3) $min(r+1, m) = r+1$	From (2)
(4) $dist(v_i^{r+1}, v_j^m) \leq dist(v_k^\ell, v_j^m)$	From (1)
(5) $diam(T) - m \leq diam(T) - (r+1)$	From (2)
(6) $dist(v_k^\ell, v_j^m) \leq diam(T) - min(\ell, m)$	I.H.
(7) $dist(v_k^\ell, v_j^m) \leq diam(T) - m$	(6) and $\ell \geq m$
(8) $dist(v_k^\ell, v_j^m) \leq diam(T) - (r+1)$	(7) y (5)
(9) $dist(v_k^\ell, v_j^m) \leq diam(T) - min(r+1, m)$	(8) and (3)
(10) $dist(v_i^{r+1}, v_j^m) \leq diam(T) - min(r+1, m)$	(4) and (9)

- Sub-case 2.3: $dist(v_i^{r+1}, v_j^m) = dist(v_i^r, v_j^m)$
  Note that this case only happens when $v_i^{r+1} = v_i^r$, i.e., p_i did not move, so at least one of the two conditions in line 11 of Algorithm 4 was not fulfilled. Therefore, $diam(T_i^{r+1}) = 0$ or v_i^r is not a leaf in T_i^{r+1} with $diam(T_i^{r+1}) \geq 2$ (it is necessary at least 3 vertices in T_i^{r+1} so that v_i^r is not a leaf).

  Also, observe that if $r \geq m$, then $min(r, m) = m = min(r+1, m)$ and $dist(v_i^{r+1}, v_j^m) = dist(v_i^r, v_j^m) \leq_{I.H.} diam(T) - min(r, m) = diam(T) - m = diam(T) - min(r+1, m)$.

Thus, assume that $r < m$, then $min(r, m) = r$, $min(r + 1, m) = r + 1$ and $diam(T) - m \leq diam(T) - (r + 1)$. By I.H. $dist(v_i^r, v_j^m) \leq diam(T) - \min(r, m) = diam(T) - r$. In what follows we show that $dist(v_i^{r+1}, v_j^m) \leq diam(T) - \min(r + 1, m) = diam(T) - (r + 1)$.

Sub-case 2.3.1: $diam(T_i^{r+1}) = 0$
- If $v_j^m \in T_i^{r+1}$ then $v_j^m = v_i^{r+1}$. Therefore, $dist(v_i^{r+1}, v_j^m) = 0 \leq diam(T) - (r + 1)$ because $(r + 1)$ is a valid round number and $0 \leq (r + 1) \leq diam(T) - 1$.
- If $v_j^m \notin T_i^{r+1}$ but $v_j^{m-1} \in T_i^{r+1}$, then $v_j^{m-1} = v_i^{r+1}$ with $dist(v_i^{r+1}, v_j^{m-1}) = 0$, but as $dist(v_j^{m-1}, v_j^m) = 1$ we have $dist(v_i^{r+1}, v_j^m) = 1 \leq diam(T) - (r + 1)$ because $(r + 1)$ is a valid round number and $0 \leq (r + 1) \leq diam(T) - 1$.
- If $v_j^m \notin T_i^{r+1}$ and $v_j^{m-1} \notin T_i^{r+1}$, it was because p_i detected at least one robot p_k over a vertex v_k^ℓ executing a round greater to or equal to m (depending on the real position of p_j). Hence, $\ell = max\_round_i^{r+1} \geq m$ and $m = min(\ell, m)$. Now, as $diam(T_i^{r+1}) = 0$ we have that $v_k^\ell = v_i^{r+1}$, thus $dist(v_i^{r+1}, v_j^m) = dist(v_k^\ell, v_j^m) \leq_{I.H.} diam(T) - min(\ell, m) = diam(T) - m \leq diam(T) - (r + 1)$.

Sub-case 2.3.2: v_i^r is not a leaf in T_i^{r+1} and $diam(T_i^{r+1}) \geq 2$.
Due to p_j is an active robot its real position (v_j^m or v_j^{m-1}) must be in $view_i^{r+1}$ and hence $\ell = max\_round_i^{r+1} \geq m - 1$.
On the other hand, as v_i^r is not a leaf its degree in T_i^{r+1} is at least 2, then there exists a vertex v_k^ℓ belonging to a robot p_k that is a leaf in T_i^{r+1} where $v_i^{r+1} \in Path(v_k^\ell, v_j^m)$ (See figure 3.7).

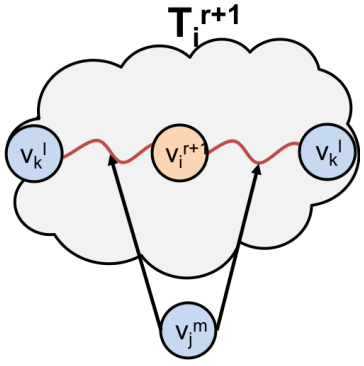

Figure 3.7: If v_i^r is not a leaf in T_i^{r+1} then there exists a vertex v_k^ℓ such that $v_i^{r+1} \in Path(v_k^\ell, v_j^m)$.

Observe that $v_i^{r+1} \neq v_k^\ell$, hence $dist(v_i^{r+1}, v_j^m) < dist(v_k^\ell, v_j^m)$ and by I.H. $dist(v_k^\ell, v_j^m) \leq diam(T) - min(l, m)$. Now, as $r \leq (m - 1) \leq min(l, m)$ then $diam(T) - min(l, m) \leq diam(T) - r$, thus

$dist(v_i^{r+1}, v_j^m) < dist(v_k^\ell, v_j^m) \leq diam(T) - r$. Finally, $dist(v_i^{r+1}, v_j^m) \leq diam(T) - (r+1)$

□

Lemma 3.2.5. *Algorithm 4 satisfies the* Edge-Agreement *property of Edge-Gathering.*

Proof. Consider any execution of Algorithm 4, and let α be any prefix of it in which every non-faulty robot has decided. Let $v_i^{diam(T)-1}$, $v_j^{diam(T)-1}$ be the real positions in α of two non-faulty robots p_i and p_j. Then, $v_i^{diam(T)-1}$ (resp. $v_j^{diam(T)-1}$) is the decision of p_i (resp. p_j). By Lemma 3.2.4:

$$dist(v_i^{diam(T)-1}, v_j^{diam(T)-1}) \leq 1$$

which implies that the decided vertices of p_i and p_j are at distance at most 1. Finally, observe that all decided vertices cover a vertex or an edge, because T is a tree and it has no triangles. □

Theorem 3.2.1. *Algorithm 4 solves Edge-Gathering in the EALR model for $N \geq 2$ luminous, oblivious and anonymous robots, on any tree T in $diam(T) - 1$ rounds using $diam(T) - 1$ distinct light colors and tolerating $N - 1$ crash failures.*

Proof. The proof follows directly from lemmas [3.2.1, 3.2.3, 3.2.5] □

3.3 Algorithms for $\mathcal{K}$-Gathering

In this section, we expose three algorithms to solve the $\mathcal{K}$-Gathering problem on particular graph families. The first solves the problem on any tree. The second one works on graphs with at least one dominating vertex. And the last one deals with graphs whose related clique is also a tree.

3.3.1 Solution on trees

Corollary 3.3.1. Algorithm 4 solves the $\mathcal{K}$-Gathering problem in the EALR model for $N \geq 2$ luminous, oblivious and anonymous robots, on any tree T in $diam(T) - 1$ rounds using $diam(T) - 1$ distinct light colors and tolerating $N - 1$ crash failures.

Proof. Theorem 3.2.1 shows that Algorithm 4 solves the Edge-Gathering problem on trees. But, observe that in the case of trees, the Edge-Gathering and $\mathcal{K}$-Gathering problems are equivalent: a tree has no triangles, hence, if the robots' decisions are at distance 1, then all of them must belong to the same edge. Thus, Algorithm 4 also solves $\mathcal{K}$-Gathering on trees. □

3.3.2 Solution on graphs with a dominant vertex

Algorithm 5 solves $\mathcal{K}$-Gathering for $N \geq 2$ robots on any connected graph G with at least one dominating vertex. The algorithm works as follows: every robot simply collects the positions of all active robots and decides its initial position if it sees no conflict (i.e., all vertices belong to the same complete graph). Otherwise, it moves to any dominating vertex and decides it.

Observe that in Algorithm 5 robots do not use lights to communicate information, but lets remember that there is always a fictitious light that it is used only to indicate the participation of the robot on the execution.

Algorithm 5 $\mathcal{K}$-Gathering algorithm for $N \geq 2$ robots on any connected graph $G = (V, E)$ with a dominating vertex $\widehat{v}$. Code for robot p_i.

Function KGatheringDominatingVertex(v_i, G)

1: // p_i becomes visible to others
2: **Move**(v_i)
3: $view_i \leftarrow$ **Look**(G)
4: // if p_i sees a conflict
5: **if** $\exists v_j \in view_i$ such that $dist(v_i, v_j) \geq 2$ **then**
6: // p_i moves to some dominating vertex $\widehat{v}$
7: $v_i \leftarrow \widehat{v}$
8: // p_i makes visible its new position
9: **Move**(v_i)
10: **end if**
11: **return** v_i

Lemma 3.3.1. *Every execution of Algorithm 5 satisfies the* Termination *property of the $\mathcal{K}$-Gathering problem.*

Proof. Algorithm 5 terminates because it does not contain any loop, so every correct robot decides a vertex in one LCM round. □

Lemma 3.3.2. *Every execution of Algorithm 5 satisfies the* Validity *property of the $\mathcal{K}$-Gathering problem.*

Proof. If the initial positions of all active robots belong to the same complete subgraph, then the distance between any pair of positions is at most one. Thus, the condition in line 5 is not satisfy and no robot executes line 7. Therefore, all decided vertices belong to the initial subgraph. □

Lemma 3.3.3. *Algorithm 5 satisfies the* $\mathcal{K}-$AGREEMENT *property of the* $\mathcal{K}$*-Gathering problem.*

Proof. Due to G has at least one dominating vertex, the diameter of G is at most 2. And two robots cannot decide vertices at distance 2, because the robot that takes the last snapshot among them will notice that there is a conflict and will move to one of the dominating vertices. Therefore, any pair of robots ends in the same complete subgraph. □

Theorem 3.3.1. *Let* G *be a graph with at least one dominating vertex, then* $\mathcal{K}$*-Gathering is solvable in the ALR model for* $N \geq 2$ *oblivious and anonymous robots on* G *in only* 1 *round, without the use of lights and tolerating* $N-1$ *crash failures.*

Proof. The proof follows directly from lemmas [3.3.1, 3.3.2, 3.3.3] □

3.3.3 Solution when the clique graph ($K(G)$) is a tree

The *clique graph* of G, represented by $K(G)$, is the graph whose vertices are the maximal complete subgraphs of G, and there is an edge between two vertices if the intersection of the corresponding graphs is non-empty. Formally, the clique graph of G is the graph $K(G) = (V_K, E_K)$ with $V_K = \{K_x : K_x \text{ is a maximal complete subgraph of } G\}$ and $E_K = \{(K_x, K_y) : V(K_x) \cap V(K_y) \neq \emptyset\}$. See figure 3.8 for an example of a clique graph.

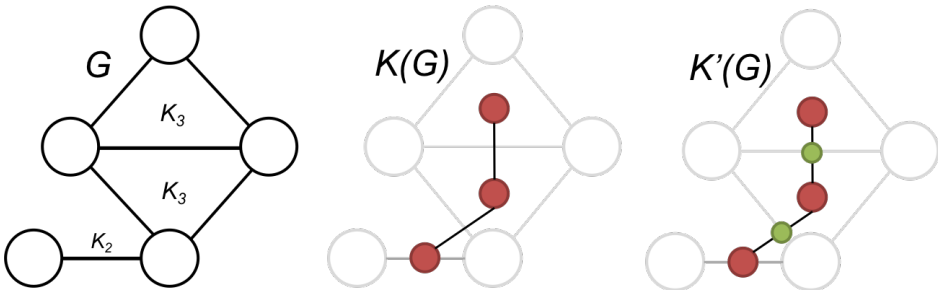

Figure 3.8: Example of clique graph $K(G)$ and the subdivision $K'(G)$.

Let $K'(G)$ be the graph obtained from $K(G)$ by subdividing each of its edges: every $(K_x, K_y) \in E(K(G))$ is replaced with (K_x, K_{xy}) and (K_{xy}, K_y)

in $K'(G)$, where K_{xy} is the complete subgraph defined by the intersection of K_x and K_y. Thus, every vertex of $K'(G)$ is a complete subgraph of G. For example, in figure 3.8 the upper green vertex corresponds to the intersection of the two completed K_3 graphs, while the other intersects the lower K_3 graph with K_2.

Remark 1. If $K(G)$ is a tree then $K'(G)$ is a tree. Also, $diam(K'(G)) \leq 2 * diam(K(G))$ and $|V(K'(G))| \leq 2 * |V(K(G))|$.

In what follows we prove that algorithm 6 solves the $\mathcal{K}$-Gathering problem for $N \geq 2$ robots on any graph G whenever $K(G)$ (the clique graph) is a tree. The idea behind of the algorithm is that robots simulate Algorithm 4 over the graph $K'(G)$. In the simulation, robots do not actually move, they simple announce in their lights where they move in the simulation. Finally, to solve $\mathcal{K}$-Gathering on G every robot computes its decision based on the result of the simulation and then moves towards it. To do that, robots use any pair of functions $f_{in} : V(G) \to V(K(G))$ and $f_{out} : V(K'(G)) \to V(G)$ such that every $v \in V(G)$ belongs to the subgraph $f_{in}(v)$, and for every subgraph $K \in V(K'(G))$, $f_{out}(K)$ is a vertex of K. The aim of these functions is to transform inputs for $\mathcal{K}$-Gathering on G to inputs for algorithm 4, and outputs of that algorithm to outputs for $\mathcal{K}$-Gathering on G. Note that the codomain of f_{in} is $V(K(G))$ because we want to map each vertex of G to a maximal complete subgraph of G containing that vertex. See figure 3.10 for an example of the application of the functions.

Lemma 3.3.4. *Function f_{in} can only map each vertex $v \in V(G)$ to at most two different and adjacent vertices in $K(G)$.*

Proof. By contradiction. Suppose that there exists a vertex $v \in V(G)$ such that function f_{in} can map v to three or more vertices in $K(G)$ and let C_1, C_2 and C_3 any three of those vertices. By definition of f_{in}, v must be a vertex of the maximal subgraphs represented by C_1, C_2 and C_3, and by definition of $K(G)$ there must be an edge between all of these vertices (See figure 3.9). Thus, $K(G)$ contains K_3. Contradiction!, because $K(G)$ is a tree. Therefore, function f_{in} can map vertex v to at most two vertices of $K(G)$.

Now, observe that by definition of $K(G)$, if some vertex v can be mapped to two different vertices $x, y \in V(K(G))$ by function f_{in}, then x and y must be adjacent. Because, by definition of f_{in}, v must appears in the maximal complete subgraphs x and y, and hence, there must be an edge connecting them. □

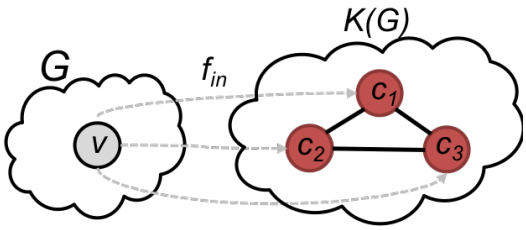

Figure 3.9: Each vertex $v \in V(G)$ can only be mapped by function f_{in} to at most two adjacent vertex in $K(G)$, because if we assume the opposite then $K(G)$ contains a cycle.

Algorithm 6 $\mathcal{K}$-Gathering algorithm for $N \geq 2$ robots on any connected graph $G = (V, E)$ whose clique graph $K(G)$ is a tree. Code for robot p_i.

Function KGatheringCliqueTree$(v_i, G, f_{in}, f_{out})$

1: // p_i becomes visible to others
2: **Move**$(v_i, v_i|\perp|\perp)$
3: /* p_i simulates algorithm 4 over the subdivision of the clique graph and stores the result vertex
4: $u_i \leftarrow$ Simulate EdgeGatheringTree$(f_{in}(v_i), K'(G))$
5: /* p_i stores the value of its light, because it will start moving but it cannot change the status of its light because it is possible that other robots are still simulating algorithm 4 */
6: $r_i \leftarrow$ light value of p_i
7: /* p_i gets the initial positions of the robots to decide the complete subgraph where to move. Observe that we need to retrieve the initial positions of the robots through the light because it is possible that some robots are currently moving on the graph */
8: $S_i \leftarrow \{ w_i : (*, w_i \mid *|*) \in Look(G) \}$
9: /* Instead of applying f_{out} directly, p_i first checks is there is an initial position of a robot that belongs to the subgraph u_i. This instruction is to guarantee the VALIDITY property*/
10: **if** exists an initial position $w_i \in S_i$ such that w_i belongs to the complete subgraph u_i, i.e. $w_i \in V(u_i)$ **then**
11: $\quad x_i \leftarrow w_i$
12: **else**
13: $\quad x_i \leftarrow f_{out}(u_i)$
14: **end if**
15: // p_i moves until it reach the decided vertex
16: **while** $x_i \neq v_i$ **do**
17: $\quad v_i \leftarrow$ closest vertex to x_i
18: $\quad$ /* Observe that while p_i is moving to the decided vertex, it conserves the status of its light, to be according to those robots that are in the simulating phase */
19: $\quad$ **Move**(v_i, r_i)
20: **end while**
21: **return** v_i

To perform the simulation of Algorithm 4, the light's value of each robot p_i will be divided in three parts. The first one will permanently show

its input vertex v_i (hence, $|V(G)|$ light colors are needed), the second part will show the vertex where p_i is standing on in the simulated graph $K'(G)$ ($2 * |V(K(G))|$ additional light colors) and the last one will store the round number of the simulated algorithm 4 (at most $2 * diam(K(G))$ light colors).

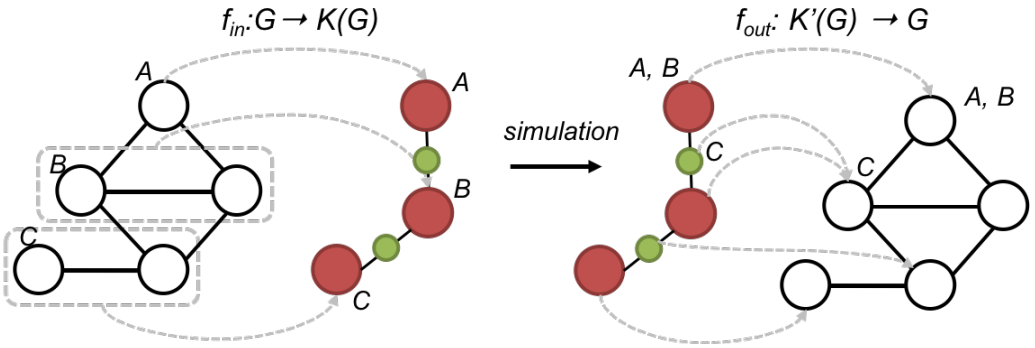

Figure 3.10: Example of an execution of Algorithm 6. Letters A, B and C represent the positions of active robots.

Lemma 3.3.5. *Every execution of Algorithm 6 satisfies the* Termination *property of the $\mathcal{K}$-Gathering problem.*

Proof. We know that every invocation of Algorithm 4 always terminates. On the other hand, we know that vertex $x_i \in V(G)$ because both w_i and $f_{out}(u_i)$ are vertices of G. Hence, if p_i is not in x_i then it moves to one vertex closest to it, and due to G is connected, at some moment p_i will reach x_i. Therefore, the loop in line 15 always terminates and algorithm 6 satisfies the Termination property. □

Lemma 3.3.6. *Every execution of algorithm 6 satisfies the* Validity *property of the $\mathcal{K}$-Gathering problem.*

Proof. Consider any execution α of Algorithm 6 in which robots start on vertices in the same complete graph. Let I be the set with the initial input of all active robots in α. As I forms a complete graph, then there exists at least one maximal complete subgraph C where $I \subseteq C$. Note that C must be a vertex in $K(G)$.

Now, as every vertex $v \in I$ belongs to C, f_{in} can map v to C, and by lemma 3.3.4, v also could be mapped to at most one adjacent vertex K of C. Thus, the minimal subtree T generated by $f_{in}(I)$ must be a star with center C. Observe that C contains the initial positions of all the robots and if v is mapped to K instead of C, then K contains v that is the initial position of some robot because $v \in I$. Also, note that the only way v belong to K is that v lives in the intersection between C and K.

On the other hand, when T is subdivided in $K'(G)$, namely T', one can see that for every vertex of T' there is at least an input of a robot that belongs to that vertex (which is a subgraph of G), because the vertices in the subdivision correspond exactly to the vertices that live in the intersection of C and other maximal complete subgraph. Observe, that T' correspond to the subgraph generated by the initial positions of robots in $K'(G)$, hence, by the property of Algorithm 4 explained in lemma 3.2.2, every decided vertex in the simulation must belong to T'.

Therefore, each robot must decides the input of a robot, because as we explained previously, every vertex in T' contains at least one initial position of a robot. Thus, the decided vertices is a subset of I that is a complete subgraph. Hence, the algorithm satisfy the VALIDITY property. □

Lemma 3.3.7. *Algorithm 6 satisfies the $\mathcal{K}-$AGREEMENT property of the $\mathcal{K}$-Gathering problem.*

Proof. It is enough to observe that after the simulation of Algorithm 4 each robot decides a vertex of the of maximal complete subgraph u_i that it obtains after the simulation, through the line 10 or the f_{out} function.

Now, by the EDGE-AGREEMENT property of Algorithm 4, every robot decides either the same vertex K of $K'(G)$ (which is a complete subgraph of G), or decide an edge (K, K') of $K'(G)$ (with K' being a subgraph of the complete subgraph K). In both cases, robots decide a subgraph of K, which is complete (See figure 3.11).

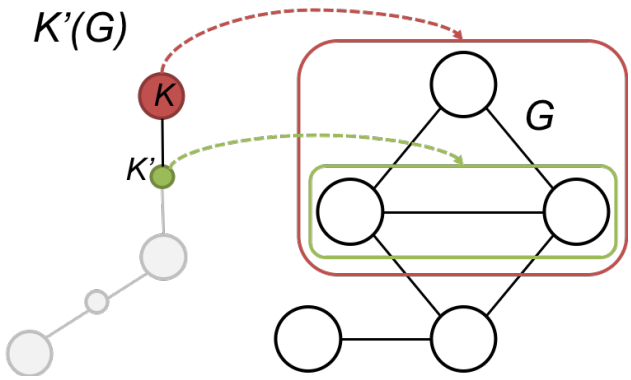

Figure 3.11: By the EDGE-AGREEMENT property of Algorithm 4 robots must decide the same vertex or the vertices spanning an edge in $K'(G)$. In both cases, robots decide a subgraph of a complete subgraph.

□

Theorem 3.3.2. *Let G be a graph such that $K(G)$ is a tree. Then, the $\mathcal{K}$-Gathering problem is solvable in the ALR model for $N \geq 2$ oblivious and anonymous robots on G in at most $2 * diam(K(G)) + diam(G)$ rounds, using at most $4 * diam(K(G)) * |V(G)| * |V(K(G))||$ light colors and tolerating $N-1$ crash failures.*

Proof. The proof follows directly from lemmas [3.3.5, 3.3.6, 3.3.7] □

Chapter 4

Impossibility Results

In this section, we present our impossibility results. We first define a general notion of a robot task that includes many variants of Gathering type problems. We show that as far as robot task solvability, the EALR model is equivalent to WFSM, connecting with this result two areas that have been studied independently for long time.

4.1 Robot Tasks

Definition 4.1.1. Given a graph G, we say that σ is a ***input simplex*** if it represents the set of vertices of G where robots can start their execution. Similarly, σ is called an ***output simplex*** if σ is the set of vertices where the robots are allowed to end.

Definition 4.1.2. The set $\mathcal{I}$ that contains all the possible input simplexes is called ***input complex***. Similarly, the ***output complex*** $\mathcal{O}$ is the set that consist of all output simplexes. We assume that both $\mathcal{I}$ and $\mathcal{O}$ are closed under containment, which is why they are called complexes: if σ is an input /output simplex, then for any simplex $\sigma' \subseteq \sigma$ it holds that $\sigma' \in \mathcal{I}/\mathcal{O}$. Namely, if robots may start (or end) on a set of vertices, they may start (end) in any subset of them.

Definition 4.1.3. A ***robot task*** on G is a triple $\langle \mathcal{I}, \mathcal{O}, \Delta \rangle$ where $\mathcal{I}$ is an input complex, $\mathcal{O}$ is an output complex and $\Delta : \mathcal{I} \to \mathcal{O}$ is the function that abstracts the constraints of the problem, with the property that if $\sigma' \subseteq \sigma$ then $\Delta(\sigma') \subseteq \Delta(\sigma)$. In other words, Δ is a monotonic function.

Robot tasks can be considered as *colorless* [1] tasks, because they simply specify input/output relations and not specific input/output assignments to robots. As an example, we can define the Gathering and Edge-Gathering problems as a robot tasks specifications, observe that the definition of the output complex $\mathcal{O}$ captures the AGREEMENT property of each problem:

Definition 4.1.4. Given a graph $G = (V, E)$, the ***Gathering*** problem as a robot task specification is defined by the triple $\langle \mathcal{I}, \mathcal{O}, \Delta \rangle$ where:

- $\mathcal{I} = \mathcal{P}(V)$ is the power set of V, i.e. $\mathcal{I}$ is the set that contains all the possible subsets of vertices in V.
- $\mathcal{O} = \{ \{ v \} : v \in V \}$. The output simplexes in $\mathcal{O}$ are all the sets consisting of a single vertex of V.
- For each $\sigma \in \mathcal{I}$: $\Delta(\sigma) = \tau \in \mathcal{O}$

Definition 4.1.5. Given a graph $G = (V, E)$, the ***Edge-Gathering*** problem as a robot task specification is defined by the triple $\langle \mathcal{I}, \mathcal{O}, \Delta \rangle$ where:

- $\mathcal{I} = \mathcal{P}(V)$ is the power set of V, i.e. $\mathcal{I}$ is the set that contains all the possible subsets of vertices of V.
- $\mathcal{O} = \{ \{ v \} : v \in V \} \cup \{ \{ u, v \} : (u, v) \in E \}$. In other words, the output complex $\mathcal{O}$ includes in addition to all the singletons, all the sets of two vertices that belong to the same edge on G.
- For each $\sigma \in \mathcal{I}$:

$$\Delta(\sigma) = \begin{cases} \sigma & \text{if } |\sigma| = 1 \\ \tau \in \{ \sigma, \{ v_1 \}, \{ v_2 \} \} & \text{if } \sigma = \{ v_1, v_2 \} \text{ with } (v_1, v_2) \in E \\ \tau \in \mathcal{O} & \text{otherwise} \end{cases}$$

There are many other robot Gathering tasks that one could consider, notably $\mathcal{K}$-Gathering. For example, one may define tasks where robots should end up at distance d or maybe decide a set of vertices forming an independent set. But not all the problems are robot Gathering tasks, for instance, requiring that no two robots end up in the same vertex or requiring that robots end in vertices inducing a connected subgraph of G cannot be specified as a robot task (In both cases the output complex is not closed under by containment).

4.2 Equivalence between EALR and WFSM

Before proving the equivalence, we will describe the concept of robot task solubility on each model.

Definition 4.2.1. An algorithm $\mathcal{A}$ ***solves*** a robot task $\langle \mathcal{I}, \mathcal{O}, \Delta \rangle$ in the EALR model, if for any $\sigma \in \mathcal{I}$ and for any execution where the correct robots start with entries in σ, the following conditions are satisfy:

- TERMINATION: Every correct robot decides a value in a bounded number of LCM cycles.
- VALIDITY: The set of all the decided vertices forms a simplex $\tau \in \Delta(\mathcal{I})$.

Definition 4.2.2. A distributed algorithm in the WFSM model ***solves*** a robot task on G, $\langle \mathcal{I}, \mathcal{O}, \Delta \rangle$, if the following holds:

- Each process q_i starts with a vertex v_i of G as input, such that the set of the inputs forms a simplex σ with $\sigma \in \mathcal{I}$.
- In any execution, each correct process decides a value v_i, such that the set of the decided values forms a simplex τ, with $\tau \in \Delta(\sigma)$.

Notice that according to the previous definition, processes actually do not "move" in any way on G; they simply communicate with each other as many times (and as much information) as they want through the shared memory M, and then they decide vertices of G.

The following theorem proves that EALR and WFSM models are equivalent, that means that any algorithm that solves a robot task in the former can be used to solve the same problem on the latter, and viceversa. The result is based on the lemmas presented in the next subsections but we have decided to put it here for readability purposes.

Theorem 4.2.1. *A robot task $\mathcal{T}$ on G is solvable in the WFSM model with N luminous robots and tolerating f failures* ***if and only if*** *$\mathcal{T}$ is solvable in the EALR model with N robots tolerating up to f failures.*

Proof. The proof follows directly from lemmas[4.2.1, 4.2.2]. □

4.2.1 From EALR to WFSM

Lemma 4.2.1. *If a robot task $\mathcal{T}$ on G is solvable in the EALR model with N luminous robots tolerating f failures, then $\mathcal{T}$ is solvable in the WFSM model with N processes tolerating up to f failures.*

Proof. Consider an algorithm $\mathcal{A}$ that solves the robot task $\mathcal{T}$ in the EALR model with N luminous robots, tolerating f failures. We will prove that $\mathcal{T}$ is solvable in the WFSM model with N processes, tolerating up to f failures by simulating algorithm $\mathcal{A}$. To do that, each process q_i simulates the code of robot p_i, so, we need to specify how processes are able to simulate the atomic MOVE and LOOK operations in EALR, as WRITE and SNAPSHOT operations on WFSM. Note that the COMPUTE operation is a local deterministic algorithm that can be used in both models.

At the beginning of the execution, there is an array M of shared memory registers, where each entry $M[i]$ contains the current state of robot p_i. Initially, every register contains the value $\perp$, which symbolizes the fact that p_i has not started its local computation. Now, to simulate a $\text{MOVE}(v_i, r_i)$ operation, process q_i simply executes $\text{WRITE}_i((v_i, m_i))$, namely, it atomically writes the pair (v_i, m_i) into $M[i]$. On the other hand, to simulate a $\text{LOOK}(G)$ operation, process q_i takes an atomic snapshot of M, i.e., it executes $\text{SNAPSHOT}(M)$ and returns the set of pairs (v_j, r_j) that it detected in the array M.

Thus, to solve $\mathcal{T}$, each process q_i simply simulates step by step the code of algorithm $\mathcal{A}$ for robot p_i and decides whatever p_i decides in the resulting simulated execution. The simulation solves $\mathcal{T}$, because (1) $\mathcal{A}$ solves $\mathcal{T}$ in the EALR model by assumption, and (2) as already argued, processes in the WFSM model can atomically simulate the MOVE and LOOK operations in the EALR model. □

4.2.2 From WFSM to EALR

We have seen in lemma 4.2.1 that if a robot task $\mathcal{T} = \langle \mathcal{I}, \mathcal{O}, \Delta \rangle$, is solvable in EALR with N luminous robots, then $\mathcal{T}$ is solvable in the WFSM model. In this subsection we are going to prove that the other direction is also true.

Lemma 4.2.2. *If a robot task $\mathcal{T}$ on G is solvable in the WFSM model with N processes tolerating f failures, then $\mathcal{T}$ is solvable in the EALR model with N luminous robots tolerating up to f failures.*

Proof. Consider an algorithm $\mathcal{A}$ that solves $\mathcal{T}$ with N processes tolerating f failures in the WFSM model. We will prove that $\mathcal{T}$ is solvable in the EALR model with N robots tolerating f failures by simulating algorithm $\mathcal{A}$. To do that, each robot p_i simulates the code of process q_i, so, we need to specify how robots are able to simulate the WRITE and SNAPSHOT operations in WFSM as LOOK and MOVE operations on EALR.

To perform the simulation, the light of each robot p_i will save the value of the i-th register of the share memory M, and to keep the integrity of the information, each robot p_i will be assigned to a single vertex v_i, where it will remain throughout all the execution. The graph G consists of only N vertices and it is possible to add edges, however, it is not necessary, since as mentioned previously no robot will move from its designated vertex.

To simulate the WRITE$_i(x)$ operation, robot p_i uses its light to save the value x, so it invokes the MOVE(v_i, x) operation. Now, to replicate an SNAPSHOT(M) operation, it is sufficient to execute a LOOK(G) operation returning the light's value of each active robot, and $\perp$ from those that has not started its execution.

Thus, to solve $\mathcal{T}$, each robot p_i simulates merely step by step the code of algorithm $\mathcal{A}$ for process q_i, and decides whatever q_i decides in the resulting simulated execution. The simulation solves $\mathcal{T}$, because (1) $\mathcal{A}$ solves $\mathcal{T}$ in the WFSM model by assumption, and (2) as already argued, robots in EALR can atomically simulate the WRITE and SNAPSHOT operations defined in the WFSM model. □

4.2.3 The semi-synchronous case

In this subsection, we show that Theorem 4.2.2 proves that the semi-synchronous version of EALR and the immediate snapshot of WFSM are equivalent. Also, Theorem 4.2.3 shows that EALR and its semi-synchronous version are equivalent too.

In the semi-synchronous version of EALR, the execution consists of a sequence of concurrency classes, where in each of them, k robots are activated, where they first execute a MOVE operation (in any arbitrary order), and then execute a LOOK operation (in any arbitrary order too). Notice that the first operation of every robot must be a MOVE operation, because that is the only way that robots can turn on their light and become visible in G.

On the other hand, in the immediate snapshot version of WFSM model, each step consists of a *concurrency class* of k processes, $1 \leq k \leq N$, where processes first concurrently execute their WRITE operations (each one to its own cell of M), and then concurrently execute a SNAPSHOT(M) operation. An execution is an arbitrary sequence of concurrency classes. We stress that the immediate snapshot version of WFSM can solve exactly the same set of tasks as its fully asynchronous version [44].

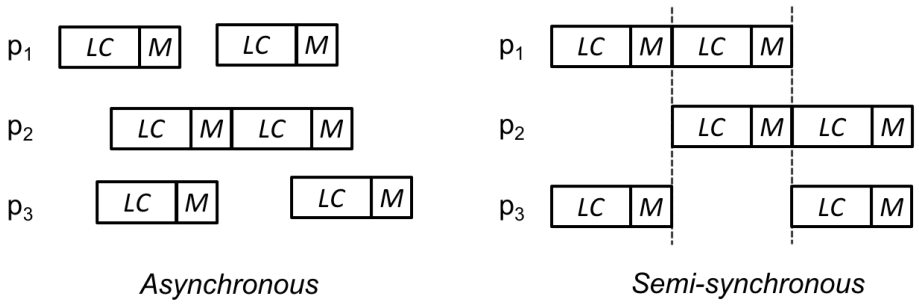

Figure 4.1: In the asynchronous version robots run at its own speed, while in the semi-synchronous version the execution consists of a sequence of concurrency classes, where in each of them k robots are activated.

Theorem 4.2.2. *A* robot task $\mathcal{T}$ *on* G *is solvable in the semi-synchronous EALR model* ***if and only if*** *it is solvable in the immediately snapshot version of WFSM.*

Proof. The proof follows directly from lemmas [4.2.4, 4.2.3] and the fact that the WFSM model and the immediate snapshot of WFSM are equivalent [44]. □

Theorem 4.2.3. *A* robot task $\mathcal{T}$ *on* G *is solvable asynchronously in the EALR model* ***if and only if*** *it is solvable in the semi-synchronous version of EALR.*

Proof. The proof follows directly from lemmas [4.2.1, 4.2.2] and the fact that the WFSM model and its immediate snapshot version are equivalent [44]. □

4.2.3.1 From EALR to immediate snapshot WFSM

Lemma 4.2.3. *If a robot task* $\mathcal{T}$ *on* G *is solvable in the EALR model with* N *luminous robots tolerating* f *failures, then* $\mathcal{T}$ *is solvable in the immediate snapshot WFSM model with* N *processes tolerating* f *failures.*

Proof. The proof is almost the same as lemma 4.2.1. The only difference is that to perform the simulation, we need to consider only a subset of executions of $\mathcal{A}$, in which the scheduler runs the robots in semi-synchronous steps. Since $\mathcal{A}$ solves $\mathcal{T}$, then, particularly $\mathcal{A}$ solves $\mathcal{T}$ in the semi-synchronous set of executions. □

4.2.3.2 From WFSM to semi-synchronous EALR

Lemma 4.2.4. *If a robot task $\mathcal{T}$ on G is solvable in WFSM model with N processes tolerating f failures, then $\mathcal{T}$ is solvable in the semi-synchronous EALR model with N robots tolerating f failures.*

Proof. The simulation is the same as the one in the proof of Lemma 4.2.2, the only difference is that we will consider only a subset of executions of $\mathcal{A}$ where the scheduler runs the processes in semi-synchronous steps. Since $\mathcal{A}$ solves $\mathcal{T}$, then, particularly $\mathcal{A}$ solves $\mathcal{T}$ in the semi-synchronous set of executions. □

4.3 Impossibility of Edge-Gathering

In this section, we are going prove that if one intends to solve the Edge-Gathering problem, robots must use lights, even if at most 2 robots fail. Also, we are going to show that Edge-Gathering is unsolvable in the EALR model on any graph G with a cycle. With this last result and the Algorithm 4 presented in subsection 3.2, we can fully characterize the problem: it is solvable for three or more robots on any graph G if and only if G is a tree.

The idea of both impossibilities results are divided into three parts: first, we prove that the problem is unsolvable for three robots tolerating two failures in the WFSM model. Then, using the solubility result of the *BG simulation* [45], we extend the result for N robots tolerating two failures. Finally, we applied the reduction presented in subsection 4.2.2 that transform any algorithm from WFSM to EALR, getting a contradiction and completing the proof.

Roughly speaking, the BG simulation reduces the solvability of a robot task for $(t+1)$ processes tolerating t failures, to the solvability of the same robot task for $N > t$ processes tolerating t failures. In other words, the

BG simulation result says that: given an algorithm $\mathcal{A}$ for $N > t$ processes tolerating t failures that solves a robot task $\mathcal{T}$ (observe that $\mathcal{A}$ is not necessary wait-free), then it is possible to construct an algorithm $\mathcal{B}$ where $t+1$ processes out of which t might fail can simulate algorithm $\mathcal{A}$ to solve the same task $\mathcal{T}$. Note that algorithm $\mathcal{B}$ is wait-free.

However, the BG simulation is typically used for proving impossibility results (using the contrapositive) by showing first an impossibility result for the wait-free case and the extend it to the case with only t failures.

4.3.1 Impossibility without the use of lights

Definition 4.3.1. In the WFSM model, an Edge-Gathering algorithm $\mathcal{A}$ is ***restricted*** if each process q_i writes a vertex v_i of G (it does not store any other information) in the shared memory every time it executes a WRITE operation. Moreover, whenever q_i writes v_i in the shared memory, v_i depends only on the set generated by the SNAPSHOT(M) operation, i.e., every vertex that q_i writes in its position of the shared memory depends only on the set of vertices q_i "sees" from its latest SNAPSHOT operation.

Lemma 4.3.1. *Let G be a graph with $diam(G) \geq 3$, then there is no restricted algorithm $\mathcal{A}$ that solves Edge-Gathering for $N = 3$ processes tolerating up to 2 failures in the WFSM model, even if processes share the same labeling or orientation of G.*

Proof. By contradiction. Suppose there is such an algorithm $\mathcal{A}$.
Let q_1, q_2, q_3 be the name of the processes and v_1, v_2, v_3 three vertices of G, such that $dist(v_1, v_2) = 2$ and $dist(v_1, v_3) \geq 3$. Note that these vertices exist due to $diam(G) \geq 3$.

Now, consider the execution α of algorithm $\mathcal{A}$, in which first q_1 starts on v_1 and runs solo until it decides. Then, q_3 starts on v_3, executes a single WRITE operation and crashes. Finally, q_2 starts on v_2 and runs until it decides. Let α' be the shortest prefix of α in which q_1 has already decided a value. Note that α' is also a prefix of the execution γ in which q_1 starts on v_1 and runs solo, and all other processes crashed without taking any step. Due to the VALIDITY property of Edge-Gathering, q_1 must decide v_1 in γ, and thus, q_1 must decide v_1 in α. Hence, by the EDGE-AGREEMENT property, q_2 decides a vertex w in α such that $dist(v_1, w) \leq 1$, and since $dist(v_1, v_3) \geq 3$, we have that $dist(w, v_3) \geq 2$.

On the other hand, consider the execution β of $\mathcal{A}$ that is symmetric to α: first q_3 starts on v_3 and runs solo until it decides, then q_1 starts on v_1, executes a single WRITE operation and crashes. Finally, q_2 starts on v_2 and runs until it decides. Observe that v_3 must decide v_3 in β by the same reason of the previous scenario. Following, we argue that the decision of q_2 in β is at distance more than 1.

Let α'', β'' the prefixes of α and β respectively, in which q_2 is about to take its first SNAPSHOT operation. Observe that due to the robots do not have lights, the only information that q_2 detect from the environment at the end of both α'' and β'' is the set $\{ v_1, v_3 \}$, forming by the vertices of q_1 and q_3. Namely, q_2 cannot distinguish between executions α and β. Hence, q_2 writes in its register the same vertex in both scenarios in its next WRITE operation (since $\mathcal{A}$ is restricted); and again, q_3 sees the same set of vertices in both extensions of α'' and β'', and so on.

Therefore, the decision of q_2 in β is the same as its decision in α, that is the vertex w such that $dist(w, v_3) \geq 2$. But this contradicts the EDGE-AGREEMENT property of Edge-Gathering, because in the execution β, q_2 and q_3 are correct robots and there must happen that $dist(w, v_3) \leq 1$. □

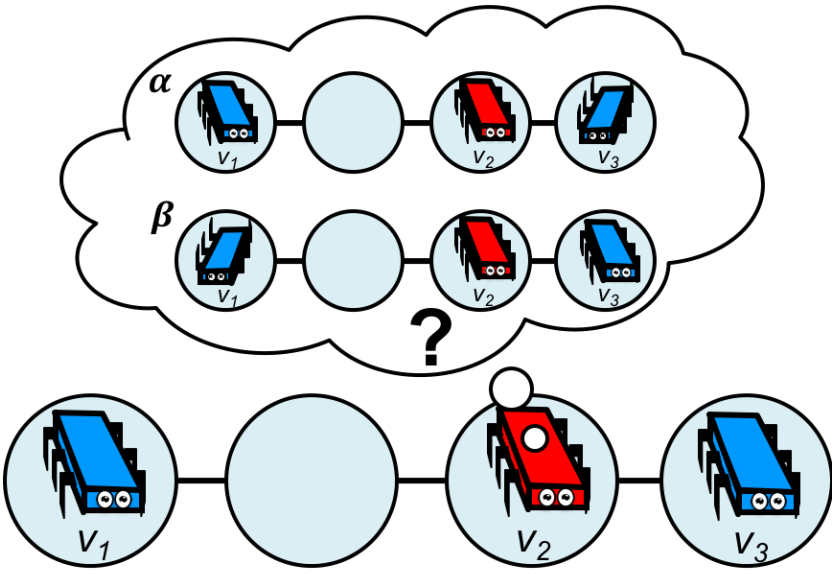

Figure 4.2: Due to robots do not have lights, then the red robot is not able to discern between α and β executions. Therefore, in one of them the final position of the red robot will not satisfy the VALIDITY property of Edge-Gathering.

Theorem 4.3.1. *Let G be a graph with $diam(G) \geq 3$, then, there is no restricted algorithm that solves the Edge-Gathering problem for $N \geq 3$ processes tolerating 2 failures in the WFSM model, even if processes share the same labeling or orientation of G.*

Proof. Suppose for the sake of contradiction, that there is a restricted algorithm $\mathcal{A}$ that solves the Edge-Gathering problem for $N \geq 3$ processes tolerating 2 failures.

Hence, using the BG simulation, it is possible to use $\mathcal{A}$ to construct an algorithm $\mathcal{B}$ for 3 processes tolerating 2 failures. Algorithm $\mathcal{B}$ is wait-free, restricted and solves Edge-Gathering, because:

1. $\mathcal{A}$ solves Edge-Gathering and tolerates two failures, hence $\mathcal{B}$ solves Edge-Gathering and tolerates two failures.
2. $\mathcal{A}$ is restricted, hence, in the simulation each simulated WRITE operation of $\mathcal{A}$ writes a vertex of G (though processes use an additional shared memory to implement the commit-adopt function, however this is part of the simulation)

But by Lemma 4.3.1, $\mathcal{B}$ does not exist. Contradiction! □

Theorem 4.3.2. *Let G be a graph with $diam(G) \geq 3$. Then, there is no algorithm $\mathcal{A}$ that solves the Edge-Gathering problem for $N \geq 3$ robots without lights in the* STRONG [1] *version of the EALR model, with at most two robots failing.*

Proof. By contradiction. Suppose there is such an algorithm $\mathcal{A}$. By lemma 4.2.2, the existence of $\mathcal{A}$ implies an Edge-Gathering algorithm $\mathcal{B}$ on G for $N \geq 3$ processes tolerating two failures in the WFSM model.

Moreover, the proof of lemma 4.2.2 shows that in the simulation of $\mathcal{A}$ in the WFSM model, every time a process writes into the shared memory, it writes only a vertex of G: since robots in $\mathcal{A}$ do not have lights then the information that is written in the shared memory in the simulation is only the positions of robots, i.e., vertices of G. Thus, $\mathcal{B}$ is restricted. Finally, by lemma 4.3.1, $\mathcal{B}$ does not exist. Hence, $\mathcal{A}$ cannot exist. □

The same result can be extended to $\mathcal{K}$-Gathering over graphs with a diameter at least three and no triangles since in this case Edge-Gathering and $\mathcal{K}$-Gathering are equivalent.

[1] In the STRONG version of the EARL, we assume that robots are non-oblivious, non-anonymous, non-disoriented, share the same labeling of G and can detect multiplicities

Corollary 4.3.1. Let G be a connected graph with $diam(G) \geq 3$ and no triangles. Then, there is no algorithm $\mathcal{A}$ that solves $\mathcal{K}$-Gathering problem on G for $N \geq 3$ robots without lights in the STRONG version of the EALR model, with at most two robots failing.

4.3.2 Impossibility on cyclic graphs

So far, we have seen that (1) it is impossible that robots gather on a single vertex, and (2) for three or more robots the Edge-Gathering problem requires the use of lights. Now, we show that even using an unbounded number of lights, the existence of a single cycle in G is an obstruction for solving Edge-Gathering.

Definition 4.3.2. In the k-Set Agreement problem [46], each process starts with a private input value v, and after communicating with other processes, the following properties are satisfied:

- TERMINATION: Every correct process decides a value.
- VALIDITY: Every correct process decides a value that it is proposed by at least one process.
- K-SET-AGREEMENT: At most k different values are decided.

Lemma 4.3.2. *Let G be a connected graph with a cycle $\mathcal{C}$. There is no algorithm $\mathcal{A}$ that solves Edge-Gathering on G for $N = 3$ processes tolerating two failures in the WFSM model.*

Proof. By contradiction. Suppose there is such an algorithm $\mathcal{A}$. Let $\mathcal{C}$ be a simple cycle of G and v_1, v_2, v_3 three distinct and consecutive vertices of $\mathcal{C}$. Consider P as the simple path obtained by removing v_3 from $\mathcal{C}$.

It is straightforward to prove that two robots can solve Edge-Gathering in the EALR model on any path P in at most $length(P)$ rounds without using lights. Hence, by Lemma 4.2.1 there is an algorithm $\mathcal{B}$ that solves Edge-Gathering on P for processes q_1 and q_2 in the WFSM model. We use algorithm $\mathcal{A}$ and $\mathcal{B}$ to solve the 2-Set Agreement problem for three processes tolerating two failures in the WFSM model that it is well-known to be impossible by [47]

We claim that Algorithm 7 solves the 2-Set Agreement problem for three processes q_1, q_2, q_3 using algorithms $\mathcal{A}$ and $\mathcal{B}$. The idea of the algorithm is the following: first, the vertices of G are mapped to indexes of the shared

memory of processes: v_3 is mapped to index 3, v_2 is mapped to 2, and the remaining vertices are mapped to 1, as illustrated in figure 4.3. Then, as a first action, every process proposes its value by writing it in its register of the shared memory. Subsequently, only processes q_1 and q_2 simulate algorithm $\mathcal{B}$ on the path P, taking v_1 and v_2 as its input vertices. At the end of that invocation two important properties are satisfied. The first one is that q_1 and q_2 must decide vertices at a distance at most one on the path P; and the second is that if q_1 or q_2 decides a vertex other than v_1 and v_2, it is because both processes were activated. Finally, the three processes execute algorithm $\mathcal{A}$ to gather in a vertex or in an edge of G. And hence, at most to distinct indexes (and then, at most two different values) are decided.

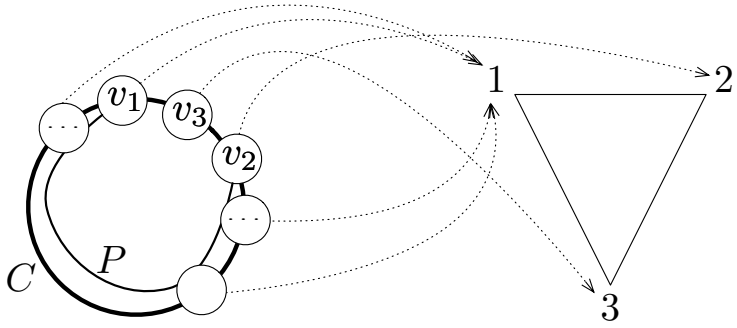

Figure 4.3: Mapping vertices of the cycle C to indexes of the shared memory for the 2-Set Agreement problem.

Now, we need to show that Algorithm 7 is correct. The TERMINATION and 2-SET-AGREEMENT property are straightforward to prove. What is more complicated to achieve is the VALIDITY property: only proposed values from participating processes can be decided; and that is the aim of algorithm $\mathcal{B}$. Actually, the most complicated case is when p_1 and p_2 participate; but by the way in we arranged the initial positions, when algorithm $\mathcal{A}$ executes, algorithm $\mathcal{B}$ will prevent that q_1 and q_2 move in the case when q_3 fails before starting the algorithm; and thus, preventing them from decide vertex v_3 that corresponds to a value that has not been proposed.

- TERMINATION: By assumption, $\mathcal{A}$ and $\mathcal{B}$ terminate in all invocations, and there are not loops in Algorithm 7. Thus, in every execution all the correct processes return a value.
- 2-SET-AGREEMENT: If at most two processes return a value, then the property is trivially satisfied. So, the only interesting case is when the three processes return a value. But, since $\mathcal{A}$ solves the Edge-Gathering problem on G, then the three robots end in the same vertex or on the

Algorithm 7 2-Set Agreement algorithm for $N = 3$ processes tolerating 2 failures. Code for process q_i.

Function 2SetAgreement(v_i)

1: // process q_i proposes its value in the shared memory
2: $M[i] \leftarrow v_i$
3: // The initial vertex of q_3 in the algorithm $\mathcal{A}$ is always v_3
4: **if** $i = 3$ **then**
5: $u \leftarrow v_3$
6: **else**
7: /* The initial vertex of q_1 and q_2 in the algorithm $\mathcal{A}$ is the result of the simulated Edge-Gathering algorithm over the path P */
8: $u \leftarrow \mathcal{B}$.**EdgeGathering2Robots**(v_i, P)
9: **end if**
10: // All the robots simulate algorithm $\mathcal{A}$ and at most two values are decided
11: $y_j \leftarrow \mathcal{A}$.**EdgeGatheringCycle**$(u, G)$
12: /* If the decided vertex is v_1, v_2 or v_3 then process decides the corresponding proposed value */
13: **if** $y_j \in \{ v_1, v_2, v_3 \}$ **then**
14: **return** $M[j]$
15: **else**
16: /* The only way that some process chooses a value other than v_1, v_2 or v_3 is that at least processes q_1 and q_2 participate in the algorithm. So, we can choose the value associated to any of those processes; in this case, we have decided to choose the value of q_1 */
17: **return** $M[1]$
18: **end if**

same edge, and hence, at most two distinct values are decided.

- VALIDITY: We identify three cases according to the number of processes that participate and decide in the execution.

  Case 1: Only one process q_i participates in the execution.
  If $q_i = q_3$, then q_3 invokes $\mathcal{A}$ with input v_3, and consequently obtains v_3 from it by the VALIDITY property of the Edge-Gathering problem. Thus, q_3 returns $M[3]$ that corresponds to its proposed value.

  If q_i is either q_1 or q_2, then q_i first invokes $\mathcal{B}$ with input v_i, and due to the VALIDITY property of the Edge-Gathering problem, q_i must get the same vertex v_i from $\mathcal{B}$. Next, q_i invokes $\mathcal{A}$ with input v_i and obtains v_i as well, for the same above reason. Therefore, q_i returns its own value $M[i]$.

  Case 2: Two processes q_i and q_j participate in the execution.
  Suppose without loss of generality that $q_i = q_3$ and q_j is either q_1 or q_2. Then, process q_j first invoke algorithm $\mathcal{B}$ with input v_j, it

runs solo (because process q_3 does not executes $\mathcal{B}$) and obtains v_j. Subsequently, q_3 and q_j invoke $\mathcal{A}$ with inputs v_3 and v_j, respectively. And since v_3 and v_j form an edge, then they must obtain from $\mathcal{A}$ only those vertices. Therefore, we concluded that q_3 and q_j could return only the values stored in $M[3]$ and $M[j]$, which both are proposed values.

Now, if q_i and q_j are the processes q_1 and q_2, then when they invoke $\mathcal{B}$ they decide the same vertex or the vertices that cover an edge over the path P, and by the VALIDITY property of Edge-Gathering, q_i and q_j must obtain any subset of those vertices in the invocation of $\mathcal{A}$. Observe that vertex v_3 is not in P, so, q_i and q_j could only return the values stored in $M[1]$ and $M[2]$, which both were proposed.

Case 3: The three processes become active. Which implies that any value that is decided was proposed by someone. Therefore, the VALIDITY property is satisfy.

□

Theorem 4.3.3. *Let G be any cyclic graph. Then, there is no algorithm $\mathcal{A}$ that solves Edge-Gathering on G for $N \geq 3$ processes tolerating up to two failures in the WFSM model.*

Proof. By contradiction. Suppose that there is such an algorithm $\mathcal{A}$. Now, using the BG simulation, it is possible to use $\mathcal{A}$ to construct and algorithm $\mathcal{B}$ that solves the same task for three processes tolerating two failures. But, by lemma 4.3.2 algorithm $\mathcal{B}$ does not exist, so we have a contradiction. □

Theorem 4.3.4. *Let G be a connected graph with a cycle, then there is no algorithm $\mathcal{A}$ that solves the Edge-Gathering problem on G for $N \geq 3$ robots in the* STRONG *ALR model, even if at most two robots might fail.*

Proof. By contradiction. Suppose there is such an algorithm $\mathcal{A}$. By lemma 4.2.2, the existence of such an algorithm $\mathcal{A}$ implies a solution for Edge-Gathering on G for $N \geq 3$ processes tolerating two failures in the WFSM model. However, that algorithm cannot exist by theorem 4.3.3. □

Theorem 4.3.5. *Let G be a connected graph with cycles and no triangles. There is no algorithm that solves $\mathcal{K}$-Gathering on G for $N \geq 3$ robots in the* STRONG *ALR model, even if at most two robots might fail.*

Proof. Theorem 4.3.4 directly implies that the $\mathcal{K}$-Gathering problem is impossible on cyclic graphs with no triangles: if there is such an algorithm,

then in every execution processes decide a vertex or an edge (since the graph has no triangles), which contradicts the theorem. □

4.4 Characterization of solvable Robot Tasks

This section presents a topological characterization of the solvability of robots tasks in the EALR model. It is assumed that the reader has a basic knowledge of the study of distributed computing through combinatorial topology (see [1]).

4.4.1 The characterization

The following combinatorial characterization is expressed in the topological approach to distributed computing (see for example [1]).

Theorem 4.4.1. *A robot task on G, $\langle \mathcal{I}, \mathcal{O}, \Delta \rangle$ is solvable for N robots in the EALR model tolerating $N-1$ failures if and only if there is a subdivision $Subd(\mathcal{I})$ and a simplicial map δ from $Subd(\mathcal{I})$ to $\mathcal{O}$, such that for every input simplex σ, $\delta(Subd(\sigma)) \subseteq \Delta(\sigma)$.*

This follows because (1) the EALR and WFSM models are equivalent and (2) the wait-free computability theorem [47] states that a robot task is solvable in WFSM if and only if there is a subdivision of $\mathcal{I}$ and a simplicial decision map to $\mathcal{O}$ respecting Δ [1]. More in detail, the wait-free computability theorem describes when a general task $\langle \mathcal{I}, \mathcal{O}, \Delta \rangle$ has a wait-free read/write protocol. Such a general task permits the use of processes IDs, both to specify the task and to solve it. Thus, input and output simplexes are defined over vertices colored with process IDs. We reproduce here Figure 11.7 from [1] for the reader's convenience, where the theorem is illustrated.

The theorem states that $(\mathcal{I}, \mathcal{O}, \Delta)$ has a wait-free read/write protocol if and only if there is a chromatic subdivision X of $\mathcal{I}$ and a color-preserving simplicial map $\delta : X(\mathcal{I}) \to \mathcal{O}$ such that for each simplex $s' \in X(s)$, $\delta(s')$ is carried by $\Delta(s)$. In Figure 4.4 we see an input simplex s in $\mathcal{I}$ at the top, and how a distributed algorithm subdivides it into $X(s)$, part of the subdivision $X(\mathcal{I})$. Each vertex of $X(s)$ represents the final state of a process in one of the executions starting in s. In the figure, s' is an example of a set of process'

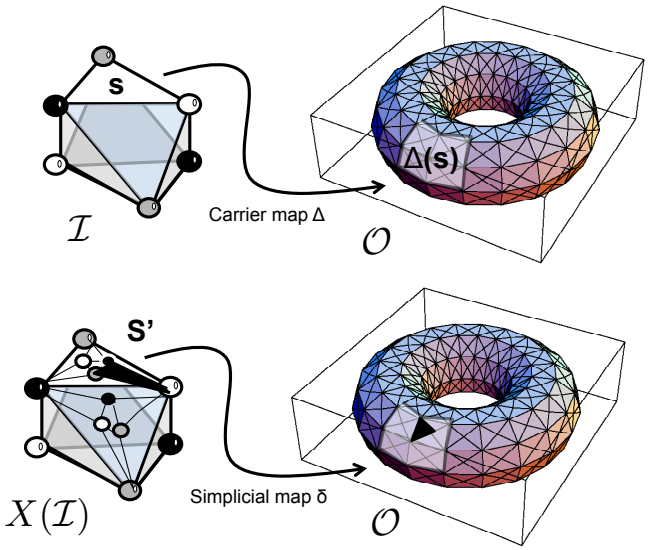

Figure 4.4: The wait-free computability Theorem, Fig 11.7 from [1].

final states in one of the possible executions starting in s. The decision map δ defines the decisions taken by the set of processes in s'. These decisions are $\delta(s')$, thus, for each $s' \in X(s)$, the decisions $\delta(s')$ should belong to $\Delta(s)$, as the rule Δ states the legal outputs when starting in each initial simplex s.

Corollary 4.4.1. No sequential algorithm decides if a given robot task on G for three processes tolerating two failures is solvable in the EALR model.

The proof follows from Theorem 4.2.1, and by reduction to the classic loop contractibility problem in a 2-dimensional complex, which is undecidable, following the techniques in [48, 49]. Notice, however, that if the robots have a bounded number of lights (as in [34]), then their set of possible views returned by Look operations on G is finite, hence:

Corollary 4.4.2. It is decidable if a given robot task on G for three processes with no lights, tolerating two failures is solvable in the EALR model.

Chapter 5

Conclusions

In this work, we studied the hardness of termination in Gathering type problems. We considered a strong termination property in which each robot terminates in a bounded number of its LCM cycles, regardless of delays of other robots.

We showed that the classic Gathering problem with termination is unsolvable by any algorithm in the standard Asynchronous Luminous Robot model (ALR), even if robots have stronger capabilities.

Then, we introduced a novel model called EALR by extending ALR to include crash failures and asynchronous appearing times. In this new model, we studied the solvability of two Gathering type problems with termination: Edge-Gathering and $\mathcal{K}$-Gathering. We proved that Edge-Gathering is possible on a graph G if and only if G is a tree. We also gave solutions for the $\mathcal{K}$-Gathering problem when G has at least one dominating vertex or its related clique graph is a tree.

Our impossibility results were obtained through an equivalence between models: EALR is equally powerful to solve general robot tasks as the standard asynchronous crash-prone wait-free multiprocess shared memory (WFSM) model. This equivalence allowed us to use powerful topological techniques in the ALR and EALR models, deriving a full combinatorial topology characterization of the solvability of general robot tasks in EALR.

There are many interesting research avenues that arise from this work. On the complexity side, two of our algorithms for Edge-Gathering and $\mathcal{K}$-Gathering use a non-constant number of lights. An interesting question is whether the same problems can be solved using a constant number of lights;

hence showing that strong termination for these problems can be achieved at the cost of an extra small communication mechanism. We also proved that it is impossible to solve $\mathcal{K}$-Gathering on any connected graph G with cycles and no triangles, but remains open if it is solvable or not if G has triangles (for example, $\mathcal{K}$-Gathering is solvable by a trivial algorithm if G is a wheel.).

Related to the number of lights, our equivalence between EALR and WFSM uses an unbounded amount of lights. This comes from the fact that the size of registers in WFSM is unbounded. However, it would be interesting to study the effect of restricting the number of lights in EALR and its relation with asynchronous multiprocessing shared memory systems (possibly with bounded size registers). Also, it would be interesting to study other variants of the model. For example, robots moving in a two (or higher) dimensional space, synchronous variants, or variants with Byzantine failures.

Bibliography

[1] Maurice Herlihy, Dmitry Kozlov, and Sergio Rajsbaum. *Distributed Computing Through Combinatorial Topology*. Morgan Kaufmann, San Francisco, CA, USA, 2013.

[2] Maria Potop-Butucaru, Michel Raynal, and Sebastien Tixeuil. Distributed computing with mobile robots: An introductory survey. In *Proceedings of the 2011 14th International Conference on Network-Based Information Systems*, NBIS '11, pages 318–324, Washington, DC, USA, 2011. IEEE Computer Society.

[3] Levent Bayındır. A review of swarm robotics tasks. *Neurocomputing*, 172:292 – 321, 2016.

[4] Ichiro Suzuki and Masafumi Yamashita. Distributed anonymous mobile robots: Formation of geometric patterns. *SIAM J. Comput.*, 28(4):1347–1363, March 1999.

[5] Shantanu Das, Paola Flocchini, Giuseppe Prencipe, Nicola Santoro, and Masafumi Yamashita. Autonomous mobile robots with lights. *Theor. Comput. Sci.*, 609(P1):171–184, January 2016.

[6] Giuseppe Antonio Di Luna, Paola Flocchini, Sruti Gan Chaudhuri, Nicola Santoro, and Giovanni Viglietta. Robots with lights: Overcoming obstructed visibility without colliding. *CoRR*, abs/1405.2430, 2014.

[7] Mark Cieliebak, Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro. Distributed computing by mobile robots: Gathering. *SIAM J. on Computing*, 41(4):829–879, 2012.

[8] Paola Flocchini, Giuseppe Prencipe, Nicola Santoro, and Peter Widmayer. Gathering of asynchronous robots with limited visibility. *Theoretical Computer Science*, 337(1):147 – 168, 2005.

[9] Mark Cieliebak. *Gathering Non-oblivious Mobile Robots*, pages 577–588. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[10] Noa Agmon and David Peleg. Fault-tolerant gathering algorithms for autonomous mobile robots. *SIAM Journal on Computing*, 36(1):56–82, 2006.

[11] Zohir Bouzid, Shantanu Das, and Sébastien Tixeuil. Gathering of mobile robots tolerating multiple crash faults. In *IEEE 33rd International Conference on Distributed Computing Systems (ICDCS)*, pages 337–346, Washington, DC, USA, 2013. IEEE Computer Society.

[12] S. Rajsbaum, A. Castañeda, D. F. Peñaloza, and M. Alcántara. Fault-tolerant robot gathering problems on graphs with arbitrary appearing times. In *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 493–502, May 2017.

[13] David Peleg Asaf Efrima. *Distributed Models and Algorithms for Mobile Robot Systems*, pages 70–87. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[14] P. Flocchini, G. Prencipe, and N. Santoro. *Distributed Computing by Oblivious Mobile Robots*. Synthesis lectures on distributed computing theory. Morgan & Claypool, 2012.

[15] Giuseppe Prencipe. Autonomous mobile robots: A distributed computing perspective. In Paola Flocchini, Jie Gao, Evangelos Kranakis, and Friedhelm Meyer auf der Heide, editors, *ALGOSENSORS*, volume 8243 of *Lecture Notes in Computer Science*, pages 6–21. Springer, 2013.

[16] Giuseppe Prencipe and Nicola Santoro. *Distributed Algorithms for Autonomous Mobile Robots*, pages 47–62. Springer US, Boston, MA, 2006.

[17] Giuseppe Prencipe. Impossibility of gathering by a set of autonomous mobile robots. *Theoretical Computer Science*, 384(2):222 – 231, 2007. Structural Information and Communication Complexity (SIROCCO 2005).

[18] Ralf Klasing, Euripides Markou, and Andrzej Pelc. *Gathering Asynchronous Oblivious Mobile Robots in a Ring*, pages 744–753. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[19] François Bonnet, Maria Potop-Butucaru, and Sebastien Tixeuil. *Asynchronous Gathering in Rings with 4 Robots*, pages 311–324. Springer International Publishing, Cham, 2016.

[20] Gianlorenzo D'Angelo, Gabriele Di Stefano, and Alfredo Navarra. Gathering on rings under the look–compute–move model. *Distributed Computing*, 27(4):255–285, 2014.

[21] Gianlorenzo D'Angelo, Gabriele Di Stefano, Ralf Klasing, and Alfredo Navarra. *Gathering of Robots on Anonymous Grids without Multiplicity Detection*, pages 327–338. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[22] Gianlorenzo D'Angelo, Gabriele Di Stefano, and Alfredo Navarra. *Gathering Asynchronous and Oblivious Robots on Basic Graph Topologies Under the Look-Compute-Move Model*, pages 197–222. Springer New York, New York, NY, 2013.

[23] Paola Flocchini, Nicola Santoro, Giovanni Viglietta, and M. Yamashita. Rendezvous with constant memory. *Theor. Comput. Sci.*, 621:57–72, 2016.

[24] Giovanni Viglietta. Rendezvous of two robots with visible bits. *CoRR*, abs/1211.6039, 2012.

[25] Taisuke Izumi, Samia Souissi, Yoshiaki Katayama, Nobuhiro Inuzuka, Xavier Défago, Koichi Wada, and Masafumi Yamashita. The gathering problem for two oblivious robots with unreliable compasses. *CoRR*, abs/1111.1492, 2011.

[26] Quentin Bramas and Sébastien Tixeuil. *Wait-Free Gathering Without Chirality*, pages 313–327. Springer International Publishing, Cham, 2015.

[27] Armando Castañeda, Sergio Rajsbaum, and Matthieu Roy. Convergence and covering on graphs for wait-free robots. *Journal of the Brazilian Computer Society*, 24(1):1, Jan 2018.

[28] Reuven Cohen and David Peleg. Convergence properties of the gravitational algorithm in asynchronous robot systems. *SIAM Journal on Computing*, 34(6):1516–1528, 2005.

[29] Gianluca De Marco, Luisa Gargano, Evangelos Kranakis, Danny Krizanc, Andrzej Pelc, and Ugo Vaccaro. Asynchronous deterministic rendezvous in graphs. *Theoretical Computer Science*, 355(3):315 – 326, 2006.

[30] Anders Dessmark, Pierre Fraigniaud, Dariusz R. Kowalski, and Andrzej Pelc. Deterministic rendezvous in graphs. *Algorithmica*, 46(1):69–96, Sep 2006.

[31] Sébastien Bouchard, Yoann Dieudonné, and Bertrand Ducourthial. Byzantine gathering in networks. *Distrib. Comput.*, 29(6):435–457, November 2016.

[32] Yoann Dieudonné, Andrzej Pelc, and David Peleg. Gathering despite mischief. *ACM Trans. Algorithms*, 11(1):1:1–1:28, August 2014.

[33] Jérémie Chalopin, Yoann Dieudonné, Arnaud Labourel, and Andrzej Pelc. Fault-tolerant rendezvous in networks. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Automata, Languages, and Programming*, pages 411–422, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[34] Shantanu Das, Paola Flocchini, Giuseppe Prencipe, Nicola Santoro, and Masafumi Yamashita. Autonomous mobile robots with lights. *Theor. Comput. Sci.*, 609(P1):171–184, January 2016.

[35] Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro. *Distributed Computing by Oblivious Mobile Robots*. Synthesis Lectures on Distributed Computing Theory 3(2). Morgan & Claypool, 2012.

[36] Chrysovalandis Agathangelou, Chryssis Georgiou, and Marios Mavronicolas. A distributed algorithm for gathering many fat mobile robots in the plane. In *ACM Symposium on Principles of Distributed Computing, PODC '13, Montreal, QC, Canada, July 22-24, 2013*, pages 250–259, 2013.

[37] Mattia D'Emidio, Daniele Frigioni, and Alfredo Navarra. Synchronous robots vs asynchronous lights-enhanced robots on graphs. *Electron. Notes Theor. Comput. Sci.*, 322(C):169–180, April 2016.

[38] François Bonnet, Maria Potop-Butucaru, and Sebastien Tixeuil. Asynchronous gathering in rings with 4 robots. In *Proc. Ad-hoc, Mobile, and Wireless Networks: 15th International Conference (ADHOC-NOW)*, number 9724 in LNCS, pages 311–324. Springer, 2016.

[39] Hagit Attiya and Jennifer Welch. *Distributed Computing: Fundamentals, Simulations and Advanced Topics*. John Wiley & Sons, 2004.

[40] Michel Raynal. *Safe, Regular, and Atomic Read/Write Registers*, pages 305–328. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[41] Michael J. Fischer, Nancy A. Lynch, and Mike Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, 1985.

[42] Noa Agmon and David Peleg. Fault-tolerant gathering algorithms for autonomous mobile robots. *SIAM Journal on Computing*, 36(1):56–82, 2006.

[43] Kaustav Bose, Ranendu Adhikary, Sruti Gan Chaudhuri, and Buddhadeb Sau. Crash tolerant gathering on grid by asynchronous oblivious robots. *CoRR*, abs/1709.00877, 2017.

[44] Elizabeth Borowsky and Eli Gafni. Generalized FLP impossibility result for t-resilient asynchronous computations. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing (STOC)*, pages 91–100, 1993.

[45] E. Borowsky, E. Gafni, N. Lynch, and S. Rajsbaum. The BG distributed simulation algorithm. *Distrib. Comput.*, 14(3):127–146, October 2001.

[46] Soma Chaudhuri. More choices allow more faults: Set consensus problems in totally asynchronous systems. *Inf. Comput.*, 105(1):132–158, July 1993.

[47] Shavit N. Herlihy, Maurice and. The topological structure of asynchronous computability. *J. ACM*, 46(6):858–923, November 1999.

[48] Eli Gafni and Elias Koutsoupias. Three-processor tasks are undecidable. *SIAM J. Comput.*, 28(3):970–983, 1999.

[49] Maurice Herlihy and Sergio Rajsbaum. The decidability of distributed decision tasks (extended abstract). In *Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing (STOC)*, pages 589–598, 1997.