Università degli Studi di Milano

Dipartimento di informatica "Giovanni Degli Antoni"

Corso di dottorato in informatica
Ph.D. program in computer science

38th cycle
INF/01

Distributed Computing by Mobile Robots: Computational Power and Algorithm Design

Author: Caterina Feletti
ORCID: 0009-0004-1813-8056
Matr.: R13932

Supervisor: Prof. Beatrice Palano
Co-supervisor: Prof. Carlo Mereghetti
Ph.D. Coordinator: Prof. Roberto Sassi

Academic year 2024–2025

Abstract

The *Look-Compute-Move* (LCM) model is a theoretical framework widely used to formalize and study distributed systems of mobile autonomous robots, namely *swarms of robots*. Generally, the LCM model assumes robots are anonymous, indistinguishable, homogeneous, and punctiform entities acting on the Euclidean plane through LCM cycles. Other features may limit robots' power: e.g., mostly, literature considers disoriented robots, i.e., devoid of a global coordinate system, and with limited or absent storage and communication means. Each robot is embedded with the same deterministic algorithm, which is executed at any robot activation within an LCM cycle. Thanks to such an algorithm, the swarm solves a *problem*; typically, research focuses on problems requiring, for instance, the swarm to arrange in specific geometric patterns, or to move to satisfy some conditions.

This thesis encompasses two research topics mainly involved in this field. The first part investigates the *computational power* of different LCM sub-models, which vary in certain robot features (i.e., memory, communication, visibility, and synchronization). Notably, we study whether different sub-models are able to solve the same set of problems or specific problems (e.g., the Fault Detection problem in fault-prone systems). The second part concerns the *design and analysis of algorithms* for solving specific formation problems under selected sub-models. Specifically, we provide two algorithms for the `Uniform Circle Formation` problem, which requires a swarm of n robots—that are luminous (i.e., equipped with constant-size memory and communication means), opaque (i.e., subject to obstructed visibility in the case of collinearity), and asynchronous—to form a regular n-gon. In particular, the first algorithm is $O(\log n)$-time, while the second one is time-optimal. Finally, we constructively prove that a swarm of luminous and sequential robots (i.e., activated one at a time) can solve the `Universal Dancing` problem, which requires the swarm to form sequences of any patterns, starting from any initial configuration.

Contents

Overview

Within the developing field of *distributed computing by mobile entities*, various theoretical models have been proposed in the last two decades to explore algorithmic strategies and the computational capabilities of such systems [FPS19a]. Notably, the *Look-Compute-Move* (LCM) model has been introduced [SY99] and deeply studied to model swarms of mobile entities, called *robots*, that have to arrange in the environment to solve a common task collaboratively. Robots are traditionally assumed to be punctiform entities deployed on the Euclidean plane, each one operating through an infinite sequence of LCM cycles: as soon as activated, a robot takes a snapshot of the environment (*Look*), executes a deterministic algorithm (*Compute*), and travels straight towards the computed destination point (*Move*).

Robots can be activated according to different synchronization modes: in the *fully synchronous* mode (`FSYNCH`), all robots execute their cycle in lockstep; in the *semi-synchronous* mode (`SSYNCH`), robots are synchronously activated in groups; in the *asynchronous* mode (`ASYNCH`), robots are activated without any synchronization assumption. However, the *fairness condition* is always assumed to guarantee that each robot is activated infinitely often, and it allows time to be segmented into consecutive and finite intervals called *epochs*. Each epoch has the minimal duration required to contain the activations of all the robots in the swarm. A more recent and relatively unexplored class of synchronization modes is the *sequential* one. Starting from the generic `SEQ` mode, in which only one robot is activated at a time, two sub-modes can be defined: `PERM`, in which an epoch activates exactly a *permutation* of the swarm, and the well-known *round-robin* (`RROBIN`), where the permutation activated in each epoch is invariant.

With the aim to spot the most limited conditions under which a swarm can operate, the LCM model generally assumes minimal systems: robots are *anonymous* and *indistinguishable* (i.e., with neither internal nor external identifiers), *homogeneous* (i.e., executing the same algorithm), *autonomous* (i.e., without any central control), *disoriented* (i.e., devoid of a global coordinate system), *oblivious* (i.e., without a local persistent memory), and *silent* (i.e., without explicit means to communicate).

Starting from this very limited model, called $\mathcal{OBLOT}$ after its *obliviousness*, research has investigated algorithmic solutions to specific tasks, aka *problems*: `Pattern Formation`, requiring a swarm to form a given geometric pattern, is the most studied problem, together with `Gathering`, requiring robots to meet at the same point. Despite its simplicity, the $\mathcal{OBLOT}$ model is capable of solving several non-trivial problems. However, it also exhibits significant limitations: for instance, `Gathering` is impossible to solve for two robots under the `SSYNCH` mode, without any additional assumption.

To overcome these impossibility results, an enhanced model, called $\mathcal{LUMI}$, has been introduced: here robots are embedded with a *constant-size persistent light* which plays the role of both an internal state and an external communication means. The intermediate models $\mathcal{FSTA}$ and $\mathcal{FCOM}$ have been introduced to study how the sole memory and the sole communication capability, respectively, affect the computational power of robots. In addition to memory and communication, several other features

(e.g., orientation agreement, visibility, faultiness, movement rigidity, etc.), have been introduced, contributing to the development of various LCM (sub-)models.

Research lines and thesis contribution

Two research lines have mainly dominated the study on the LCM framework. The first one aims to study the *computational power* of different models, varying in a chosen combination of features, by comparing the set of problems solvable under each of them. Interestingly, it has been shown that some models are computationally equivalent through the construction of simulators that demonstrate one model can replicate the behavior of another. Otherwise, some witness problems, called *separators*, have been exhibited to prove that a model dominates another, or to prove that two models are computationally incomparable. Indeed, besides finding the computational relations between different models, this research line aims to characterize the class of problems solvable under certain model assumptions, and, conversely, to identify the minimal model assumptions required to solve a given problem. In line with this later objective, the second main research direction focuses on *developing algorithmic strategies* for solving typical problems for such systems. Due to the distributed nature of the swarms and the limited capabilities of the robots, algorithms must be designed to handle fundamental tasks such as leader election, coordination, swarm orientation, agreement, information encoding, collision avoidance, and more.

This thesis, thematically divided into two parts, contributes to both research lines, each explored in its respective section. Before them, a first introductory chapter, Chapter 1, surveys the LCM framework, rigorously presenting all the features by which a model can be defined and have been mainly studied in the literature. It introduces the concept of computability for a swarm of robots, defines the notion of a problem, its solvability, the computational metrics used in algorithm analysis, and explores the computational relationships that can exist between different models. This chapter serves as a toolbox for the subsequent ones.

Part I - Computational power

Part I collects three chapters that explore the *computational power* of different LCM models. Chapter 2 conducts a cross-model comparison among 12 transparent models mainly considered in literature (combining the four base models $\mathcal{OBLOT}$, $\mathcal{FSTA}$, $\mathcal{FCOM}$, and $\mathcal{LUMI}$ with the three main synchronization modes `FSYNCH`, `SSYNCH`, and `ASYNCH`) and the same 12 models in their *opaque* version, i.e., featuring robots that cannot see through collinear robots. Indeed, this chapter compares and contrasts how two different visibility settings (i.e., complete visibility vs. obstructed visibility) impact the power of robots with the change of memory/communication capabilities.

Chapter 3 focuses on the three sequential modes (`SEQ`, `PERM`, `RROBIN`), analyzing how the computational power of each base model ($\mathcal{OBLOT}$, $\mathcal{FSTA}$, $\mathcal{FCOM}$, and $\mathcal{LUMI}$) is affected by changes in the pattern of the activation sequence.

Chapter 4, rather than comparing the *general* capability for a set of models to solve problems, focuses on two *specific* problems, novel within the LCM framework. Notably, it assumes crash-prone robots, and studies how various LCM models ($\mathcal{OBLOT}$, $\mathcal{FSTA}$, $\mathcal{FCOM}$, and $\mathcal{LUMI}$, combined with `FSYNCH`, `RROBIN`, and `SSYNCH`) can solve the Fault Detection problem (i.e., detect the presence of a crashed robot) and the Fault Identification problem (i.e., identify all the crashed robots).

Part II - Algorithm design

Part II includes three chapters that focuses on the *algorithm design* for solving two problems: `Uniform Circle Formation` (`UCF`) and `Universal Dancing`.

`UCF` requires a swarm of n robots to arrange on the vertices of a regular n-gon. This problem, well-known for its importance and relevance to real-world applications, has been widely studied in literature, especially for transparent robots. Chapter 5 proposes an algorithm solving `UCF` assuming *opaque* robots, without any orientation agreement, avoiding collisions, and under the `ASYNCH` mode. However, to compensate for these constrained assumptions, robots are considered in the $\mathcal{LUMI}$ model. The proposed algorithm solves `UCF` in $O(\log n)$ epochs.

Chapter 6 proposes an improved solution for `UCF` under the same model: despite the model restrictions and the nature of the problem, it designs and exploits ad hoc algorithmic strategies to solve `UCF` in $O(1)$ epochs, that is, in constant time regardless of swarm size n.

Lastly, Chapter 7 focuses on the `Dancing` problem, which requires a swarm to form a sequence of patterns, that is, to perform a *choreography*. Previous studies have shown that the solvability of this problem depends on various factors and conditions, both in the choreography and in the swarm's initial configuration. This chapter shows that, under $\mathcal{LUMI}$ and the `SEQ` synchronization mode, a swarm is able to perform a finite or periodic sequence of *any pattern* (trivially assuming to have a sufficient number of robots), starting from *any initial configuration*, thus solving what we call the `Universal Dancing` problem.

Publications

We list the publications that constitute the source of the chapters:

- Chapter 2 is based on the journal paper [Fel+25c], which is in turn an extended version of the conference paper [Fel+24].
  - [Fel+25c] Caterina Feletti, Lucia Mambretti, Carlo Mereghetti, and Beatrice Palano. "Computational power of autonomous robots: Transparency vs. opaqueness". In: *Theorerical Computer Science* 1036 (2025), p. 115153.
  - [Fel+24] Caterina Feletti, Lucia Mambretti, Carlo Mereghetti, and Beatrice Palano. "Computational Power of Opaque Robots". In: *Proc. of 3rd Symposium on Algorithmic Foundations of Dynamic Networks (SAND)*. vol. 292. LIPIcs. 2024, 13:1–13:19.

  Some of the findings reported in these works have been developed in the context of the master's thesis [Mam23].
- Chapter 3 is based on and extends the conference paper [FFS25][1].
  - [FFS25] Caterina Feletti, Paola Flocchini, and Nicola Santoro. "On the Computational Power of Mobile Robots under Sequential Schedulers". In: *Proc. of 27th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*. vol. 16350. LNCS. Springer Nature Switzerland, 2025.
- Chapter 4 is based from the conference paper [CF25].

[1] Reproduced with permission from Springer Nature.

  - [CF25] Stefano Clemente and Caterina Feletti. "Fault Detection and Identification by Autonomous Mobile Robots". In: *Proc. of 4th Symposium on Algorithmic Foundations of Dynamic Networks (SAND)*. vol. 330. LIPIcs. 2025, 10:1—10:20.

  Some of the findings reported in this work have been developed in the context of the master's thesis [Cle24].

- Chapter 5 is based from the conference paper [FMP23a].

  - [FMP23a] Caterina Feletti, Carlo Mereghetti, and Beatrice Palano. "$O(\log n)$-Time Uniform Circle Formation for Asynchronous Opaque Luminous Robots". In: *Proc. of 27th International Conference on Principles of Distributed Systems (OPODIS)*. vol. 286. LIPIcs. 2023, 5:1–5:21.

- Chapter 6 is based on the conference paper [FPS24a], which is in turn an extracted version from the manuscript [FPS24b].

  - [FPS24a] Caterina Feletti, Debasish Pattanayak, and Gokarna Sharma. "Brief Announcement: Optimal Uniform Circle Formation by Asynchronous Luminous Robots". In: *Proc. of 38th International Symposium on Distributed Computing (DISC)*. vol. 319. LIPIcs. 2024, 46:1–46:7.
  - [FPS24b] Caterina Feletti, Debasish Pattanayak, and Gokarna Sharma. *Optimal Uniform Circle Formation by Asynchronous Luminous Robots*. 2024. arXiv: 2405.06617 [cs.DC].

- Chapter 7 is based on the conference paper [Fel+25a], which is in turn an extracted version from the manuscript [Fel+25b].

  - [Fel+25a] Caterina Feletti, Paola Flocchini, Debasish Pattanayak, Giuseppe Prencipe, and Nicola Santoro. "Brief Announcement: Luminous Robots under Sequential Schedulers: Universal Dancing". In: *Proc. of 39th International Symposium on Distributed Computing (DISC)*. vol. 356. LIPIcs. 2025, 56:1–56:7.
  - [Fel+25b] Caterina Feletti, Paola Flocchini, Debasish Pattanayak, Giuseppe Prencipe, and Nicola Santoro. *Universal Dancing by Luminous Robots under Sequential Schedulers*. 2025. arXiv: 2508.15484 [cs.DC].

Some preliminary parts of the mentioned papers have been integrated and extended in Chapter 1.

Chapter 1

Distributed Computing by Mobile Robots

1.1 Introduction

1.1.1 Background and motivation

Starting from the mid-1980s, computer science witnessed a major paradigm shift from standalone systems to *distributed* ones [TS07]. Until then, systems were mainly composed of a unique monolithic machine equipped with all the resources (data, memory, processors) needed to compute the given tasks. The subsequent evolution in the field of communication engineering has indeed paved the way for no longer considering a unique machine, but for evaluating a network of them, which can distribute the resources and the workload: this has led to the emergence of data grids, clusters of machines, and the Internet.

Generally, the distributed approach brings some benefits to a system: these include improved scalability, enhanced performance, increased reliability, and fault tolerance. Yet, these valuable advantages come with a price. The system is responsible for making the machines collaborate (sometimes by electing leaders), synchronizing their clocks and actions, reconfiguring after new arrivals, faults, or removals, and disseminating information. Thus, in parallel with the engineering study, this new technology approach has required an ad hoc theoretical investigation about rigorous models, distributed algorithms, and their complexity; this investigation has established (and continues to establish) the *distributed computing theory* [San07].

More recently, the distributed approach has also been adopted for systems where machines (here called *entities*) are mobile, making the system topology dynamic. This is the case, for example, of mobile software agents that can migrate across the nodes of a network, systems of drones or mobile sensors, and the emergent field of *swarm robotics*. As a subfield—and evolution—of robotics, swarm robotics aims to shift from monolithic solutions to distributed ones, where a team of simple robots collaborates to achieve a common task. Swarm robotics has been the subject of great interest and numerous studies in recent years; this is due to the advantages that such a technology brings (a distributed system of tiny, simple, and relatively cheap computational units) and, consequently, to the variety of applications that include patrolling, environmental exploration and data collection, search and rescue missions, chemical application in agriculture. Indeed, advances in robotics and nanotechnology in the last two decades have fueled research for these applications.

Even in this case, the engineering research has been accompanied by a proper theoretical study, which has been necessary to address the specific issues for mobile systems: e.g., motion synchronization, collision avoidance, orientation and positioning in the

environment, and system reconfiguration after movements. Hence, the establishment of the research area about *distributed computing by mobile entities* [FPS19a].

1.1.2 Theoretical approach

The theoretical study in the field of distributed computing by mobile entities [FPS19a] aims to formalize distributed systems of mobile entities through proper models, and design distributed algorithms to make systems solve a common task. The typical tasks—formally called *problems*—investigated by the scientific research require the entities to coordinate in order to arrange in a particular position or pattern, gather at the same point, separate or scatter, explore the environment, search, or coat an object. Indeed, the traditional problems for distributed systems, e.g., communication, synchronization, election, and consensus, are to be considered as well.

A vast and established area in this field, with a long history of research, has modeled distributed systems through sets of entities—customarily called *agents*—deployed on networks (formally, graphs). Agents are located on the nodes of the network and can move along the linking edges. Various operating/interaction methods have been theorized to model different scenarios and agents' behaviors: e.g., *face-to-face* communication (where two agents have to meet on the same node to share some information), with the use of *whiteboards* (shared memory area on each node, which can be used to read and write messages), or with the use of tokens (aka *pebbles*) left by agents on the nodes as implicit messages for the other agents (we refer the reader to [DS19] for a deep survey). Notably, the study of these models has had a valuable impact in other fields, such as AI and software engineering.

A more recent research area in distributed computing focuses on mobile entities operating under the *Look-Compute-Move* (LCM) model—which is the subject of this thesis. Since this model draws its motivation and inspiration from the application field of swarm robotics, the computational entities are commonly referred to as *robots* and the system is called *swarm* [FPS19b]. To design algorithms as robust as possible, the literature considers swarms of very limited robots: namely, robots are assumed to be *autonomous* (i.e., without any central control), *anonymous* and *indistinguishable* (i.e., without any internal or external ids), and *homogeneous* (i.e., with the same features and executing the same algorithm). Each robot operates through an infinite sequence of *Look-Compute-Move* cycles; thus, as soon as activated, it *looks* at its surroundings, it *computes* the next position, and it *moves* to the computed position.

The above features represent the core assumptions of the LCM model. In the last two decades, the research has theorized different LCM sub-models, varying in robots' capabilities (e.g., type of interaction, memory, agreement of a global coordinate system, etc.), synchronization policies, the environment where they act, and possibly adversarial factors (e.g, faults, impediments in movements). Notably, the research objective is twofold: *(i)* compare the behavior of swarms (i.e., their ability to solve problems) under different sub-models, and *(ii)* find the most limited sub-model under which a given problem can be solvable.

Typically, robots are assumed to be punctiform entities operating in *continuous spaces*; except for a few works considering the Euclidean space $\mathbb{R}^3$ [BCM18; Yam+17], the state-of-the-art mostly considers robots deployed in the *Euclidean plane* $\mathbb{R}^2$. However, there is no lack of work considering discrete spaces, thus robots operating within a network. This network can be finite (usually represented as a generic graph [DFN15; D'E+18], a tree [D'A+16], a ring [Das+19; Dob+03; MMJ20; Oos+24], a hypercube [Bos+18; Cic+25], etc.), or infinite (e.g., a grid [BDL20; D'A+16; KT24; MC22; Shi+21]).

Regarding memory/communication capabilities, the basic model—first introduced in the pioneering and seminal work [SY99] and later named $\mathcal{OBLOT}$—assumes robots are silent (i.e., without any direct communication means), and oblivious (i.e., without any persistent memory to retain data about past cycles). A more recent variant have considered robots with memory and communication means, albeit limited [Das+12; Das+16]: notably, here, robots are assumed to be *luminous*—hence the name $\mathcal{LUMI}$ model—i.e., embedded with a persistent register, called *light*, whose value, called *color*, can be updated at each LCM cycle choosing from a $O(1)$-size palette of colors, that is, of constant size regardless of the number of robots in the swarm. Being visible to the other robots in the swarm, each robot's light serves as an internal persistent memory and an external communication means. Halfway between $\mathcal{OBLOT}$ and $\mathcal{LUMI}$, other works have explored the impact on robots of having either memory or communication capabilities alone, by considering models where the light of each robot can be either external or internal [BDS08; FSW19].

In addition to memory and communication, numerous other features have been analyzed in the literature; among them, the synchronization modes and the partial or total agreement on a global coordinate system among robots can significantly influence their computational capabilities and, consequently, the design of distributed algorithms for them.

Following the lead of the survey [FPS19b], Section 1.2 provides a detailed description of the main features and their variants considered in the literature for the LCM framework, and analyzed throughout this thesis. Subsequently, Section 1.3 defines the computability concepts: what a problem is, when a problem is solved by a swarm and under a model, which are the metrics defining the complexity of an algorithm and thus of a problem, and, lastly, how to compare two LCM sub-models.

Note that the research for the LCM model has mostly focused on designing *deterministic* algorithms to solve a given problem. Two such well-studied problems are `Pattern Formation` and `Gathering`, the formal definition of which, although guessable from their names, will be provided in Section 1.3. However, *probabilistic* approaches have been explored for those problems that cannot be solved deterministically [BT16; Cle+10; DPBT19; DP09; YY14a] or to provide less constrained solutions w.r.t. the deterministic ones [DPT13].

1.1.3 Research areas

Correspondingly to what happens within the classic *theory of computation* [HMU06], the study on distributed computing by mobile robots regards and impacts two research areas. The first one concerns *computability* issues and focuses on the formal definition of the models and problems as abstractions of real-world scenarios, and the design of exact algorithms solving the problems under a given model. The second one concerns *complexity* issues, and deals with the definition of proper metrics to compute the cost, aka complexity, of an algorithm executed within a model, compare the costs for different models, and to reduce the model assumptions to solve a given problem.

More specifically, we can outline the following main lines of attack by which research has contributed to both areas:

- *Modeling novel and ad hoc features* in order to formalize more adversarial or realistic scenarios. An example of this direction is the introduction of *fault-prone* robots [AP06; CF25; DPBT19; Pat+20; PAS21], in contraposition with the standard model of fault-free robots, or *fat robots* [AGR25; CM15; Cor+11; DM23; Dat+13], i.e., modeled as solid discs on the plane with a non-null radius, in contraposition with the standard model of punctiform robots;

- *Formalizing and studying new problems* (as the `Dancing` problem, introduced in [Das+14] and dealt with in Chapter 7). Some problems typically studied in other areas may be considered for the LCM model too, after due adaptations. This is the case of Fault Detection and Fault Identification, typical problems in the engineering area and considered in some distributed models, as we will see in Chapter 4: recently, these problems have been introduced for the LCM model [CF25];

- *Adopting model checking methods* [Bal+19; Cou+16; Pot+19] or *formal tools* drawn from other fields (e.g., computational geometry [CMN04; LHP21; Lun+17; Lun+14; PAS21], formal languages [DP07a; DP07b; Fel+25a; FMP23a], graph theory [Fuj+10]) to analyze the correctness of the algorithms. We point out that the use of strings (e.g., Swing words [DP07b], Lyndon words [DP07a], and the Dyck words as we will see in Chapter 7) has been a valuable tool to analyze properties of the swarm configurations and to define algorithmic strategies.

- *Designing (more efficient) algorithms* for a given problem, considering different sub-models. E.g., `Gathering` [AP06; BCM18; CSN15; Cie+12; Déf+06; Flo19; PAM21] and `Pattern Formation` [AGR25; Bos+21; CSN21b; Col+20; Flo+25; Flo+08; Flo+99; Pre19; SS96; SY99; VST22; YY14b] have been the subject of a series of algorithmic solutions, from the less efficient to the optimal one, under different combinations of model assumptions;

- *Providing new complexity metrics* and trying to *optimize* the algorithmic solutions according to these metrics [BMR25; Bos+18; CSN15; Col+20; OWD23; Sha+25]. An example of this research direction is [FPS24a] that defines the performance SEC[1] as spatial metrics to analyze algorithms;

- *Comparing different LCM sub-models* according to their ability to solve problems—formally, their computational power. This comparison has been the subject of a recent and prolific line of research, as we will see in Part I.

1.2 The *Look-Compute-Move* model

The *Look-Compute-Move* (LCM) model has been widely used to formalize swarms of robots, especially operating in continuous spaces. Robots present some *core features* that are always assumed in the general LCM model. Starting from this latter, several sub-models have been theorized and studied, which differ in some *variable features*.

This section provides a detailed description and proper formalization of all the core (variable, resp.) features assumed (considered, resp.) in this thesis. Note that, although literature has also considered robots deployed in different topological spaces ($\mathbb{R}^3$ or discrete spaces), or fat robots (thus, non-punctiform), this thesis will only focus on punctiform robots deployed on the *Euclidean plane* $\mathbb{R}^2$.

1.2.1 Core features

Robots. A *swarm of robots* is a set of n computational entities $\mathcal{R} = \{r_1, \ldots, r_n\}$, which act in the Euclidean plane $\mathbb{R}^2$. Robots are *autonomous* (without any central control), *indistinguishable* (they cannot be distinguished by external appearance), *anonymous* (they are not provided with any id), *homogeneous* (they execute the same deterministic algorithm), and *punctiform* entities.

[1] In the paper, *computational SEC*.

Time domain. Robots perform their actions throughout a time domain $\mathbb{T}$. As will be explained later, $\mathbb{T}$ can be discrete (i.e., $\mathbb{T} = \mathbb{N}_0$, where $\mathbb{N}_0$ represents the set of all non-negative integers) or continuous (i.e., $\mathbb{T} = \mathbb{R}_{\geq 0}$).

Lights. Each robot $r \in \mathcal{R}$ is embedded with a persistent light $\texttt{lig}_r$ which can assume a color at a time. The color is chosen from a *palette*, namely, a set of colors. According to the sub-model (see below for the different variants), the color of $\texttt{lig}_r$ can be *perceivable*[2] by a subset of $\mathcal{R}$. In particular, if the light is *internal*, then its color is perceivable by r itself; if the light is *external*, then its color is perceivable by the other robots in $\mathcal{R} \setminus \{r\}$. Note that the two properties are independent; thus, a light can be both internal and external, or neither.

Local coordinate system. Let $\hat{\Xi}$ be a fixed *absolute coordinate system* on the Euclidean plane, so that each point is mapped into a position in $\mathbb{R}^2$. Being punctiform, each robot lies on one point of the plane at a time, and thus is located in one position at a time according to $\hat{\Xi}$. Each robot $r \in \mathcal{R}$ is equipped with a *local coordinate system* (which may differ from that of other robots and be time-varying, as we will see later) through which it maps the positions on the plane. Let $\Xi_t(r) = (o, \theta_x, \theta_y, u)$ denote the local coordinate system of r at time $t \in \mathbb{T}$, where o, θ_x, θ_y, u stand for the position of the origin, the direction in $[0, 2\pi)$ of the positive x-axis and the positive y-axis, and the unit of measure, respectively, w.r.t. $\hat{\Xi}$. Note that r is unaware of $\hat{\Xi}$ and thus of the values o, θ_x, θ_y, u.

Sensor system. Each robot $r \in \mathcal{R}$ is equipped with a *sensor system* that allows sensing the positions and (possibly) the colors of the *visible* robots in $\mathcal{R}$ at time t, according to a *visibility relation* $\bowtie_t \subseteq \mathcal{R}^2$ whose meaning will be made clear below. We always assume that $\bowtie_t$ is reflexive (thus a robot always sees itself) and symmetric (if r sees a robot s, then s sees r).

Algorithm. All robots in $\mathcal{R}$ are provided with the same *deterministic* algorithm: any robot executes such an algorithm every time the robot itself is activated. Let $\mathbb{A}$ be the deterministic algorithm executed by $\mathcal{R}$: $\mathbb{A}$ defines also the *palette* $\mathfrak{L}$ which contains the colors used for the robots' lights. We assume that $\mathfrak{L}$ is a *non-empty* $O(1)$-size set (thus, of constant-size regardless of the swarm size n), and that it contains at least the *null color* $\emptyset$, with which all robots' lights are initialized[3]. The algorithm also defines a total order of $\mathfrak{L}$.

LCM cycle. Robots are idle by default and become active as soon as an *activation scheduler* activates them (we will often refer to it as the *scheduler* when no ambiguity arises). Assume that a robot r is activated at a time $t \in \mathbb{T}$. Then, r executes a *Look-Compute-Move* (LCM) cycle, consisting of the following steps (see Algorithm 1):

- **Look:** let $\bowtie_t (r) = \{r_{i_0}, \ldots, r_{i_l}\}$ with $0 \leq l < n$ be the set of robots visible to r at time t. So r takes the instantaneous *snapshot* of the robots in $\bowtie_t (r)$, according to its local coordinate system $\Xi_t(r)$. Precisely, this snapshot is the

[2] We use the term "perceivable" instead of "visible" to avoid ambiguities with the visibility relation $\bowtie$ explained later.

[3] For the sake of clarity, the initialization color `OFF` will sometimes be used instead.

Algorithm 1: LCM cycle executed by a robot r.

***Look*:**
  $\sigma \leftarrow \langle (\underline{x}_{i_0}, c_{i_0}), \ldots, (\underline{x}_{i_l}, c_{i_l}) \rangle$ snapshot taken by r;
***Compute*:**
  $(\underline{x}', c') \leftarrow \mathbb{A}(\sigma)$;
***Move*:**
  if $c' \neq -$ **then**
    $\mathtt{lig}_r \leftarrow c'$;
  r reaches $\underline{x}'$ moving along a straight trajectory;

tuple[4]

$$\sigma = \langle (\underline{x}_{i_0}, c_{i_0}), \ldots, (\underline{x}_{i_l}, c_{i_l}) \rangle \tag{1.1}$$

where, for any $0 \leq j \leq l$, $\underline{x}_{i_j} \in \mathbb{R}^2$ is the position of r_{i_j} according to $\Xi_t(r)$ and $c_{i_j} \in \mathfrak{L}$ is its color. We always assume that $(\underline{x}_{i_0}, c_{i_0})$ refers to the position and color of r itself[5]. The other values of the tuple are ordered according to the local coordinate system of r and the total order of the colors in $\mathfrak{L}$. We remark that *no other information* about the swarm (e.g., whether robots are idle/active, still/moving, ...) is displayed in the snapshot taken by r.

- **Compute:** r executes the algorithm $\mathbb{A}$ with σ as the *sole* input. Specifically, r computes $\mathbb{A}(\sigma) = (\underline{x}', c')$, where $\underline{x}' \in \mathbb{R}^2$ is the position according to $\Xi_t(r)$ to be reached and $c' \in \mathfrak{L} \cup \{-\}$ is the possible color to be assumed. We assume that $-$ is a special symbol never contained in $\mathfrak{L}$ which indicates to r not to update its light.

- **Move:** if $c' \in \mathfrak{L}$, r updates its light color to c'; then, r travels along a *straight* trajectory towards the computed destination $\underline{x}'$. If the computed destination position is equal to the current one, r is said to perform a *null movement*.

After the Move step, r becomes idle again, ready for the next activation.

Observation 1 (Idempotent Compute)**.** Most of the literature, consistent with the works that introduced the $\mathcal{LUMI}$ model [Das+16; Das+12], assumes that the light update occurs at the end of the Compute step. In this thesis, as well as in the related papers and other works [Bos+19; FMP18; PS24; PAS21; SVT21; Sha+16], we assume the light update occurs as the first action in the Move step. Indeed, from a computational perspective, the two assumptions are equivalent: this is clear since these assumptions only concern the naming and grouping of the robot's actions, not the nature or the effects of such actions. However, we prefer to keep the Compute step *idempotent* (as well as Look), and to make Move the only step that changes the state of the robot (i.e., color and position). We feel that decoupling the idempotent phase from the non-idempotent phase of a LCM cycle (as shown in Figure 1.1) encourages possible evolutions of the LCM model where the Move step abstracts a step where the entity updates its state, regardless of the domain to which the state belongs (e.g., position, internal memory, color, state of charge, etc.).

[4] We need a locally ordered data structure to manage snapshots so that r can always spot its state (i.e., position plus light color) among the states of the other robots.

[5] Since $\bowtie_t$ is reflexive, $\bowtie(r)$ contains at least r.

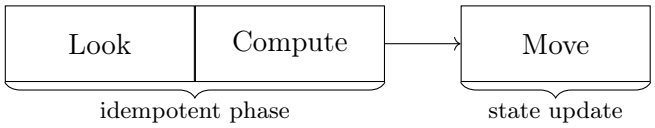


FIGURE 1.1: Decoupling of the idempotent steps of the LCM cycle from the non-idempotent one.

1.2.2 Orientation

A robot senses the environment (i.e., the plane and the robots arranged on it) according to its own local coordinate system. Most of the literature has considered swarms of robots where no or few assumptions on their local coordinate systems are made: in other words, robots may be partially or totally *disoriented*, i.e., devoid of a global coordinate system. In the most limited scenario, the local coordinate systems of the robots may differ from each other in *any of their components* (i.e., origin, unit distance, axes position, and axes orientation) [BT15; CDSN19b; Fel+25a; Sha+16]. The rationale behind this strong limitation is to design algorithms that are robust to orientation system failures. However, to solve some problems, some orientation assumptions are necessary [Flo+99] or are not yet proved to be unnecessary [Das+20; PAS21]; e.g., one of the most used assumptions is the chirality agreement, i.e., the agreement on the clockwise direction of the coordinate systems [BDL20; BLD23; CSN21b; Das+15].

Formally, let $\mathcal{R}$ be a swarm of robots acting on a time domain $\mathbb{T}$. Let r and r' be any robots on $\mathcal{R}$, and let $\Xi_t(r) = (o, \theta_x, \theta_y, u)$ and $\Xi_t(r') = (o', \theta'_x, \theta'_y, u')$ be their local coordinate systems for any given time $t \in \mathbb{T}$. We can make different assumptions on the different components of $\Xi_t(r)$ and $\Xi_t(r')$, possibly in conjunction (some depicted in Figure 1.2):

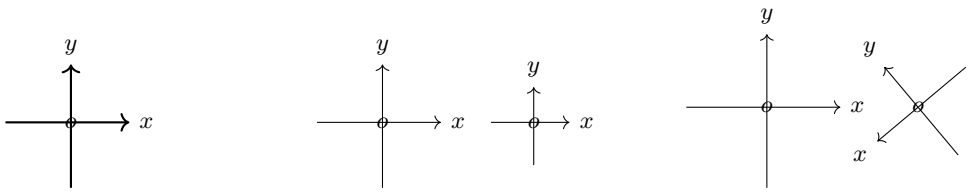
- *origin* agreement: $o = o'$;
- x-axis (y-axis, resp.) agreement: $\theta_x = \theta'_x$ ($\theta_y = \theta'_y$, resp.);
- *chirality* agreement: $\theta_y - \theta_x \equiv_{\mathrm{mod}\ 2\pi} \theta'_y - \theta'_x$;
- *unit distance* agreement: $u = u'$.

Combining them, we obtain a lattice of orientation settings: the strongest setting assumes robots are *totally oriented*, i.e., $\Xi_t(r) = \Xi_t(r')$ for any t and $r, r' \in \mathcal{R}$. This means they possess a global coordinate system. On the other end of the lattice, the weakest setting assumes nothing about the local coordinate systems, thus $\Xi_t(r)$ and $\Xi_t(r')$ may differ in any component. In this case, we say that the robots are *totally disoriented*.

Note that the previous settings are on the level of orientation agreement among the robots at a given time. Another important aspect which can be considered is the consistency of $\Xi_t(r)$ as $t \in \mathbb{T}$ varies. In this case, two settings can be assumed:

- **Fixed disorientation** (FIXDIS): this setting assumes that the local coordinate of each robot remains invariant at each activation. Formally, $\Xi_t(r) = \Xi_{t'}(r)$ for any $r \in \mathcal{R}$ and for any $t, t' \in \mathbb{T}$. In this case, we can simply define $\Xi(r)$ by omitting the time parameter;
- **Variable disorientation** (VARDIS): this setting assumes nothing about $\Xi_t(r)$ and $\Xi'_t(r)$ for any $t, t' \in \mathbb{T}$. Thus, the local coordinate system of a robot r may change at any activation.

Indeed, the level of orientation agreement (i.e., agreement on the origin, unit distance, or axes orientation) and the consistency over time (thus, FIXDIS and VARDIS) are two independent features that can be assumed independently, in any combination.



(A) Total agreement, $\Xi = \Xi'$.

(B) Different origin and unit distance. Agreement on axes orientation, thus on chirality.

(C) No orientation agreement.

FIGURE 1.2: Some examples of orientation agreements between two coordinate systems Ξ and Ξ'.

Egocentrism. A typical assumption made on disoriented robots is that each local coordinate system is *egocentric*, i.e., centered in the current position of the corresponding robot [SY99]. Formally, if $\Xi_t(r) = (o, \theta_x, \theta_y, u)$, then r is located in o at time t. The *egocentrism assumption*, here named EGO, aims at simplifying the formalization of coordinate systems, under the VARDIS setting. As a matter of fact, it is always possible for r to translate its $\Xi_t(r)$ into the egocentric version, without altering the other three components of the coordinate system. Thus, EGO can be assumed *without loss of generality* under VARDIS.

We denote with $\text{FIXDIS}_{\text{EGO}}$ the FIXDIS orientation setting where however the EGO assumption holds: more specifically, under $\text{FIXDIS}_{\text{EGO}}$, the orientation of the coordinate axes and the unit distance for a robot r are assumed to be invariant from cycle to cycle, while the origin always corresponds to the current position of r.

1.2.3 Memory and communication

Regarding the *memory* and *communication* features of robots, four models have been mainly proposed in the literature that differ in the (im)possibility of storing some information cycle by cycle, or of communicating with other robots [Das+16; Das+12; Flo+23; FSW19] (see Table 1.1). Such a (im)possibility has been formalized by the literature through the properties of the robots' lights, and thus of the snapshot in Equation (1.1) [Fel+25c]. Notably, for any robot, an internal light can play the role of a constant-size (namely, independent of the cardinality of the swarm $\mathcal{R}$) persistent memory; an external light can be exploited to communicate constant-size information between visible robots.

In the $\mathcal{OBLOT}$ model, robots are assumed to be *oblivious* (i.e., they do not have any persistent memory to store data about past cycles) and *silent* (i.e., they do not have any means of communicating with other robots). In this model, robots' lights are *neither internal nor external*; indeed, this is equivalent to stating that robots do not have any light, or, more elegantly, $\mathfrak{L} = \{\not{b}\}$ for any algorithm under $\mathcal{OBLOT}$. Therefore, any snapshot under $\mathcal{OBLOT}$ will be in the form

$$\sigma = \langle (\underline{x}_{i_0}, \not{b}), \ldots, (\underline{x}_{i_l}, \not{b}) \rangle$$

In the $\mathcal{FSTA}$ (*finite-state*) model, each $\texttt{lig}_r$ is *internal, but not external*, i.e., perceivable only by r. Thus, under $\mathcal{FSTA}$, any snapshot in Equation (1.1) will be in the form

$$\sigma = \langle (\underline{x}_{i_0}, c_{i_0}), (\underline{x}_{i_1}, \not{b}), \ldots, (\underline{x}_{i_l}, \not{b}) \rangle$$

i.e., $c_{i_j} = \not{b}$ for any $1 \leq j \leq l$.

Setting	no memory	memory
no communication	$\mathcal{OBLOT}$	$\mathcal{FSTA}$
communication	$\mathcal{FCOM}$	$\mathcal{LUMI}$

TABLE 1.1: Four main models, differing in memory/communication settings.

In the $\mathcal{FCOM}$ (*finite-communication*) model, each $\texttt{lig}_r$ is *external, but not internal*, i.e., perceivable by all robots in $\bowtie(r) \setminus \{r\}$. This means that r can only see the light color of the other robots visible to itself, but not its own light. Under $\mathcal{FCOM}$, any snapshot in Equation (1.1) will be in the form

$$\sigma = \langle (\underline{x}_{i_0}, \not{\flat}), (\underline{x}_{i_1}, c_{i_1}), \ldots, (\underline{x}_{i_l}, c_{i_l}) \rangle$$

i.e., $c_{i_0} = \not{\flat}$.

Finally, the $\mathcal{LUMI}$ model encompasses the features of both $\mathcal{FSTA}$ and $\mathcal{FCOM}$, where each $\texttt{lig}_r$ is *both external and internal*. Robots in this model are said to be *luminous*, and can exploit both memory and communication capabilities. Therefore, under $\mathcal{LUMI}$, snapshots have the form in Equation (1.1), without any restriction on colors.

1.2.4 Synchronization modes

Robots are idle by default and become active as soon as an *activation scheduler* activates them. Once activated, a robot starts executing an LCM cycle; after that, it becomes idle again, ready to be reactivated by the scheduler. Formally,

Definition 1 (Activation Scheduler)**.** *An* activation scheduler *(often abbreviated as just* scheduler*) for a swarm $\mathcal{R}$ over the time domain $\mathbb{T} \in \{\mathbb{N}_0, \mathbb{R}_{\geq 0}\}$ is a function $\mathfrak{A} : \mathbb{T} \to 2^{\mathcal{R}}$, so that $\mathfrak{A}(t)$ represents the subset of robots activated at time $t \in \mathbb{T}$.*

A scheduler can undergo different synchronization policies, which we will refer to as (synchronization) *modes*. Such modes define some conditions on both the robots' activations (e.g., time domain, frequency, periodicity, and concurrency) and the LCM steps (e.g., duration and synchronization). Generally, the literature always assumes the following basic conditions:

- the Look step is always instantaneous;
- the update of the color light is instantaneous;
- the Compute and Move steps are finite[6].

Another example of a condition on robots' activation (specifically, on their frequency) is the *fairness condition* which requires that any robot in the swarm is activated infinitely often. Formally,

Definition 2 (Fairness Condition)**.** *A scheduler $\mathfrak{A} : \mathbb{T} \to \mathcal{R}$ is said to be* fair*, or, equivalently, to satisfy the* fairness condition*, if for any $t \in \mathbb{T}$ and for any $r \in \mathcal{R}$, there exists a finite time $t' > t$ such that $r \in \mathfrak{A}(t')$.*

We say that a mode is fair if it admits only fair schedulers. The fairness condition is generally assumed for twofold reasons: firstly, it avoids deadlock situations caused

[6]This is true assuming that both the algorithm executed by the robots and the computed destination are finite.

by robots waiting forever; then, it allows for measuring time $\mathbb{T}$ in consecutive and finite time units called *epochs*. The first epoch starts at time 0; each epoch ends when all the robots of the swarm are activated at least once since the start of the current epoch; after that, a new epoch starts with the subsequent activation. Let us provide a formal definition.

Definition 3 (Epoch)**.** *Let $\mathfrak{A} : \mathbb{T} \to 2^{\mathcal{R}}$ be an activation scheduler for a swarm $\mathcal{R}$. Then, $\mathbb{T}$ can be partitioned (or* segmented*) in consecutive time frames $\mathcal{E}_0, \mathcal{E}_1, \ldots, \mathcal{E}_i, \ldots$, where each $\mathcal{E}_i = [t_{\mathcal{E}_i}, t_{\mathcal{E}_{i+1}})$ is called* epoch *and is defined recursively:*

- $t_{\mathcal{E}_0} = 0$*;*
- $t_{\mathcal{E}_{i+1}} = \min\{t \in \mathbb{T} \text{ s.t. } \mathfrak{A}(t) \neq \varnothing \ \wedge \ \forall r \in \mathcal{R} \ \exists t' \in [t_{\mathcal{E}_i}, t) \text{ s.t. } r \in \mathfrak{A}(t')\}$.

We now describe the synchronization modes mainly present in the state-of-the-art (some are depicted in Figure 1.3).

The most general mode is the *asynchronous* mode (`ASYNCH`) where robots (and schedulers) act in the continuous time domain $\mathbb{T} = \mathbb{R}_{\geq 0}$, without any assumption of synchronization. Thus, idle robots can be activated at any time, independently of other robots or past activations; their Compute and Move steps last an unpredictable but finite time. However, the Look step is always assumed to be instantaneous.

The first step towards synchronism is given by the *semi-synchronous* mode (`SSYNCH`). Under `SSYNCH`, time is globally discretized in unit times called *rounds* (in this case, $\mathbb{T} = \mathbb{N}_0$) and the robots activated at time t perform one LCM cycle within round t. All the LCM steps are assumed to be instantaneous and performed in perfect synchrony within each round. A `SSYNCH` scheduler activates an arbitrary non-empty subset of the swarm at each round. Formally, $\mathfrak{A} : \mathbb{T} \to 2^{\mathcal{R}}$ is `SSYNCH` if $\mathbb{T} = \mathbb{N}_0$ and $\mathfrak{A}(t) \neq \varnothing$ for any t.

The *energy-constrained* mode `RSYNCH` [Buc+25] is a sub-mode of `SSYNCH` which assumes that no robot can be reactivated in the subsequent round after an activation: formally, $\mathfrak{A}(t) \cap \mathfrak{A}(t+1) = \varnothing$ for any $t \in \mathbb{N}_0$. This mode has been introduced to formalize systems where robots have limited but renewable energy, so that they need at least one round of inactivity to recharge their battery.

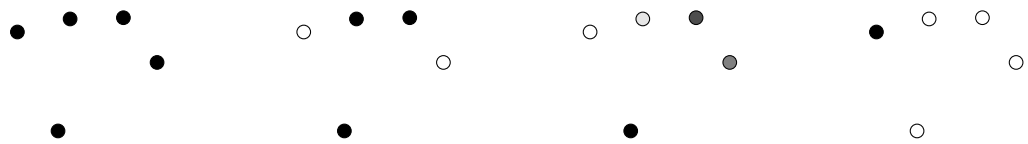
The *k-bounded* (`kBOUND`) mode [DPBT19; SYD08] is a sub-mode of `SSYNCH`[7] which assumes that, between any two consecutive activations of a robot r, any other robot $s \neq r$ is activated at most k times, for a $k > 0$ constant. Formally, $\mathfrak{A}$ is `kBOUND` for a given $k > 0$ if $\mathfrak{A}$ is `SSYNCH` and if $r \in \mathfrak{A}(t') \cap \mathfrak{A}(t'')$ and $r \notin \mathfrak{A}(t)$ for any $t' < t < t''$, it holds that for any $s \in \mathcal{R} \setminus \{r\}$, $|\{t \text{ s.t. } s \in \mathfrak{A}(t) \wedge t' \leq t < t''\}| \leq k$.

One of the most studied sub-modes of `SSYNCH` is the *fully synchronous* mode (`FSYNCH`), which assumes that any robot is activated at any round (in this case, any round is an epoch). Trivially, for any swarm $\mathcal{R}$ there exists only one `FSYNCH` scheduler, which maps $t \mapsto \mathcal{R}$ for any $t \in \mathbb{N}_0$.

A recently studied sub-mode of `SSYNCH` is the *sequential* mode (`SEQ`) [DPBT19; FFS25], which, as the name suggests, activates only one robot at round. Formally, a sequential scheduler can be written as $\mathfrak{A} : \mathbb{N}_0 \to \mathcal{R}$. In turn, different sub-modes of `SEQ` can be defined.

The *permutation* mode `PERM` [FFS25] is a sequential mode where any epoch lasts exactly n rounds. In other words, all the robots are activated only once in some order (i.e., permutation) before starting again the activation of the swarm (with possibly another permutation).

[7] Even if it can be extended also to `ASYNCH`, we consider only the synchrnonous version.



(A) FSYNCH. (B) SSYNCH. (C) ASYNCH. (D) SEQ.

FIGURE 1.3: Some synchronization modes. Gray-scale colors represent different phases during asynchronous LCM cycles.

The well-known *round-robin* mode RROBIN [CF25; DPBT19; FFS25; FW24; NP25] is a particular sub-mode of PERM where the permutation of each epoch is invariant; thus, robots are activated with the same order at any epoch. Note that RROBIN is a sequential 1-bounded mode [DPBT19].

Remark 1. Due to its properties, the fairness condition is generally always assumed in the literature. Coherently, this work always assumes the fair version of ASYNCH, SSYNCH, RSYNCH, and SEQ. Indeed, kBOUND, FSYNCH, PERM, and RROBIN are fair by definition.

Adversarial power of schedulers. The literature has dealt with a plethora of modes with the aim of investigating the behavior of swarms under different synchronization policies. Schedulers have been seen as adversarial entities that control the robots' activations and make the system more or less challenging. Indeed, ASYNCH represents the mode where the schedulers have more adversarial power, in that it is the least constrained. On the contrary, the FSYNCH mode has no power at all, since the activation scheduler for each swarm is fixed and unique.

1.2.5 Motion

The LCM model generally assumes that, during each Move step, a robot moves along a straight trajectory connecting the starting position (where the robot was still performing the Look and Compute steps) and the computed destination position. Thus, if a robot r must travel from point p to point p' during a cycle, it will maintain the same trajectory direction $\vec{pp'}$ along its Move step (except for when r has speed 0, where no direction is defined).

On the contrary, robots may not have any means to define or adjust the speed and acceleration at which they move along the trajectory. In fact, while the Move steps are assumed to be instantaneous in the synchronous modes, asynchronous robots move along the trajectories with arbitrary and unpredictable speeds and accelerations, which are beyond the control of the robots. Note that, during its path from p to p', the speed of r may be zero even while the robot is located in the open segment $\overline{pp'}$.

1.2.6 Rigid movements

In the standard setting, any robot always reaches the computed destination in finite time during the corresponding Move step.

However, to examine different scenarios, the literature proposes some variant settings where robots' motion may be interrupted by an adversary:

- **Rigid Setting** (RIG). This is the standard setting, which assumes that all robots reach their destination point during the Move step. Robots are said to perform *rigid* or, equivalently, undisturbed movements.

- **Fixed-Rigid Setting** (FixRig). An adversary stops a robot as soon as the robot travels a maximum distance $\gamma > 0$, where γ is a constant not known by the robots. Formally, if a robot r lies on p and must reach a destination point p' so that $dist(p, p') = d$, then r will cover a d'-length path during its Move step, where $d' = \min\{d, \gamma\}$.
- **Non-Rigid Setting**, (NonRig). In this setting, each robot may be stopped unpredictably by an adversary only after having traveled at least a constant distance $\epsilon > 0$ not known by the robots. Formally, if a robot r lies on p and must reach a destination point p' so that $dist(p, p') = d$, then r will cover a d'-length path during its Move step, where $\min\{d, \epsilon\} \leq d' \leq d$.

Indeed, there exists a hierarchy among the three settings: Rig is a special case of FixRig, which is in turn a special case of NonRig. This leads to a hierarchy of their adversarial power, where Rig represents the most optimistic case since the adversary has no power to interrupt robots' movements. On the contrary, FixRig and NonRig aim at formalizing more realistic scenarios: in the first setting, robots can exploit a fixed quantity of energy for the movement (e.g., robots are powered by fixed-energy batteries), while the second setting describes more hostile environments where robots may encounter unpredictable obstacles.

1.2.7 Multiplicities and collisions

Robots are punctiform entities on the plane. The standard model allows a group of robots to form a *multiplicity*, namely to occupy the same position at the same time. In this case, different assumptions can be made according to the capability of a robot r to distinguish the robots of a multiplicity during its Look step:

- **strong multiplicity detection**: r can distinguish all the robots forming a multiplicity (thus count them and see their colors);
- **weak multiplicity detection**: r can only detect if a given position is occupied by zero, one, or more robots, without detecting their precise number in the latter case. If robots in the multiplicity have different colors, r can distinguish the colors, without counting their occurrence.
- **no multiplicity detection**: r can only detect if a given position is occupied or not. Colors are generally not assumed in this setting [Bal+19].

More finely, the capabilities of r of detecting multiplicities can be limited to its own position (*local multiplicity detection*) or extended to all visible positions (*global multiplicity detection*).

Intuitively, a multiplicity implies that some robots coexist or have collided at the same point. However, in some contexts, it may be necessary to investigate systems where collisions are not tolerated and thus design *collision-free algorithms* to maintain these systems intact. Formally, a *collision-intolerant system* do not tolerate either multiplicities (i.e., no robot can occupy the same position of another robot in the swarm at the same time) and *overlapping trajectories* (i.e., robots r and s have overlapping trajectories if *(i)* r is moving from point a to point a', *(ii)* s is moving from point b to point b', and *(iii)* the line segments $\overline{aa'}$ and $\overline{bb'}$ have points in common) [FMP23a; FMP23b]. We refer to both multiplicity and overlapping trajectories as *collisions*. We remark that the fact that $\overline{aa'}$ and $\overline{bb'}$ are overlapping trajectories do not imply necessarily that r and s will collide in some point during their movements. However, since the speed of the robots is unpredictable, a collision may occur. For this reason, a collision-intolerant algorithm must avoid these situations at all.

1.2.8 Visibility

During its Look step, a robot perceives the positions and, possibly, the light colors of all robots *visible* from r. As seen in Section 1.2.1, the *visibility relation* $\bowtie_t \subseteq \mathcal{R}^2$, which is assumed to be reflexive and symmetric, defines the pairs of mutually visible robots at time $t \in \mathbb{T}$. Naturally, $\bowtie_t$ can be expressed by the corresponding *visibility graph* $G_{\bowtie_t}$ such that $V(G_{\bowtie_t}) = \mathcal{R}$ and $\{r, s\} \in E(G_{\bowtie_t}) \iff r \neq s \wedge r \bowtie_t s$.[8] We will use the notation $\bowtie$ instead of $\bowtie_t$ when it is not necessary to express the time parameter. We remind that $r \bowtie s$ does not imply that r can perceive the color of $\texttt{lig}_s$ (this holds even if $r = s$).

In the standard setting, robots are assumed to have *complete visibility*, namely $\bowtie_t = \mathcal{R}^2$ for any $t \in \mathbb{T}$. In other words, robots are said to be *transparent* since they do not obstruct the visibility of other robots in case of collinearities (three robots r, p, q form a collinearity, aka are collinear, if they lie on distinct positions of the same straight line in that order), and *non-myopic* since they can see robots independently of their mutual distance.

However, these strong assumptions can be dropped to study more limited and realistic settings:

- **obstructed visibility**: all robots are assumed to be *opaque*, thus if three robots are collinear, say r, p, s, then r and s cannot see each other since the robot p obstructs their visibility [BCM18; Bos+21; BLD23; Fel+25c; FMP23a; FPS24a; SVT21]. Formally,

$$\bowtie_t (r) = \{r\} \cup \{s \in \mathcal{R} \text{ s.t. } \nexists p \in \mathcal{R} \setminus \{r, s\} \text{ s.t. } r, p, s \text{ are collinear at time } t\}$$

- **limited visibility**: all robots are assumed to be *myopic*, thus they can see only robots whose mutual distance is at most ρ_V, which is called *visibility radius*. Formally,

$$\bowtie_t (r) = \{s \in \mathcal{R} \text{ s.t. } dist(r, s) \leq \rho_V \text{ at time } t\}.$$

  Generally, the limited visibility assumption is always in conjunction with the hypothesis that the swarm starts from an initial configuration where, for any robot r, there exists another robot s such that $dist(r, s) \leq \rho_V$. In other words, the visibility graph $G_{\bowtie_0}$ is assumed to be always connected. Myopic robots have usually been considered in discrete environments, where the value of ρ_V is known by the robots [Bra+24a; Kam+19; NOI23]. In continuous spaces, if the value of ρ_V is a known parameter, this implies that robots agree on the unit distance [Cas+22]; instead, other works do not assume robots are aware of their visibility radius [Flo+05; Kir+24].

Indeed, the two features (i.e., transparent vs. opaque and non-myopic vs. myopic) are independent of each other; thus, they can be combined to analyze systems of opaque myopic robots.

Opaqueness and multiplicities

Generally, literature considers the opaque model (i.e., with obstructed visibility) in conjunction with the collision-intolerant assumption, namely, where robots are not allowed to form multiplicities [Bos+21; FMP23a; FPS24a; PS24; SVT21; Sha+16]. As explained in [Fel+25c], the rationale behind this choice is to avoid theoretical inconsistencies: in fact, a multiplicity of two robots forms a "degenerate" collinearity

[8] Since $\bowtie_t$ is always reflexive, loops can be omitted.

with any other robot of the swarm, thus making the definition of the visibility relation among robots ambiguous.

1.2.9 Faultiness

Typically, robots are assumed to be always *correct*, or equivalently *non-faulty*; namely, they always execute a complete LCM cycle at any activation, performing each LCM step without inaccuracies or errors. However, failures may occur in many guises in the real world, and their occurrence must be taken into account in order to analyze the fault tolerance of these systems. For this purpose, literature has mainly formalized and examined these types of faults [DPBT19].

- **Crash faults**. A correct robot r may unpredictably crash at any time. From that moment on, r is a *crashed* robot, and it will no longer execute an LCM cycle when activated, thus will no longer update its state (position and color). Typically, crashes are supposed to be irreversible [CF25; DPP20; Pat+19].
- **Byzantine faults**: A correct robot r may start behaving unpredictably, e.g., moving/setting to a wrong destination/color during an LCM cycle. Such faults are generally symptoms of malicious behaviors (e.g., corrupted robots) [Ash+19; Aug+13; DPP20; MMJ20].
- **Inaccurate robots**: A robot may have imprecise sensors or actuators, and therefore perform inaccurate actions: taking partial snapshots [Oos+24] or imprecise measurements [CP08; SDY06] through defective sensor systems, processing algorithms without infinite algebraic precision, or making movements with slight deviations [BDS21; CP08]. However, the inaccuracy is generally bounded by an accuracy parameter.

Some variants may assume that the above faults are *transient* (i.e., persisting only for a limited, maybe unpredictable, amount of time), or *fixable* (where thus correct robots can fix the faulty one). The first variant opens the way to the study of *self-stabilizing* algorithms; the second to *fault recovery* algorithms.

1.2.10 (Sub-)Models

In the previous sections, we have described all the main features that can characterize an LCM sub-model. Indeed, the LCM model is an umbrella model encompassing a plethora of different sub-models differing in a combination of features (memory, communication, visibility, orientation, synchronization,...). The literature has explored several possible sub-models; the aim is to investigate how specific combinations of features affect the capability of robots in performing certain tasks (aka solving certain problems). Consequently, any scientific work studying the behavior of a given sub-model must accurately introduce all the features that characterize it. When no ambiguity arises, we will simply use the term model instead of sub-model.

For a better understanding and nomenclature standardization, the following notation has been introduced and widely adopted to denote the basic features of a model [Buc+25; CF25; FFS25; Flo+23; FSW19]. We indicate with $M = X^Y$ an LCM (sub-)model which enjoys all the core features explained in Section 1.2.1, and that has

- $X \in \{\mathcal{OBLOT}, \mathcal{FSTA}, \mathcal{FCOM}, \mathcal{LUMI}\}$ as memory-communication setting;
- $Y \in \{\texttt{ASYNCH}, \texttt{SSYNCH}, \texttt{FSYNCH}, \texttt{kBOUND}, \texttt{SEQ}, \texttt{PERM}, \texttt{RROBIN}, \dots\}$ as synchronization mode.

All the other features (visibility, orientation, ...) must be explicitly specified. In some cases [Das+24; Fel+25c], the notation X_V^Y has been adopted to indicate through the superscript the visibility setting $V \in \{\mathcal{T}, \mathcal{O}, \dots\}$ (e.g., where $\mathcal{T}$ refers to the transparent setting, and $\mathcal{O}$ refers to the opaque one). When no superscript is specified, we assume the standard model with transparent and non-myopic robots. As we will see in the next section, these compact notations turn out to be convenient and functional when comparing different models.

1.3 Computability

Robots are computational entities, i.e., able to perform an algorithm and thus collaboratively solve a given task, formally, a *problem*. Here, we identify two classes of problems: the first one includes the so-called *agreement-related problems*, which constitute the primitive problems for any general distributed system, while the second one comprises the ad hoc problems for the LCM model.

In this section, after a brief overview of the first class, we provide all the elements necessary to define what a problem is for the LCM model and when a swarm solves it.

1.3.1 Agreement-related problems

The *agreement-related* problems [ASY95; DPBT19; San07] represent a basic class of problems for the conventional distributed systems, thus not necessarily formed by mobile entities. To solve an agreement-related problem, a swarm (or a part of it) must reach an agreement about some information. We list the main agreement-related problems, whose definitions are inherited from the general distributed computing theory [San07].

- `Leader Election` [CM15; Die+13], where the swarm must agree on one / some robots as leaders;
- `Consensus`, where the swarm must agree about some information. A typical example in the LCM model, especially as a subroutine for pattern formation problems, is to totally/partially agree on a global coordinate system [Fel+25a], or on the location of the meeting point in `Gathering` [Cle+10];
- `Broadcast`, where the swarm must agree about some information generated by a robot. A valid example can be the broadcast of some information among different connected components in a swarm of myopic robots.
- `Fault Detection/Identification`, where the (correct part of the) swarm must agree on the presence of at least one faulty robot (*detection*) or on the classification of robots between the correct and the faulty ones (*identification*). These two problems will be deeply treated in Chapter 4.

Note that swarms of silent robots (as in $\mathcal{FSTA}$ or $\mathcal{OBLOT}$) may reach an agreement through *stigmergy*[9], i.e., an indirect communication strategy using movements and configurations as coordination messages [Die+19].

1.3.2 Configuration-related problems

For the LCM model, the typical problems proposed in the literature require swarms to form a (finite or infinite) sequence of configurations, possibly imposing some constraints

[9] Term coined to refer to termite behaviors.

in the formation of such configurations (e.g., traveling along specific trajectories, or avoiding collisions or forbidden patterns/configurations). As stated in [Buc+21; FFS25; Fel+25c], such problems can be defined as temporal predicates on the configurations formed by the swarm over time.

Definition 4 (Configuration)**.** *Let $\mathcal{R}$ be a swarm arranged on the Euclidean plane $\mathbb{R}^2$. We define the* configuration *of $\mathcal{R}$ at time t as the function $\mathcal{C}_t : \mathcal{R} \to \mathbb{R}^2 \times \mathfrak{L}$, where $\mathcal{C}_t(r) = (\underline{x}, c)$ represents the position $\underline{x}$ of r according to a global coordinate system $\hat{\Xi}$ and the color c of its light, both at time t. We refer to $\mathcal{C}_t(r)$ as the* state *of r at time t. We will omit t when irrelevant.*

We say that $\mathcal{C}_t$ is *static* if no robot is in its Move step at time t. Formally, a *configuration-related problem* for a swarm is defined [Fel+25c] as a (finite or infinite) sequence

$$P = (\phi_0, \tau_0, \phi_1, \tau_1, \dots, \phi_m, \tau_m, \dots)$$

where ϕ_i is a predicate on the swarm configuration and τ_i is a predicate that must be satisfied by every configuration assumed by the swarm while transitioning to a new configuration satisfying ϕ_{i+1}. The sequence $(\phi_0, \tau_0, \phi_1, \tau_1, \dots, \phi_m, \tau_m, \dots)$ is also said to be the *request of the problem* P. Any ϕ_i must include the clause requiring the configuration to be static. Moreover, the initial predicate ϕ_0 generally includes the *color uniformity clause* requiring that the light of all the robots in the swarm is initialized with the default color $\not{b}$. Except for these clauses, the conditions ϕ_i and τ_i cannot impose any restriction on light colors or absolute times, since P might in principle be solved under $\mathcal{OBLOT}$ (where thus $\mathfrak{L} = \{\not{b}\}$) and under any synchronization mode.

1.3.3 Solutions

We now formally state the notion of *solving* a configuration-related problem by a robot swarm [Fel+25c]. We write $\mathcal{C} \vDash \phi$ to state that the configuration $\mathcal{C}$ satisfies the predicate ϕ. So, let $\mathcal{R}$ be a robot swarm under a model $M = X^Y$ starting from an initial configuration $\mathcal{C}_0$ and executing an algorithm $\mathbb{A}$ which complies with the features of M. Let $\mathfrak{A} : \mathbb{T} \to 2^{\mathcal{R}}$ be an activation scheduling under the Y mode. We define the *evolution* induced by $\mathbb{A}$ w.r.t. $\mathfrak{A}$ and $\mathcal{C}_0$ as the sequence $\{\mathcal{C}_t\}_{t \in \mathbb{T}}$ such that $\mathcal{C}_t$ is the configuration reached by the swarm $\mathcal{R}$ at time t starting from the initial configuration $\mathcal{C}_0$ upon executing $\mathbb{A}$ according to the activation scheduling $\mathfrak{A}$. Such an evolution is a *solving* evolution for the problem $P = (\phi_0, \tau_0, \phi_1, \tau_1, \dots, \phi_m, \tau_m \dots)$ whenever a sequence $0 = t_0 < t_1 < t_2 < \cdots < t_m < \cdots$ of time instants exists, along which $\mathcal{C}_{t_i} \vDash \phi_i$ and $\mathcal{C}_t \vDash \tau_i$ for every $t_i \leq t < t_{i+1}$. More generally, we say that the problem P is *solved* under a given model X^Y whenever there exists an algorithm $\mathbb{A}$ such that, for any robot swarm under the model M:

- $\mathbb{A}$ works with the memory-communication X;
- for any $\mathcal{C}_0$ satisfying $\mathcal{C}_0 \vDash \phi_0$ and any $\mathfrak{A}$ under Y, $\mathbb{A}$ induces a solving evolution for P.

If the request of the problem is *finite*, the last condition τ_m requires the swarm to stay still after entering a configuration satisfying ϕ_m.

1.3.4 Classic problems

We here describe the main problems studied for the LCM model.

Formation of patterns

The most studied class of problems for the LCM model is pattern formation [AGR25; Bos+21; CDSN19a; CSN21b; Col+20; Flo+25; Flo+08; Pre19; Sha+25; SS96; SY99; YS10]. Informally, robots are required to arrange themselves on the plane so that their positions form a given pattern, which can be arbitrary or specific.

Formally, a *pattern* $\Pi = \{\!\{v_1, \ldots, v_n\}\!\}$ is a multiset[10] of positions $v_i \in \mathbb{R}^2$ [Fel+25a; Fel+25b]. We call these positions *vertices* of Π. Given a configuration $\mathcal{C} : \mathcal{R} \to (\mathbb{R}^2 \times \mathfrak{L})$, we denote with $\mathcal{C}_{|\mathbb{R}^2}$ the multiset of positions formed by $\mathcal{C}$, i.e., $\mathcal{C}_{|\mathbb{R}^2} = \{\!\{\underline{x} \text{ s.t. } r \in \mathcal{R} \wedge \mathcal{C}(r) = (\underline{x}, c)\}\!\}$. We denote with $\mathscr{P}(\Pi)$ the set of patterns similar to Π, i.e., all the patterns obtained from Π by any composition of non-degenerate similarity transformations: translation, rotation, reflection, and uniform scaling with a non-zero scale factor[11]. Formally, $\Pi' \in \mathscr{P}(\Pi)$ if there exists a non-degenerate similarity transformation $\tau : \Pi \to \Pi'$.

In the `Pattern Formation` problem [Pre19], a swarm $\mathcal{R}$ of n robots is given a pattern Π of n vertices, and robots are required to terminate in a configuration $\mathcal{C}$ such that $\mathcal{C}_{|\mathbb{R}^2} \in \mathscr{P}(\Pi)$. Trivially, for a Π pattern to be formable by a swarm, the number of vertices must match the number of robots in the swarm. We refer to this necessary condition as the *cardinality condition*: unless noted, we always consider it as assumed.

More generally, the `Arbitrary Pattern Formation` problem [Bos+21; BT16; CDSN19a; Flo+08; Sha+25] requires an n-robot swarm to form any arbitrary n-vertex pattern Π, starting from *any* initial configuration.

Specific patterns

In some cases, the research has focused on investigating the formation of specific patterns. This choice may be driven by two main motivations and goals: *(i)* it has been proved that `Arbitrary Pattern Formation` is not solvable under a given model, thus it may be interesting to characterize the subset of patterns formable under such a model [YS10], or *(ii)* the existing `Arbitrary Pattern Formation` algorithms are inefficient for specific patterns, thus research aims to propound an ad hoc and adequate algorithm for those cases [FPS24b].

The `Gathering` problem [Bal+19; BCM18; Bos+18; Flo19; Cas+22; Cie+12; CP02; Cor+11; Flo+05; NSW21; NP25; PAM21; SDY06] requires all the robots to meet at the same point, not known in advance. Indeed, it is a special case of `Pattern Formation` where Π is composed of n vertices located in the same position. By its interesting properties, `Gathering` has been deeply studied for robots moving in discrete environments [Bra+24a; Bos+18; Bra+24b; CSN21a; NP25]. The `Rendezvous` problem [Déf+23; Dob+03; OWD23] is the special case of `Gathering` where the swarm is composed of only two robots. Despite its simple request, `Gathering` has gained a lot of attention due to its several impossibility results in weak or fault-prone models. For example, it is self-evident that, under $\mathcal{OBLOT}^{\texttt{SSYNCH}}$, robots may never meet in the deterministic paradigm, without any additional assumption. Also, if two robots crash in different positions, `Gathering` cannot be achieved even assuming the strongest model ($\mathcal{LUMI}^{\texttt{FSYNCH}}$, completely oriented, etc.). Hence, the interest in probabilistic approaches and in defining weaker versions of the `Gathering` problem suitable for fault-prone systems [Bra+24a; Bra+24b; BLT23].

The `Uniform Circle Formation` (`UCF`) is another well-studied problem within the class of pattern formation. It requires a swarm of n robots to form a regular n-gon,

[10] We will use the notations $\{\!\{\}\!\}$ and $\{\}$ to indicate a multiset and a set, respectively.

[11] To avoid to degenerate `Pattern Formation` into `Gathering`, which are being presented shortly.

where each robot lies on one vertex of the target polygon. Indeed, the well-known properties (uniform distancing within a circle, equidistance from a common center, complete symmetry) of the target configuration have encouraged research to investigate this problem under several models [FMP23a; FPS24b; FPS24a; Flo+17; PS24] (a deeper discussion will be done in Chapters 5 and 6).

Patterns by properties

A more relaxed version of `Pattern Formation` is the class of problems where robots have to terminate forming a pattern which is not specified by its vertices, but by its properties; we provide some examples.

`Circle Formation` [CMN04; DK02] requires n robots to arrange on n arbitrary but distinct points of the boundary of the same (non-degenerate) circle, whose origin and radius can be arbitrary (as long as the radius is non-zero).

`Mutual Visibility` [BMR25; Lun+17; Lun+14] (also called `Complete Visibility` [PAS21; SVT21; Sha+16]) requires the swarm to terminate in a configuration where all robots are mutually visible. In the opaque model, thus suffering from obstructed visibility, this means that no collinearities must exist in the final configuration; in the myopic model, this could require that all robots are pairwise within their visibility range. For the opaque model, literature has often solved `Mutual Visibility` virtually arranging the robots on *convex polygons* [SVT21; Sha+16]. Together with `Circle Formation`, `Mutual Visibility` has been used as a sub-problem in some solutions for `Uniform Circle Formation` [FMP23a; FPS24a; FPS24b; Flo+17; PS24].

`Scattering` requires n robots to arrange on n distinct arbitrary points of the plane, thus breaking all existing multiplicities [BT17; Izu+18a; Izu+18b; MC22]. As for its "dual" problem `Gathering`, `Scattering` also suffer from some impossibility results even in strong models. E.g., a multiplicity cannot be broken under the `FSYNCH` mode, since the co-located robots may behave like *clones* if they all have the same coordinate system, thus always move at the same point; hence the need for probabilistic algorithms [BT17].

Perpetual problems

As seen in Section 1.3.2, not all the problems require robots to terminate. If the request is infinite, we say that the problem is *perpetual*.

Within this class of problems, `Flocking` [Can+16; CP07; GP04; SYD08; Yan+11] is one of the most studied due to its important applications. It requires a swarm of robots to perpetually move on the plane like a flock, maintaining the given flock pattern. Two approaches have been explored to cope with this problem. The first assumption is that there is an a priori leader that is identifiable by the swarm and that all the other robots must follow [GP04]. Indeed, while this assumption represents an advantage for the swarm, it also poses a risk in fault-prone systems. To overcome the single point of failure risk, the second approach does not assume a leader and so implements uniform algorithms (thus executed homogeneously by all the robots) [Can+16; CP07; SYD08; Yan+11]. Even this problem is particularly suitable for investigating probabilistic and/or fault-tolerant solutions [SYD08; Yan+11].

A quite novel problem and less studied problem is `Dancing`, where robots are asked to perform a (finite or infinite) sequence of patterns, called *choreography* [Das+20; Das+14; Das+15; Fel+25a]. This problem is specifically treated in Chapter 7.

For the sake of completeness, it is worth citing the `Exploration` problem where robots are required to perpetually visit some locations of the environment where they

operate. Due to its nature, this problem has been studied for robots deployed in graphs, rings, or grids [BDL20; NOI23; Oos+24].

1.3.5 Scalability

Most of the studied problems for swarms of robots (for example `Circle Formation`, `Uniform Circle Formation`, or `Gathering`) do not depend on a particular cardinality of the swarm. Accordingly, the solving algorithms are designed without assuming a known swarm size and must therefore be *scalable* to operate correctly regardless of the number of robots. Note that, for models lacking in complete visibility (e.g., opaque or myopic robots), even individual robots may be unaware of the total number of robots in the swarm during an LCM cycle: whilst it can restrict the computational power of the system, such an unawareness condition makes the system scalable and open to robot insertions or removals.

However, we cite some works which, due to the nature of the problem, have considered swarms of fixed and known cardinality [DP08; MV16; Oos+24; Shi+21].

1.3.6 Complexity metrics

Beyond the computability of a problem P under a given model M (i.e., if there exists an algorithm $\mathbb{A}$ which solves P under M), the subsequent question concerns the *complexity* (or efficiency) of $\mathbb{A}$. We list the main metrics usually considered to analyze the complexity of an algorithm $\mathbb{A}$:

- **Runtime**, i.e., the time needed to $\mathbb{A}$ to solve the problem. If the problem is perpetual, time will be infinite. Otherwise, the runtime is measured in epochs (or in rounds under `FSYNCH`), generally considering the asymptotical behavior w.r.t. n in the worst case. The objective is to find an algorithm for P with the minimum runtime [FMP23a; FPS24b; SVT17].

- **Amount of colors**, i.e., the size of the palette $\mathfrak{L}$ used by $\mathbb{A}$. Note that the optimal case $|\mathfrak{L}| = 1$ corresponds to the model $\mathcal{OBLOT}$. Even if the majority of the literature has assumed $O(1)$-size palettes, some work has considered other asymptotic sizes w.r.t. n. Examples can be found in [PS24] where $|\mathfrak{L}| = O(n^{1/2^x})$ for $x \in [1, O(\log \log n)]$, and in [Das+16; Yan+11] where robots are allowed to store a constant number of past snapshots of the swarm. In any case, the objective is to minimize the asymptotic value or the exact amount of $|\mathfrak{L}|$;

- **Maximum distance traveled** by a robot during the evolution of $\mathbb{A}$ [BM18; CSN15; Col+20]. The objective is to solve P by making robots travel the shortest (total) distance.

- **Performance SEC**[12], i.e., the smallest circular area enclosing all the points touched by the robots during the evolution of the algorithm. Indeed, the optimal case is when $\mathbb{A}$ makes all robots operate within the SEC of the initial configuration [FPS24a; FPS24b][13].

Indeed, the objective is to find the most efficient algorithm solving a given problem P, and thus to prove that it is impossible to solve the problem with fewer resources (time, colors, space). In that case, the found algorithm is said to be an optimal solution for P. Indeed, one can decide to optimize only one metric at a time, thus

[12] Smallest enclosing circle.
[13] In [FPS24a; FPS24b], this metric has been called *computational SEC*.

providing an optimal solution for P w.r.t. the chosen metric. It may happen that optimizing a metric causes the worsening of another one: in that case, a tradeoff of the complexities should be evaluated. An example is given in [PS24], where the authors present a time-color tradeoff algorithm for the `Uniform Circle Formation` problem: their algorithm solves the problem in $O(1)$ time using $O(\sqrt{n})$ colors, or in $O(\log \log n)$ time using $O(1)$ colors.

Note that, while runtime and colors are two well-considered metrics usually analyzed in literature, spatial metrics are only now gaining some popularity.

1.3.7 Fault tolerance

As the term suggests, fault tolerance is the ability of a system to continue operating correctly even in the presence of faults. For the LCM model, the objective is to determine if a problem P is solvable even in the presence of faults, to identify the level of fault tolerance for a given problem (e.g., type and maximum number of tolerated faults), and lastly, design distributed algorithms that solve the problem under such tolerance conditions.

The most studied problems in this context are `Flocking` [SYD08], and `Gathering`, for which different fault-tolerant solutions have been provided [AP06; Das+19; Déf+06; DPP20; Pat+19]. Not surprisingly, not all the problems admit a fault-tolerant solution under some settings. The `Gathering` problem is a typical example that proves this: if two robots crash at two different positions, the swarm will never gather at the same point; the same impossibility may happen if robots experience inaccurate movements. For these settings, theoretical research has proposed weaker versions of the problem, thus allowing a fault-prone swarm to solve the problem (see Figure 1.4). In the crash-prone setting, two versions have been considered: `Weak Gathering` [AP06; DPBT19] requires all correct robots to gather at the same position, while `Stand-up Indulgent Gathering` (`SUIG`) [Bra+24a; Bra+24b; BLT23; Bra+25] assumes robots can only crash at the same position and thus requires all robots to gather there. In the inaccurate-movement setting, `Gathering` has been relaxed to the `Convergence` problem, which requires all the robots to meet within a disk of diameter $\epsilon > 0$ in finite time [CP08].

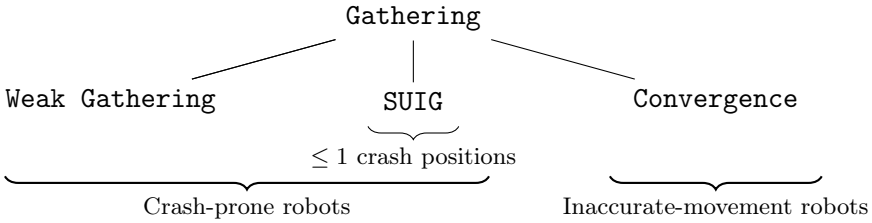
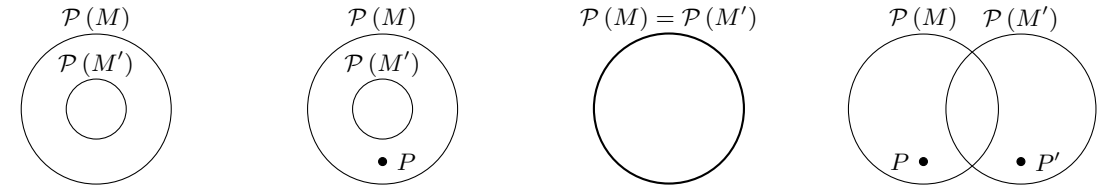


FIGURE 1.4: `Gathering` and some weaker versions for different fault-prone systems.

Self-stabilization

As stated in Section 1.2.9, faults can be transient (e.g., due to blackouts, particular environmental conditions) and/or fixable. In these cases, a desirable capability for robots is to recover from temporary failures and resume the solution of the problem, possibly performing the algorithm from scratch. This capability was introduced by Dijkstra in 1974 as *self-stabilization* [Dij74]. Specifically, for the LCM model, an algorithm is self-stabilizing if it solves the problem starting from any initial configuration [Ash+19; Déf+06; DPP20; DP12].



(A) $M \geq M'$ (equi-dominance) (B) $M > M'$ (dominance) (C) $M \equiv M'$ (equivalence) (D) $M \perp M'$ (orthogonality)

FIGURE 1.5: Computational power relations.

As stated in Section 1.3.2, a problem for a swarm $\mathcal{R}$ is defined to start from an initial configuration in which all robots have the same default color $\not\flat$ (*color uniformity clause*). Formally, it assumes that $\mathcal{C}_0(r) \in (\mathbb{R}^2 \times \{\not\flat\})$ for any $r \in \mathcal{R}$. Other hypotheses can be assumed on $\mathcal{C}_0$ in order to make the problem solvable. Indeed, the best case considered in literature is when the problem is solvable starting from an *arbitrary initial configuration* $\mathcal{C}_0$ only satisfying the color uniformity clause. In other words, robots' positions in $\mathcal{C}_0$ can be arbitrary. In this case, we say that the algorithm has self-stabilization capabilities after system blackouts, where the swarm restarts with the default color $\not\flat$ from their last positions before the system crash, or in case of non-rigid movements, where robots may be stopped during their motions. However, it has been proved for some problems to be unsolvable if starting from configurations with particular properties: e.g., presence of multiplicities or symmetries [SY99]. In these cases, research has characterized the condition on $\mathcal{C}_0$ that makes a problem solvable. E.g., it is often assumed that all the robots lie in distinct positions in $\mathcal{C}_0$ [Das+20; Das+15; YS10]. Indeed, this clause is essential in collision-intolerant models [FMP23a; FPS24a; FPS24b].

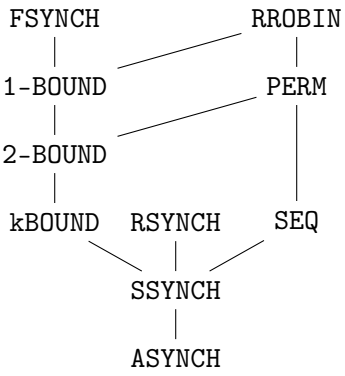
Note that the color uniformity clause is generally assumed to make the problem and the setting directly portable also in $\mathcal{OBLOT}$, where no further colors are available for robots (in fact, $\mathfrak{L} = \{\not\flat\}$). However, in the other models ($\mathcal{LUMI}$, $\mathcal{FCOM}$, and $\mathcal{FSTA}$), this clause can be removed to study the behavior of swarms starting from *arbitrarily colored* configurations. This investigation, for the sake of our knowledge, is still unexplored, and it might analyze the self-stabilization capability of swarms where robots may recover after a crash and start with the last set color before the crash. We emphasize that, in this work, we will always assume the color uniformity clause.

1.3.8 Computational power

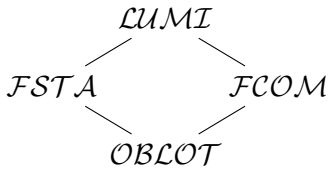
Given a robot model M, we denote with $\mathcal{P}(M)$ its *computational power*, namely, the set of problems solvable under M.

A main research branch on the LCM model has focused on studying and comparing the computational power of different LCM (sub-)models [Buc+21; Buc+25; Das+16; Das+24; DFN16; DFN15; FFS25; Fel+25c; Flo+23; FSW19]. The objective is to analyze how different combinations of features affect the ability of robots to solve problems. Given two models M and M', this analysis leads to a set-theoretic comparison of $\mathcal{P}(M)$ and $\mathcal{P}(M')$. Accordingly, the literature has defined these four relations (refer to Figure 1.5):

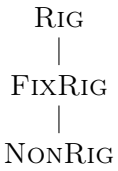
- (**equi-dominance**) M is *computationally not less powerful* than M', formally $M \geq M'$, if $\mathcal{P}(M) \supseteq \mathcal{P}(M')$, i.e., any problem solvable in M' is solvable in M;
- (**dominance**) M is *computationally more powerful* than M', formally $M > M'$, if $\mathcal{P}(M) \supset \mathcal{P}(M')$, i.e., any problem solvable in M' is solvable in M and there



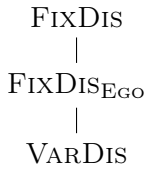
(A) Synchronization modes.



(B) Memory/communication setting.



(C) Movement rigidity.



(D) Orientation setting.

FIGURE 1.6: Equi-dominances for models only differing in one feature.

exists a problem P solvable in M but not solvable in M'. Equivalently, we say that M *computationally dominates* M';

- (**equivalence**) M is *computationally equivalent* to M', formally $M \equiv M'$, if $\mathcal{P}(M) = \mathcal{P}(M')$, i.e., any problem solvable in M (M', resp.) is solvable in M' (M, resp.);
- (**orthogonality**) M is *computationally orthogonal*[14] to M', formally $M \perp M'$, if $\mathcal{P}(M) \setminus \mathcal{P}(M') \neq \varnothing$ and $\mathcal{P}(M') \setminus \mathcal{P}(M) \neq \varnothing$, i.e., there exists a problem P (P', resp.) solvable in M (M', resp.) that is not solvable in M' (M, resp.).

In the remainder, we often omit the adverb "computationally" when referring to these relations. Indeed, the relations $<$ and $\leq$ are defined as the converse relations of, respectively, $>$ and $\geq$.

To identify which relation occurs between two models, the research has either designed some *simulators* proving the equivalence between models, or provided some *separators*, i.e., a problem solvable in a model but not in the other one, witnessing a dominance or orthogonality. Worthy to be mentioned is the simulator shown in [Das+16], which proves the equivalence between $\mathcal{LUMI}^{\texttt{ASYNCH}}$ and $\mathcal{LUMI}^{\texttt{SSYNCH}}$ for transparent robots. A more in-depth review of the existing results in this area will be given in Chapter 2.

1.3.9 Trivial equi-dominances

A model is defined by a combination of features (memory/communication settings, synchronization modes, visibility, orientation, rigidity, multiplicities, etc.). Indeed, for some pairs of models differing in only one feature, it is trivial to note the equi-dominance relation; this straightforwardly derives from the definition of the features.

For example, since SSYNCH is a special case of ASYNCH, we have that the same model under ASYNCH is equi-dominated by the same model under SSYNCH. RSYNCH is

[14] Or, equivalently, *computationally incomparable*.

a sub-mode of SSYNCH, incomparable with both kBOUND and SEQ. Again, since PERM sequentially activates all the robots in any permutation before restarting a new epoch, it is clear that between two consecutive activations of a robot r, any other robot can be activated at most 2 times; thus, PERM is a special case of 2-BOUND (which is in turn the sub-mode of kBOUND with $k = 2$). Instead, RROBIN is a sub-mode of 1-BOUND, since between two consecutive activations of a robot, any other robot is activated exactly once. The same property holds for FSYNCH, which is another sub-mode of 1-BOUND [DPBT19]. However, it is obvious that FSYNCH and RROBIN are incomparable.

According to the memory/communication settings, $\mathcal{OBLOT}$ is a special case of $\mathcal{FSTA}$, $\mathcal{FCOM}$, and $\mathcal{LUMI}$ by assuming that the palette only contains $\emptyset$. At the same time, $\mathcal{FSTA}$ ($\mathcal{FCOM}$, resp.) is a subcase of $\mathcal{LUMI}$ by assuming that the color lights are only internally (externally, resp.) perceivable.

According to robots' movements, Rig is a special case of FixRig, where the maximum distance allowed to travel is $\gamma = \infty$; in other terms, robots will never be stopped before reaching their destination. In turn, FixRig is a special case of NonRig by assuming that $\epsilon = \gamma$, and that the adversary always stops a robot after having traveled exactly a ϵ distance.

According to the robots' orientation, FixDis$_{\text{Ego}}$ is a special case of VarDis, by assuming that the axes direction, and the unit distance of $\Xi_t(r)$ never change with t. A little less evident is the equi-dominance between FixDis$_{\text{Ego}}$ and FixDis. The following lemma proves it.

Lemma 1. *If a problem P is solved under a model assuming FixDis$_{\text{Ego}}$, then it can be solved under the same model but assuming FixDis.*

Proof. Let $\mathbb{A}$ be an algorithm solving P under M assuming FixDis$_{ego}$. Thus, a swarm $\mathcal{R}$ solves P by executing $\mathbb{A}$. Consider the same swarm under M assuming FixDis. Each activated robot can translate the taken snapshot so that its current position becomes the origin of the coordinate system. Thus, the robot can execute $\mathbb{A}$, and solve the problem. □

The Hasse diagrams in Figure 1.6 depict the just-mentioned trivial equi-dominance relations among models. These relations will be used as a basis to prove much less evident relations in Part I. Other relations may be derived by the transitivity property enjoyed by $>$, $\geq$, and $\equiv$.

Part I

Computational Power

Chapter 2

Transparency vs. Opaqueness

2.1 Introduction

Within the field of distributed computing by mobile robots, a wide effort has been dedicated to the study of the *computational power* of a given model (i.e., the set of problems it can solve) and outlining the hierarchical relations (dominance, equivalence, or orthogonality) among different models [Buc+21; Das+16; Das+24; DFN16; DFN15; D'E+18; Flo+23; FSW19]. Notably, research has inspected how the power of a swarm is affected by different robot features, with a deep focus on memory/communication settings and synchronization modes. According to the memory/communication setting, the four main base models (i.e., $\mathcal{OBLOT}$, $\mathcal{FSTA}$, $\mathcal{FCOM}$, and $\mathcal{LUMI}$) have been investigated. According to synchronization, each base model has been considered under the three main modes `FSYNCH`, `SSYNCH`, and `ASYNCH`.

In [Buc+21; Das+16; Flo+23; FSW19], the authors compare the computational power of the 12 models defined as X^Y with $X \in \{\mathcal{OBLOT}, \mathcal{FSTA}, \mathcal{FCOM}, \mathcal{LUMI}\}$ and $Y \in \{\mathtt{F}, \mathtt{S}, \mathtt{A}\}$ (standing for `FSYNCH`, `SSYNCH`, `ASYNCH`, resp.), and provide the comprehensive relation map among them. In all these models, robots are deployed on the Euclidean plane, are *collision-tolerant*, and have *complete visibility* (namely, they are transparent and non-myopic). Among the results, we cite two noteworthy equivalences: $\mathcal{LUMI}^{\mathtt{S}} \equiv \mathcal{LUMI}^{\mathtt{A}}$ (proved by the simulator explained in [Das+16]) and $\mathcal{LUMI}^{\mathtt{F}} \equiv \mathcal{FCOM}^{\mathtt{F}}$ (proved by the simulator explained in [FSW19]). The simulator presented in [Das+16] is a meta-algorithm that, given in input any algorithm $\mathbb{A}$ solving a problem P under $\mathcal{OBLOT}^{\mathtt{S}}$, it simulates the behavior of $\mathbb{A}$ under the $\mathcal{LUMI}^{\mathtt{A}}$ model, thus showing that $\mathcal{LUMI}^{\mathtt{A}} \geq \mathcal{OBLOT}^{\mathtt{S}}$. This simulation is achieved through the usage of 5 light colors, which allow the robots to synchronize their LCM cycles. Indeed, the same simulator can be exploited to solve a problem $P \in \mathcal{P}\left(\mathcal{LUMI}^{\mathtt{S}}\right)$ in the `ASYNCH` mode: the 5 synchronization colors are used in combination with the original palette of the algorithm solving P. Hence, the equivalence $\mathcal{LUMI}^{\mathtt{S}} \equiv \mathcal{LUMI}^{\mathtt{A}}$. This equivalence leads to an interesting fact: since it has been shown that $X^{\mathtt{S}} > X^{\mathtt{A}}$ for all the other three base models $X \in \{\mathcal{OBLOT}, \mathcal{FSTA}, \mathcal{FCOM}\}$, *both memory and communication are needed* to simulate a `SSYNCH` scheduler under `ASYNCH`.

The simulator for $\mathcal{LUMI}^{\mathtt{F}} \equiv \mathcal{FCOM}^{\mathtt{F}}$ [FSW19], instead, makes $\mathcal{FCOM}$ robots emulate each round of luminous robots into two rounds, by using the external lights of the robots are mirrors for the light color of the other robots. In the first round, robots stay still and use their external lights to copy the value the light color of the two adjacent robots in the configuration; in the second round, each robot can detect its own color by looking at the copied colors of the two adjacent robots, and thus compute the next action by executing the original $\mathcal{LUMI}$ algorithm. The `FSYNCH` mode is necessary to ensure the synchronization of the light updates. Even in this case,

[0]The content of this chapter refers to the works [Fel+25c; Fel+24].

the resulting equivalence leads to a surprising fact: under FSYNCH, *it is more powerful to communicate instead of remembering*. Formally, $\mathcal{FCOM}^{\mathtt{F}} > \mathcal{FSTA}^{\mathtt{F}}$. Instead, the two base models $\mathcal{FCOM}$ and $\mathcal{FSTA}$ turn out to be orthogonal under SSYNCH and ASYNCH.

In [DFN16; DFN15; D'E+18], the same investigation is performed on robots moving on graphs; operating in discrete spaces, rather than continuous ones, has been shown to significantly impact their computational power, showing, for example, that $\mathcal{LUMI}^{\mathtt{A}}$ and $\mathcal{OBLOT}^{\mathtt{F}}$ are orthogonal on graphs [DFN15], differently from what happens in the Euclidean plane (i.e., $\mathcal{LUMI}^{\mathtt{A}} > \mathcal{OBLOT}^{\mathtt{F}}$ [Das+16]).

In [Buc+25], the research on continuous models is extended to include the RSYNCH mode, in addition to the other synchronization modes previously studied. We remind that RSYNCH is a special sub-mode of SSYNCH where no robot can be activated for two consecutive rounds; this mode formalizes *energy-constrained* robots, i.e., requiring an idle round to restore energy for a new LCM cycle. Specifically, the authors show both the relations among the base models under RSYNCH, and the relations between energy-constrained and unrestricted models.

Recently, in [Das+24], a comparative analysis of models with myopic robots is proposed, focusing on how *short-sightedness* affects their computational power.

New research directions about the computational landscape of swarm models have been recently outlined in [Flo25].

Contribution

The study conducted in this chapter is inspired by [Buc+21; Das+16; Flo+23; FSW19], where the authors exhibit the complete taxonomy of the computational power of the 12 models of *transparent* robots that can freely move on the Euclidean plane. Such models differ for the synchronization mode (FSYNCH, SSYNCH, and ASYNCH), and for the (im)possibility of memorizing and communicating ($\mathcal{OBLOT}$, $\mathcal{FSTA}$, $\mathcal{FCOM}$, $\mathcal{LUMI}$). However, the models in [Buc+21; Das+16; Flo+23; FSW19] deal with *(i) collision-tolerant* robots (i.e., they can occupy the same position at the same time), *(ii) transparent* and *non-myopic* robots (thus enjoying complete and unlimited visibility).

In this chapter, we conduct a cross-model comparison among the 12 transparent models considered in [Buc+21; Das+16; Flo+23; FSW19] and the same 12 models in their *opaque* version, i.e., featuring robots that cannot see through collinearities. The objective is to examine how opaqueness affects the power of robots in contrast with transparency.

Opaqueness introduces a remarkable difficulty in the design of correct algorithms to solve some classic problems [Bos+21; FMP23a; FPS24a; PS24]. In fact, the *obstructed visibility* leads to critical issues to be addressed in the algorithmic strategies: robots may not be aware of the swarm cardinality as well as of moving robots in the ASYNCH mode, robots may not know the complete topology of the current configuration, robots may (incorrectly) compute the next action based on partial information. Clearly, these issues may also be experienced by myopic robots; their computational power has recently been studied in [Das+24], where the authors provide an initial analysis of the computational impact of *short-sightedness*. However, although many issues hold for both opaque and myopic robots, *ad hoc* techniques are separately needed to deal with these two different visibility limitations [Bra+24a; Kam+19; KT24; SVT21; Sha+16].

For the reasons explained in Section 1.2.8, besides the *opaqueness* feature, all our 24 models (thus, both the transparent and the opaque ones) differ from those presented in [Buc+21; Das+16; Flo+23; FSW19] in that robots are *collision-intolerant*. According to this, some problems introduced in [Buc+21; Flo+23; FSW19] cannot be used as

separators under our comparison since they assume robot multiplicities, and a new study with specific separators is thus required. We remark that the collision-intolerance assumption not only avoids theoretical inconsistencies in opaque models, but it also enables the development of theoretical models for more realistic systems, perfectly aligned with the spirit of assuming robot opaqueness.

All the 24 models under study will be presented in detail in Section 2.2. In Section 2.3, we expose a preliminary study of the relations between transparent and opaque models. Intuitively, a transparent model seems to computationally dominate the corresponding opaque model. Here, we formally prove this dominance: endowing a model with transparency increases its computational power, allowing the model to solve more problems. In addition, this result enables us to show that constant-size lights are *not always* sufficient to compensate for robots' obstructed visibility. In Section 2.4, we present five separator problems, showing the majority of the hierarchical relations among models of *collision-intolerant opaque robots*, thus yielding a first overview of their computational taxonomy. Such separators also lead to defining the cross-model relations among transparent and opaque models, providing us with an analysis of the impact of obstructed visibility on the computational power of robots. We then establish a sixth witness problem, pointing out a peculiar issue occurring under `ASYNCH` and opaqueness. Finally, we focus on model relations still to be proved, suggesting some lines of attack. In Section 2.5, we summarize our results showing the *almost* complete hierarchy of the 24 models under study, while, in Section 2.6, we draw some concluding remarks and address possible research outlooks.

2.2 Models

We compare 24 robot models that differ in some features. We here introduce all the *fixed features* that such models share[1] and the *variable features* under study.

Fixed features. All the 24 models under study share the core features presented in Section 1.2.1. Moreover, we fix some other features. In particular, given a swarm $\mathcal{R}$, we always assume:

- robots are completely disoriented, thus, they do not have any agreement on their local coordinate systems. Formally, $\Xi_t(r)$ and $\Xi_t(r')$ may differ in any component for any $r \neq r' \in \mathcal{R}$;
- the *variable disorientation* setting (VarDis), i.e., the local coordinate system $\Xi_t(r)$ of any robot r at time t may differ from $\Xi_{t'}(r)$ for any $t \neq t'$;
- movements are *rigid* (Rig), thus robots always reach their computed destination during a LCM cycle;
- robots are *collision intolerant*, thus they start from multiplicity-free configurations and they have to avoid collisions. We refer the reader to the definition of collisions in Section 1.2.7.

Variable features. We consider four features: visibility, memory, communication, and synchronization. By varying them, we obtain the claimed 24 models.

- (*visibility)* We consider *transparent* and *opaque* swarms. In the former case, all robots are assumed to be transparent so that they always enjoy *complete visibility* of the entire swarm in their Look step. In the latter case, all robots

[1] Note that, in [Buc+21; Das+16; Flo+23; FSW19], models with the chirality agreement and non-rigid movements are also considered.

are assumed to be opaque, so that for three collinear robots p, q, r, the endpoint robots p, r cannot see each other (we call this condition *obstructed visibility*). In both cases, robots are non-myopic, i.e., their visibility does not depend on the mutual distance.

- (*memory* and *communication*) We consider the four base models $\mathcal{OBLOT}$, $\mathcal{FSTA}$, $\mathcal{FCOM}$, and $\mathcal{LUMI}$.
- *(synchronization modes)* We consider the three modes `FSYNCH` (fully synchronous), `SSYNCH` (semi-synchronous), and `ASYNCH` (asynchronous).

Notation for the 24 models. Let us set $\mathcal{X} = \{\mathcal{OBLOT}, \mathcal{FSTA}, \mathcal{FCOM}, \mathcal{LUMI}\}$ and $\mathcal{Y} = \{\mathtt{F}, \mathtt{S}, \mathtt{A}\}$ (standing for `FSYNCH`, `SSYNCH`, `ASYNCH`, resp.). For $X \in \mathcal{X}$ and $Y \in \mathcal{Y}$, we let $X_{\mathcal{T}}^{Y}$ (resp., $X_{\mathcal{O}}^{Y}$) denote the model for transparent (resp., opaque) robots that possess all the above fixed features, and that has X as memory-communication setting and Y as synchronization mode. Let $\mathcal{M}_{\mathcal{T}} = \{X_{\mathcal{T}}^{Y} \text{ s.t. } X \in \mathcal{X}, Y \in \mathcal{Y}\}$ and $\mathcal{M}_{\mathcal{O}} = \{X_{\mathcal{O}}^{Y} \text{ s.t. } X \in \mathcal{X}, Y \in \mathcal{Y}\}$: we collectively refer to these two classes of models as the *transparent* and *opaque framework*. This chapter analyzes the computational power of the 24 models contained in the set $\mathcal{M} = \mathcal{M}_{\mathcal{T}} \cup \mathcal{M}_{\mathcal{O}}$.

For the sake of conciseness, we will compactly denote by $X_{V}^{Y_1,\dots,Y_h}$ the set of models $\{X_{V}^{Y_1}, \dots, X_{V}^{Y_h}\}$, where $X \in \mathcal{X}$, $Y_j \in \mathcal{Y}$, and $V \in \{\mathcal{T}, \mathcal{O}\}$. Moreover, we will sometimes use X_{*}^{Y} as a shorthand for the following notation: X_{V}^{Y}, where V can be both $\mathcal{T}$ and $\mathcal{O}$.

2.3 Preliminary relations

In this section, we begin by establishing some basic facts concerning the computational power of our robot models.

2.3.1 Model-based equi-dominances

The following theorem extends the relations stated in Section 1.3.9.

Theorem 1. *Given a visibility setting $V \in \{\mathcal{T}, \mathcal{O}\}$ and synchronization modes $Y_1 = \mathtt{F}$, $Y_2 = \mathtt{S}$, $Y_3 = \mathtt{A}$, for each $i \leq j \leq k$, we have*

$$\mathcal{LUMI}_{V}^{Y_i} \geq \mathcal{FSTA}_{V}^{Y_j} \geq \mathcal{OBLOT}_{V}^{Y_k}, \quad \mathcal{LUMI}_{V}^{Y_i} \geq \mathcal{FCOM}_{V}^{Y_j} \geq \mathcal{OBLOT}_{V}^{Y_k}.$$

Proof. By model definition, for any $Y \in \mathcal{Y}$ and $V \in \{\mathcal{T}, \mathcal{O}\}$, we know that $\mathcal{LUMI}_{V}^{Y} \geq \mathcal{FSTA}_{V}^{Y} \geq \mathcal{OBLOT}_{V}^{Y}$ and $\mathcal{LUMI}_{V}^{Y} \geq \mathcal{FCOM}_{V}^{Y} \geq \mathcal{OBLOT}_{V}^{Y}$. Moreover, for any $X \in \mathcal{X}$ and $V \in \{\mathcal{T}, \mathcal{O}\}$, we have that $X_{V}^{\mathtt{F}} \geq X_{V}^{\mathtt{S}} \geq X_{V}^{\mathtt{A}}$. The other relations follow by transitivity. □

2.3.2 Dominance of transparency

Theorem 2. *For any $X \in \mathcal{X}$ and $Y \in \mathcal{Y}$, if $P \in \mathcal{P}\left(X_{\mathcal{O}}^{Y}\right)$ then $P \in \mathcal{P}\left(X_{\mathcal{T}}^{Y}\right)$.*

Proof. Let $\mathbb{A}_{\mathcal{O}}$ be an algorithm solving a problem P under $X_{\mathcal{O}}^{Y}$. We can easily construct an algorithm $\mathbb{A}_{\mathcal{T}}$ solving P under $X_{\mathcal{T}}^{Y}$, which works as follows. After taking its snapshot $\sigma_{\mathcal{T}}$ (where clearly all robots in the swarm are visible), any transparent robot r first computes the reduced snapshot $\sigma_{\mathcal{O}}$, obtained from $\sigma_{\mathcal{T}}$ by removing those robots which would be geometrically not visible to r due to opaqueness, and then runs $\mathbb{A}_{\mathcal{O}}(\sigma_{\mathcal{O}})$. Clearly, $\mathbb{A}_{\mathcal{T}}$ perfectly simulates $\mathbb{A}_{\mathcal{O}}$, thus correctly solving P for transparent robots. □

Corollary 1. *For any $X \in \mathcal{X}$ and $Y \in \mathcal{Y}$,*

$$X_{\mathcal{T}}^{Y} \geq X_{\mathcal{O}}^{Y}.$$

Corollary 2. *For any model $M \in \mathcal{M}$, $M \geq \mathcal{OBLOT}_{\mathcal{O}}^{\mathtt{A}}$.*

Proof. It follows from Theorem 1 and Corollary 1. □

Problem 1 (`Line-Stretch`)**.** Let us consider an initial configuration where $n > 3$ robots are equally spaced along the same line, say γ. Let d be the distance between two adjacent robots. The problem asks the endpoint robots to move away from their adjacent robot and stop at a distance $d + \frac{d}{n}$ from them. The endpoint robots are allowed to travel only along γ. The other robots must stay still. See Figure 2.1.

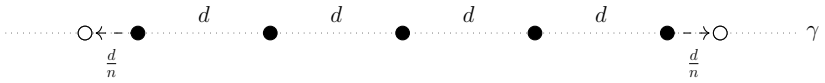


FIGURE 2.1: `Line-Stretch` for $n = 5$ robots.

Lemma 2. *`Line-Stretch`* $\in \mathcal{P}\left(\mathcal{OBLOT}_{\mathcal{T}}^{\mathtt{A}}\right)$.

Proof. The problem is solved under the weakest model of the transparent framework. In fact, the endpoint robots can compute and head to their destination since they can count all the robots and spot the distance d among two adjacent robots. The final configuration is stable (i.e., no robot will move anymore). □

Lemma 3. *`Line-Stretch`* $\notin \mathcal{P}\left(\mathcal{LUMI}_{\mathcal{O}}^{\mathtt{F}}\right)$.

Proof. The problem cannot be solved under the strongest model of the opaque framework. By contradiction, suppose an algorithm $\mathbb{A}$ solves the problem under $\mathcal{LUMI}_{\mathcal{O}}^{\mathtt{F}}$ using the constant-size palette $\mathfrak{L}$. Consider an endpoint robot e that has not moved yet, and let a be its adjacent robot. Suppose e always has a local coordinate system such that the positions of e and a are respectively $(0, 0)$ and $(1, 0)$, regardless of the problem instance and the activation time. Thus, the snapshot taken by e will always have the form $\langle((0, 0), c_e), ((1, 0), c_a)\rangle$ with $c_e, c_a \in \mathfrak{L}$. Accordingly, the colors c_e, c_a are the only way for the swarm to code information needed for e to compute n. Let t be the time where e moves to its destination point at a distance $1 + \frac{1}{n}$ from a (being $d = 1$ according to the coordinate system of e). Let σ be the snapshot taken by e at time t, which has been used by $\mathbb{A}$ to compute the destination point. So, $\mathbb{A}$ must implement a function $f : \mathfrak{L}^2 \to \mathbb{N}$ mapping the light combination in a snapshot (i.e., c_e, c_a) with the correct natural number n. However, since $\mathbb{A}$ solves the problem for any size of the swarm, f must be surjective. Contradiction achieved. □

Theorem 3. *For any $X \in \mathcal{X}$ and $Y \in \mathcal{Y}$, we have*

$$X_{\mathcal{T}}^{Y} > X_{\mathcal{O}}^{Y}.$$

Proof. By Lemma 2 and Lemma 3, respectively, we get that `Line-Stretch` $\in \mathcal{P}\left(X_{\mathcal{T}}^{Y}\right)$ and `Line-Stretch` $\notin \mathcal{P}\left(X_{\mathcal{O}}^{Y}\right)$. Thus, the result follows from Corollary 1. □

Theorem 4. *Let P be a problem solved by an algorithm $\mathbb{A}$ under $X_{\mathcal{T}}^{Y}$, which guarantees that collinearities never occur along any solving evolution. Then, P can be solved under $X_{\mathcal{O}}^{Y}$.*

Proof. Running $\mathbb{A}$ under $X_{\mathcal{O}}^{Y}$ perfectly replicates the evolution of $\mathbb{A}$ by transparent robots. This is due to the fact that, by hypothesis, the snapshots taken in both the opaque and transparent frameworks are identical. □

2.3.3 Non-emptiness and non-disjointness

Problem 2 (Iso-Equi)**.** Let three robots initially form a non-degenerate isosceles triangle. The problem requires the robots to form an equilateral triangle and terminate.

Lemma 4. *Iso-Equi* $\in \mathcal{P}\left(\mathcal{OBLOT}_{\mathcal{O}}^{\mathtt{A}}\right)$.

Proof. We provide this simple algorithm: firstly, the three robots can check whether they form an equilateral triangle and, in case, stand still. Otherwise, the problem can be easily solved by making the apex robot move along the axis and stop to form an equilateral triangle with the base robots (which remain still). □

As a consequence of Lemma 4 and Corollary 2, we obtain that Iso-Equi $\in \mathcal{P}(M)$ for each $M \in \mathcal{M}$. This proves the following properties of the computational power of the models under study in this chapter:

Corollary 3 (Non-emptiness)**.** *For any model $M \in \mathcal{M}$, we have $\mathcal{P}(M) \neq \varnothing$.*

Corollary 4 (Non-disjointness)**.** *For any two models $M_1, M_2 \in \mathcal{M}$, we have $\mathcal{P}(M_1) \cap \mathcal{P}(M_2) \neq \varnothing$.*

2.3.4 Collinearities vs. awareness

A natural question for the opaque framework is how collinearities may affect the computational power of robots. The following observations highlight that a non-obstructed snapshot of an opaque swarm (i.e., taken from a collinearity-less configuration) may be neither necessary nor sufficient for a robot to correctly perceive the whole current configuration.

Observation 2. Let r be a robot under a model in $\mathcal{M}_{\mathcal{O}}$, activated in a configuration $\mathcal{C}$. Then, there are cases in which r can correctly perceive the whole $\mathcal{C}$ even if $\mathcal{C}$ contains collinear robots.

This observation can be understood by considering the following example. Let us define the perpetual problem for an $\mathcal{OBLOT}_{\mathcal{O}}^{\mathtt{F}}$ swarm of three robots, starting from an equally spaced line configuration. The problem asks the endpoint robots to move along the line and double the previous distance from the center robot at every round. Even if the endpoint robots cannot see each other, at each activation, they know the exact position of the hidden robot. This specific problem shows that the absence of collinearities in a configuration *is not always necessary* for a robot to reconstruct the whole configuration.

On the other hand:

Observation 3. Let r be a robot under a model in $\mathcal{M}_{\mathcal{O}}$, activated in a collinearity-less configuration $\mathcal{C}$. This latter property of $\mathcal{C}$ does not guarantee that r is aware of enjoying a non-obstructed snapshot of the whole swarm.

To understand this observation, let us consider the problem defined for an $\mathcal{OBLOT}_{\mathcal{O}}^{\mathtt{F}}$ swarm, which can start from two different configurations: *(i)* two robots or *(ii)* three equally spaced and aligned robots. From *(i)*, the problem asks the swarm to stand still. From *(ii)*, the problem asks the endpoint robots to double their distance from

the center robot at every round, always moving along the original alignment line. This problem is unsolvable in $\mathcal{OBLOT}^{\mathtt{F}}_{\mathcal{O}}$, since each activated endpoint robot has no means to understand whether or not the current configuration describes a two or three-robot system. In general, we can state that the absence of collinearities in a configuration (as in scenario *(i)*) *is not always sufficient* for a robot to reconstruct the whole configuration.

2.3.5 Transparency vs. opaqueness

Lemma 5. *Let $A_{\mathcal{O}} \in \mathcal{M}_{\mathcal{O}}$ and $B_{\mathcal{T}} \in \mathcal{M}_{\mathcal{T}}$. Then, neither $A_{\mathcal{O}} \equiv B_{\mathcal{T}}$ nor $A_{\mathcal{O}} > B_{\mathcal{T}}$ can hold.*

Proof. This trivially follows from Lemmas 2 and 3, showing the separator problem `Line-Stretch` is solved by any transparent model but by no opaque model. □

Theorem 5. *Let $A_{\mathcal{T}}, B_{\mathcal{T}} \in \mathcal{M}_{\mathcal{T}}$. If $A_{\mathcal{T}} \geq B_{\mathcal{T}}$ (i.e., either $A_{\mathcal{T}} > B_{\mathcal{T}}$ or $A_{\mathcal{T}} \equiv B_{\mathcal{T}}$), then $A_{\mathcal{T}} > B_{\mathcal{O}}$.*

Proof. This trivially holds since $B_{\mathcal{T}} > B_{\mathcal{O}}$ by Theorem 3. □

Theorem 6. *Let $A_{\mathcal{T}}, B_{\mathcal{T}} \in \mathcal{M}_{\mathcal{T}}$. If $A_{\mathcal{T}} \geq B_{\mathcal{T}}$ and $A_{\mathcal{O}} \geq B_{\mathcal{O}}$, and there exists a problem $P \in \mathcal{P}(A_{\mathcal{O}}) \setminus \mathcal{P}(B_{\mathcal{T}})$, then we have:*

$$A_{\mathcal{T}} > B_{\mathcal{T}}, \quad A_{\mathcal{T}} > B_{\mathcal{O}},$$
$$A_{\mathcal{O}} \perp B_{\mathcal{T}}, \quad A_{\mathcal{O}} > B_{\mathcal{O}}.$$

Proof. The claimed relations are depicted in Figure 2.2a. The dominance relations $A_{\mathcal{T}} > B_{\mathcal{T}}$ and $A_{\mathcal{O}} > B_{\mathcal{O}}$ straightforwardly derives from the fact that $P \in \mathcal{P}(A_{\mathcal{T}}) \setminus \mathcal{P}(B_{\mathcal{T}})$ and that $P \in \mathcal{P}(A_{\mathcal{O}}) \setminus \mathcal{P}(B_{\mathcal{O}})$ by Theorem 3. The dominance relation $A_{\mathcal{T}} > B_{\mathcal{O}}$ holds by Theorem 5. The orthogonality relation $A_{\mathcal{O}} \perp B_{\mathcal{T}}$ holds true since $P \in \mathcal{P}(A_{\mathcal{O}}) \setminus \mathcal{P}(B_{\mathcal{T}})$ and `Line-Stretch` $\in \mathcal{P}(B_{\mathcal{T}}) \setminus \mathcal{P}(A_{\mathcal{O}})$ by Lemmas 2 and 3. □

Theorem 7. *Let $A_{\mathcal{T}}, B_{\mathcal{T}} \in \mathcal{M}_{\mathcal{T}}$. If there exists a problem $P \in \mathcal{P}(A_{\mathcal{O}}) \setminus \mathcal{P}(B_{\mathcal{T}})$ and a problem $Q \in \mathcal{P}(B_{\mathcal{O}}) \setminus \mathcal{P}(A_{\mathcal{T}})$, then we have that $A_* \perp B_*$, i.e.,*

$$A_{\mathcal{T}} \perp B_{\mathcal{T}}, \quad A_{\mathcal{T}} \perp B_{\mathcal{O}}, \quad A_{\mathcal{O}} \perp B_{\mathcal{T}}, \quad A_{\mathcal{O}} \perp B_{\mathcal{O}}.$$

Proof. The claimed relations are depicted in Figure 2.2b. By Theorem 3, we get that $P \in \mathcal{P}(A_{\mathcal{T}}) \setminus \mathcal{P}(B_{\mathcal{T}})$, $P \in \mathcal{P}(A_{\mathcal{O}}) \setminus \mathcal{P}(B_{\mathcal{O}})$ and $P \in \mathcal{P}(A_{\mathcal{T}}) \setminus \mathcal{P}(B_{\mathcal{O}})$. The corresponding relations hold for the problem Q, so $Q \in \mathcal{P}(B_{\mathcal{T}}) \setminus \mathcal{P}(A_{\mathcal{T}})$, $Q \in \mathcal{P}(B_{\mathcal{O}}) \setminus \mathcal{P}(A_{\mathcal{O}})$ and $Q \in \mathcal{P}(B_{\mathcal{T}}) \setminus \mathcal{P}(A_{\mathcal{O}})$. By combining these results, the four claimed orthogonality relations follow. □

2.4 Taxonomy of the 24 models

Firstly, we cite two main computational equivalences proved in [Das+16; FSW19] for transparent robots. Namely, the authors proved such equivalences through the construction of two simulators which allow to emulate a $\mathcal{LUMI}^{\mathtt{S}}_{\mathcal{T}}$ ($\mathcal{LUMI}^{\mathtt{F}}_{\mathcal{T}}$, resp.) algorithm in the $\mathcal{LUMI}^{\mathtt{A}}_{\mathcal{T}}$ ($\mathcal{FCOM}^{\mathtt{F}}_{\mathcal{T}}$, resp.) model. Since such simulators work regardless of the presence of collisions, they can be used to prove the same equivalences for our collision-intolerant models as well.

Theorem 8 ([Das+16])**.** $\mathcal{LUMI}^{\mathtt{S}}_{\mathcal{T}} \equiv \mathcal{LUMI}^{\mathtt{A}}_{\mathcal{T}}$.

(A) Relation diagram in Theorem 6.

(B) Relation diagram in Theorem 7.

FIGURE 2.2: Relation diagrams: a straight (dashed, resp.) line denotes a dominance (orthogonality, resp.) relation.

Theorem 9 ([FSW19]). $\mathcal{LUMI}^{\mathtt{F}}_{\mathcal{T}} \equiv \mathcal{FCOM}^{\mathtt{F}}_{\mathcal{T}}$.

We now present six separator problems to prove some dominance ($>$) and orthogonality ($\perp$) relations among the 24 models in $\mathcal{M}$. For each separator problem, we identify the *most limited* models under which the problem can be solved, and the *most powerful* models under which the problem cannot be solved.

The first separator problem we introduce, `Triangle Round-Trip`, is a restriction of a problem devised in [Buc+21] for the transparent framework. Thanks to Theorem 2 and Theorem 4, we can borrow it to prove that some hierarchical relations hold in our opaque framework as well. However, other problems in [Buc+21; Flo+23; FSW19] are not compliant with our collision-intolerant models. Thus, we present other four specific problems that fit our assumptions. Eventually, we introduce an *ad hoc* problem, `Pseudo-Polygon`, which raises a critical issue occurring under obstructed visibility.

2.4.1 Weakness of $\mathcal{OBLOT}$

Problem 3 (`Triangle Round-Trip`). Let $\mathcal{C}$ be a configuration with three robots, two of which lie on the vertices of an equilateral triangle, while the third robot lies at the triangle center. Let a be the empty triangle vertex. From $\mathcal{C}$, the robot in the center has to move to a, forming the new configuration $\mathcal{C}'$. Then, robots have to form $\mathcal{C}$ again, where a is again the empty vertex. Afterwards, the robots must stay still. See Table 2.1.

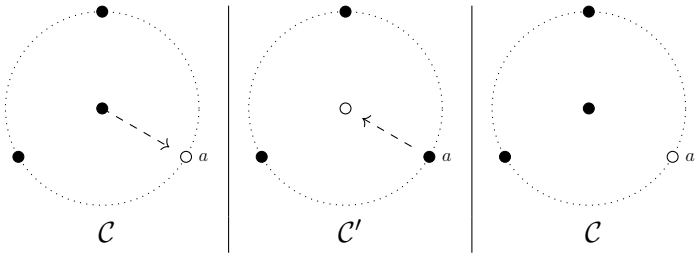


TABLE 2.1: Configurations in `Triangle Round-Trip`.

`Triangle Round-Trip` is a sub-case of the problem `N-gon Round-Trip` defined in [Buc+21], Definition 1.

Lemma 6. *`Triangle Round-Trip`* $\notin \mathcal{P}\left(\mathcal{OBLOT}^{\mathtt{F}}_{\mathcal{T}}\right)$.

Proof. The problem cannot be solved in $\mathcal{OBLOT}^{\mathtt{F}}_{\mathcal{T}}$ (see Lemma 3 in [Buc+21]). In fact, using oblivious and silent robots, there is no way to identify the former empty vertex a due to the full symmetry of $\mathcal{C}'$. □

Lemma 7. *Triangle Round-Trip* $\in \left(\mathcal{P}\left(\mathcal{FSTA}^{\mathtt{A}}_{\mathcal{O}}\right) \cap \mathcal{P}\left(\mathcal{FCOM}^{\mathtt{A}}_{\mathcal{O}}\right)\right)$.

Proof. The problem has been solved in $\mathcal{FSTA}^{\mathtt{A}}_{\mathcal{T}}$ and $\mathcal{FCOM}^{\mathtt{A}}_{\mathcal{T}}$ by Lemmas 4 and 5 in [Buc+21], where solving algorithms never create collinearities or collisions. This enables us to apply Theorem 4 and obtain that Triangle Round-Trip can be solved both in $\mathcal{FSTA}^{\mathtt{A}}_{\mathcal{O}}$ and $\mathcal{FCOM}^{\mathtt{A}}_{\mathcal{O}}$ as well. □

Theorem 10. *For any* $X \in \{\mathcal{FSTA}, \mathcal{FCOM}, \mathcal{LUMI}\}$ *and given the modes* $Y_1 = \mathtt{F}$, $Y_2 = \mathtt{S}$, $Y_3 = \mathtt{A}$, *we have the following relations for any* $1 \leq i \leq j \leq 3$*:*

$$X^{Y_i}_{\mathcal{T}} > \mathcal{OBLOT}^{Y_j}_{\mathcal{T}}, \quad X^{Y_i}_{\mathcal{T}} > \mathcal{OBLOT}^{Y_j}_{\mathcal{O}},$$
$$X^{Y_i}_{\mathcal{O}} \perp \mathcal{OBLOT}^{Y_j}_{\mathcal{T}}, \quad X^{Y_i}_{\mathcal{O}} > \mathcal{OBLOT}^{Y_j}_{\mathcal{O}}.$$

Proof. Let us consider any $X \in \{\mathcal{FSTA}, \mathcal{FCOM}, \mathcal{LUMI}\}$. By Theorem 1, we know that $X^{Y_i}_{\mathcal{T}} \geq \mathcal{OBLOT}^{Y_j}_{\mathcal{T}}$ and $X^{Y_i}_{\mathcal{O}} \geq \mathcal{OBLOT}^{Y_j}_{\mathcal{O}}$. Moreover, we have that Triangle Round-Trip cannot be solved under $\mathcal{OBLOT}^{\mathtt{F},\mathtt{S},\mathtt{A}}_{\mathcal{T}}$ (by Lemma 6) but it can be solved under $X^{\mathtt{A},\mathtt{S},\mathtt{F}}_{\mathcal{O}}$ (by Lemma 7). Thus, the claimed relations follow from Theorem 6. □

2.4.2 Orthogonality between $\mathcal{FSTA}$ and $\mathcal{FCOM}$

Problem 4 (Flip-Flop-Flip)**.** Let p, q, r be three robots forming a strictly isosceles triangle such that $dist(p, r) = dist(q, r)$. Let γ be the perpendicular bisector to the line segment $\overline{pq}$ passing through the point $b \in \overline{pq}$. Let γ' (γ'', resp.) be the semi-line of γ starting from b and containing (not containing, resp.) r. The problem requires r to perpetually perform three subsequent actions (see Table 2.2), in an infinite loop: *(i)* r must reach its symmetric position r' on γ''; *(ii)* r must reach a different point r'' on γ'' such that $dist(r'', b) > dist(r', b)$; *(iii)* r must reach its symmetric position r''' on γ'. Along with these actions, r can never leave γ and can never stop in a configuration where robots p, q, r form an equilateral triangle. The robots p, q must always stay still.

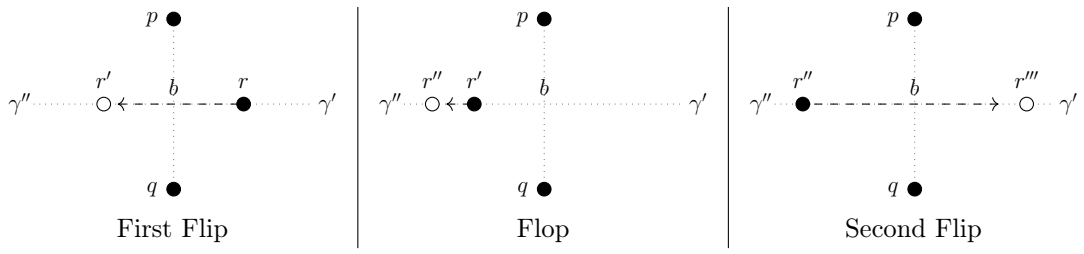


TABLE 2.2: Configurations in Flip-Flop-Flip where robot r reaches the points r', r'' and r'''.

Lemma 8. *Flip-Flop-Flip* $\in \left(\mathcal{P}\left(\mathcal{FSTA}^{\mathtt{A}}_{\mathcal{O}}\right) \cap \mathcal{P}\left(\mathcal{FCOM}^{\mathtt{F}}_{\mathcal{O}}\right)\right)$.

Proof. We solve the problem in both the models using the colors `flip1`, `flop`, and `flip2`, assuming w.l.o.g. all robots start with the color `flip1`. Note that the request of the problem guarantees that each robot can recognize its role by geometric conditions only, and that possible collinearities under $\mathcal{FSTA}^{\mathtt{A}}_{\mathcal{O}}$ do not affect the solvability of the problem. In $\mathcal{FSTA}^{\mathtt{A}}_{\mathcal{O}}$, the robot r moves along γ changing its internal color according

to the perpetual scheme $(\texttt{flip1} - \texttt{flop} - \texttt{flip2})^{\infty}$, so that at each activation, r knows which is the current action to be performed. The robots p, q do not need to change their colors. In the $\mathcal{FCOM}^{\mathtt{F}}_{\mathcal{O}}$ model, all the robots synchronously update their external colors following the above scheme, so that at each round each robot knows what actions (color setting and movement) have to be accomplished by looking at the light of the other robots. □

Lemma 9. *Flip-Flop-Flip* $\notin \left(\mathcal{P}\left(\mathcal{OBLOT}^{\mathtt{F}}_{\mathcal{T}}\right) \cup \mathcal{P}\left(\mathcal{FCOM}^{\mathtt{S}}_{\mathcal{T}}\right)\right)$.

Proof. By contradiction, suppose an algorithm $\mathbb{A}$ solves Flip-Flop-Flip under $\mathcal{OBLOT}^{\mathtt{F}}_{\mathcal{T}}$. Suppose that r always maintains the same local coordinate system Ξ. Let σ be the snapshot taken by r according to which $\mathbb{A}$ computes the first Flop action. Being in $\mathcal{OBLOT}^{\mathtt{F}}_{\mathcal{T}}$, σ contains the positions of the three robots according to Ξ, while all the colors are set to $\not\flat$. Clearly, such a snapshot could perfectly describe an initial configuration for the problem. So, assume that Flip-Flop-Flip now starts from an initial configuration where the snapshot taken by r w.r.t. Ξ coincides with σ. Since $\mathbb{A}$ has no further information as input, its output is still a Flop, which causes r to perform an erroneous action. Contradiction achieved.

By contradiction, suppose an algorithm $\mathbb{A}$ solves Flip-Flop-Flip under $\mathcal{FCOM}^{\mathtt{S}}_{\mathcal{T}}$, and let $\mathfrak{A}$ be a SSYNCH activation scheduler under which $\mathbb{A}$ solves the problem. We show that there exists a SSYNCH activation scheduler $\mathfrak{A}'$ such that Flip-Flop-Flip is not solved by $\mathbb{A}$. Let t be the first round in $\mathfrak{A}$ where r executes the first Flip. Let Ξ be a coordinate system having its origin in b and, w.l.o.g., the positive x-axis along the semi-line γ'. Assume the coordinate system of r for any activation time $t' \leq t$ is always Ξ, and let $\underline{x}_p$, $\underline{x}_q$, and $\underline{x}_r$ be the positions of p, q, and r at time t according to Ξ. So, at time t, r computes $\mathbb{A}(\sigma) = (-\underline{x}_r, c'_r)$ where $\sigma = \langle(\underline{x}_r, \not\flat), (\underline{x}_p, c_p), (\underline{x}_q, c_q)\rangle$ is the snapshot taken by r at time t, and it reaches its symmetrical position $-\underline{x}_r$ on γ''. Now, consider the scheduler $\mathfrak{A}'$ defined as $\mathfrak{A}'(t') = \mathfrak{A}(t')$, $\forall t' \leq t$. Clearly, r executes its first Flip at the t-th round under $\mathfrak{A}'$. Suppose that, in the $(t+1)$-th activation round under $\mathfrak{A}'$, r is the only robot that gets activated, namely $\mathfrak{A}'(t+1) = \{r\}$, and suppose that the coordinate system of r is the same as Ξ but now the positive x-axis is directed from b to r along γ''. Let σ' be the snapshot taken by r at time $t+1$. Being under $\mathcal{FCOM}$ and being the coordinate system mirror-symmetric w.r.t. Ξ, we have $\sigma' = \sigma$. As a consequence, r re-computes $\mathbb{A}(\sigma') = \mathbb{A}(\sigma)$ and makes again a Flip at time $t+1$. Contradiction achieved. □

Theorem 11.

$$\begin{array}{lll}
\mathcal{LUMI}^{\mathtt{A}}_{\mathcal{T}} > \mathcal{FCOM}^{\mathtt{A}}_{*}, & \mathcal{LUMI}^{\mathtt{A}}_{\mathcal{O}} > \mathcal{FCOM}^{\mathtt{A}}_{\mathcal{O}}, & \mathcal{LUMI}^{\mathtt{A}}_{\mathcal{O}} \perp \mathcal{FCOM}^{\mathtt{A}}_{\mathcal{T}}, \\
\mathcal{LUMI}^{\mathtt{S}}_{\mathcal{T}} > \mathcal{FCOM}^{\mathtt{S,A}}_{*}, & \mathcal{LUMI}^{\mathtt{S}}_{\mathcal{O}} > \mathcal{FCOM}^{\mathtt{S,A}}_{\mathcal{O}}, & \mathcal{LUMI}^{\mathtt{S}}_{\mathcal{O}} \perp \mathcal{FCOM}^{\mathtt{S,A}}_{\mathcal{T}}, \\
\mathcal{LUMI}^{\mathtt{F}}_{\mathcal{T}} > \mathcal{FCOM}^{\mathtt{S,A}}_{*}, & \mathcal{LUMI}^{\mathtt{F}}_{\mathcal{O}} > \mathcal{FCOM}^{\mathtt{S,A}}_{\mathcal{O}}, & \mathcal{LUMI}^{\mathtt{F}}_{\mathcal{O}} \perp \mathcal{FCOM}^{\mathtt{S,A}}_{\mathcal{T}}, \\
\mathcal{FCOM}^{\mathtt{F}}_{\mathcal{T}} > \mathcal{FCOM}^{\mathtt{S,A}}_{*}, & \mathcal{FCOM}^{\mathtt{F}}_{\mathcal{O}} > \mathcal{FCOM}^{\mathtt{S,A}}_{\mathcal{O}}, & \mathcal{FCOM}^{\mathtt{F}}_{\mathcal{O}} \perp \mathcal{FCOM}^{\mathtt{S,A}}_{\mathcal{T}}.
\end{array}$$

Proof. By Theorem 1, we know that:

$$\begin{array}{ll}
\mathcal{LUMI}^{\mathtt{A}}_{\mathcal{T}} \geq \mathcal{FCOM}^{\mathtt{A}}_{\mathcal{T}}, & \mathcal{LUMI}^{\mathtt{A}}_{\mathcal{O}} \geq \mathcal{FCOM}^{\mathtt{A}}_{\mathcal{O}} \\
\mathcal{LUMI}^{\mathtt{S}}_{\mathcal{T}} \geq \mathcal{FCOM}^{\mathtt{S,A}}_{\mathcal{T}}, & \mathcal{LUMI}^{\mathtt{S}}_{\mathcal{O}} \geq \mathcal{FCOM}^{\mathtt{S,A}}_{\mathcal{O}} \\
\mathcal{LUMI}^{\mathtt{F}}_{\mathcal{T}} \geq \mathcal{FCOM}^{\mathtt{S,A}}_{\mathcal{T}}, & \mathcal{LUMI}^{\mathtt{F}}_{\mathcal{O}} \geq \mathcal{FCOM}^{\mathtt{S,A}}_{\mathcal{O}} \\
\mathcal{FCOM}^{\mathtt{F}}_{\mathcal{T}} \geq \mathcal{FCOM}^{\mathtt{S,A}}_{\mathcal{T}}, & \mathcal{FCOM}^{\mathtt{F}}_{\mathcal{O}} \geq \mathcal{FCOM}^{\mathtt{S,A}}_{\mathcal{O}}.
\end{array}$$

By Lemma 8, we have that Flip-Flop-Flip is solved under $\mathcal{FCOM}^{\mathtt{F}}_{\mathcal{O}}$ and $\mathcal{LUMI}^{\mathtt{A},\mathtt{S},\mathtt{F}}_{\mathcal{O}}$. In addition, by Lemma 9, Flip-Flop-Flip cannot be solved under $\mathcal{FCOM}^{\mathtt{S},\mathtt{A}}_{\mathcal{T}}$. By using Theorem 6 with Flip-Flop-Flip as a separator, the claimed relations follow. □

Problem 5 (Newcomer Introducing)**.** Consider $n+2$ robots, with $n \geq 7$, of which n are placed on the same circle of radius of length ρ. Then, a robot p lies in the center of the circle, and the last robot s sits outside the circle so that no robot lies on the segment $\overline{sp}$. The problem requires sequentially the swarm to form two configurations and then stop. First, s must travel along the segment $\overline{sp}$ and stop on the boundary of the circle. Second, p must travel along the radius defined with s and stop at the position p' satisfying $dist(s, p') = \frac{1}{2}\rho$. All the other robots must stay still. See Table 2.3.

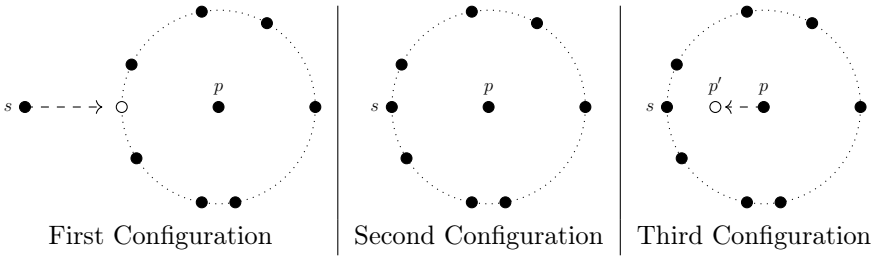


TABLE 2.3: Configurations in Newcomer Introducing.

Lemma 10. *Newcomer Introducing* $\notin \mathcal{P}\left(\mathcal{FSTA}^{\mathtt{F}}_{\mathcal{T}}\right)$.

Proof. By contradiction, let us assume there exists an $\mathcal{FSTA}^{\mathtt{F}}_{\mathcal{T}}$ algorithm $\mathbb{A}$ solving Newcomer Introducing with a constant-size color palette $\mathfrak{L}$, so that $|\mathfrak{L}| = k$ for some $k \in \mathbb{N}$. Let us consider a swarm with $n = k+2$ robots, and let $\mathcal{C}$ be a second configuration consisting of the robot p at the center of $k+1$ robots $r_1, \ldots, r_{k+1}$. Specifically, $\mathcal{C}$ can derive from $k+1$ different types of configurations, say $\mathcal{C}_1, \ldots, \mathcal{C}_{k+1}$, where $\mathcal{C}_i$ denotes a type of initial configuration in which robot r_i is external to the circle. So, whenever in $\mathcal{C}$, the robot p must detect which robot among $r_1, \ldots, r_{k+1}$ is the newcomer to move towards. Let us assume that p has a fixed coordinate system, and let σ be the snapshot p takes in $\mathcal{C}$. For notation ease, we write this snapshot as $\sigma = \langle \underline{x}_p, \underline{x}_1, \ldots, \underline{x}_{k+1}, c_p \rangle$, where $\underline{x}_p$ and c_p are, respectively, the position and the internal color of p, while $\underline{x}_i$ is the position of r_i, for $1 \leq i \leq k+1$. Notice that the same positions $\underline{x}_p, \underline{x}_1, \ldots, \underline{x}_{k+1}$ would have been observed by p in its snapshot independently of the actual $\mathcal{C}_i$ which $\mathcal{C}$ derives from. So the only element in σ which could be used by p to distinguish the newcomer among the $k+1$ robots is its internal light color c_p. However, since there are $k+1$ robots and only k colors to be used, at least 2 robots, say r_j and r_h, would be associated with the same color, say c'. So, if by hypothesis we assume that $\mathbb{A}(\langle \underline{x}_p, \underline{x}_1, \ldots, \underline{x}_{k+1}, c' \rangle)$ correctly aims at the newcomer r_j when $\mathcal{C}$ derives from $\mathcal{C}_j$, then the algorithm returns the wrong target position to p (i.e., r_j) when $\mathcal{C}$ derives from $\mathcal{C}_h$. Contradiction achieved. □

Lemma 11. *Newcomer Introducing* $\in \mathcal{P}\left(\mathcal{FCOM}^{\mathtt{A}}_{\mathcal{O}}\right)$.

Proof. We show an $\mathcal{FCOM}^{\mathtt{A}}_{\mathcal{O}}$ algorithm solving Newcomer Introducing using the palette $\mathfrak{L} = \{\not{b}, \mathtt{s}\}$. All the robots are initially set to color $\not{b}$. Each robot can determine its role by the geometry of the configurations (p realizes to be at the center of $n \geq 7$ robots lying on a common circle, and spots an external robot, s sees at least four robots forming a circle with a robot at its center, the other robots can see they lie on

a circle with at least $n - 2 \geq 5$ other robots). When s is activated, it sets its light to $\mathtt{s}$ and starts to move. This color is maintained during its subsequent activations. When activated, upon seeing a robot with color $\mathtt{s}$ on the circle, p computes the correct destination and moves there. The last configuration is stable: no other robot will move. □

Theorem 12. *Given the schedulers $Y_1 = \mathtt{F}$, $Y_2 = \mathtt{S}$, $Y_3 = \mathtt{A}$, we have the following relations for any $1 \leq i \leq j \leq 3$:*

$$\mathcal{LUMI}^{Y_i}_{\mathcal{T}} > \mathcal{FSTA}^{Y_j}_{\mathcal{T}}, \quad \mathcal{LUMI}^{Y_i}_{\mathcal{T}} > \mathcal{FSTA}^{Y_j}_{\mathcal{O}}$$
$$\mathcal{LUMI}^{Y_i}_{\mathcal{O}} \perp \mathcal{FSTA}^{Y_j}_{\mathcal{T}}, \quad \mathcal{LUMI}^{Y_i}_{\mathcal{O}} > \mathcal{FSTA}^{Y_j}_{\mathcal{O}}.$$

Proof. Trivially, we know that $\mathcal{LUMI}^{Y_i}_{\mathcal{T}} \geq \mathcal{FSTA}^{Y_j}_{\mathcal{T}}$ and $\mathcal{LUMI}^{Y_i}_{\mathcal{O}} \geq \mathcal{FSTA}^{Y_j}_{\mathcal{O}}$. By Lemma 11, `Newcomer Introducing` is solved under $\mathcal{LUMI}^{\mathtt{A},\mathtt{S},\mathtt{F}}_{\mathcal{O}}$. By Lemma 10, `Newcomer Introducing` cannot be solved under $\mathcal{FSTA}^{\mathtt{F},\mathtt{S},\mathtt{A}}_{\mathcal{T}}$. Thus, the stated relations hold by applying Theorem 6. □

Theorem 13. $\mathcal{FSTA}^{\mathtt{F},\mathtt{S},\mathtt{A}}_{*} \perp \mathcal{FCOM}^{\mathtt{S},\mathtt{A}}_{*}$.

Proof. By Lemmas 8 and 9, `Flip-Flop-Flip` is solved in $\mathcal{FSTA}^{\mathtt{F},\mathtt{S},\mathtt{A}}_{\mathcal{O}}$ but not in $\mathcal{FCOM}^{\mathtt{S},\mathtt{A}}_{\mathcal{T}}$. By Lemmas 10 and 11, `Newcomer Introducing` is solved in $\mathcal{FCOM}^{\mathtt{S},\mathtt{A}}_{\mathcal{O}}$ but not in $\mathcal{FSTA}^{\mathtt{F},\mathtt{S},\mathtt{A}}_{\mathcal{T}}$. Thus, the stated relations hold by applying Theorem 7. □

2.4.3 The power of FSYNCH

Problem 6 (`Angle-Shift`)**.** Consider an initial configuration with three robots forming an acute and scalene triangle. Let a, b, c be the three robots, where a is placed on the greatest angle, say α, whereas c is placed on the smallest angle. Fixing a as the rotation center and following the direction given by $a \to b \to c$, the problem requires b to rotate by α and c to rotate by $\pi - \alpha$. Robot a must never change its position, while robots b and c, once left their initial positions, are not allowed to stop in any other positions but the target ones. Afterwards, the robots must stay still. See Table 2.4.

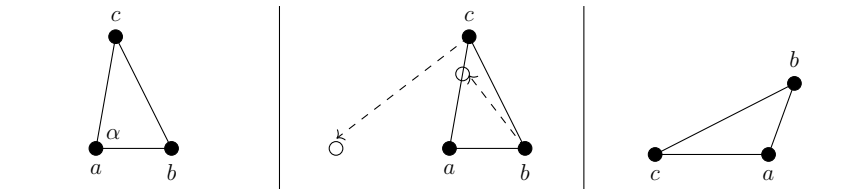


TABLE 2.4: `Angle-Shift`.

Lemma 12. *`Angle-Shift`* $\in \left(\mathcal{P}\left(\mathcal{OBLOT}^{\mathtt{F}}_{\mathcal{O}}\right) \setminus \mathcal{P}\left(\mathcal{LUMI}^{\mathtt{S}}_{\mathcal{T}}\right)\right)$.

Proof. `Angle-Shift` is solvable under any `FSYNCH` model: if b and c perform their cycles at the same time, then they correctly compute their target position. The final configuration is stable since it always forms an obtuse triangle (terminal condition).

Instead, the swarm can suffer from information loss under `SSYNCH`, making the problem unsolvable even under $\mathcal{LUMI}^{\mathtt{S}}_{\mathcal{T}}$. Suppose by contradiction that there exists an algorithm $\mathbb{A}$ that solves the problem in $\mathcal{LUMI}^{\mathtt{S}}_{\mathcal{T}}$ using a constant-size palette $\mathfrak{L}$.

Consider a SEQ scheduler (i.e., only one robot per round is activated). Suppose that b is the first robot that moves (the situation is analogous if c is the first to move). Let $\mathcal{C}'$ be the resulting configuration where the three robots are aligned. By problem request, b would have reached the same position as in $\mathcal{C}'$ by starting from an infinite range of initial configurations, where $\alpha \in (\frac{\pi}{3}, \frac{\pi}{2})$. So, from $\mathcal{C}'$, c infers the angle α, getting information only from the light color combination in $\mathcal{C}'$. Thus, the range of α must have at most the same cardinality as $|\mathfrak{L}|^3$. Contradiction achieved. □

Theorem 14. *For any $X \in \mathcal{X}$, we have:*

$$X_{\mathcal{T}}^{\mathtt{F}} > X_{\mathcal{T}}^{\mathtt{S},\mathtt{A}}, \quad X_{\mathcal{T}}^{\mathtt{F}} > X_{\mathcal{O}}^{\mathtt{S},\mathtt{A}},$$
$$X_{\mathcal{O}}^{\mathtt{F}} \perp X_{\mathcal{T}}^{\mathtt{S},\mathtt{A}}, \quad X_{\mathcal{O}}^{\mathtt{F}} > X_{\mathcal{O}}^{\mathtt{S},\mathtt{A}}.$$

Proof. We know that $X_{\mathcal{T}}^{\mathtt{F}} \geq X_{\mathcal{T}}^{\mathtt{S},\mathtt{A}}$ and $X_{\mathcal{O}}^{\mathtt{F}} \geq X_{\mathcal{O}}^{\mathtt{S},\mathtt{A}}$. By Lemma 12, Angle-Shift is solved under $\mathcal{OBLOT}_{\mathcal{O}}^{\mathtt{F}}$, and so any $X_{\mathcal{O}}^{\mathtt{F}}$, but it cannot be solved under $\mathcal{LUMI}_{\mathcal{T}}^{\mathtt{S}}$, and so any $X_{\mathcal{T}}^{\mathtt{S},\mathtt{A}}$. Thus, the stated relations hold by Theorem 6. □

Theorem 15.

$$\mathcal{OBLOT}_{*}^{\mathtt{F}} \perp \mathcal{FCOM}_{*}^{\mathtt{S},\mathtt{A}}, \quad \mathcal{OBLOT}_{*}^{\mathtt{F}} \perp \mathcal{FSTA}_{*}^{\mathtt{S},\mathtt{A}},$$
$$\mathcal{OBLOT}_{*}^{\mathtt{F}} \perp \mathcal{LUMI}_{*}^{\mathtt{S},\mathtt{A}}, \quad \mathcal{FSTA}_{*}^{\mathtt{F}} \perp \mathcal{LUMI}_{*}^{\mathtt{S},\mathtt{A}}.$$

Proof. These orthogonality relations follow by combining the previous lemmas with Theorem 7:

- $\mathcal{OBLOT}_{*}^{\mathtt{F}} \perp \mathcal{FCOM}_{*}^{\mathtt{S},\mathtt{A}}$ holds since Angle-Shift is solved in $\mathcal{OBLOT}_{\mathcal{O}}^{\mathtt{F}}$ but not in $\mathcal{FCOM}_{\mathcal{T}}^{\mathtt{S},\mathtt{A}}$, and since Newcomer Introducing is solved in $\mathcal{FCOM}_{\mathcal{O}}^{\mathtt{S},\mathtt{A}}$ but not in $\mathcal{OBLOT}_{\mathcal{T}}^{\mathtt{F}}$ (by Lemmas 10 to 12);
- $\mathcal{OBLOT}_{*}^{\mathtt{F}} \perp \mathcal{FSTA}_{*}^{\mathtt{S},\mathtt{A}}$ holds since Angle-Shift is solved in $\mathcal{OBLOT}_{\mathcal{O}}^{\mathtt{F}}$ but not in $\mathcal{FSTA}_{\mathcal{T}}^{\mathtt{S},\mathtt{A}}$, and since Triangle Round-Trip is solved in $\mathcal{FSTA}_{\mathcal{O}}^{\mathtt{S},\mathtt{A}}$ but not in $\mathcal{OBLOT}_{\mathcal{T}}^{\mathtt{F}}$ (by Lemmas 6, 7 and 12);
- $\mathcal{OBLOT}_{*}^{\mathtt{F}} \perp \mathcal{LUMI}_{*}^{\mathtt{S},\mathtt{A}}$ holds since Angle-Shift is solved in $\mathcal{OBLOT}_{\mathcal{O}}^{\mathtt{F}}$ but not in $\mathcal{LUMI}_{\mathcal{T}}^{\mathtt{S},\mathtt{A}}$, and since Triangle Round-Trip is solved in $\mathcal{LUMI}_{\mathcal{O}}^{\mathtt{S},\mathtt{A}}$ but not in $\mathcal{OBLOT}_{\mathcal{T}}^{\mathtt{F}}$ (by Lemmas 6, 7 and 12);
- $\mathcal{FSTA}_{*}^{\mathtt{F}} \perp \mathcal{LUMI}_{*}^{\mathtt{S},\mathtt{A}}$ holds since Angle-Shift is solved in $\mathcal{FSTA}_{\mathcal{O}}^{\mathtt{F}}$ but not in $\mathcal{LUMI}_{\mathcal{T}}^{\mathtt{S},\mathtt{A}}$, and since Newcomer Introducing is solved in $\mathcal{LUMI}_{\mathcal{O}}^{\mathtt{S},\mathtt{A}}$ but not in $\mathcal{FSTA}_{\mathcal{T}}^{\mathtt{F}}$ (by Lemmas 10 to 12).

□

2.4.4 $\mathcal{OBLOT}$ and asynchrony

Problem 7 (Semi-Expansion)**.** The problem is defined recursively without any stop conditions, for a swarm of two robots. Consider an arbitrary configuration $\mathcal{C}$ where such robots a, b are located in distinct positions. Let $u = dist(a, b)$ and let γ be the supporting line of a, b. Robots are required to move along γ *at most once*, and to stop to form a new static configuration $\mathcal{C}'$ where $dist(a, b) \in \{2u, \frac{3}{2}u\}$. Recursively, the problem demands the same request starting from $\mathcal{C}'$. See Table 2.5.

Lemma 13. *Semi-Expansion* $\in \left(\mathcal{P}\left(\mathcal{OBLOT}_{\mathcal{O}}^{\mathtt{S}}\right) \cap \mathcal{P}\left(\mathcal{LUMI}_{\mathcal{O}}^{\mathtt{A}}\right)\right)$.

Configuration $\mathcal{C}$. | At most one movement per robot to reach $\mathcal{C}'$.

TABLE 2.5: Configurations in Semi-Expansion.

Proof. Semi-Expansion can be trivially solved as follows under $\mathcal{OBLOT}^{\mathtt{S}}_{\mathcal{O}}$: when a robot is activated, it computes γ and the current distance u, and it travels along γ in order to form a new distance $u + \frac{1}{2}u$ with the other robot. If only one robot is activated, the new distance between the robots is $\frac{3}{2}u$. If both robots are activated, the new distance is $2u$.

We can exploit the first part of the lemma to state that Semi-Expansion is solved under $X^{\mathtt{S}}_{\mathcal{O}}$, and thus under $X^{\mathtt{S}}_{\mathcal{T}}$, for any $X \in \mathcal{X}$. By applying Theorem 8 (which mentions the valuable result $\mathcal{LUMI}^{\mathtt{S}}_{\mathcal{T}} \equiv \mathcal{LUMI}^{\mathtt{A}}_{\mathcal{T}}$ proved in [Das+16]), we get that Semi-Expansion can be solved also under $\mathcal{LUMI}^{\mathtt{A}}_{\mathcal{T}}$. Being only two robots, the swarm never suffers from obstructed visibility, thus allowing us to conclude that Semi-Expansion $\in \mathcal{P}\left(\mathcal{LUMI}^{\mathtt{A}}_{\mathcal{O}}\right)$. □

Lemma 14. *Semi-Expansion* $\notin \left(\mathcal{P}\left(\mathcal{FSTA}^{\mathtt{A}}_{\mathcal{T}}\right) \cup \mathcal{P}\left(\mathcal{FCOM}^{\mathtt{A}}_{\mathcal{T}}\right)\right)$.

Proof. Semi-Expansion cannot be solved under $\mathcal{FSTA}^{\mathtt{A}}_{\mathcal{T}}$. The impossibility derives from the fact that a robot cannot distinguish between a static configuration and a transient one (i.e., where the other robot is moving), so an activated robot has no means to compute the correct distance it has to travel.

Semi-Expansion cannot be solved under $\mathcal{FCOM}^{\mathtt{A}}_{\mathcal{T}}$. By contradiction, let us assume that there exists an algorithm $\mathbb{A}$ under $\mathcal{FCOM}^{\mathtt{A}}_{\mathcal{T}}$ solving Semi-Expansion. Suppose the local coordinate system of each activated robot is always so that the origin corresponds to the position of the robot, and the other robot is placed at position $(1, 0)$. By definition, $\mathbb{A}$ must solve the problem even under a FSYNCH scheduler: being completely symmetric, both robots synchronously move at each round to form a new configuration where the distance is doubled w.r.t. the previous round, and they always set the same color. In fact, they both compute $\mathbb{A}(\sigma)$ where $\sigma = \langle((0,0),\not{0}),((1,0),c)\rangle)$ and c is the color of the two robots at a given round.

Let us now consider a scheduler that works as FSYNCH until a time t, and let $\mathcal{C}$ be the static configuration obtained at time t where the two robots a, b have the same color and are at a distance $u = dist(a, b)$ according to a global coordinate system. Suppose at time t both robots perform the instantaneous Look step where they get the same snapshot, say σ'. Now, let us suppose robot b computes the next position and moves there, whereas a is still executing its Compute step. Let $\mathcal{C}'$ be the static configuration obtained after the movement of b, where now $dist(a, b) = \frac{3}{2}u = u'$. Suppose b is activated again while a is still computing. Since a has the same color as at time t, then b takes the same snapshot σ' and deterministically re-executes the same action as before, moving a distance $\frac{1}{2}u' = \frac{3}{4}u$. Let $\mathcal{C}''$ be the static configuration obtained after the second movement of b, where now $dist(a, b) = \frac{3}{2}u + \frac{3}{4}u = \frac{9}{4}u = u''$. At this time, suppose b stays still and a eventually executes the Move step w.r.t. the snapshot at time t. So a travels a distance $\frac{1}{2}u$, obtaining a static configuration $\mathcal{C}'''$ where $dist(a, b) = \frac{9}{4}u + \frac{1}{2}u = \frac{11}{4}u = u'''$. By request, u''' must be in $\{2u'', \frac{3}{2}u''\}$. However, since $\frac{11}{4}u \neq 2 \cdot \frac{9}{4}u$ and $\frac{11}{4}u \neq \frac{3}{2} \cdot \frac{9}{4}u$, the request of the problem has not been fulfilled. Contradiction achieved. □

Theorem 16. *For each* $X \in \{\mathcal{OBLOT}, \mathcal{FSTA}, \mathcal{FCOM}\}$*, we have:*

$$X^{\mathtt{S}}_{\mathcal{T}} > X^{\mathtt{A}}_{\mathcal{T}}, \quad X^{\mathtt{S}}_{\mathcal{T}} > X^{\mathtt{A}}_{\mathcal{O}}$$
$$X^{\mathtt{S}}_{\mathcal{O}} \perp X^{\mathtt{A}}_{\mathcal{T}} \quad X^{\mathtt{S}}_{\mathcal{O}} > X^{\mathtt{A}}_{\mathcal{O}}.$$

Proof. Trivially, we know that $X^{\mathtt{S}}_{\mathcal{T}} \geq X^{\mathtt{A}}_{\mathcal{T}}$ and $X^{\mathtt{S}}_{\mathcal{O}} \geq X^{\mathtt{A}}_{\mathcal{O}}$. By Lemmas 13 and 14, `Semi-Expansion` can be solved in $X^{\mathtt{S}}_{\mathcal{O}}$ but not in $X^{\mathtt{A}}_{\mathcal{T}}$. By applying Theorem 6, the claimed relations hold. □

Theorem 17.

$$\mathcal{FSTA}^{\mathtt{A}}_{*} \perp \mathcal{OBLOT}^{\mathtt{S}}_{*},$$
$$\mathcal{FCOM}^{\mathtt{A}}_{*} \perp \mathcal{OBLOT}^{\mathtt{S}}_{*}.$$

Proof. By Lemmas 13 and 14, we know that `Semi-Expansion` is solved under $\mathcal{OBLOT}^{\mathtt{S}}_{\mathcal{O}}$ but under neither $\mathcal{FSTA}^{\mathtt{A}}_{\mathcal{T}}$ nor $\mathcal{FCOM}^{\mathtt{A}}_{\mathcal{T}}$. By Lemmas 6 and 7, we know that `Triangle Round-Trip` is solved under both $\mathcal{FSTA}^{\mathtt{A}}_{\mathcal{O}}$ and $\mathcal{FCOM}^{\mathtt{A}}_{\mathcal{O}}$ but not under $\mathcal{OBLOT}^{\mathtt{S}}_{\mathcal{T}}$. By applying Theorem 7, the orthogonalities follow. □

2.4.5 Opaqueness and asynchrony

We now introduce the `Pseudo-Polygon` problem which shows a peculiar issue occurring in case of obstructed visibility and asynchrony.

Definition 5. *A* safe zone *of a regular polygon* $\mathcal{N}$ *is the locus of all points* x *in the plane such that:*

- x *does not belong to the closed region delimited by* $\mathcal{N}$*;*
- x *is not aligned with any pair of vertices of* $\mathcal{N}$*;*
- x *does not lie on the bisector of any edge of* $\mathcal{N}$ *(equivalently,* x *is not equally distanced from any two adjacent vertices);*
- *if* ℓ *is the length of the edge of* $\mathcal{N}$*, then the distance between* x *and any vertex of* $\mathcal{N}$ *is at least* ℓ*.*

The blue regions plus infinite lines in Figure 2.3 depicts the complement of the safe zone of a square.

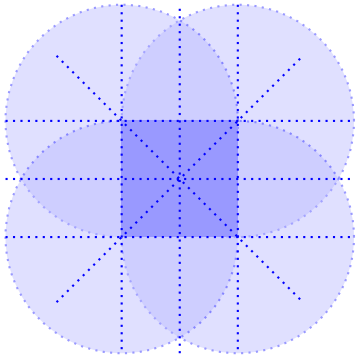


FIGURE 2.3: The safe zone of the square comprehends all the points not belonging to the blue-colored regions and (infinite) lines.

Definition 6. *Given a regular n-gon $\mathcal{N}$, for any $n \geq 4$, a* pseudo-polygon $\mathcal{Q}$ *is a subset of vertices of $\mathcal{N}$, such that $|\mathcal{Q}| \geq \lfloor \frac{n}{2} \rfloor + 1$. We call $\mathcal{N}$ the* polygon associated *with $\mathcal{Q}$.*

Given a pseudo-polygon $\mathcal{Q}$, it is always possible to determine the associated polygon, which is unique. In fact, as $\mathcal{Q}$ contains at least three vertices, the circumscribed circle is univocally defined. Moreover, since $\mathcal{Q}$ contains more than half of the vertices of the associated n-gon $\mathcal{N}$, there always exist at least two vertices in $\mathcal{Q}$ that are adjacent in $\mathcal{N}$, thus enabling the reconstruction of all the edges of $\mathcal{N}$.

Problem 8 (Pseudo-Polygon)**.** Let $\mathcal{N}$ be a regular n-gon with $n \geq 6$ vertices. Let $\mathcal{Q}$ be a pseudo-polygon of $m \geq \lfloor \frac{n}{2} \rfloor + 2$ vertices, with which $\mathcal{N}$ is associated. Consider a swarm of $m+1$ robots, where m robots lie on $\mathcal{Q}$ and let the last robot, w, lie in the safe zone of $\mathcal{N}$. Let a be the farthest robot from w. Let b be the first robot encountered while moving from a along the perimeter of $\mathcal{N}$ in one direction, and c be the first robot encountered while proceeding in the opposite direction. Assume $dist(b, w) > dist(c, w)$. Note that the two distances are always different since w belongs to the safe zone of $\mathcal{N}$. The problem requires a to move away from b towards a point x such that *(i)* x belongs to the safe zone of $\mathcal{N}$, *(ii)* x belongs to the halfplane delimited by the line $\overline{bc}$ that does not contain a, and *(iii)* x must not lie on any line passing through w and any other robot on $\mathcal{Q}$. Note that the requests *(i)* and *(iii)* are imposed in order to have x visible by every robot in the swarm. See Figure 2.4.

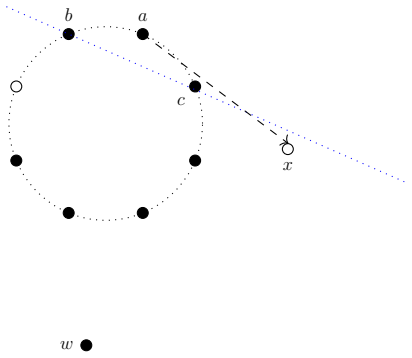


FIGURE 2.4: An instance of the Pseudo-Polygon problem built on an octagon.

Lemma 15. *Pseudo-Polygon* $\notin \mathcal{P}\left(\mathcal{FSTA}^{\mathtt{A}}_{\mathcal{O}}\right)$.

Proof. Pseudo-Polygon cannot be solved in the ASYNCH opaque model, only using internal lights. Let us consider the problem instance shown in Figure 2.4, where the pseudo-polygon of the initial configuration consists $\lfloor \frac{n}{2} \rfloor + 3$ vertices, with $n = 8$. Let us assume b is activated for the first time during the movement of a towards x, when a is hidden by c (i.e., b, c, a are collinear). When b looks at its snapshot, it recognizes a feasible initial configuration (it sees a pseudo-polygon with $\lfloor \frac{n}{2} \rfloor + 2$ robots, and the robot w). According to this configuration, b erroneously elects itself as the robot that has to move away from the pseudo-polygon. It has no means to understand whether or not a exists: this is true since a may form a collinearity with b and c for a finite but unpredictable time range. On the other hand, a cannot postpone its movement and wait for b to memorize that it is not the elected robot to move. In fact, a has no means of knowing if b has updated its internal light. □

False election. The impossibility of solving Pseudo-Polygon with opaque and asynchronous robots with only internal lights (Lemma 15) derives from a critical issue that is peculiar for swarms with obstructed visibility. This critical issue can be described as the *false election* phenomenon. Such a phenomenon can be informally explained as follows: from a static configuration (i.e., where no robot moves), a given problem requires, using a *leader election* routine, to single out the unique robot (the *true leader*) which has to execute a non-null movement to reach the next configuration. All the other robots have to stay still. In ASYNCH, a robot r executes its Look step while the true leader is moving and is hidden from r. Thus, if r cannot detect the presence of the true leader from its snapshot, r might elect itself as the (*false*) leader by applying the same leader election routine. This clearly would start a non-requested movement.

We observe that attention must be paid to the false election phenomenon, e.g., when trying to adapt a SSYNCH algorithm to the ASYNCH framework. In particular, the use of lights must be considered as a possible method to avoid false elections. As we have shown in Lemma 15 for Pseudo-Polygon, internal lights are not sufficient to cope with them. Instead, we are now going to prove that external lights are required (and sufficient) to correctly solve the Pseudo-Polygon problem in ASYNCH.

Lemma 16. *Pseudo-Polygon* $\in \left(\mathcal{P}\left(\mathcal{OBLOT}^{\mathtt{S}}_{\mathcal{O}}\right) \cap \mathcal{P}\left(\mathcal{FCOM}^{\mathtt{A}}_{\mathcal{O}}\right)\right)$.

Proof. Pseudo-Polygon is solvable in $\mathcal{OBLOT}^{\mathtt{S}}_{\mathcal{O}}$ (i.e., in any synchronous model), since complete visibility is guaranteed at any activation time and all the movements (null and non-null) are univocally determined by geometric conditions. In fact, each robot can determine $\mathcal{Q}$, robot w, and robot a (the farthest from w). The robot a can compute its final destination and move there. If a robot is not the farthest from w, or if it sees two robots that are not part of the pseudo-polygon, then it stands still.

Pseudo-Polygon needs at least external lights to be solvable in the ASYNCH mode by opaque robots. We show here an algorithm that uses 4 colors: `off` (assuming it as default, w.l.o.g.), `on`, `a`, `b`. In the first phase, every robot updates its color according to its role: robot a turns into `a`, robot b turns into `b`, whereas the remainder turns into `on`. Afterward, let r be an activated robot that sees no `off` robots and that notes there is only one robot (i.e., w) out of the pseudo-polygon. Let V_r be the set of colors of the other robots that r can see.

- if $V_r = \{\mathtt{a}, \mathtt{b}, \mathtt{on}\}$, r turns into `on` and stays still;
- if $V_r = \{\mathtt{a}, \mathtt{on}\}$, r turns into `b` and stays still;
- if $V_r = \{\mathtt{b}, \mathtt{on}\}$, and if r is the farthest robot from w, it turns into `a` and starts moving;
- if $V_r = \{\mathtt{on}\}$, it means r is `b` and stays still (robot a is hidden).

If a robot r sees two robots not belonging to the pseudo-polygon, then r does not move (the final configuration is already formed or is about to be formed). □

2.4.6 Unknown relations

So far, all the theorems exhibited in this section prove, for each considered pair $A = X^Y$ and $B = Z^W$ with $X, Z \in \mathcal{X}$ and $Y, W \in \mathcal{Y}$, the four relations $A_{\mathcal{T}}$ vs. $B_{\mathcal{T}}$, $A_{\mathcal{T}}$ vs. $B_{\mathcal{O}}$, $A_{\mathcal{O}}$ vs. $B_{\mathcal{T}}$, $A_{\mathcal{O}}$ vs. $B_{\mathcal{O}}$, thus providing the complete hierarchy for the four models obtained by tuning the visibility setting. Notably, each relation $A_{\mathcal{T}}$ vs. $B_{\mathcal{T}}$ here proved is identical to the corresponding relation $A_{\mathcal{O}}$ vs. $B_{\mathcal{O}}$.

However, there exist other pairs of models A, B for which the relation $A_{\mathcal{O}}$ vs. $B_{\mathcal{O}}$ is unknown to date and, in particular, we point out the relations $\mathcal{LUMI}^{\mathtt{S}}_{\mathcal{O}}$ vs. $\mathcal{LUMI}^{\mathtt{A}}_{\mathcal{O}}$ and $\mathcal{LUMI}^{\mathtt{F}}_{\mathcal{O}}$ vs. $\mathcal{FCOM}^{\mathtt{F}}_{\mathcal{O}}$. The only facts we can state up to now are $\mathcal{LUMI}^{\mathtt{S}}_{\mathcal{O}} \geq \mathcal{LUMI}^{\mathtt{A}}_{\mathcal{O}}$ and $\mathcal{LUMI}^{\mathtt{F}}_{\mathcal{O}} \geq \mathcal{FCOM}^{\mathtt{F}}_{\mathcal{O}}$, without discerning between $>$ or $\equiv$. Both these relations, in their transparent version, boil down to equivalences. This has been proved in [Das+16; FSW19] by designing model simulators. However, such simulators cannot be used *as they are* in the related opaque models due to the obstructed visibility. Those simulators, in fact, rely on the assumptions that robots can see all the robots in the swarm. In particular, in the simulator for $\mathcal{LUMI}^{\mathtt{S}}_{\mathcal{T}} \equiv \mathcal{LUMI}^{\mathtt{A}}_{\mathcal{T}}$ [Das+16], the complete visibility assumption ensures a robot can see all robots' lights, which are used to synchronize robots' actions; in the simulator for $\mathcal{LUMI}^{\mathtt{F}}_{\mathcal{T}} \equiv \mathcal{FCOM}^{\mathtt{F}}_{\mathcal{T}}$ [FSW19], the complete visibility assumption ensures the swarm to reconstruct a global circular arrangement of the robots in the current configuration, and thus to identify the two adjacent robots which will mirror its own color light. Dropping the complete visibility assumption does not guarantee the correctness of these simulators in the opaque framework.

For other pairs of models, two possible relations can exist: $>$ or $\perp$. For instance, for $\mathcal{FCOM}^{\mathtt{F}}_{\mathcal{O}}$ and $\mathcal{FSTA}^{\mathtt{F}}_{\mathcal{O}}$, we have built the separator `Newcomer Introducing` $\in \left(\mathcal{P}\left(\mathcal{FCOM}^{\mathtt{F}}_{\mathcal{O}}\right) \setminus \mathcal{P}\left(\mathcal{FSTA}^{\mathtt{F}}_{\mathcal{O}}\right)\right)$. To prove the orthogonality relation, we should further design a separator to be settled within $\left(\mathcal{P}\left(\mathcal{FSTA}^{\mathtt{F}}_{\mathcal{O}}\right) \setminus \mathcal{P}\left(\mathcal{FCOM}^{\mathtt{F}}_{\mathcal{O}}\right)\right)$. Instead, to obtain a dominance, we should prove that any problem in $\mathcal{P}\left(\mathcal{FSTA}^{\mathtt{F}}_{\mathcal{O}}\right)$ can be solved in the model $\mathcal{FCOM}^{\mathtt{F}}_{\mathcal{O}}$ as well. This could be achieved, e.g., by showing how to simulate any $\mathcal{FSTA}^{\mathtt{F}}_{\mathcal{O}}$ algorithm under the model $\mathcal{FCOM}^{\mathtt{F}}_{\mathcal{O}}$.

For the sake of completeness, we here list all the relations A_* vs. B_* for which the relation between $A_{\mathcal{O}}$ vs. $B_{\mathcal{O}}$ is so far unknown.

Theorem 18.

$$\mathcal{LUMI}^{\mathtt{S}}_{\mathcal{T}} \equiv \mathcal{LUMI}^{\mathtt{A}}_{\mathcal{T}},\ \ \mathcal{LUMI}^{\mathtt{S}}_{\mathcal{T}} > \mathcal{LUMI}^{\mathtt{A}}_{\mathcal{O}},$$
$$\mathcal{LUMI}^{\mathtt{S}}_{\mathcal{O}} < \mathcal{LUMI}^{\mathtt{A}}_{\mathcal{T}},\ \ \mathcal{LUMI}^{\mathtt{S}}_{\mathcal{O}} > \textit{or} \equiv \mathcal{LUMI}^{\mathtt{A}}_{\mathcal{O}}.$$

Proof. The equivalence $\mathcal{LUMI}^{\mathtt{S}}_{\mathcal{T}} \equiv \mathcal{LUMI}^{\mathtt{A}}_{\mathcal{T}}$ (see Theorem 8) has been proved in [Das+16]. The two dominances directly follow from this equivalence and Theorem 5. Finally, we know that $\mathcal{LUMI}^{\mathtt{S}}_{\mathcal{O}} \geq \mathcal{LUMI}^{\mathtt{A}}_{\mathcal{O}}$: whether it is a dominance or an equivalence remains to be proved. □

Theorem 19.

$$\mathcal{LUMI}^{\mathtt{F}}_{\mathcal{T}} \equiv \mathcal{FCOM}^{\mathtt{F}}_{\mathcal{T}},\ \ \mathcal{LUMI}^{\mathtt{F}}_{\mathcal{T}} > \mathcal{FCOM}^{\mathtt{F}}_{\mathcal{O}},$$
$$\mathcal{LUMI}^{\mathtt{F}}_{\mathcal{O}} < \mathcal{FCOM}^{\mathtt{F}}_{\mathcal{T}},\ \ \mathcal{LUMI}^{\mathtt{F}}_{\mathcal{O}} > \textit{or} \equiv \mathcal{FCOM}^{\mathtt{F}}_{\mathcal{O}}.$$

Proof. The equivalence $\mathcal{LUMI}^{\mathtt{F}}_{\mathcal{T}} \equiv \mathcal{FCOM}^{\mathtt{F}}_{\mathcal{T}}$ (see Theorem 9) has been proved in [FSW19]. The other relations follow from the same reasoning as the proof of the previous theorem. □

Theorem 20. *For any* $X \in \{\mathcal{OBLOT}, \mathcal{FSTA}, \mathcal{FCOM}\}$*, we have*

$$\mathcal{LUMI}^{\mathtt{A}}_{\mathcal{T}} > X^{\mathtt{S}}_{\mathcal{T}},\ \ \mathcal{LUMI}^{\mathtt{A}}_{\mathcal{T}} > X^{\mathtt{S}}_{\mathcal{O}},$$
$$\mathcal{LUMI}^{\mathtt{A}}_{\mathcal{O}} \perp X^{\mathtt{S}}_{\mathcal{T}},\ \ \mathcal{LUMI}^{\mathtt{A}}_{\mathcal{O}} >\ \textit{or}\ \perp X^{\mathtt{S}}_{\mathcal{O}}.$$

Proof. Let us proceed considering $X = \mathcal{OBLOT}$. Using the equivalence $\mathcal{LUMI}^{\mathtt{A}}_{\mathcal{T}} \equiv \mathcal{LUMI}^{\mathtt{S}}_{\mathcal{T}}$ and `Triangle Round-Trip` as separator (see Lemma 7), we can conclude that $\mathcal{LUMI}^{\mathtt{A}}_{\mathcal{T}} > \mathcal{OBLOT}^{\mathtt{S}}_{\mathcal{T}} > \mathcal{OBLOT}^{\mathtt{S}}_{\mathcal{O}}$ (by Theorem 5). The orthogonality $\mathcal{LUMI}^{\mathtt{A}}_{\mathcal{O}} \perp \mathcal{OBLOT}^{\mathtt{S}}_{\mathcal{T}}$ follows using `Triangle Round-Trip` and `Line-Stretch` as separators (remember that `Line-Stretch` is solved under any transparent model but by no opaque model). Between $\mathcal{LUMI}^{\mathtt{A}}_{\mathcal{O}}$ and $\mathcal{OBLOT}^{\mathtt{S}}_{\mathcal{O}}$ we only know that `Triangle Round-Trip` is solved under $\mathcal{OBLOT}^{\mathtt{S}}_{\mathcal{O}}$ but not under $\mathcal{LUMI}^{\mathtt{A}}_{\mathcal{O}}$: whether they are related by a dominance or an orthogonality remains to be proved.

The proof for $X = \mathcal{FSTA}$ ($X = \mathcal{FCOM}$, resp.) has the same structure, using `Newcomer Introducing` (`Flip-Flop-Flip`, resp.) as separator, instead of `Triangle Round-Trip`. □

Theorem 21. *For any $X \in \{\mathcal{FSTA}, \mathcal{LUMI}\}$, we have*

$$\mathcal{FCOM}^{\mathtt{F}}_{\mathcal{T}} > X^{\mathtt{S,A}}_{\mathcal{T}}, \;\; \mathcal{FCOM}^{\mathtt{F}}_{\mathcal{T}} > X^{\mathtt{S,A}}_{\mathcal{O}},$$
$$\mathcal{FCOM}^{\mathtt{F}}_{\mathcal{O}} \perp X^{\mathtt{S,A}}_{\mathcal{T}}, \;\; \mathcal{FCOM}^{\mathtt{F}}_{\mathcal{O}} > \;\; \textit{or} \;\; \perp X^{\mathtt{S,A}}_{\mathcal{O}}.$$

Proof. We know that $\mathcal{LUMI}^{\mathtt{F}}_{\mathcal{T}} \equiv \mathcal{FCOM}^{\mathtt{F}}_{\mathcal{T}}$ [FSW19], thus concluding that $\mathcal{FCOM}^{\mathtt{F}}_{\mathcal{T}} \geq X^{\mathtt{S,A}}_{\mathcal{T}}$. By Lemma 12, we know that `Angle-Shift` is solved under $\mathcal{FCOM}^{\mathtt{F}}_{\mathcal{O}}$ but not under $X^{\mathtt{S,A}}_{\mathcal{T}}$, thus making self-evident the dominance $\mathcal{FCOM}^{\mathtt{F}}_{\mathcal{T}} > X^{\mathtt{S,A}}_{\mathcal{T}}$. The second dominance $\mathcal{FCOM}^{\mathtt{F}}_{\mathcal{T}} > X^{\mathtt{S,A}}_{\mathcal{O}}$ follows by Theorem 5. The orthogonality $\mathcal{FCOM}^{\mathtt{F}}_{\mathcal{O}} \perp X^{\mathtt{S,A}}_{\mathcal{T}}$ follows using `Angle-Shift` and `Line-Stretch` as separators. Between $\mathcal{FCOM}^{\mathtt{F}}_{\mathcal{O}}$ and $X^{\mathtt{S,A}}_{\mathcal{O}}$ we only know that a problem (`Angle-Shift`) is solved under $\mathcal{FCOM}^{\mathtt{F}}_{\mathcal{O}}$ but not under $X^{\mathtt{S,A}}_{\mathcal{O}}$: whether they are related by a dominance or an orthogonality remains to be proved. □

Theorem 22.

$$\mathcal{FCOM}^{\mathtt{F}}_{\mathcal{T}} > \mathcal{FSTA}^{\mathtt{F}}_{\mathcal{T}}, \;\; \mathcal{FCOM}^{\mathtt{F}}_{\mathcal{T}} > \mathcal{FSTA}^{\mathtt{F}}_{\mathcal{O}},$$
$$\mathcal{FCOM}^{\mathtt{F}}_{\mathcal{O}} \perp \mathcal{FSTA}^{\mathtt{F}}_{\mathcal{T}}, \;\; \mathcal{FCOM}^{\mathtt{F}}_{\mathcal{O}} > \;\; \textit{or} \;\; \perp \mathcal{FSTA}^{\mathtt{F}}_{\mathcal{O}}.$$

Proof. The proof has the same structure as in Theorem 21, using `Newcomer Introducing`, instead of `Angle-Shift`, as separator between $\mathcal{FCOM}^{\mathtt{F}}_{\mathcal{O}}$ and $\mathcal{FSTA}^{\mathtt{F}}_{\mathcal{T}}$ (Lemmas 10 and 11). □

2.5 Relation map

Table 2.6 summarizes the results proved in this chapter, showing the relations ($>$, $<$, $\perp$, and $\equiv$) holding between any two models from the set $\mathcal{M}$ of the 24 models here under study. Specifically, the cell related to the pair of models A_* and B_* shows the relation existing between $A_{\mathcal{T}}, B_{\mathcal{T}}$ and $A_{\mathcal{T}}, B_{\mathcal{O}}$ (in the first row) and between $A_{\mathcal{O}}, B_{\mathcal{T}}$ and $A_{\mathcal{O}}, B_{\mathcal{O}}$ (in the second row). To avoid redundancy, we have specified the relation $\perp$ for the blue cells only once, since it holds for all four combinations. The relation map also shows which separator problems have been used to prove dominances and orthogonalities: `TRT` stands for `Triangle Round-Trip`, `FFF` for `Flip-Flop-Flip`, `NWC` for `Newcomer Introducing`, `ASH` for `Angle-Shift`, `SEP` for `Semi-Expansion`. Red cells represent the relations of type in Figure 2.2a, while yellow cells in the diagonal represent the trivial equivalences and the transparent-opaque dominance demonstrated in Theorem 3.

Figure 2.5 shows the *intra-framework* hierarchies of the models in $\mathcal{M}_{\mathcal{T}}$ (Figure 2.5a) and $\mathcal{M}_{\mathcal{O}}$ (Figure 2.5b), depicting the relations among them. The unknown relations

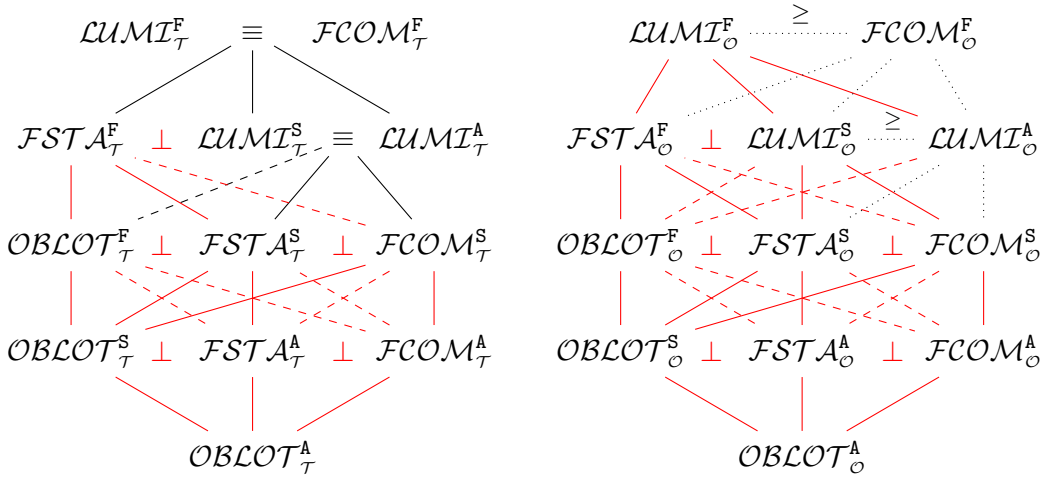
shown in Section 2.4.6 are highlighted using the gray cells in Table 2.6 and the dotted lines in Figure 2.5b.

TABLE 2.6: Relation map. Each cell shows the relation holding between the $\mathcal{T}$ – $\mathcal{T}$ and $\mathcal{T}$ – $\mathcal{O}$ models (first row), and $\mathcal{O}$ – $\mathcal{T}$ and $\mathcal{O}$ – $\mathcal{O}$ models (second row). The acronyms refer to problems witnessing dominances and orthogonalities: TRT for Triangle Round-Trip, FFF for Flip-Flop-Flip, NWC for Newcomer Introducing, ASH for Angle-Shift, SEP for Semi-Expansion.

↱	$\mathcal{LUMI}^{\mathrm{F}}_{*}$	$\mathcal{FCOM}^{\mathrm{F}}_{*}$	$\mathcal{FSTA}^{\mathrm{F}}_{*}$	$\mathcal{OBLOT}^{\mathrm{F}}_{*}$	$\mathcal{LUMI}^{\mathrm{S}}_{*}$	$\mathcal{FCOM}^{\mathrm{S}}_{*}$	$\mathcal{FSTA}^{\mathrm{S}}_{*}$	$\mathcal{OBLOT}^{\mathrm{S}}_{*}$	$\mathcal{LUMI}^{\mathrm{A}}_{*}$	$\mathcal{FCOM}^{\mathrm{A}}_{*}$	$\mathcal{FSTA}^{\mathrm{A}}_{*}$	$\mathcal{OBLOT}^{\mathrm{A}}_{*}$
$\mathcal{OBLOT}^{\mathrm{A}}_{*}$	$<,\perp$ $<,<$ TRT	$<,\perp$ $<,<$ TRT	$<,\perp$ $<,<$ TRT	$<,\perp$ $<,<$ ASH	$<,\perp$ $<,<$ TRT	$<,\perp$ $<,<$ TRT	$<,\perp$ $<,<$ TRT	$<,\perp$ $<,<$ SEP	$<,\perp$ $<,<$ TRT	$<,\perp$ $<,<$ TRT	$<,\perp$ $<,<$ TRT	$\equiv,>$ $<,\equiv$
$\mathcal{FSTA}^{\mathrm{A}}_{*}$	$<,\perp$ $<,<$ NWC	$<,\perp$ $<,<$ or $\perp$ ASH	$<,\perp$ $<,<$ ASH	$\perp$ TRT, ASH	$<,\perp$ $<,<$ NWC	$\perp$ NWC, FFF	$<,\perp$ $<,<$ SEP	$\perp$ TRT, SEP	$<,\perp$ $<,<$ NWC	$\perp$ NWC, FFF	$\equiv,>$ $<,\equiv$	
$\mathcal{FCOM}^{\mathrm{A}}_{*}$	$<,\perp$ $<,<$ FFF	$<,\perp$ $<,<$ FFF	$\perp$ NWC, FFF	$\perp$ NWC, ASH	$<,\perp$ $<,<$ FFF	$<,\perp$ $<,<$ SEP	$\perp$ NWC, FFF	$\perp$ TRT, SEP	$<,\perp$ $<,<$ FFF	$\equiv,>$ $<,\equiv$		
$\mathcal{LUMI}^{\mathrm{A}}_{*}$	$<,\perp$ $<,<$ ASH	$<,\perp$ $<,<$ or $\perp$ ASH	$\perp$ NWC, ASH	$\perp$ TRT, ASH	$\equiv,>$ $<,\equiv$ or $<$	$>,>$ $\perp,>$ or $\perp$ FFF,	$>,>$ $\perp,>$ or $\perp$ NWC,	$>,>$ $\perp,>$ or $\perp$ TRT,	$\equiv,>$ $<,\equiv$			
$\mathcal{OBLOT}^{\mathrm{S}}_{*}$	$<,\perp$ $<,<$ TRT	$<,\perp$ $<,<$ TRT	$<,\perp$ $<,<$ TRT	$<,\perp$ $<,<$ ASH	$<,\perp$ $<,<$ TRT	$<,\perp$ $<,<$ TRT	$<,\perp$ $<,<$ TRT	$\equiv,>$ $<,\equiv$				
$\mathcal{FSTA}^{\mathrm{S}}_{*}$	$<,\perp$ $<,<$ NWC	$<,\perp$ $<,<$ or $\perp$ ASH	$<,\perp$ $<,<$ ASH	$\perp$ TRT, ASH	$<,\perp$ $<,<$ NWC	$\perp$ NWC, FFF	$\equiv,>$ $<,\equiv$					
$\mathcal{FCOM}^{\mathrm{S}}_{*}$	$<,\perp$ $<,<$ FFF	$<,\perp$ $<,<$ FFF	$\perp$ NWC, FFF	$\perp$ NWC, ASH	$<,\perp$ $<,<$ FFF	$\equiv,>$ $<,\equiv$						
$\mathcal{LUMI}^{\mathrm{S}}_{*}$	$<,\perp$ $<,<$ ASH	$<,\perp$ $<,<$ or $\perp$ ASH	$\perp$ NWC, ASH	$\perp$ TRT, ASH	$\equiv,>$ $<,\equiv$							
$\mathcal{OBLOT}^{\mathrm{F}}_{*}$	$<,\perp$ $<,<$ TRT	$<,\perp$ $<,<$ TRT	$<,\perp$ $<,<$ TRT	$\equiv,>$ $<,\equiv$								
$\mathcal{FSTA}^{\mathrm{F}}_{*}$	$<,\perp$ $<,<$ NWC	$<,\perp$ $<,<$ or $\perp$ NWC	$\equiv,>$ $<,\equiv$									
$\mathcal{FCOM}^{\mathrm{F}}_{*}$	$\equiv,>$ $<,\equiv$ or $<$	$\equiv,>$ $<,\equiv$										
$\mathcal{LUMI}^{\mathrm{F}}_{*}$	$\equiv,>$ $<,\equiv$											

2.6 Conclusions

We have investigated the computational power of 24 models of *collision-intolerant* robots, obtained by tuning four features: memory and communication ($\mathcal{OBLOT}$, $\mathcal{FSTA}$, $\mathcal{FCOM}$, and $\mathcal{LUMI}$), synchronization (FSYNCH, SSYNCH, and ASYNCH), and visibility (*transparent* and *opaque*). We have been inspired by [Buc+21; Das+16; Flo+23; FSW19], where the authors provide the complete map of the relations among the 12 *transparent* models, but considering *collision-tolerant* robots. In this chapter, we have extended their investigation by considering also opaque robots and showing the relations between the transparent and the opaque framework. We have introduced six separator problems to prove the dominance or orthogonality relations between models: such problems work as separators both in the transparent and opaque models, showing that the relations existing in the opaque models still hold for the same transparent models. Thus far, the relations proved here in our opaque framework are the same as in the corresponding transparent framework. So, a natural question that arises is



(A) Hierarchy of the transparent models.

(B) Hierarchy of the opaque models.

FIGURE 2.5: Relation maps for the 12 models, in the transparent framework (Figure 2.5a) and in the opaque framework (Figure 2.5b). *Straight* (*dashed*, resp.) lines represent dominances (orthogonalities, resp.). *Dotted* lines join models whose relation is still unknown. The *red* relations hold both in case of transparency and opaqueness. The *black* relations are currently proved for the transparent models only.

whether the relation map for the opaque model is identical to that of the transparent model.

To answer this question, future research should address the missing relations among the 12 opaque models, in order to obtain the complete hierarchy in the opaque framework. Among others, the yet unknown relation between $\mathcal{LUMI}^{\mathtt{S}}_{\mathcal{O}}$ and $\mathcal{LUMI}^{\mathtt{A}}_{\mathcal{O}}$ is worth mentioning. In the transparent framework, these two models were proved in [Das+16] to be computationally equivalent by designing a simulator which, thanks to extra light colors, turns any `SSYNCH` algorithm into an equivalent `ASYNCH` algorithm. This simulator cannot be directly used to prove the same relation for opaque robots, mainly because of the obstructed visibility. With the `Pseudo-Polygon` problem, we have presented the *false election* phenomenon whose formalization and investigation will be preparatory to answer this interesting open question: is it possible to simulate a $\mathcal{LUMI}^{\mathtt{S}}_{\mathcal{O}}$ algorithm in the `ASYNCH` mode, thus proving that $\mathcal{LUMI}^{\mathtt{S}}_{\mathcal{O}}$ and $\mathcal{LUMI}^{\mathtt{A}}_{\mathcal{O}}$ are two equivalent models also in the opaque framework? Are constant-size lights sufficient to always avoid the phenomenon of false elections? In addition, it would be crucial to formalize and study all the critical issues caused by obstructed visibility: such formalizations may be essential for the correct investigation of the missing relations.

In conclusion, to obtain an even more complete analysis of the computational power of models under different visibility settings, it would be well worth extending this line of research to *myopic* models [Das+24]. Specifically, future work may focus on how the combination of transparency/opaqueness with unlimited visibility/short-sightedness can affect the computational power of robots.

Chapter 3

Sequential Robots

3.1 Introduction

As seen in Chapter 2, most of the research on the computational power of models, as well as the algorithmic research for mobile robots, has been conducted on the three synchronization modes `FSYNCH`, `SSYNCH`, and `ASYNCH`. Instead, the class of *sequential* modes (`SEQ`, `PERM`, and `RROBIN`) has received less attention. While broadly studied in other areas (e.g., cellular automata and self-stabilization), the investigation on the sequential modes in the LCM model has been confined to fault-tolerance [CF25; DPBT19], and more recently to gathering and pattern formation [Fel+25a; Flo+25; FW24; NP25]. Nonetheless, the limited research on such synchronization modes has already recognized their importance.

The interest and rationale behind sequential schedulers in distributed systems like swarms of robots might seem counterintuitive. As a matter of fact, one of the main desired and researched properties of distributed systems is *concurrency*. Increasing the concurrency of the robots' computation and the movements of robots may lead to undeniable advantages in terms of time. Yet, this is not always true: a (safe) parallelization has enormous costs since it relies on a proper synchronization of the robots and the avoidance of collisions or interferences. Therefore, the `FSYNCH` scheduler seems optimal in favoring the swarm performance in solving a task. However, in some cases, `FSYNCH` turns out to be an adversarial policy, especially if the task requires robots to never collide: in this case, synchronization may be left to the robots themselves, which have to decide whether and where to move to accomplish the task, avoiding crossing trajectories. Yet, this decision is not always possible. E.g., if in a mirror-symmetric configuration two symmetric robots are activated in the same round, they will execute the same (mirrored) action: in fact, they have no way to elect one of them, thus the two robots may travel synchronously along crossing trajectories. Besides collision avoidance, moving sequentially may be needed for further aims, for example, due to space limitations (robots must pass through narrow corridors). Or, robots need to stay idle for some periods to recharge (as theorized in [Buc+25]): to exploit the maximum idle period but to guarantee at least one operating robot at each time, a round-robin scheduler can be adopted. Hence, these practical advantages have been subject to recent theoretical studies, which have investigated how some problems can be solved under different sequential modes [CF25; Fel+25a; Flo+25; FW24; NP25].

Contribution

In this chapter, we start a general investigation of how the computational power of a swarm changes with the change of the sequential mode. Notably, we determine the exact relationship (dominance or equivalence) between X^{Y_1} and X^{Y_2}, for any

[0]The content of this chapter refers to the work [FFS25].

$X \in \{\mathcal{OBLOT}, \mathcal{FSTA}, \mathcal{FCOM}, \mathcal{LUMI}\}$ and $Y_1, Y_2 \in \{\texttt{SEQ}, \texttt{PERM}, \texttt{RROBIN}\}$. This study provides a first characterization of the computational landscape of the four base models under the sequential modes, and naturally extends the work of [Buc+21; Das+16; Flo+23; FSW19], which provides the computational landscape of the same four models under `FSYNCH`, `SSYNCH`, and `ASYNCH`. Note that here we consider the standard model of transparent and non-myopic robots.

Noteworthily, we prove that $\mathcal{LUMI}^{\texttt{SEQ}} \equiv \mathcal{LUMI}^{\texttt{PERM}}$ by providing a *simulator* that allows a swarm to correctly execute under `SEQ` any algorithm designed to run under `PERM`. Instead, all the other models are related by a dominance; to prove this, we provide a separator problem that is solvable in one model but unsolvable in the other. Except for one, the found relations (whether equivalences and dominances) hold independently from the fact that, under the hypothesis of X, we assume rigid or non-rigid movements, and fixed or variable disorientation.

Observation 4. In sequential modes, computational dominance implies that some problems require a specific class of activation sequences to be solved. Thus, the separators highlight some characteristics that make problems (un)solvable under specific sequences of activations, and allow us to discover valuable properties of the computational power of sequential schedulers. For example, being a problem solvable under `SEQ` means that the computability of the problem does not rely on any specific order of robot activations. Such problems are *robust to temporal blackouts* in a sequential scheduler; that is, even if the scheduler temporarily forgets the previous activations and resumes with a different sequence, the problem remains solvable. Instead, problems requiring `RROBIN` or `PERM` schedulers may not tolerate such interruptions, as their computability depends on a specific activation sequence pattern.

3.2 Models

This chapter compares 12 models: we here introduce all the *fixed features* that such models share, and the *variable features* under study.

Fixed features. Given a swarm $\mathcal{R}$, we assume that all robots enjoy the core features explained in Section 1.2.1 and:

- enjoy *complete visibility*, i.e., they are *transparent* and *non-myopic*;
- are *completely disoriented*, thus do not have any agreement on their local coordinate systems.
- are *collision-tolerant*, thus are allowed to form multiplicities at any time;
- have *strong multiplicity detection*, i.e., they can distinguish the number of robots with the same color in each multiplicity.

Variable features. We consider the 12 models in the form X^Y, where:

- $X \in \{\mathcal{OBLOT}, \mathcal{FSTA}, \mathcal{FCOM}, \mathcal{LUMI}\}$ represents the base model.
- For any X, we will specify if we consider it under NonRig or Rig, and FixDis or VarDis. We remind their meaning:
  1. Under Rig, each robot reaches the computed destination during the *Move* step (*rigid movements*). Under NonRig, each robot may be stopped during its *Move* step before reaching its destination, but after having traveled at least a fixed distance $\delta > 0$, whose value is unknown to the robots (*non-rigid movements*).

$\mathcal{OBLOT}$	RROBIN	$>$	PERM	$>$	SEQ
$\mathcal{FSTA}$	RROBIN	$>$	PERM	$>$	SEQ
$\mathcal{FCOM}$	RROBIN	$>$	PERM	$>^*$	SEQ
$\mathcal{LUMI}$	RROBIN	$>$	PERM	$\equiv$	SEQ

TABLE 3.1: Computational relations within each base model. All the relations (except for that denoted with *) hold independently from the fact that we assume FIXDIS/VARDIS and RIG/NONRIG.

2. Under FIXDIS, the local coordinate of each robot does not vary (*fixed disorientation*); under VARDIS, the local coordinate of each robot may vary from an activation to the next one (*variable disorientation*).

- $Y \in \{\texttt{SEQ}, \texttt{PERM}, \texttt{RROBIN}\}$ is the synchronization mode. We consider the following three sequential modes, one sub-mode of the previous one:
  - SEQ (i.e., only one robot is *fairly* activated at any round);
  - PERM (i.e., each epoch activates a permutation of the swarm in n rounds);
  - RROBIN (i.e., each epoch activates all robots in the same order).

  With a slight abuse of notation, we denote with SEQ, PERM, and RROBIN the respective sets of activation schedulers under the corresponding mode. So,
  - $\mathfrak{A} \in \texttt{SEQ}$ if $\mathfrak{A} : \mathbb{N}_0 \to \mathcal{R}$ where only the fair condition is satisfied, that is, for any $t \in \mathbb{N}_0$ and for any $r \in \mathcal{R}$, there exists a finite time $t' > t$ such that $\mathfrak{A}(t') = r$. Clearly, $\mathfrak{A}$ can be equivalently defined as the infinite sequence $\mathfrak{A} = r_{i_0} r_{i_1} \dots$ describing for any round $t \in \mathbb{N}_0$ the activated robot $r_{i_t} \in \mathcal{R}$;
  - $\mathfrak{A} \in \texttt{PERM}$ if $\mathfrak{A} = (r_{0_1} \dots r_{0_n})(r_{1_1} \dots r_{1_n}) \dots$ where $(r_{i_1} \dots r_{i_n})$ is a permutation of $\mathcal{R}$ for any $i \in \mathbb{N}_0$;
  - $\mathfrak{A} \in \texttt{RROBIN}$ if $\mathfrak{A} = (r_{i_1} \dots r_{i_n})^\infty$ for a permutation $(r_{i_1} \dots r_{i_n})$ of $\mathcal{R}$.

  It is self-evident that

$$\texttt{RROBIN} \subset \texttt{PERM} \subset \texttt{SEQ} \subset \texttt{SSYNCH}.$$

3.3 Taxonomy of the 12 sequential models

For any base model $X \in \{\mathcal{OBLOT}, \mathcal{FSTA}, \mathcal{FCOM}, \mathcal{LUMI}\}$ enjoying the fixed features explained above, we study how its computational power changes with the change of the sequential mode. Specifically, we find the computational relation that holds between the two models X^{Y_1}, X^{Y_2}, with $Y_1, Y_2 \in \{\texttt{SEQ}, \texttt{PERM}, \texttt{RROBIN}\}$. Such relations are summarized in Table 3.1.

Our results (except for one, highlighted with a star in Table 3.1) are true independently of the fact that, under the hypothesis of X, we assume RIG or NONRIG, and FIXDIS or VARDIS. More specifically, this means that when we prove the relation $X^{Y_1} \star X^{Y_2}$ with $\star \in \{\geq, >, \equiv\}$, we mean that

$$(X^{Y_1} \text{ assuming } H) \star (X^{Y_2} \text{ assuming } H)$$

for any hypothesis $H \in \{\text{RIG}, \text{NONRIG}\} \times \{\text{FIXDIS}, \text{VARDIS}\}$. Notably, this means that, if we prove that $X^{Y_1} \equiv X^{Y_2}$, the provided simulator is completely agnostic w.r.t. H. If, instead, X^{Y_1} and X^{Y_2} are separated by a problem, then the separator property does not lie on H. In other terms, even if we change H, the provided problem

remains a separator for X^{Y_1} and X^{Y_2}, thus highlighting the peculiar differences in the computational power under different synchronization modes.

The following theorem states the trivial equi-dominances that hold by definition and from which we start our study:

Theorem 23 (Trivial equi-dominances)**.** *Independently from* RIG-NONRIG *and* FIXDIS-VARDIS*, for any* $X \in \{\mathcal{OBLOT}, \mathcal{FSTA}, \mathcal{FCOM}, \mathcal{LUMI}\}$*, we have that*

$$X^{\texttt{RROBIN}} \geq X^{\texttt{PERM}} \geq X^{\texttt{SEQ}}.$$

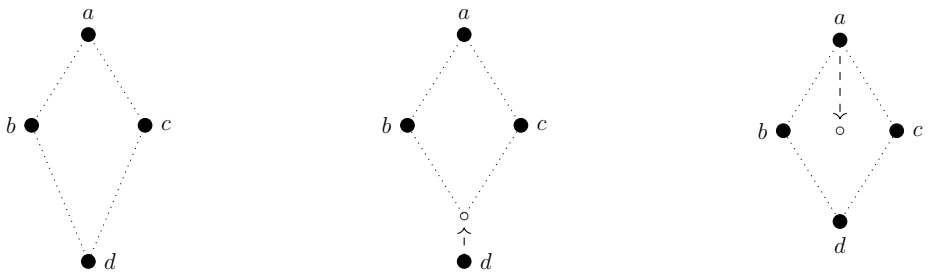
Note that the strict set-inclusion $\texttt{RROBIN} \subset \texttt{PERM} \subset \texttt{SEQ}$ does not necessarily imply a dominance in the power of the related models. As we will prove, $\mathcal{LUMI}$ robots can exploit lights to simulate the behavior of a more general scheduler mode (SEQ) within a more restricted one (PERM), thus demonstrating that $\mathcal{LUMI}^{\texttt{SEQ}} \equiv \mathcal{LUMI}^{\texttt{PERM}}$.

3.3.1 Relations in $\mathcal{OBLOT}$

We prove that, under $\mathcal{OBLOT}$, the relation between the three sequential modes is one of dominance.

$\mathcal{OBLOT}$	RROBIN	$>$	PERM	$>$	SEQ

Problem 9 (Kite)**.** Let $\mathcal{C}$ be an initial configuration where four robots a, b, c, d form a kite with $\angle bac = \frac{\pi}{3}$ and $\angle bdc = \frac{\pi}{6}$. Let h be the axis passing through a, d. From $\mathcal{C}$, robot d must move along h and stop so that the robots form a diamond (configuration $\mathcal{C}'$). From $\mathcal{C}'$, robot a must move along h and stop in the center of the diamond, thus forming the final configuration $\mathcal{C}''$. Refer Figure 3.1.



(A) Initial configuration $\mathcal{C}$. (B) Formation of $\mathcal{C}'$. (C) Formation of $\mathcal{C}''$.

FIGURE 3.1: Configurations in Kite.

Lemma 17. *Kite* $\in \mathcal{P}\left(\mathcal{OBLOT}^{\texttt{RROBIN}}\right)$ *even assuming* NONRIG *and* VARDIS*.*

Proof. From $\mathcal{C}$, as soon as d is activated, it correctly elects itself as the robot that has to move and forms $\mathcal{C}'$. After its activation, the robots a, b, c will be activated (in some order) before the re-activation of d. Thus, when a is activated in $\mathcal{C}'$, it can safely move to form the final configuration. Note that the problem is solvable even considering non-rigid movements: in fact, if d or a are stopped along their trajectories, their angles belong to two different ranges ($\angle bdc \in (\frac{\pi}{6}, \frac{\pi}{3}]$, $\angle bac \in (\frac{\pi}{3}, \pi]$), thus making the configuration unambiguous. □

Lemma 18. *Kite* $\notin \mathcal{P}\left(\mathcal{OBLOT}^{\texttt{PERM}}\right)$ *even assuming* FIXDIS *and* RIG*.*

Proof. By contradiction, let $\mathbb{A}$ be an algorithm solving Kite under PERM. Thus, $\mathbb{A}$ must solve the problem even under the PERM scheduler $\mathfrak{A} = (abcd)^\infty$. In the first epoch of $\mathfrak{A}$, d moves to form $\mathcal{C}'$. In the second epoch of $\mathfrak{A}$, a performs the last movement running $\mathbb{A}(\sigma)$ where σ is the snapshot taken by a. Now, let $\mathfrak{A}' = (abcd)(dabc)^\infty$ be a PERM scheduler that behaves like $\mathfrak{A}$ for the first epoch, and then starts the second epoch with d. So, in the first epoch of $\mathfrak{A}'$, d moves and reaches $\mathcal{C}'$ as under $\mathfrak{A}$. At the beginning of the second epoch of $\mathfrak{A}'$, d may perceive the swarm with the same snapshot σ as a under $\mathfrak{A}$. This is due to the mirror-symmetric configuration, and the fact that d may have a coordinate system which is symmetric w.r.t. the coordinate system of a. Thus, d performs a movement according to $\mathbb{A}(\sigma)$. This contradicts the problem's request. □

Theorem 24. *Independently from* Rig-NonRig *and* FixDis-VarDis*:*

$$\mathcal{OBLOT}^{RROBIN} > \mathcal{OBLOT}^{PERM}.$$

Proof. By Theorem 23, we know that $\mathcal{OBLOT}^{\texttt{RROBIN}} \geq \mathcal{OBLOT}^{\texttt{PERM}}$. Kite plays the role of separator between the two models (by Lemmas 17 and 18). □

Problem 10 (Pinwheel)**.** The problem is perpetual, and it is defined recursively. Let $\mathcal{C}$ be a configuration where a swarm of robots is arranged on the vertices of two different acute scalene triangles, say $\triangle ABC$ and $\triangle abc$, with the same shape, orientation, and circumcenter. Assume that $\triangle abc$ is completely contained in $\triangle ABC$, and only one robot lies on each vertex of $\triangle abc$. Let $m \geq 3$ be the total number of robots lying on the vertices of $\triangle ABC$. A multiplicity of $m+1$ robots lies on the shared circumcenter of the two triangles. Assume that $\angle bac > \angle abc > \angle acb$: thus, $\overset{\frown}{abc}$ establishes a global clockwise orientation to the swarm. Starting from $\mathcal{C}$, the robots are asked to form a configuration $\mathcal{C}''$ where each multiplicity of $\triangle ABC$ has shifted to the adjacent vertex following the orientation $\overset{\frown}{abc}$. From $\mathcal{C}$ to $\mathcal{C}'$, each robot of $\triangle ABC$ must travel only along the edge connecting itself with the related target vertex, following the clockwise direction. Recursively, the problem demands the same request starting from $\mathcal{C}'$. Refer to Figure 3.2.

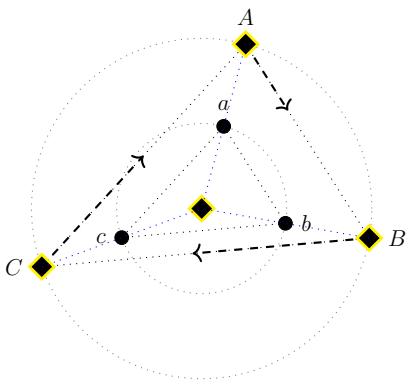


Figure 3.2: Pinwheel. Multiplicities are represented through yellow-shaded diamonds.

Lemma 19. *Pinwheel* $\in \mathcal{P}\left(\mathcal{OBLOT}^{PERM}\right)$ *even assuming* NonRig *and* VarDis*.*

Proof. Firstly, note that for any correct algorithm solving Pinwheel, the $m+1$ robots on the circumcenter of the two triangles will always form the most crowed multiplicity

in the swarm: this invariant ensures that, at any time, the robots can easily reconstruct the circles where the two triangles are inscribed. We now show how Pinwheel is solved under $\mathcal{OBLOT}^{\texttt{PERM}}$. Every time a robot is activated from a vertex of $\triangle ABC$, it can easily compute its target vertex according to the clockwise direction given by the angles of the $\triangle abc$ and move there. Suppose a vertex of the original $\triangle ABC$ gets empty (since all its robots have migrated to the adjacent vertex). In that case, it is still possible for a robot to recompute $\triangle ABC$ thanks to the presence of the similar triangle $\triangle abc$ (which provides the angles and the position of $\triangle ABC$). Note also that, under NonRig, if a robot r is stopped along one edge of $\triangle ABC$, then all the other robots will recognize this situation (since there are 8 distinct occupied points on the plane or since the 7 occupied points of the plane do not compose a feasible configuration) and will do nothing until r is activated in the next epoch(s) and it will correctly recover its trajectory. A PERM scheduling guarantees all the robots are activated only once at each epoch, thus forming $\mathcal{C}'$ from $\mathcal{C}$ according to the problem requirements. □

Lemma 20. *Pinwheel* $\notin \mathcal{P}\left(\mathcal{OBLOT}^{SEQ}\right)$ *even assuming* Rig *and* FixDis.

Proof. By contradiction, let $\mathbb{A}$ be an algorithm solving the problem under SEQ. Suppose that all the robots have the same coordinate system. Let t be a time when a robot r is activated and correctly moves from A to B (w.l.o.g.), starting from an intermediate configuration (between configuration $\mathcal{C}$ and configuration $\mathcal{C}'$) where A contains some robots $\mathcal{R}_1$ that already moved in the current epoch and others ($\mathcal{R}_2$) that have not moved yet. However, if at time t the scheduler had activated a robot in A from the set $\mathcal{R}_1$, such a robot would have taken the same snapshot as r, thus performing the same action by $\mathbb{A}$. This contradicts the request of the problem. □

Theorem 25. *Independently from* Rig-NonRig *and* FixDis-VarDis*:*

$$\mathcal{OBLOT}^{PERM} > \mathcal{OBLOT}^{PERM}.$$

Proof. By Theorem 23, we know that $\mathcal{OBLOT}^{\texttt{PERM}} \geq \mathcal{OBLOT}^{\texttt{SEQ}}$. Pinwheel plays the role of separator between the two models (by Lemmas 19 and 20). □

3.3.2 Relations in $\mathcal{LUMI}$

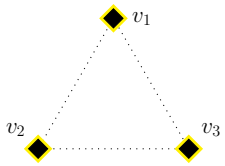
For $\mathcal{LUMI}$, we prove that RROBIN and PERM are related by a dominance, while any problem solved in PERM can be solved under SEQ, thus proving the equivalence for these two models.

$\mathcal{LUMI}$	RROBIN	$>$	PERM	$\equiv$	SEQ

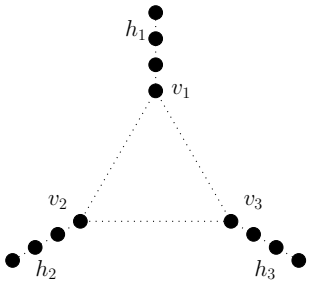
Problem 11 (Triline)**.** Let us assume a swarm is arranged on the vertices of an equilateral triangle $\triangle v_1v_2v_3$, so that at least four robots lie on each vertex. The robots of each vertex v_i are required to (1) arrange along a line h_i, each robot in a distinct point; (2) come back to v_i; (3) arrange on h_i, occupying each the same position as before. The sequence of these tasks must be performed perpetually. Refer to Figure 3.3.

Lemma 21. *Triline* $\in \mathcal{P}\left(\mathcal{FCOM}^{RROBIN}\right)$ *even assuming* VarDis *and* NonRig.

Proof. For any vertex v_i of $\triangle v_1v_2v_3$, we define h_i as the axis of the triangle passing through v_i. We propose an algorithm that makes robots on v_i uniformly arranged on the external semi-line of h_i starting from v_i, and so that their uniform mutual distance



(A) Initial configuration of three multiplicities (diamonds with yellow shadows) forming $\triangle v_1 v_2 v_3$.



(B) Possible arrangement on the axes h_1, h_2, h_3 solving task (1).

FIGURE 3.3: Configurations in Triline.

is $\frac{\ell}{a}$ where ℓ is the length of the edge of the triangle and $a > 1$ is a constant of the algorithm (see Figure 3.3b). Assuming m robots lie on v_i, we define the target points for the m robots as the points $U_1, U_2, \ldots, U_m$ on h_i such that $dist(U_j, v_i) = \frac{\ell}{a}(m - j)$ (see Figure 3.4). Indeed, U_m coincides with v_i. We solve the problem using the colors $\{\texttt{OFF}, \texttt{FIRST}, \texttt{LAST}, \texttt{ON}, \texttt{MOVING}\}$. We describe the evolution of the algorithm for a single vertex v_i: the same strategy is applied to the other two vertices independently. Starting from the initial OFF-colored configuration[1], in the first initialization epoch, robots stay still and the first (last, resp.) robot activated initializes to FIRST (LAST, resp.) while all the other $m-2$ robots turn into ON. Being under RROBIN, the robots will always be activated in the same order. The FIRST robot is intended for the target point U_1, while the LAST one is intended for U_m (thus, it never moves); the ON ones must sequentially arrange on $U_2, \ldots, U_{m-1}$ following their activation order. Note that, even if a robot cannot see its own light, it can detect its color and therefore its role, by excluding the other colors on h_i.

After the initialization epoch, the FIRST robot is activated and travels to U_1. The other robots can note if FIRST has not reached U_1: in that case, they skip their turn, allowing FIRST to continue its path towards its destination. Once FIRST has arranged on U_1, each ON robot moves and reaches the farthest free target point among $U_2, \ldots, U_{m-1}$. Even in this case, if a robot is stopped before reaching its U_j, all the other robots will skip their turn: in other words, a robot starts moving towards U_{j+1} only if the previous points $U_1, \ldots, U_j$ are covered by robots. The LAST robot will always occupy the last target point U_m.

Once all robots have reached the m target points, a new epoch begins and FIRST moves towards v_i. Then, it is the turn of the ON robots: before moving, they set their color to MOVING. This extra color is needed to avoid ambiguity in case, during the inward migration from U_j to v_i, a robot is stopped and creates a multiplicity in $U_{j+1}, \ldots, U_{m-1}$. As a matter of fact, if a FIRST or a MOVING robot creates a multiplicity with a ON robot, it is always evident by the colors which is the stopped robot. No other robot moves until it sees a MOVING robot: once a MOVING robot reaches v_i, it resets its light to ON. Being activated in the same order, robots are guaranteed to always reach the same target points. Algorithm 2 shows the pseudo-code of the presented algorithm. □

From Lemma 21, it straightforwardly follows that:

Corollary 5. *Triline* $\in \mathcal{P}\left(\mathcal{LUMI}^{\texttt{RROBIN}}\right)$ *even assuming* NONRIG *and* VARDIS.

[1]For the sake of clarity, we often use the OFF color in place of $\not\emptyset$ as the starting color.

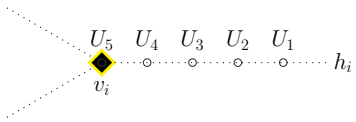


FIGURE 3.4: Triline. Target points along h_i of the m robots belonging to v_i (here, $m = 5$).

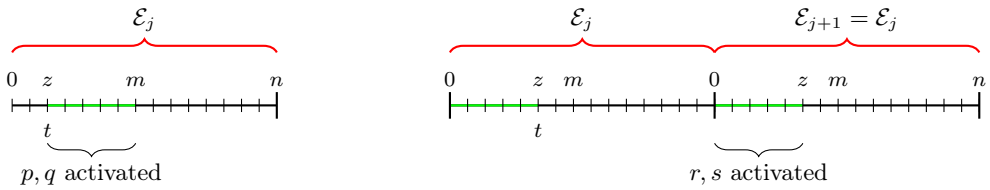
Algorithm 2: Solution for Triline under $\mathcal{FCOM}^{\texttt{RROBIN}}$.

Input: σ snapshot of the swarm $\mathcal{R}$ taken by a robot r
Output: $(\underline{x}, c) \in (\mathbb{R}^2 \times \mathfrak{L})$ new state of r to be set in the Move step, namely
- $\underline{x}$ is the target position to be reached by r;
- c is the color to which $\texttt{lig}_r$ will be set;

We use the special value $-$ to indicate that the relative state value remains unchanged.
Procedure $\mathbb{A}(\sigma)$**:**
- $V_r \leftarrow$ colors of the other robots seen by r along h_i;
- **switch** V_r **do**
  - **case** $V_r = \{\texttt{OFF}\}$ **do**
    - `// Initialization epoch`
    - **return** $(-, \texttt{FIRST})$;
  - **case** $V_r \subseteq \{\texttt{OFF}, \texttt{FIRST}, \texttt{ON}\}$ and $\texttt{OFF}, \texttt{FIRST} \in V_r$ **do**
    - **return** $(-, \texttt{ON})$;
  - **case** $V_r = \{\texttt{FIRST}, \texttt{ON}\}$ **do**
    - **return** $(-, \texttt{LAST})$;
  - **case** $V_r = \{\texttt{LAST}, \texttt{ON}\}$ **do**
    - `// Then` r `is FIRST`
    - **if** $r \in h_i \setminus \{U_1\}$ and all other robots are on v_i **then**
      - **return** $(U_1, -)$;
    - **if** $r \in h_i \setminus \{v_i\}$ and all $U_2, \ldots, U_m$ are occupied **then**
      - **return** $(v_i, -)$;
  - **case** $V_r = \{\texttt{FIRST}, \texttt{ON}, \texttt{LAST}\}$ **do**
    - `// Then` r `is ON/MOVING`
    - **if** $\texttt{FIRST} \in \{U_1\}$ and $U_1 \ldots U_j$ are occupied and $U_{j+1} \ldots U_{m-1}$ are free **then**
      - **if** $r \in h_i \setminus \{U_2, \ldots, U_j\}$ **then**
        - **return** $(U_{j+1}, -)$;
    - **if** $\texttt{FIRST} \in \{v_i\}$ and $U_1 \ldots U_j$ are free and $U_{j+1} \ldots U_{m-1}$ are occupied **then**
      - **if** $r \in h_i \setminus \{v_i\}$ **then**
        - **return** $(v_i, \texttt{MOVING})$;
      - **else**
        - **return** $(-, \texttt{ON})$;
  - **otherwise do**
    - `// In all the other cases, do nothing`
    - **return** $(-, -)$;

Lemma 22. *Triline* $\notin \mathcal{P}\left(\mathcal{LUMI}^{\texttt{PERM}}\right)$ *even assuming* RIG *and* FIXDIS.

Proof. By contradiction, assume Triline is solved under $\mathcal{LUMI}^{\texttt{PERM}}$ by an algorithm $\mathbb{A}$ which uses k colors. Let us consider an initial configuration of the problem with a swarm of n robots and $m \geq 2k+1$ robots on v_1. Let $\mathfrak{A} = \mathcal{E}_1\mathcal{E}_2 \ldots$ be a PERM scheduler



(A) Case $z \leq k$. (B) Case $z > k$.

FIGURE 3.5: Epochs in $\mathfrak{A}$ for Triline.

where the epoch $\mathcal{E}_i$ defines a permutation of the swarm. For the sake of simplicity, we assume that each $\mathcal{E}_i$ activates first the m robots in v_1. Let ℓ_1 be the line chosen by $\mathbb{A}$ on which the robots of v_1 arrange themselves. Let t be the time when all m robots have gathered again in v_1, and have to rearrange on ℓ_1 in their previous positions. Assume t occurs during epoch $\mathcal{E}_j$, and let $z = t \bmod n$ be the round within $\mathcal{E}_j$ corresponding to t. Note that, since we assume that the robots of v_1 are activated first in $\mathcal{E}_j$, we have that $z < m$. Suppose that $\mathcal{E}_{j+1} = \mathcal{E}_j$ in $\mathfrak{A}$. Reminding that $m \geq 2k + 1$, we show that two clone robots (thus with the same color and same position) exist in both scenarios (refer to Figure 3.5):

- if $z \leq k$, then $m - z > k$. So, at least two robots p, q have the same color in v_1 at time t, and they will be activated in the next $m - z$ rounds of $\mathcal{E}_j$ from t onward (see Figure 3.5a);
- otherwise, at least two robots r, s have the same color in v_1 at time t, and they will be activated in the first z rounds of $\mathcal{E}_{j+1}$, which is equal to $\mathcal{E}_j$ (see Figure 3.5b).

Thus, we construct two PERM schedulers, namely $\mathfrak{A}'$ (for $z \leq k$) and $\mathfrak{A}''$ (for $z > k$), that make the swarm fail to satisfy the problem request in the corresponding scenario. Notably, $\mathfrak{A}'$ is equal to $\mathfrak{A}$ but the activations of p and q are swapped from $\mathcal{E}_j$ onward; instead, $\mathfrak{A}''$ is obtained by $\mathfrak{A}$ by swapping the activation of r and s from $\mathcal{E}_{j+1}$ onward. Thus, under $\mathfrak{A}'$, from t onward, robot p (q, resp.) will execute the same actions as q (p, resp.) under $\mathfrak{A}$, and so it will reach the position intended for q (p, resp.). This contradicts the request of the problem in the case $z \leq k$. For the case $z > k$, we achieve the same contradiction using $\mathfrak{A}''$. □

Theorem 26. *Independently from* RIG-NONRIG *and* FIXDIS-VARDIS*:*

$$\mathcal{LUMI}^{\mathit{RROBIN}} > \mathcal{LUMI}^{\mathit{PERM}}.$$

Proof. By Theorem 23, we know that $\mathcal{LUMI}^{\texttt{RROBIN}} \geq \mathcal{LUMI}^{\texttt{PERM}}$. Triline plays the role of separator between the two models (by Corollary 5 and Lemma 22). □

Theorem 27. *Independently from* RIG-NONRIG *and* FIXDIS-VARDIS*:*

$$\mathcal{LUMI}^{\mathit{PERM}} \equiv \mathcal{LUMI}^{\mathit{SEQ}}.$$

Proof. By Theorem 23, we already know the trivial equi-dominance $\mathcal{LUMI}^{\texttt{PERM}} \geq \mathcal{LUMI}^{\texttt{SEQ}}$. We now constructively prove the reverse equi-dominance $\mathcal{LUMI}^{\texttt{SEQ}} \geq \mathcal{LUMI}^{\texttt{PERM}}$, which states that any problem P solvable in $\mathcal{LUMI}^{\texttt{PERM}}$ can be solved under $\mathcal{LUMI}^{\texttt{SEQ}}$. Let $\mathbb{A}$ be an algorithm solving P under $\mathcal{LUMI}^{\texttt{PERM}}$ using the colors in $\mathfrak{L}$. We design a simulator algorithm, $\mathbb{A}_{\texttt{SEQ}}$ for solving P under $\mathcal{LUMI}^{\texttt{SEQ}}$ using an

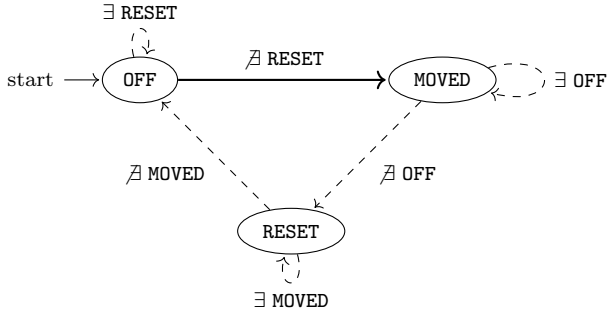


FIGURE 3.6: Transition diagram of the color of light $\texttt{aux}_r$ according to $\mathbb{A}_{\texttt{SEQ}}$. Transition conditions refer only to auxiliary lights. Dashed edges represent color transition occurring in LCM cycles where r does nothing except change $\texttt{aux}_r$.

extended palette $\mathfrak{L}_{\texttt{SEQ}} = \mathfrak{L} \times \{\texttt{OFF}, \texttt{MOVED}, \texttt{RESET}\}$. Thus, without loss of generality, we assume that each robot r is endowed with an auxiliary light $\texttt{aux}_r$ (in addition to the standard light $\texttt{lig}_r$), initially set to OFF. The update of $\texttt{aux}_r$ occurs simultaneously with the update of $\texttt{lig}_r$ (when it occurs). The key idea is to simulate one epoch of $\mathcal{LUMI}^{\texttt{PERM}}$ within a mega-epoch[2] in $\mathcal{LUMI}^{\texttt{SEQ}}$. A mega-epoch consists of three consecutive $\mathcal{LUMI}^{\texttt{SEQ}}$ epochs:

1. **Execution epoch:** each robot r performs the original algorithm $\mathbb{A}$ only once, by updating its $\texttt{aux}_r$ from OFF to MOVED. This color marks that r has already been activated within the first epoch; therefore, it will skip its turn if it is activated further times during this epoch.
2. **Synchronization epoch:** when all the robots are set to MOVED, each robot r updates its $\texttt{aux}_r$ to RESET.
3. **Reset epoch:** when all the robots are set to RESET, each robot r resets its $\texttt{aux}_r$ to OFF, thus allowing the swarm to perpetually perform a mega-epoch $(\texttt{OFF} \to \texttt{MOVED} \to \texttt{RESET})^{\infty}$.

The three epochs are marked and conditioned by the colors of the auxiliary lights. We now describe in detail the actions of a robot r executing $\mathbb{A}_{\texttt{SEQ}}$:

1. (Case $\texttt{aux}_r = \texttt{OFF}$) If r observes no RESET-colored robots, it is aware that this is its first activation during the first epoch of the current mega-epoch. So, r acts as follows:

   - r takes the snapshot σ of the swarm, and computes the reduced version $\sigma_{\mathbb{A}}$, removing the colors of the aux variables;
   - r executes the original algorithm $\mathbb{A}(\sigma_{\mathbb{A}})$, and it possibly updates the color of its $\texttt{lig}_r$, and moves to the computed position;
   - simultaneously to the update of $\texttt{lig}_r$, it sets $\texttt{aux}_r \leftarrow \texttt{MOVED}$ to mark that it has already performed its LCM cycle for this epoch.

   Otherwise (i.e., r observes some RESET-colored robots), r does nothing.

2. (Case $\texttt{aux}_r = \texttt{MOVED}$) If r observes no OFF-colored robots, it updates $\texttt{aux}_r \leftarrow \texttt{RESET}$, doing nothing else. Otherwise, r is aware that it has already been activated during the current epoch and that other robots (those with the aux

[2] Note that the concept of simulating an epoch through a mega-epoch has already been investigated in [Das+16] for proving $\mathcal{OBLOT}^{\texttt{S}} \leq \mathcal{LUMI}^{\texttt{A}}$.

light to OFF) must still be activated in this epoch. Thus, r skips its turn (i.e., it does nothing).

3. (Case $\texttt{aux}_r = \texttt{RESET}$) If r observes no MOVED robots, then it updates $\texttt{aux}_r \leftarrow \texttt{OFF}$, doing nothing else. Otherwise, it does nothing.

Algorithm 3 shows the pseudo-code of $\mathbb{A}_{\texttt{SEQ}}$, while Figure 3.6 depicts the transition diagram of color of $\texttt{aux}_r$ according to $\mathbb{A}_{\texttt{SEQ}}$. It is straightforward to note that a mega-epoch of $\mathbb{A}_{\texttt{SEQ}}$ simulates the behavior of an epoch of $\mathbb{A}$ under PERM. In fact, the main algorithm $\mathbb{A}$ is executed only once in the execution epoch (OFF $\rightarrow$ MOVED). The MOVED color is used both to memorize the already-done action during the first epoch, and to synchronize robots to proceed with the next epoch (MOVED $\rightarrow$ RESET). The reset epoch (thus a third color transition RESET $\rightarrow$ OFF) is necessary for the correctness of the simulator: having only two values (OFF and MOVED), robots may be unable to distinguish between a configuration transitioning from OFF $\rightarrow$ MOVED and one transitioning from MOVED $\rightarrow$ OFF. □

Algorithm 3: Algorithm $\mathbb{A}_{\texttt{SEQ}}$ simulating an $\mathcal{LUMI}^{\texttt{PERM}}$ algorithm $\mathbb{A}$ under the $\mathcal{LUMI}^{\texttt{SEQ}}$ model.

Input: σ snapshot of the swarm $\mathcal{R}$ taken by a robot r
Output: $(\underline{x}, c, a) \in (\mathbb{R}^2 \times \mathfrak{L} \times \{\texttt{OFF}, \texttt{MOVED}, \texttt{RESET}\})$ such that
- $\underline{x}$ is the target position to be reached by r;
- c and a are the colors to which $\texttt{lig}_r$ and $\texttt{aux}_r$ will be set simultaneously;

We use the special value $-$ to indicate that the relative state value remains unchanged.
Procedure $\mathbb{A}_{SEQ}(\sigma)$**:**

```
switch aux_r do
    case OFF do
        if ∄ r' ∈ R such that aux_r' = RESET then
            σ_A ← σ without the aux lights;
            // Simulate the execution of A
            (x, c) ← A(σ_A);
            // Set lights to c and MOVED, and move to x
            return (x, c, MOVED);
        // Do nothing
        return (−, −, −);
    case MOVED do
        if ∄ r' ∈ R such that aux_r' = OFF then
            // Reset aux light to RESET
            return (−, −, RESET);
        // Do nothing
        return (−, −, −);
    case RESET do
        if ∄ r' ∈ R such that aux_r' = MOVED then
            // Reset aux light to OFF
            return (−, −, OFF);
        // Do nothing
        return (−, −, −);
```

3.3.3 Relations in $\mathcal{FSTA}$

We prove that, under $\mathcal{FSTA}$, the relation between the three sequential modes is one of dominance.

$\mathcal{FSTA}$	RROBIN	$>$	PERM	$>$	SEQ

Lemma 23. *Triline* $\in \mathcal{P}\left(\mathcal{FSTA}^{RROBIN}\right)$ *even assuming* NonRig *and* VarDis.

Proof. Let us design an algorithm solving Triline with {OFF, OUT, M_OUT, IN, LAST} as palette, with OFF being the initialization color of any robot r. As in Lemma 21, we make the m robots of a vertex v_i arrange themselves on the points $U_1, U_2, \ldots, U_m$ of the axis h_i (see Figure 3.4). However, instead of implementing a FIFO strategy to make robots arrange themselves on h_i and gather again on v_i, we here use a LIFO one. In the first epoch, all robots simply change their state from OFF to OUT, indicating their next action of moving out of v_i. Now, the robots, one at a time, travel along h_i and reach their target point: specifically, for any $j = 1, \ldots, m-1$, the j-th robot activated in v_i sets to M_OUT and heads U_j. Note that the j-th OUT robot starts moving along h_i only when the previous robots have reached their target points $U_1, \ldots, U_{j-1}$. Being under NonRig, this arrangement may last multiple epochs. The last robot remaining on v_i will set its color to LAST: from now on, the LAST robot will never change either its state or its position.

Once all the M_OUT robots have uniformly arranged along h_i, they sequentially update their color into IN. Now, starting from the IN robot in U_{m-1}, all the IN robots, one at a time, set their color to OFF and travel along h_i to reach v_i. Even in this case, the IN robot on U_j starts moving towards v_i only if the robots that were on $U_{j+1} \ldots U_{m-1}$ have reached v_i (possibly in multiple epochs). Lastly, the robot on U_1 (which is the last to come back to v_i) sets its color to OUT (instead of OFF) before heading to v_i. Being under NonRig, the LIFO scheme ensures robots never create multiplicities with other robots if they are stopped along their trajectories on $h_i \setminus \{v_i\}$. After the robot on U_1 (i.e., the first to be activated in the group) has reached v_i with the color OUT, the task can be repeated: so, the other $m-1$ robots of v_i will be activated in the same order and thus they will change their color from OFF to OUT, thereby repeating the entire task. Figure 3.7 shows the transition diagram of the light of a robot r: the conditions defined on the transition edges unambiguously lead r to identify its next action. □

The next corollary follows directly from Lemma 22.

Corollary 6. *Triline* $\notin \mathcal{P}\left(\mathcal{FSTA}^{PERM}\right)$ *even assuming* Rig *and* FixDis.

Theorem 28. *Independently from* Rig-NonRig *and* FixDis-VarDis*:*

$$\mathcal{FSTA}^{RROBIN} > \mathcal{FSTA}^{PERM}.$$

Proof. By Theorem 23, we know that $\mathcal{FSTA}^{\texttt{RROBIN}} \geq \mathcal{FSTA}^{\texttt{PERM}}$. Triline plays the role of separator between the two models (by Corollary 6 and Lemma 23). □

Problem 12 (Go-If-Odd)**.** Consider an initial configuration with a swarm arranged on three aligned points, say a, b, c, so that $dist(a, b) = 2dist(b, c)$. Only one robot, say r_a, lies on a, while two groups of robots lie respectively on b and on c. The problem asks all the robots in b to move to c. Lastly, if the original number of robots in b was odd, then r_a is asked to move to c; otherwise, r_a must stay in a. Refer to Figure 3.8.

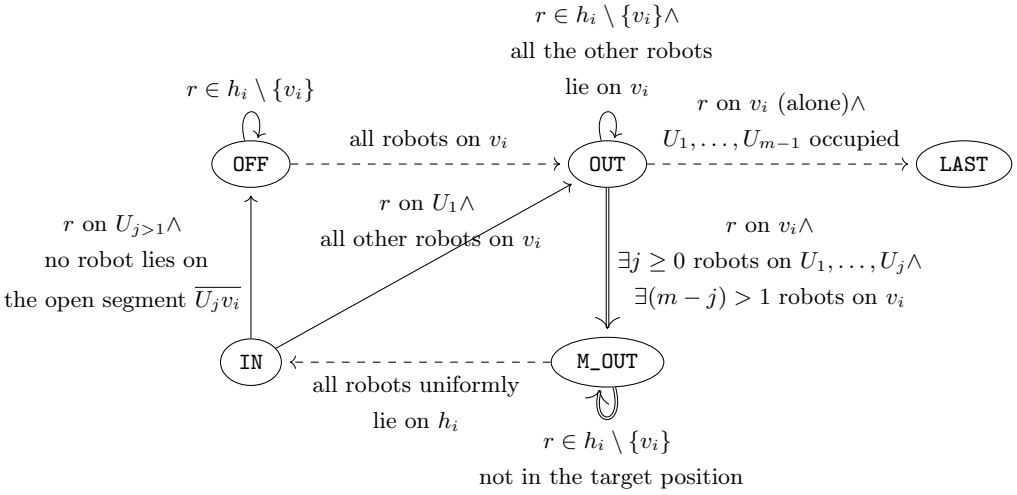
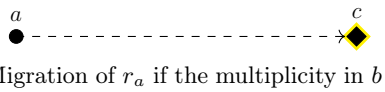


FIGURE 3.7: Transition diagram of $\texttt{lig}_r$ while solving Triline under $\mathcal{FSTA}^{\texttt{RROBIN}}$. Dashed edges represent color transition in LCM cycles where r executes a null movement. Single-line (double-line, resp.) edges represent movements inward (outward, resp.) v_i.

(A) Migration from b to c.



(B) Migration of r_a if the multiplicity in b was odd.

FIGURE 3.8: Go-If-Odd problem.

Lemma 24. *Go-If-Odd* $\in \mathcal{P}\left(\mathcal{FSTA}^{\texttt{PERM}}\right)$ *even assuming* NONRIG *and* VARDIS.

Proof. In the first epoch, the robots set their internal lights in this way: all the robots in b (c, resp.) set to B (C, resp.), while r_a sets to ODD or EVEN according to the parity of the robots in b. In the second epoch, once a robot is activated, it sees that it has already set its light, which means that the first epoch has finished. So, according to its role, the robot moves to c or stays still. Once r_a sees only two points, it reaches the other point only if it is ODD-colored. □

Lemma 25. *Go-If-Odd* $\notin \mathcal{P}\left(\mathcal{FSTA}^{\texttt{SEQ}}\right)$ *even assuming* RIG *and* FIXDIS.

Proof. Intuitively, the impossibility holds since the robots in b have no means to understand if a has been activated and if it has memorized the parity of b. Formally, let $\mathbb{A}$ be an algorithm that, by contradiction, solves the problem; assume $\mathbb{A}$ uses k colors (i.e., internal states). Let us consider a problem instance where the number of robots in b is odd. Let $\mathfrak{A}$ be a SEQ scheduler under which the swarm $\mathcal{R}$ executes $\mathbb{A}$ and reaches a configuration at time t where b is empty, and then it activates r_a for k rounds $t+1, t+2, t+3, \ldots, t+k$. Since $\mathbb{A}$ is correct, r_a must move from a towards c after a finite (and constant) number of rounds, say at $t+i$, with $i \leq k$. Let us now consider the scheduler $\mathfrak{A}'$ which removes from $\mathfrak{A}$ all the activations of r_a until round t (for the sake of simplicity and w.l.o.g., let us assume that $\mathfrak{A}'$ activates no robots in correspondence of the rounds $t' < t$ where r_a was activated under $\mathfrak{A}$). So, robots in $\mathcal{R} \setminus \{r_a\}$ throughout $\mathfrak{A}'$ will behave as under $\mathfrak{A}$ until t (in fact, the snapshots of the robots will be the same). However, at times $t+1, t+2, t+3, \ldots, t+k$, the robot in a has no information about the original parity of b (which cannot be derived by the actual number of robots in c), so it does not know the correct action to be performed. Contradiction achieved. □

Theorem 29. *Independently from* Rig-NonRig *and* FixDis-VarDis*:*

$$\mathcal{FSTA}^{PERM} > \mathcal{FSTA}^{SEQ}.$$

Proof. By Theorem 23, we know that $\mathcal{FSTA}^{\texttt{PERM}} \geq \mathcal{FSTA}^{\texttt{SEQ}}$. Go-If-Odd plays the role of separator between the two models (by Lemmas 24 and 25). □

3.3.4 Relations in $\mathcal{FCOM}$

We prove that, under $\mathcal{FCOM}$, the relation between the three sequential modes is one of dominance. However, the dominance between PERM and SEQ holds assuming VarDis.

$\mathcal{FCOM}$	RROBIN	$>$	PERM	$>^*$	SEQ

By Lemma 22, it straightforwardly follows that:

Corollary 7. *Triline* $\notin \mathcal{P}\left(\mathcal{FCOM}^{PERM}\right)$ *even assuming* Rig *and* FixDis*.*

Theorem 30. *Independently from* Rig-NonRig *and* FixDis-VarDis*:*

$$\mathcal{FCOM}^{RROBIN} > \mathcal{FCOM}^{PERM}.$$

Proof. By Theorem 23, we know that $\mathcal{FCOM}^{\texttt{RROBIN}} \geq \mathcal{FCOM}^{\texttt{PERM}}$. Triline plays the role of separator between the two models (by Lemma 21 and Corollary 7). □

Problem 13 (ApproachTwice)**.** Let $\mathcal{C}$ be a configuration where two robots p, q lie on the same point, while a third robot r lies on a distinct point. Let γ be the line where the robots lie, and let $d = dist(p, r) > 0$ be their distance[3]. Robot r must move along γ and stop in a configuration $\mathcal{C}''$ in which the new distance of the robots, say d'', is such that $0 < d'' < d$. From $\mathcal{C}$ to $\mathcal{C}''$, r is allowed to stop *at most once* and form an intermediate configuration $\mathcal{C}'$ in which the distance of the robots, say d', is such that $d'' < d' < d$. Refer to Figure 3.9.

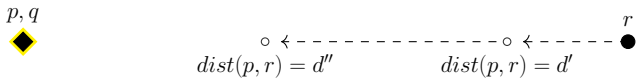


Figure 3.9: ApproachTwice problem.

Lemma 26. *ApproachTwice* $\in \mathcal{P}\left(\mathcal{FCOM}^{PERM}\right)$ *even assuming* NonRig *and* VarDis*.*

Proof. The problem is solved in at most 2 epochs, making r perform at most two non-null movements along γ towards the pair p, q. In the first epoch, r approaches p, q, setting its color to MOVED. If p or q are activated (either in the first epoch or in the second) before the reactivation of r, they see that r is MOVED-colored, they set their color to END and terminate. Upon seeing an END color, all the robots terminate, having formed $\mathcal{C}''$ directly from $\mathcal{C}$. If instead r is activated as last in the first epoch and again as first in the second epoch, it may perceive the exact same situation (being under VarDis) and it may execute another non-null movement (i.e., forming $\mathcal{C}''$ from $\mathcal{C}'$). After that, p and q will be activated and they will see that r is MOVED-colored: so they will set their color to END, thus making the algorithm terminate. □

[3]According to an absolute coordinate system.

Lemma 27. *ApproachTwice* $\notin \mathcal{P}\left(\mathcal{FCOM}^{SEQ}\right)$ *even assuming* Rig*, but assuming* VarDis*.*

Proof. By contradiction, let $\mathbb{A}$ be an algorithm solving ApproachTwice under $\mathcal{FCOM}^{\texttt{SEQ}}$. Let $\mathfrak{A}$ be a SEQ scheduler, and let t be the first time when r approaches. Informally, $\mathbb{A}(\sigma) =$ "*approach*", where σ is the taken snapshot. Let us now consider the scheduler $\mathfrak{A}'$ that activates the robots exactly as $\mathfrak{A}$ until round t, and it continues to activate r in rounds $t+1$ and $t+2$. In those rounds, r can have a coordinate system that makes it sense the swarm as in σ; so, r performs a non-null movement three times in total, which contradicts the request of the problem. □

Theorem 31. *Independently from* Rig-NonRig*, but assuming* VarDis*:*

$$\mathcal{FCOM}^{PERM} > \mathcal{FCOM}^{SEQ}.$$

Proof. By Theorem 23, we know that $\mathcal{FCOM}^{\texttt{PERM}} \geq \mathcal{FCOM}^{\texttt{SEQ}}$. ApproachTwice plays the role of separator between the two models (by Lemmas 26 and 27). □

3.4 Conclusions

In this chapter, we have widened our knowledge on the computational landscape of robot models by comparing the computational power of the four base models of robots $\mathcal{OBLOT}$, $\mathcal{FSTA}$, $\mathcal{FCOM}$, $\mathcal{LUMI}$ under three sequential modes—namely RROBIN (*round-robin*), PERM (*permutation*) and SEQ (*sequential*)—one sub-mode of the other.

Notably, we have proven the equivalence $\mathcal{LUMI}^{\texttt{SEQ}} \equiv \mathcal{LUMI}^{\texttt{PERM}}$: this result highlights the power of luminous robots under the most general sequential mode SEQ. As we will see in Chapter 7, this power will be exploited while solving the Dancing problem in its most general (aka Universal) definition.

Instead, for the other models, we have provided five separator problems, thus demonstrating that their computational power diminishes as the sequential scheduler becomes more adversarial. It is significant that, except for the dominance $\mathcal{FCOM}^{\texttt{PERM}} > \mathcal{FCOM}^{\texttt{SEQ}}$, all the computational dominances proved in this chapter hold *independently* of the assumptions on the rigidity of robot movements and the disorientation of robots. For that pair of models, we have demonstrated the related dominance assuming VarDis, i.e., that the local coordinate system of each robot may not be persistent from cycle to cycle. So far, exploring the relation $\mathcal{FCOM}^{\texttt{PERM}}$ vs. $\mathcal{FCOM}^{\texttt{SEQ}}$ assuming nothing about VarDis-FixDis and Rig-NonRig seems an interesting open question.

A subsequent research work may complete the computational relation map of the 12 models here considered, finding the cross-model relations (e.g., $\mathcal{FCOM}^{\texttt{SEQ}}$ vs. $\mathcal{FSTA}^{\texttt{PERM}}$?), or may expand the taxonomy of the models by defining further sequential modes (e.g., SEQ $\cap$ RSYNCH, the sequential schedulers which never activates a robot for two consecutive rounds). Moreover, for each sequential mode, it would also be valuable to characterize the problems (un)solvable: this characterization may serve as a preliminary study for finding algorithmic solutions for well-known problems under sequential schedulers.

Chapter 4

Fault Detection and Identification

4.1 Introduction

In the previous Chapters 2 and 3, we have compared the computational power of different models, thus their general capability of solving problems. This chapter undertakes a quite different approach to the study of the computational power of models: instead of analyzing the solvability of *generic* problems, we focus on two *specific* problems, namely Fault Detection and Fault Identification, for crash-prone systems, and we investigate the model features under which these problems are solvable.

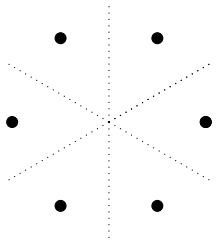
Although well-established within traditional robotics (e.g., [VS06]), these problems are novel in the distributed LCM model. Before introducing them, we begin by presenting their underlying motivation.

Motivation

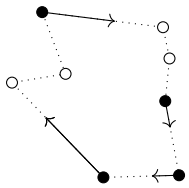
Most of the literature on the LCM model has considered *fault-free* (aka *correct*) robots: robots have infinite precision in perception and computation, they correctly compute the target destination, they always reach the computed destination, and they never crash. However, such strong assumptions are unrealistic; therefore, literature has started considering *fault-prone* systems. As a matter of fact, hardware and software imprecision in computation and motion, errors, and faults are very common in single machines. Increasing the number of involved machines and so the additional challenges that a distributed system must cope with (e.g., synchronization, collision avoidance, mutual interference) augments the chance of a fault occurrence and thus the importance of considering such possibilities while designing distributed algorithms.

As seen in Section 1.2.9, various types of faults (e.g., crashes, Byzantine faults, inaccurate actions) have been considered in the LCM model, and fault-tolerant algorithms have been designed to solve problems under fault-prone systems. Notably, a wide attention has been given to the crash faults (i.e., after those robots stop working irretrievably and thus do not execute their LCM cycle when activated) and several crash-tolerant algorithms have been provided under different models for the classic problems [Bra+24b; BLT23; BT15; Das+19; Déf+06; DPBT19; Pat+20; PAS21; SYD08; Yan+11]. As seen in Section 1.3.7, a crash-tolerant algorithm solves the problem even in the presence of crashed robots; yet, not all the problems (e.g., `Gathering`) always admit a crash-tolerant solution. In these cases, but not only, crashes should be addressed by different approaches, aimed at detecting/identifying them and intervening accordingly. Let us present two illustrative scenarios, which can be modeled through LCM.

[0]The content of this chapter refers to the work [CF25].



(A) Firefighter swarm where each robot is responsible for its zone and its adjacent ones.



(B) Fertilizing swarm of 4 robots which have to move along the cyclic path in a clockwise direction and fertilize the 8 stations.

FIGURE 4.1: Two introducing scenarios.

Example 1 (Firefighter swarm)**.** Consider a swarm of firefighter robots stationed in an area, aimed at monitoring it for fires and extinguishing them. Such robots perform an infinite sequence of computational cycles: at each clock, some of them are activated, they look at the environment, they process the snapshot just taken, and, if a fire has been sensed, they properly move to fight it. They are provided with limited, energy-saving message protocols for intra-swarm communication; external communication is allowed only in case of emergency, due to its consumption. The subset of robots activated at each clock is properly designed and scheduled to assign at least one activated robot for each zone of the surveilled area and to have a fair load balance, allowing each robot to recharge itself during idle periods (see Figure 4.1a).

Example 2 (Fertilizing swarm)**.** Consider now a swarm of robots aimed at spreading fertilizers on a field so that the activated robots at each clock reach the next station to be fertilized, following a cyclic path. Robots are synchronized and activated so that no collisions occur, i.e., if a robot moves to a station, then either the station is empty or the robot on that station is moving to the next station as well (see Figure 4.1b).

In the case of a firefighter swarm (Example 1), a lower number of functioning robots than the scheduled number may be insufficient to fight a possible fire. Due to the criticism of the scenario, all the robots should be ready to intervene; thus, it is important to design trustworthy systems where the not-crashed (aka, correct) robots can promptly and autonomously detect whether there exists a crashed robot and, in this case, send an external message to request human intervention and maintenance.

In the case of the fertilizing swarm (Example 2), robots may collide with faulty robots that have not moved as required along their prefixed path. In this scenario, the presence of faulty robots can threaten the safety of the correct robots, and the integrity of the whole system.

So, detecting and identifying faulty robots is necessary to protect the correct robots from further system damage or at least limit their occurrences. Moreover, correct robots can properly change their path to not collide with the identified faulty robots, but still accomplish their task.

Different approaches can be applied when a fault is correctly detected:

- robots stop working to avoid compromising the integrity of the system (*swarm freezing*);
- robots try to solve the original task (*fault tolerance*) or a weaker version of it (*best effort*);
- robots solve a different problem, e.g., try to fix faulty robots (*fault recovery*).

Note that freezing the swarm may be a prior step to fault recovery. Preliminary to these approaches, robots may have to solve the FAULT DETECTION (FD) and

Fault Identification (FI) problems. A swarm solves Fault Detection if the correct robots detect whether there exists at least a faulty robot in the swarm; Fault Identification requires the correct robots to identify which robots are correct and which are faulty. This chapter proposes an initial study for the FD and FI problems under the LCM model, paving the way for further work that may study the consequent reactions to recover the systems from faults.

Related work

A fault-tolerant algorithm does not necessarily contain a fault detection module; hence, it does not assume that the correct robots are aware of the presence of the faulty ones.

The concept of *failure detector* was first introduced by Chandra and Toueg in [CT96] for solving `Consensus` in asynchronous systems prone to crash faults. Although the authors considered a different distributed model than the LCM one (i.e., static system of n processes, each one equipped with a specific algorithm, and completely connected through trusty message channels), their work defines some properties (e.g., strong/weak completeness and accuracy) that a failure detector should reasonably enjoy, independently of the distributed system to which is applied.

For the LCM model, the idea of failure detector has been considered in [SYD08; Yan+11], where the authors provide some crash-fault detector modules as part of solutions for the `Flocking` problem[1]. Thus, their failure detectors are strictly based on the algorithms provided for `Flocking`. Their idea is to make correct robots compare the positions of the others, and thus understand when a robot has crashed since it has not changed position for too many activations. For this comparison, robots are allowed to store a variable $S_{PosPrevObser}$ which contains the set of positions of robots in the system during the last previous observation, thus granting each robot an infinite persistent memory: in fact, if the swarm is composed of n robots, $S_{PosPrevObser} \in (\mathbb{R}^2)^n$.

Contribution

Differently from the study proposed in [SYD08; Yan+11], we here embark on a *general investigation* of how a swarm of crash-prone robots can detect whether there are faulty robots in the swarm, and, possibly, identify them, *independently of the primary problem the swarm is solving.*

Preparatory to our investigation, the first part of this chapter formalizes the occurrence of crash faults and defines the Fault Detection (FD) and Fault Identification (FI) problems. Formalizing faults through a scheduler immediately allows us to infer some conditions (on time and the robot models) under which robots are unable to solve FD (and so FI). As in [CT96], we define two properties (i.e., reliability and punctuality) that an FD or FI algorithm should enjoy; yet, differently from [CT96], we work only on algorithms that never provide false positives, and we analyze the delays within which a fault is detected/identified. We take into account 12 robot models ($\mathcal{OBLOT}$, $\mathcal{FSTA}$, $\mathcal{FCOM}$, $\mathcal{LUMI}$, combined with the modes `FSYNCH`, `RROBIN`, `SSYNCH`), and we study generic solutions for FD or FI under such models. Note that, differently from [CT96; SYD08; Yan+11], we consider models where robots can store and communicate at most a constant-size amount of data, thus they can neither use their persistent memories to track the past snapshots of the swarm [SYD08; Yan+11], nor send the list of their suspects to the other robots [CT96]. We prove a reliable FI procedure cannot exist under $\mathcal{LUMI}^{\texttt{FSYNCH}}$. Instead, we provide three FD

[1] We remind that `Flocking` problem requires robots to move together on the plane while maintaining a pattern.

procedures for three models: $\mathcal{LUMI}^{\texttt{FSYNCH}}$ (punctual), $\mathcal{FCOM}^{\texttt{FSYNCH}}$ (punctual), and $\mathcal{LUMI}^{\texttt{RROBIN}}$ (reliable). We prove the impossibility of designing a reliable or punctual FD procedure under the other 9 models. To the best of our knowledge, this is the first work addressing fault detection in the LCM model using a general approach.

Observation 5 (Crashes in static configuration)**.** Note that we aim to detect/identify faults even when the swarm has reached a static configuration since it has terminated its task or because the swarm must stay still in a particular configuration (e.g., a regular polygon and a point). One can criticize the sense of detecting faults in a swarm where apparently robots have to do nothing. The motivation lies in the fact that robots may be required to stay in a particular configuration, ready to be activated by an event scheduler simulating an external event occurring in the environment (e.g., a fire for the firefighter swarm). Always being correct is a key requirement for trustworthy and fault-aware systems that can aptly detect and react to a failure.

4.2 Models

We investigate how FD and FI can be solved under 12 robot models. All the 12 models share some *fixed features*, while they differ in a set of *variable features*.

Fixed features. Given a swarm $\mathcal{R}$, we assume that all robots enjoy the core features explained in Section 1.2.1 and:

- perform *rigid movements* (Rig);
- enjoy *complete visibility*, i.e., they are *transparent* and *non-myopic*;
- are *completely disoriented*, thus do not have any agreement on their local coordinate systems;
- they suffer from *variable disorientation* (VarDis), thus the local coordinate system of any robot may not persist from one activation to another;
- are *collision-tolerant*, thus can form multiplicities at any time;
- have *strong multiplicity detection*, i.e., they can distinguish the number of robots with the same color[2] in each multiplicity.

Since robots can form multiplicities and have strong multiplicity detection, we use an equivalent notation rather than Equation (1.1) to express a snapshot. Specifically, a snapshot taken by a robot r has the form:

$$\sigma = \langle \{\zeta_0\}_{\#_0}, \ldots, \{\zeta_h\}_{\#_h} \rangle$$

where $\{\zeta_i\}_{\#_i}$ represents the multiset containing $\#_i > 0$ occurrences of ζ_i, and where $\zeta_i \in (\mathbb{R}^2 \times \mathfrak{L})$ is a state (position and color) perceived by r. Equivalently to Equation (1.1), we alays assume that ζ_0 corresponds to the local state of r, while all the other states are ordered according to the local coordinate system of r and the total order of $\mathfrak{L}$.

Example 3. Let r be a robot performing the Look step at time t under $\mathcal{FSTA}$. If $\sigma = \langle \{((0,0), red)\}_1, \{((0,0), \not\emptyset)\}_5, \{((1,1), \not\emptyset))\}_4 \rangle$ is the taken snapshot, it means that

- r lies in the point $(0,0)$ according to $\Xi_r(t)$ and its the internal light is `red`-colored;
- since under $\mathcal{FSTA}$, all the other robots are perceived as $\not\emptyset$-colored;
- r can count other 5 robots forming a multiplicity with itself in $(0,0)$;
- other 4 robots lie in $(1,1)$.

[2] According to the perceivable colors, not necessarily the actual ones. Perceivable colors correspond to the actual ones only under $\mathcal{LUMI}$.

Variable features. We study the 12 models in the form X^Y that possess all the above fixed features and that

- have $X \in \{\mathcal{OBLOT}, \mathcal{FSTA}, \mathcal{FCOM}, \mathcal{LUMI}\}$ as communication-memory setting;
- $Y \in \{\mathtt{S}, \mathtt{F}, \mathtt{RR}\}$ as synchronization mode (i.e., `SSYNCH`, `FSYNCH`, `RROBIN`, resp.).

We indicate with $\mathcal{M}$ the set of the resulting 12 robot models. For a model $X^Y \in \mathcal{M}$, we will consider both the *fault-free* version (i.e., where all robots are assumed to be always correct) and the *fault-prone* version (i.e., where robots may suffer from crash faults).

Note that, since this study focuses solely on synchronous modes, the time domain considered in this chapter is $\mathbb{N}_0$.

4.3 Crash schedulers

A crash fault on a robot r at time t implies that r will no longer execute an LCM cycle from time t onward. In other words, r remains idle for any time $t' \geq t$. Until t, r is said to be *correct*. According to this, we formalize the occurrence of crash faults for a swarm of robots through a scheduler of crashes.

Definition 7 (Crash Scheduler)**.** *A* crash scheduler *for a swarm $\mathcal{R}$ is a function $\mathfrak{c} : \mathcal{R} \to \mathbb{N}_0 \cup \{\infty\}$ defining for each robot r the time where r crashes (i.e., stops working). If $\mathfrak{c}(r) = \infty$, r never crashes.*

We say that a crash scheduler $\mathfrak{c}$ is *k-survival* if $|\{r \in \mathcal{R} \text{ s.t. } \mathfrak{c}(r) = \infty\}| \geq k$, i.e., it keeps at least k robots always correct. We always assume that a crash scheduler is at least 1-survival.

Definition 8 (Suppression Scheduler)**.** *Given a crash scheduler $\mathfrak{c}$ defined for a swarm $\mathcal{R}$, we define the related* suppression scheduler *as the function $\mathfrak{C}_\mathfrak{c} : \mathbb{N}_0 \to 2^\mathcal{R}$ so that*

$$\mathfrak{C}_\mathfrak{c}(t) := \{r \in \mathcal{R} \text{ s.t. } \mathfrak{c}(r) \leq t\}.$$

For a given time $t \in \mathbb{N}_0$, $\mathfrak{C}_\mathfrak{c}(t)$ is the set of robots crashed at a time $t' \leq t$ according to $\mathfrak{c}$. Obviously, it holds that $\mathfrak{C}_\mathfrak{c}(t') \subseteq \mathfrak{C}_\mathfrak{c}(t)$. When no ambiguity arises, we will use $\mathfrak{C}$ instead of $\mathfrak{C}_\mathfrak{c}$ by omitting to indicate the related crash scheduler.

A swarm is governed by both an activation scheduler and a suppression scheduler, which operate simultaneously, each independently of the other. Hence, it is convenient to define the behavior of the *combined scheduler* which determines, for each time $t \in \mathbb{N}_0$, the subset of robots that will *actually* become active and start performing their LCM cycle at time t.

Definition 9 (Combined Scheduler)**.** *Given $\mathfrak{A}, \mathfrak{C}$ respectively an activation and a suppression scheduler defined for the same swarm $\mathcal{R}$ and the same time domain $\mathbb{N}_0$, the corresponding* combined scheduler *is the function $\mathfrak{A} \setminus \mathfrak{C} : \mathbb{N}_0 \to 2^\mathcal{R}$ defined as*

$$\mathfrak{A} \setminus \mathfrak{C}(t) := \mathfrak{A}(t) \setminus \mathfrak{C}(t).$$

Note that, despite $\mathfrak{A}$ is always assumed fair, a combined scheduler $\mathfrak{A} \setminus \mathfrak{C}$ guarantees fairness iff $\mathfrak{C}(t) = \varnothing$ for each time $t \in \mathbb{N}_0$.

4.4 Fault detection and fault identification

4.4.1 Problems and procedures

Let $\mathcal{R} = \{r_0, \ldots, r_{n-1}\}$ be a swarm of robots executing a given distributed algorithm $\mathbb{A}$ for solving a problem P under a fault-free model M. If we consider the same swarm and the same problem under the fault-prone version of M, we are interested in making robots concurrently cope with two problems: the *primary problem* P, and the FAULT DETECTION problem. The FAULT DETECTION problem (FD) is an agreement-related problem (see Section 1.3.1) asking for a swarm of robots to reach awareness about the presence of a faulty robot if it exists. Basically, the swarm is required to behave in this way:

- if the first crash fault has occurred at time t_{crash}, there must exist a finite time t_{detect} when at least a robot detects there exists a fault and stays still. Afterward, all the other (correct) robots must reach the awareness of the presence of a fault and stay still;
- until t_{detect}, the robots have to continue solving the primary problem by executing $\mathbb{A}$.

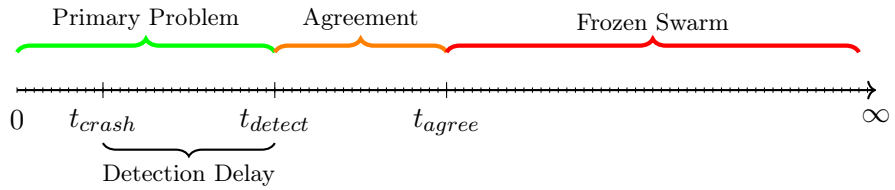
As for the other agreement-related problems typical of distributed systems (e.g., `Consensus`, `Broadcast`, `Leader Election`), FD aims to make all robots in the swarm agree on the same information (i.e., the absence or the presence of a faulty robot) and the same reaction (i.e., solve the primary problem or freeze the swarm, resp.). The *awareness* of a crash can be formally implemented in each robot by setting a local boolean variable z whose value is updated at each LCM cycle by running a FAULT DETECTION procedure during the Compute step.

FAULT DETECTION procedure

Input: A snapshot σ of the swarm.

Output: $z \leftarrow$ FAULT if a faulty robot is detected in σ; $z \leftarrow$ CORRECT otherwise.

Formally, if a crash fault has occurred for the first time at t_{crash}, then t_{detect} is the first time when at least one robot detects the fault by executing a FD procedure, sets $z \leftarrow$ FAULT, and stays still. Accordingly, a finite time $t_{agree} \geq t_{detect}$ exists where all the correct robots of the swarm have agreed on the presence of the fault by setting their variable z to FAULT and thus stay still. As we will prove later, there could be an intrinsic delay between t_{crash} and t_{detect} when the first robot detects that fault and/or t_{agree} when the swarm reaches the agreement on that fault (thus solving FD).



The FAULT IDENTIFICATION problem (FI) is more sophisticated than FD since it requires robots to distinguish the faulty robots from the correct ones in the swarm. Formally, if a robot $r_i \in \mathcal{R}$ has crashed at time t_{crash_i}, then FI requires that there exists a finite time t_{detect_i} when at least a robot identifies r_i as faulty. Afterward, as

for FD, all the correct robots are required to converge on the information "r_i is faulty" by a finite time $t_{agree_i} \geq t_{detect_i}$.

To solve FI, each robot queries a Fault Identification procedure, which returns a vector $\underline{z}$ so that $z_j = \mathsf{FAULT}$ ($z_j = \mathsf{CORRECT}$, resp.) if the corresponding j-th robot in the snapshot σ has been identified as faulty (correct, resp.). Indeed, a FI procedure simulates a FD one. For the sake of clarity, we will use the function-like notation $\underline{z}(r_i)$ to indicate the value in $\underline{z}$ which refers to the robot r_i: this allows us to ignore the local index of r_i according to the snapshot σ.

Fault Identification procedure

Input: A snapshot σ of the swarm $\mathcal{R} = \{r_0, \ldots, r_{n-1}\}$.

Output: $\underline{z} \in \{\mathsf{FAULT}, \mathsf{CORRECT}\}^n$, where $\underline{z}(r_i) = \mathsf{FAULT}$ iff r_i is identified as faulty in σ.

4.4.2 Reliability and puntuality

We here define and analyze some properties about FD/FI procedures. Let $\mathcal{F}$ be a FD procedure for $\mathcal{R} = \{r_0, \ldots, r_{n-1}\}$. We define $\Delta = t_{detect} - t_{crash}$ as the *detection delay*, where t_{crash} is the time of the first crash in $\mathcal{R}$, and t_{detect} is the first time when $\mathcal{F}$ returns FAULT. If $\mathcal{F}$ is a FI procedure, we define $\Delta_i = t_i - \mathfrak{c}(r_i)$ as the *identification delay* for a robot r_i, where $\mathfrak{c}(r_i) \neq \infty$ is the crash time of r_i and t_i is the first time $\mathcal{F}$ returns $\underline{z}(r_i) = \mathsf{FAULT}$. We say that Δ and Δ_i are undefined if the related faults have never occurred.

We firstly state that, if $\mathcal{F}$ is a FD (FI, resp.) procedure that never outputs a false positive, then a fault in the swarm (of r_i, resp.) cannot be detected (identified, resp.) in the same round of its occurrence. This means that $\Delta > 0$ ($\Delta_i > 0$, resp.). We prove the statement for FD; the proof for FI is similar.

Lemma 28 (No Instantaneous Detection)**.** *Let $\mathcal{F}$ be a FD procedure which never outputs false positives. Then, if t_{crash} is the time of the first crash, then $\mathcal{F}$ returns* $\mathsf{CORRECT}$ *at time t_{crash}.*

Proof. By contradiction, suppose that $\mathcal{F}$ detects the fault at time t_{crash}. Let σ be the snapshot through which $\mathcal{F}(\sigma)$ returns FAULT. However, σ would be the same if no robot has crashed at time t_{crash}. This contradicts the hypothesis that $\mathcal{F}$ never outputs false positives. □

We say that a FD/FI procedure guarantees *reliability* if:

- (i) (*no false positive*) it never detects/identifies a fault when it does not exist in the swarm (for FD) or in a particular robot (for FI);
- (ii) (*finite delay*) it detects/identifies a fault with a finite delay after its occurrence;
- (iii) (*coherence*) once it detects/identifies a fault, it continues to detect/identify the fault in the following LCM cycles.

Formally, let $\mathcal{F}$ be a FD procedure for a swarm $\mathcal{R} = \{r_0, \ldots, r_{n-1}\}$. We say that $\mathcal{F}$ is *reliable* when, given any crash scheduler $\mathfrak{c}$ for $\mathcal{R}$, the following conditions hold:

- (i) if $\mathcal{F}$ returns FAULT at time t then $\mathfrak{C}_{\mathfrak{c}}(t-1) \neq \varnothing$;
- (ii,iii) if $t_{crash} = \min\{\mathfrak{c}(r_i) \text{ s.t. } r_i \in \mathcal{R}\} \neq \infty$ then $\exists\, t_{detect} > t_{crash}$ s.t. $t_{detect} \in \mathbb{N}_0$ and $\mathcal{F}$ returns FAULT for any time $t' \geq t_{detect}$.

At the same time, if $\mathcal{F}$ is a FI procedure, we say that it is *reliable* if we have:

- (i) if $\mathcal{F}$ returns $\underline{z}(r_i) = \mathsf{FAULT}$ with $i \in \{0, \dots, n-1\}$ at time $t \in \mathbb{N}_0$ then $\mathfrak{c}(r_i) < t$;
- (ii,iii) if $\mathfrak{c}(r_i) = t_{crash} \neq \infty$ then $\exists\ t_i > t_{crash}$ s.t. $t_i \in \mathbb{N}_0$ so that $\mathcal{F}$ returns $\underline{z}(r_i) = \mathsf{FAULT}$ at any time $t' \geq t_i$.

We say that a FD/FI procedure $\mathcal{F}$ is *punctual* if it is reliable and it holds the stronger condition that:

- if $\mathfrak{c}(r_i) = t_{crash} \neq \infty$ then $\mathcal{F}$ returns FAULT (for FD) or $\underline{z}(r_i) = \mathsf{FAULT}$ (for FI) for any time $t' \geq t_{crash} + 1$.

In other words, $\mathcal{F}$ is punctual when it detects/identifies the fault starting from the subsequent round after a crash occurrence; thus, it never outputs false negatives (starting from $t_{crash} + 1$).

4.4.3 Time conditions

In the previous section, we proved that a fault cannot be detected/identified at the same round when it occurs. Here, we provide further time conditions on when a fault can or cannot be detected/identified.

Definition 10 (Missing Activation)**.** *Let $\mathfrak{A}, \mathfrak{C}$ be an activation and a suppression scheduler for a swarm $\mathcal{R}$, respectively. We say that there is a* missing activation *of $r \in \mathcal{R}$ at time $t \in \mathbb{N}_0$ if $r \in \mathfrak{A}(t) \cap \mathfrak{C}(t)$. We denote with $\mathfrak{fma}(r)$ the time of the* first missing activation *of r.*

Definition 11 (Fault Ignorance Time, FIT)**.** *Let $\mathfrak{A}, \mathfrak{C}_\mathfrak{c}$ be an activation and a suppression scheduler for a swarm $\mathcal{R}$, respectively. Let $r \in \mathcal{R}$ be a robot whose crash time is $\mathfrak{c}(r) \neq \infty$ and $\mathfrak{fma}(r)$ is its first missing activation time, with $\mathfrak{fma}(r) \geq \mathfrak{c}(r)$. We define the* Fault Ignorance Time *(FIT) of r as the interval $\mathcal{I}(r) = [\mathfrak{c}(r), \mathfrak{fma}(r)] \cap \mathbb{N}_0$.*

The following lemma holds since, during $\mathcal{I}(r)$, r is faulty but has not yet missed an activation according to $\mathfrak{A}$, thus its behavior would have been the same as if it had been correct. The related corollary imposes a lower bound on the time for r to be identified as faulty, without guaranteeing the actual possibility of identifying such a fault.

Lemma 29 (Missing Activation Rule)**.** *Let $r \in \mathcal{R}$ be a crashed robot. Then, the crash of r cannot be identified during $\mathcal{I}(r)$ by a reliable FI procedure.*

Proof. Assume all the robots in $\mathcal{R}$ have the same coordinate system at any activation. Consider an activation scheduler $\mathfrak{A}$ and a suppression scheduler $\mathfrak{C}_\mathfrak{c}$ such that $\mathcal{I}(r) = [\mathfrak{c}(r), \mathfrak{fma}(r)]$ is the FIT of a robot r, with $\mathfrak{c}(r) \leq \mathfrak{fma}(r)$. By contradiction, let $\mathcal{F}$ be a FI reliable procedure and let $t' \in [\mathfrak{c}(r), \mathfrak{fma}(r)]$ be the time when another robot r' identifies r as crashed by executing $\mathcal{F}(\sigma)$ where σ is the snapshot of r' taken at time t'. Now, let us consider another suppression scheduler $\mathfrak{C}'_{\mathfrak{c}'}$ which is identical to $\mathfrak{C}_\mathfrak{c}$ except that the crash time of r is $\mathfrak{c}'(r) > t'$. According to the combined scheduler $\mathfrak{A} \setminus \mathfrak{C}'_{\mathfrak{c}'}$, all the robots perform the same actions as under $\mathfrak{A} \setminus \mathfrak{C}_\mathfrak{c}$ until $t' - 1$. So, robot r' at time t' takes the same snapshot as under $\mathfrak{A} \setminus \mathfrak{C}_\mathfrak{c}$ and thus identifies r as faulty by executing $\mathcal{F}(\sigma)$. However, under $\mathfrak{C}'_{\mathfrak{c}'}$, robot r will be crashed at $\mathfrak{c}'(r) > t'$. This contradicts the hypothesis that $\mathcal{F}$ is reliable. □

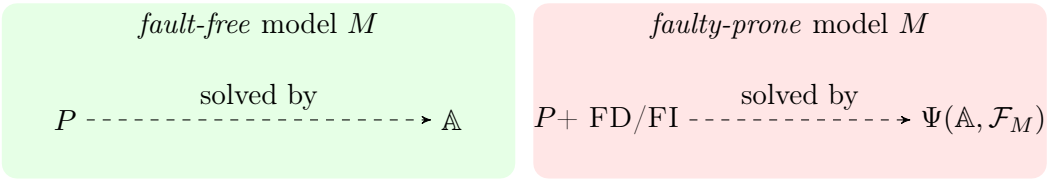
Corollary 8. *The crash of a robot r cannot reliably be identified at time $t \leq \mathfrak{fma}(r)$.*

The Missing Activation Rule can be generalized for an FD procedure.

Lemma 30. *Let t_{detect} be the first time when a reliable FD procedure detects a fault in a swarm $\mathcal{R}$. Then, $t_{detect} > \min_{r\in\mathcal{R}}\{\mathfrak{fma}(r)\}$.*

Proof. The proof is similar as in Lemma 29. Assume all the robots have the same coordinate system at any activation. Consider an activation scheduler $\mathfrak{A}$ and a suppression scheduler $\mathfrak{C}_{\mathfrak{c}}$ for $\mathcal{R}$, so that $t_{\mathfrak{fma}} = \min_{r\in\mathcal{R}}\{\mathfrak{fma}(r)\}$ and $\mathcal{R}_{\mathfrak{fma}} = \arg\min_{r\in\mathcal{R}} \mathfrak{fma}(r)$ is the subset of robots which first miss an activation (at time $t_{\mathfrak{fma}}$). By contradiction, let $t_{detect} \leq t_{\mathfrak{fma}}$ be the first time when a correct robot r' detects a fault exists in the swarm by executing a reliable FD procedure $\mathcal{F}$. Now, let us consider another suppression scheduler $\mathfrak{C}'_{\mathfrak{c}'}$ which is identical to $\mathfrak{C}_{\mathfrak{c}}$ except that the crash time of any robot $r \in \mathcal{R}$ is $\mathfrak{c}'(r) > t_{detect}$. According to the combined scheduler $\mathfrak{A} \setminus \mathfrak{C}'_{\mathfrak{c}'}$, all the robots perform the same actions until $t_{detect} - 1$. So, robot r' at time t_{detect} takes the same snapshot as under $\mathfrak{A} \setminus \mathfrak{C}_{\mathfrak{c}}$ and thus detects a fault by executing $\mathcal{F}$. However, under $\mathfrak{C}'_{\mathfrak{c}'}$, no robot has crashed before t_{detect}. This contradicts the hypothesis that $\mathcal{F}$ is reliable. □

4.4.4 Combined algorithm



Besides the reliability feature, we aim to design FD or FI procedures that are as general as possible, i.e., which can detect/identify faults independently of the primary problem that a swarm must solve. Let M be a faulty-prone model: we say that $\mathcal{F}_M$ is a *model-dependent* FD (FI, resp.) procedure if it reliably solves FD (FI, resp.) under M for *any* swarm and *any* primary problem the swarm can solve. Thus, let $\mathbb{A}$ be a solving algorithm for a (primary) problem P under the fault-free version of M. We want to combine $\mathbb{A}$ with $\mathcal{F}_M$ so that the *combined algorithm* $\Psi(\mathbb{A}, \mathcal{F}_M)$ becomes the new algorithm executed by each robot in its Compute step. The combined algorithm $\Psi(\mathbb{A}, \mathcal{F}_M)$ must behave in this way:

- *Correctness*: $\Psi(\mathbb{A}, \mathcal{F}_M)$ must continue solving P until no crash is detected/identified. If no crash occurs, $\Psi(\mathbb{A}, \mathcal{F}_M)$ eventually solves P.
- *Soundness*: $\Psi(\mathbb{A}, \mathcal{F}_M)$ must *freeze* the swarm (i.e., no robot will move anymore) when a crash is detected/identified. If a crash occurs, $\Psi(\mathbb{A}, \mathcal{F}_M)$ detects/identifies it after a finite delay after its occurrence, and will continue to detect/identify it in the next rounds.

Augmented snapshots. To solve FD/FI, $\mathcal{F}_M$ may adopt an *ad hoc* palette of colors $\mathfrak{L}_{\mathcal{F}}$ during the evolution of $\Psi(\mathbb{A}, \mathcal{F}_M)$. For the sake of clarity and w.l.o.g., we can assume that, under the fault-prone model M, each robot r is embedded with two lights, the *primary light* $\mathtt{lig}_r$ which assumes the colors in $\mathfrak{L}$ provided by the primary algorithm $\mathbb{A}$, and the *service light* $\mathtt{ser}_r$ which assumes the colors in $\mathfrak{L}_{\mathcal{F}}$ provided by $\mathcal{F}_M$. For the sake of uniformity, we assume $\mathfrak{L}_{\mathcal{F}} = \{\not\flat\}$ under $\mathcal{OBLOT}$; moreover, when the light of r is not visible to a robot, then this robot will see $\mathtt{ser}_r$ (as well as $\mathtt{lig}_r$) as $\not\flat$-colored. Thus, a combined algorithm $\Psi(\mathbb{A}, \mathcal{F}_M)$ takes in input an augmented snapshot $\sigma = \langle\{\zeta_0\}_{\#_0}, \ldots, \{\zeta_h\}_{\#_h}\rangle$ with $\zeta_i = (\underline{x}_i, c_i, f_i)$, and returns a triple $(\underline{x}', c', f')$

where $\underline{x}'$ is the position to be reached, and c' (f', resp.) is the possible color of the primary (service, resp.) light to be set during the related Move step. Remember that a robot may not update the color of its lights; in that case, the value c' (f', resp.) returned by the algorithm is denoted with the symbol $-$ (by convention, we use the same symbol also for x', thus for null movements).

Algorithm 4: Combined algorithm $\Psi(\mathbb{A}, \mathcal{F}_M)$ where $\mathcal{F}_M$ is a model-dependent FI procedure for a model M.

Input: $\sigma = \langle \{\zeta_0\}_{\#_0}, \ldots, \{\zeta_h\}_{\#_h} \rangle$ snapshot taken by r; $\zeta_0 = (\underline{x}_0, c_0, f_0)$ is the local state of r

Output: $(\underline{x}', c', f')$ new state of r

Procedure $\Psi(\mathbb{A}, \mathcal{F}_M)(\sigma)$**:**

```
(z, f') ← F_M(σ);
if ∀i ∈ {1, ..., n − 1}  z_i = CORRECT then
    σ_A ← restricted version of σ by removing all f_i components;
    // Continue solving the primary problem
    (x', c') ← A(σ_A);
    return (x', c', f');
else
    // Awareness about the faulty robots + Freezing
    return (−, −, f');
```

Let us present our combined algorithm Ψ, which uses $\mathbb{A}$ and $\mathcal{F}_M$ as black-box procedures for its implementation. The pseudo-code of Ψ is listed in Algorithm 4. We here assume that $\mathcal{F}_M$ is a FI procedure (if it were a FD procedure, the implementation would be similar). Let r be an activated robot in a swarm $\mathcal{R}$ of n robots that executes the algorithm Ψ using the just taken snapshot σ as input. At first, Ψ runs the FI procedure $\mathcal{F}_M(\sigma)$, which outputs the vector $\underline{z} \in \{\mathsf{CORRECT}, \mathsf{FAULT}\}^n$, and the new color $f' \in \mathfrak{L}_{\mathcal{F}}$ to be set for the service light of r. Then Ψ implements two different actions according to the value of $\underline{z}$:

- *(correctess)*: If the vector $\underline{z}$ contains only CORRECT values, Ψ computes the restricted snapshot $\sigma_{\mathbb{A}}$ that is obtained by σ by removing the service color f_i from any state ζ_i. Then, Ψ runs the primary algorithm $\mathbb{A}(\sigma_{\mathbb{A}})$ and obtains the pair $(\underline{x}', c')$, i.e., the possibly new position and primary color for r: this result perfectly simulates the behavior of $\mathbb{A}$ in the solution of the primary problem under the fault-free version of M.

- *(soundness)*: Otherwise, r is aware that some robots are faulty in the swarm, and it can identify them. In this case, Ψ returns the triple $(-, -, f')$ to indicate that r freezes.

This algorithm generalizes the case when $\mathcal{F}_M$ is a FD procedure: in that case, $\underline{z}$ consists of only one element.

The next theorem allows the use of $\mathcal{F}_M$ or $\Psi(\mathbb{A}, \mathcal{F}_M)$ interchangeably in the following sections.

Theorem 32. *$\mathcal{F}_M$ is a reliable model-dependent FD or FI procedure for a model $M \in \mathcal{M}$ if and only if $\Psi(\mathbb{A}, \mathcal{F}_M)$ guarantees correctness and soundness for any primary algorithm $\mathbb{A}$ under M.*

Proof. Below, we use $\mathcal{F}$ instead of $\mathcal{F}_M$ to lighten the notation.

($\Rightarrow$) By hypothesis, $\mathcal{F}$ detects/identifies a fault after a finite delay (so $\mathcal{F}(\sigma)$ returns a FAULT value after a finite delay from the fault occurrence). Moreover, after such a delay, $\mathcal{F}$ will always detect/identify the fault in the subsequent rounds. If $\mathcal{F}$ returns at least a FAULT value, then $\Psi(\mathbb{A}, \mathcal{F})$ freezes the robot. Since this happens for any correct robot activated in the swarm, the whole swarm will freeze (thus $\Psi(\mathbb{A}, \mathcal{F})$ is *sound*).

By hypothesis, $\mathcal{F}$ will always return a CORRECT value if no fault occurs: this makes $\Psi(\mathbb{A}, \mathcal{F})$ behave exactly as $\mathbb{A}$ (thus $\Psi(\mathbb{A}, \mathcal{F})$ is *correct*).

($\Leftarrow$) By hypothesis, if a crash occurs, then $\Psi(\mathbb{A}, \mathcal{F})$ makes all correct robots detect/identify it in finite delay, and freeze the swarm always maintaining coherence about the detected/identified faults in the following rounds. Let us assume that $\mathbb{A}$ is an algorithm that makes any activated robot change its position at any activation. Thus, according to Algorithm 4, $\Psi(\mathbb{A}, \mathcal{F})$ must stop entering into the "if" body and instead it must continue executing the "else" statement from a certain time onward. Thus, $\mathcal{F}$ must guarantee (ii) to detect/identify the crash after a *finite delay* and (iii) to continue detecting/identifying in the next rounds (*coherence*).

By hypothesis, if no crash occurs, then $\Psi(\mathbb{A}, \mathcal{F})$ continues solving the primary problem. So, it must always execute the "if" body. Thus, $\mathcal{F}$ must always return CORRECT, guaranteeing to provide *no false positives* (i).

□

4.4.5 Fault Identification unreliability

Here, we present a witness problem through which we prove the impossibility of designing a reliable model-dependent FI procedure for $\mathcal{LUMI}^{\mathbf{F}}$.

Problem 14 (SuperPoint)**.** Consider n robots lying on the same point p of the Euclidean plane. The problem asks them never to change position[3].

Figure 4.2: SuperPoint problem, with n robots in p.

The SuperPoint problem can be trivially solved under any fault-free model M thanks to a NOP[4] algorithm $\mathbb{A}_{NOP}$ (basically, robots do nothing). Let $\hat{\Xi}$ be a coordinate system whose origin lies in p. Let $\mathcal{F}$ be a hypothetical reliable $\mathcal{LUMI}^{\mathbf{F}}$-dependent FI procedure. In the following, we state some properties about $\Psi(\mathbb{A}_{NOP}, \mathcal{F})$ while solving SuperPoint, assuming the local coordinate system of all the robots is always $\hat{\Xi}$.

Lemma 31. *Let $\mathcal{R}$ be a swarm executing $\Psi(\mathbb{A}_{NOP}, \mathcal{F})$ to solve SuperPoint as primary problem, where $\mathcal{F}$ is a reliable model-dependent FI procedure for $\mathcal{LUMI}^{\mathbf{F}}$. Let us assume all the robots always use the coordinate system $\hat{\Xi}$, and let $\mathfrak{C}$ be a suppression scheduler for $\mathcal{R}$. Then at each round t, all the robots in $\mathcal{R} \setminus \mathfrak{C}(t-1)$ start their round with the same color.*

[3] See Observation 5 for the motivation of this kind of problem.
[4] Reference to the assembly instruction NOP, i.e., No Operation.

Proof. Let $\mathfrak{L}$ be the palette used by $\mathbb{A}_{NOP}$ to solve SuperPoint under the fault-free $\mathcal{LUMI}^{\mathbf{F}}$ model[5]. Suppose $\mathfrak{L}_{\mathcal{F}}$ is the palette used by $\mathcal{F}$. By hypothesis, the snapshot of a robot r has this form: $\sigma = \langle \{(\mathbf{O}, b_0)\}_{\#_0}, \ldots, \{(\mathbf{O}, b_h)\}_{\#_h} \rangle$ where $\mathbf{O} = (0,0)$ is the position of any robot according to $\hat{\Xi}$, and where $b_i = (c_i, f_i) \in \mathfrak{L} \times \mathfrak{L}_{\mathcal{F}}$ represents the pair of colors for the primary light and the service light of the robots. Note that b_0 represents the colors of the robot r itself. The lemma can be easily proved by induction: in fact, at time $t = 0$, all the robots have the same color $b_i = (\not{b}, \not{b})$ (by convention, we set that $\mathfrak{C}(-1) = \varnothing$). Let us now assume the fact true for a time t, i.e., all the robots in $\mathcal{R} \setminus \mathfrak{C}(t-1)$ have the same color, say $\mathbf{b}$, during the t-th round. Since $\mathcal{R} \setminus \mathfrak{C}(t) \subseteq \mathcal{R} \setminus \mathfrak{C}(t-1)$, all the active robots in the t-th Look step take the same snapshot $\sigma = \langle \{(\mathbf{O}, \mathbf{b})\}_{\#_0}, \ldots, \{(\mathbf{O}, b_h)\}_{\#_h} \rangle$, and accordingly they compute the same next color $\mathbf{b}'$ by executing $\Psi(\mathbb{A}_{NOP}, \mathcal{F})(\sigma)$. Thus, at the beginning of the $(t+1)$-th round, all the robots in $\mathcal{R} \setminus \mathfrak{C}(t)$ have set the same color $\mathbf{b}'$. □

By the contrapositive of Lemma 31, we have:

Corollary 9. *Let $\mathcal{F}$ be a reliable FI model-dependent procedure for $\mathcal{LUMI}^{\mathbf{F}}$. Let us consider the problem SuperPoint for a swarm of robots executing $\mathcal{F}$ and let us assume all the robots always use the coordinate system $\hat{\Xi}$. If at the t-th round two robots have two different colors, then at least one belongs to $\mathfrak{C}(t-1)$.*

Theorem 33. *There can be no reliable model-dependent FI procedure for $\mathcal{LUMI}^{\mathbf{F}}$.*

Proof. By contradiction, let us assume that there exists a reliable FI procedure $\mathcal{F}$ for $\mathcal{LUMI}^{\mathbf{F}}$ which uses a k-size palette $\mathfrak{L}_{\mathcal{F}}$. Let us consider the SuperPoint problem under $\mathcal{LUMI}^{\mathbf{F}}$, with $n \geq k+2$ robots, and let $\mathbb{A}_{NOP}$ be the optimal algorithm solving SuperPoint with $\mathfrak{L} = \{\not{b}\}$. By definition, the problem starts from a configuration where all the robots have the same color. Let us assume all the robots always use the coordinate system $\hat{\Xi}$. Thus, for the sake of conciseness, we can omit to state the position $(0,0)$ and the primary light color $\not{b}$ in the snapshots of the robots; instead, we only specify the colors of the service lights. As proved in Lemma 31, at each round t, all the active and correct robots take the same snapshot $\sigma = \langle \{f_0\}_{\#_0}, \ldots, \{f_h\}_{\#_h} \rangle$, execute $\mathcal{F}(\sigma) = (\underline{z}, f')$ and update their service color from f_0 to f'. Note in fact that f_0 corresponds to the current service color of all the robots in $\mathcal{R} \setminus \mathfrak{C}(t-1)$ (and thus in $\mathcal{R} \setminus \mathfrak{C}(t)$).

Let $\mathfrak{F}(t) \subseteq \mathfrak{L}_{\mathcal{F}}$ denote the set of colors of the (crashed) robots in $\mathfrak{C}_{\mathfrak{c}}(t)$. Of course, $\mathfrak{F}(t) \subseteq \mathfrak{F}(t+1)$. Let $\mathfrak{C}_{\mathfrak{c}}$ be a suppression scheduler so that, if at time t the correct robots have set their light to a color $\ell \notin \mathfrak{F}(t)$, then $\mathfrak{c}$ crashes only one robot at time $t+1$ (which is ℓ-colored). Thus, ℓ will be included in $\mathfrak{F}(t+1)$.

This crash process stops at the minimum time $\bar{t}$ after which the correct robots will periodically assume colors in $\mathfrak{F}(\bar{t})$. Note that this time exists (i.e., it is finite), and in the worst case we have that $\mathfrak{F}(\bar{t}) = \mathfrak{L}$. Then, after $\bar{t}$, with some periodicity, the correct robots will set the same color, say ℓ, as a faulty one. Being the correct robots ≥ 2, then an activated robot r will not be able to distinguish the faulty ℓ-colored robot from the correct ℓ-colored ones. This contradicts the hypothesis that $\mathcal{F}$ is a reliable FI procedure. □

4.5 Model-dependent Fault Detection

This section presents three model-dependent FD procedures for $\mathcal{LUMI}^{\mathbf{F}}$, $\mathcal{FCOM}^{\mathbf{F}}$ and $\mathcal{LUMI}^{\mathbf{RR}}$, respectively. In all of them, we assume that the related $\mathfrak{L}_{\mathcal{F}}$ always

[5] Although the most efficient NOP algorithm can solve the problem with $\mathfrak{L} = \{\not{b}\}$, we here assume nothing else about $\mathbb{A}_{NOP}$ other than that it effectively solves SuperPoint.

contains a special color ALARM which will be used as soon as a robot detects the presence of a faulty robot during its Compute step. Accordingly, in the related Move step, the robot will set its service light as ALARM to notify itself and/or the other robots that a faulty robot has been detected. When a robot r sees an ALARM in its snapshot, it automatically is aware that a fault has been detected in the previous rounds. Thus, r sets its $\texttt{ser}_r$ to ALARM too, whereas it updates neither its position nor its $\texttt{lig}_r$: this causes the alarm to be broadcast throughout the correct robots of the swarm, and the swarm to freeze at the same time.

Eventually, in Section 4.5.4, we prove that a reliable FD procedure cannot exist under the other robot models. The properties of our proposed model-dependent FD procedures are summarized in Table 4.1.

Model	Reliable	Min Survivals	Detection Delay	$\lvert\mathfrak{L}_{\mathcal{F}}\rvert$
$\mathcal{LUMI}^{\texttt{F}}$	Yes	1	$\Delta = 1$ (Punctual)	3
$\mathcal{FCOM}^{\texttt{F}}$	Yes	2	$\Delta = 1$ (Punctual)	3
$\mathcal{LUMI}^{\texttt{RR}}$	Yes	1	$1 \leq \Delta \leq 2n-1$	4
$\mathcal{FCOM}^{\texttt{F}}$	Impossible	1		
$\mathcal{FCOM}^{\texttt{RR}}$	Impossible	2		
$X^{\texttt{S}}$	Impossible			
$\{\mathcal{FSTA}, \mathcal{OBLOT}\}^{Y}$	Impossible			

Table 4.1: Proposed model-dependent FD procedures and impossibility results. $X \in \{\mathcal{OBLOT}, \mathcal{FSTA}, \mathcal{FCOM}, \mathcal{LUMI}\}$, $Y \in \{\texttt{F}, \texttt{RR}, \texttt{S}\}$.

4.5.1 Fault detection in $\mathcal{LUMI}^{\texttt{F}}$

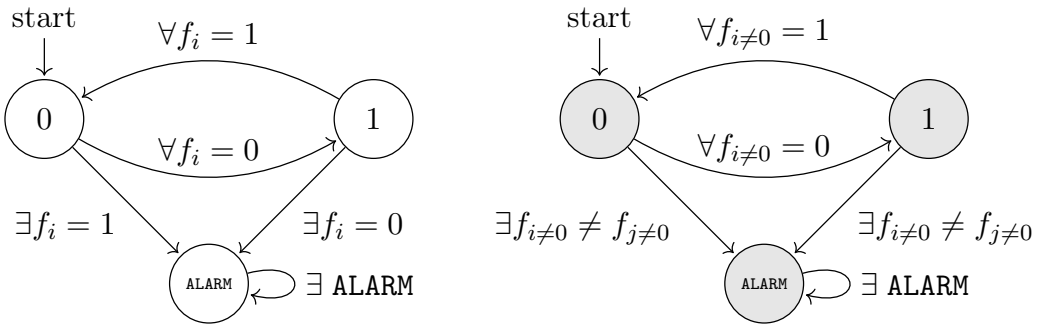
We present a reliable FD procedure $\mathcal{F}_{\mathcal{LUMI}^{\texttt{F}}}$ which can be executed by any swarm $\mathcal{R} = \{r_0, \ldots, r_{n-1}\}$ under $\mathcal{LUMI}^{\texttt{F}}$. This procedure tolerates up to $n-1$ crash faults and uses $\mathfrak{L}_{\mathcal{F}} = \mathbb{Z}_2 \cup \{\texttt{ALARM}\}$ as the service palette, 0 being the initial value[6]. Working under FSYNCH, all the correct robots become active simultaneously, and the $\mathcal{F}_{\mathcal{LUMI}^{\texttt{F}}}$ procedure makes them update their service light cycling over the values of $\mathbb{Z}_2 = \{0, 1\}$. Let $t_{crash} = \min_i\{\mathfrak{c}(r_i)\}$ be the first time when a robot crashes according to a crash time scheduler $\mathfrak{c}$ (which is 1-survival). Note that t_{crash} can be ∞. Then, all the robots synchronously alternate their service colors during the rounds before t_{crash}. If one robot crashes at time t_{crash} and so it does not update its light, its failure will be detected in the successive round, as it will display a different light than any other activated robot. Let r be an activated robot at time $t_{crash}+1$. So r detects from its snapshot that all the robots with a service light different from itself have crashed at the previous round, and thus sets $\texttt{ser}_r$ as ALARM. Because of this immediate detection, the proposed procedure is punctual. Note that, although r can identify the crashed robots at time $t_{crash}+1$, this procedure is not a model-dependent FI procedure since it cannot guarantee reliability in identifying the robots crashed in the following rounds (see Theorem 33 for the impossibility general result). Algorithm 5 and Figure 4.3a show the proposed FD procedure and the color update diagram of $\texttt{ser}_r$, respectively. Note that in Figure 4.3a (as well as for the next diagrams) the ALARM node represents a trap state, so that no outgoing transition is permitted: this property ensures the proposed procedure is coherent.

[6] $\mathbb{Z}_m$ denotes the set $\{0, \ldots, m-1\}$.

Algorithm 5: A *punctual* FD procedure for $\mathcal{LUMI}^{\mathrm{F}}$.

Input: $\sigma = \langle \{\zeta_0\}_{\#_0}, \ldots, \{\zeta_h\}_{\#_h} \rangle$ snapshot taken by a robot r, with $\zeta_i = (\underline{x}_i, c_i, f_i)$
Output: (z, y) where $z \in \{\mathsf{FAULT}, \mathsf{CORRECT}\}$ and $y \in \mathbb{Z}_2 \cup \{\mathtt{ALARM}\}$

Procedure $\mathcal{F}_{\mathcal{LUMI}^{\mathrm{F}}}(\sigma)$**:**
- **if** $\exists i \in \{0, \ldots, h\}$ s.t. $f_i = \mathtt{ALARM}$ **then**
  - `// Alarm broadcasting`
  - **return** $(\mathsf{FAULT}, \mathtt{ALARM})$;
- **if** $\exists i \in \{1, \ldots, h\}$ s.t. $f_i \neq f_0$ **then**
  - `// First detection of a fault`
  - **return** $(\mathsf{FAULT}, \mathtt{ALARM})$;
- `// Service color alternation in` $\mathbb{Z}_2$
- **return** $(\mathsf{CORRECT}, (f_0 + 1) \bmod 2)$;



(A) Algorithm 5 for $\mathcal{LUMI}^{\mathrm{F}}$. (B) Algorithm 6 for $\mathcal{FCOM}^{\mathrm{F}}$.

FIGURE 4.3: Diagrams showing the color update of $\mathtt{ser}_r$ according to the proposed FD procedures. The gray nodes represent the impossibility for r to see the color of its own $\mathtt{ser}_r$ (i.e., f_0).

4.5.2 Fault detection in $\mathcal{FCOM}^{\mathrm{F}}$

The fact that a robot under $\mathcal{FCOM}$ cannot see its own lights (`lig` and `ser`) imposes the first main difference in the number of tolerated crashes with respect to the previous FD procedure.

Theorem 34. *A reliable FD procedure for $\mathcal{FCOM}^{\mathrm{F}}$ cannot exist assuming 1-survival crash schedulers.*

Proof. By contradiction, let $\mathcal{F}$ be a reliable model-dependent procedure for $\mathcal{FCOM}^{\mathrm{F}}$ which tolerates up to $n-1$ crashes. Let us consider a swarm $\mathcal{R}$ executing $\Psi(\mathbb{A}, \mathcal{F})$ for a given primary algorithm $\mathbb{A}$. Let $\mathfrak{c}_1$ be a 1-survival crash scheduler such that $\mathfrak{c}_1(r) = \infty$, while $\mathfrak{c}_1(r') = t \neq \infty$ for any $r' \in \mathcal{R} \setminus \{r\}$. Let us assume r has a fixed local coordinate system, and let us assume that $\mathbb{A}$ makes r always stay still in its original position. By hypothesis, r is activated at time t, it executes $\mathcal{F}(\sigma)$ and detects no fault in $\mathcal{R}$ exists (by Lemma 30). From time t onward, r will always be the only robot to be activated. Since r cannot see its lights' color and since neither its coordinate system nor position changes round by round, the taken snapshot will always be σ. Thus, r will never detect the fault. Contradiction achieved. □

Assuming that no more than $n-2$ robots can crash, we can aptly apply some slight but necessary changes to Algorithm 5 and provide a *punctual* FD procedure for $\mathcal{FCOM}^{\mathrm{F}}$ (see Algorithm 6 and the corresponding color diagram in Figure 4.3b).

Notably, robots alternate their service colors in $\mathbb{Z}_2$: a robot can promptly detect a fault and set its service light to ALARM as soon as it sees two robots (different from itself) with two different service colors.

Algorithm 6: A *punctual* FD procedure for $\mathcal{FCOM}^{\mathtt{F}}$.

Input: $\sigma = \langle \{\zeta_0\}_{\#_0}, \dots, \{\zeta_h\}_{\#_h} \rangle$ snapshot taken by a robot r, with $\zeta_i = (\underline{x}_i, c_i, f_i)$
Output: (z, y) where $z \in \{\mathsf{FAULT}, \mathsf{CORRECT}\}$ and $y \in \mathbb{Z}_2 \cup \{\mathtt{ALARM}\}$

Procedure $\mathcal{F}_{\mathcal{FCOM}^{\mathtt{F}}}(\sigma)$**:**
  if $\exists i \in \{1, \dots, h\}$ s.t. $f_i = \mathtt{ALARM}$ **then**
    // Alarm broadcasting
    return $(\mathsf{FAULT}, \mathtt{ALARM})$;
  if $\exists i, j \in \{1, \dots, h\}$ s.t. $f_i \neq f_j$ **then**
    // First detection of a fault
    return $(\mathsf{FAULT}, \mathtt{ALARM})$;
  // Service color alternation in $\mathbb{Z}_2$
  return $(\mathsf{CORRECT}, (f_1 + 1) \bmod 2)$;

4.5.3 Fault detection in $\mathcal{LUMI}^{\mathtt{RR}}$

Under $\mathcal{LUMI}^{\mathtt{RR}}$, we can design a reliable FD procedure even considering 1-survival crash time schedulers. However, we prove that designing a punctual FD procedure for this model is impossible.

Theorem 35. *A punctual FD procedure for $\mathcal{LUMI}^{\mathtt{RR}}$ cannot exist, even considering $(n-1)$-survival crash time schedulers.*

Proof. By contradiction, let $\mathcal{F}$ be a punctual FD procedure for $\mathcal{LUMI}^{\mathtt{RR}}$. Let $\mathcal{R} = \{r_0, \dots, r_{n-1}\}$ be a swarm of robots under $\mathcal{LUMI}^{\mathtt{RR}}$ executing $\Psi(\mathbb{A}, \mathcal{F})$ for a given primary algorithm $\mathbb{A}$. Consider the two combined schedulers $\mathfrak{A}_0 \setminus \mathfrak{C}_0$ and $\mathfrak{A}_1 \setminus \mathfrak{C}_1$ defined as

$$\begin{aligned} \mathfrak{A}_0(t) &= \{r_{0+t}\}, \quad \mathfrak{C}_0(t) = \{r_0\} \ \text{ (i.e., } r_0 \text{ crashes at time 0)} \\ \mathfrak{A}_1(t) &= \{r_{1+t}\}, \quad \mathfrak{C}_1(t) = \varnothing \ \text{ (i.e., no robot crashes)} \end{aligned}$$

for any $t \in \mathbb{N}_0$ (indices are considered in modulo n). So, by hypothesis, r_1 is activated at time 1 under $\mathfrak{A}_0 \setminus \mathfrak{C}_0$, and it correctly detects that r_0 has crashed at time 0 by executing $\mathcal{F}$. Formally, r_1 executes $\mathcal{F}(\sigma)$ where σ is the snapshot taken by r_1 at time 1 using Ξ as coordinate system, and returns a FAULT.

Now, let us consider the same swarm under $\mathfrak{A}_1 \setminus \mathfrak{C}_1$ and suppose that r_1 has Ξ as its coordinate system. So, r_1 is activated at time 0, takes the same snapshot σ and executes $\mathcal{F}(\sigma)$ which still returns FAULT. However, no crash has occurred. Contradiction achieved. □

We now present our reliable FD procedure $\mathcal{F}_{\mathcal{LUMI}^{\mathtt{RR}}}$ (see Algorithm 7 and Figure 4.4). In this case, we need one more value in the palette $\mathfrak{L}_{\mathcal{F}}$: in particular, we make our robots cyclically take any value in $\{0, 1, 2\}$ (i.e., $\mathbb{Z}_3$), 0 being the default value. Note that it is possible to define a cyclic order $[0, 1, 2]$ on $\mathbb{Z}_3$, which would be impossible in $\mathbb{Z}_2$; this order defines our color transition rule $i \mapsto (i + 1) \bmod 3$. Thus, during the i-th epoch (with $i \geq 0$), our strategy is to make each activated robot set its $\mathtt{ser}_r$ from $i \bmod 3$ to $(i + 1) \bmod 3$.

Suppose a robot r crashes at its activation during the i-th epoch. So, its $\mathtt{ser}_r$ remains outdated to $i \bmod 3$. During the remainder of the i-th epoch, all the activated

Algorithm 7: A *reliable* FD procedure under $\mathcal{LUMI}^{\texttt{RR}}$.

Input: $\sigma = \langle \{\zeta_0\}_{\#_0}, \ldots, \{\zeta_h\}_{\#_h} \rangle$ snapshot taken by a robot r, with $\zeta_i = (\underline{x}_i, c_i, f_i)$
Output: (z, y) where $z \in \{\mathsf{FAULT}, \mathsf{CORRECT}\}$ and $y \in \mathbb{Z}_3 \cup \{\texttt{ALARM}\}$

Procedure $\mathcal{F}_{\mathcal{LUMI}^{\texttt{RR}}}(\sigma)$**:**
- **if** $\exists i \in \{0, \ldots, h\}$ s.t. $f_i = \texttt{ALARM}$ **then**
  - `// Alarm broadcasting`
  - **return** $(\mathsf{FAULT}, \texttt{ALARM})$;
- **if** $\exists i \in \{1, \ldots, h\}$ s.t. $f_i \equiv (f_0 - 1) \bmod 3$ **then**
  - `// First fault detection`
  - **return** $(\mathsf{FAULT}, \texttt{ALARM})$;
- `// Service color alternation in` $\mathbb{Z}_3$
- **return** $(\mathsf{CORRECT}, (f_0 + 1) \bmod 3)$;

robots see robots whose service lights are colored as either $i \bmod 3$ or $(i+1) \bmod 3$, thus they may have no means to understand if r has crashed or has not been activated yet. So they update their service light to $(i+1) \bmod 3$. Let r' be the first robot activated during the $(i+1)$-th epoch. Now, r' can detect the fault since it sees a robot whose service light has the preceding value (according to the cyclic order $[0, 1, 2]$) than its own. So r' sets its service light to `ALARM`, thus starting the agreement process among the other correct robots; in fact, in the following rounds, all the correct robots will set the `ALARM` light too, freezing the swarm.

To compute the maximum detection delay of $\mathcal{F}_{\mathcal{LUMI}^{\texttt{RR}}}$, consider the worst case. If $(r_0, \ldots, r_{n-1})$ is the round-robin sequence by which robots are activated, suppose that all the robots except for r_{n-1} crash during the first epoch, during their activation. So, the first crash (i.e., of r_0) will be detected by r_{n-1} in the second epoch, thus $2n-1$ rounds later. Instead, the minimum delay occurs when r_{n-1} crashes (at the end of an epoch) and r_0 detects its fault in the following round (at the beginning of the subsequent epoch).

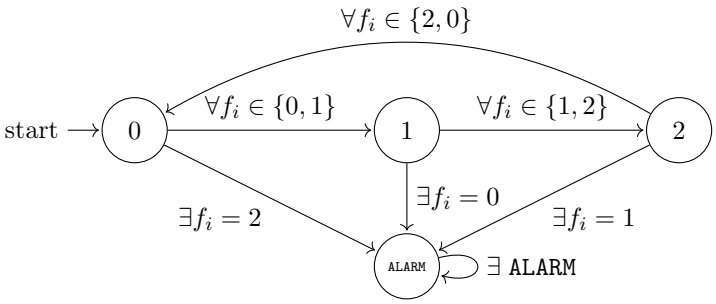


FIGURE 4.4: Color update of $\texttt{ser}_r$ in Algorithm 7.

4.5.4 Impossibility results

Using a similar argument as in Theorem 34, we prove that a reliable FD procedure for $\mathcal{FCOM}^{\texttt{RR}}$ cannot exist assuming 1-survival crash schedulers. However, unlike $\mathcal{FCOM}^{\texttt{F}}$, such impossibility holds even with 2-survival crash schedulers.

Theorem 36. *A reliable FD procedure for $\mathcal{FCOM}^{\texttt{RR}}$ cannot exist even assuming 2-survival crash schedulers.*

Proof. By contradiction, let $\mathcal{F}$ be a reliable FD procedure for $\mathcal{FCOM}^{\texttt{RR}}$. W.l.o.g., assume that $\mathfrak{L}_{\mathcal{F}} = \{f_0, \ldots, f_{k-1}\}$ for a constant $k > 0$, with f_0 being the default

color. Let us consider the SuperPoint problem where $n > k$ robots lie on the same point and have to stay still, and let us consider the algorithm $\mathbb{A}_{NOP}$ which trivially solves the problem making robots maintain their initial position and primary color $\not\emptyset$. Assume all the robots always have the same egocentric coordinate system considering the super-point as the origin $\mathbf{O} = (0, 0)$. Assume that $(r_0, \ldots, r_{n-1})$ is the round-robin activation sequence of the robots, and let $0 \leq t \leq k$ be the first time where a robot sets its `ser` light to a color $\bar{f} \in \mathfrak{L}_{\mathcal{F}}$ which is already set in at least another robot, say r_h. Note that this time always exists since $|\mathfrak{L}_{\mathcal{F}}| = k < n$. Formally, r_t is activated at time t and computes $\Psi(\mathbb{A}_{NOP}, \mathcal{F})(\sigma) = (\mathbf{O}, \not\emptyset, \bar{f})$, where σ contains the multiplicity $\{(\mathbf{O}, \not\emptyset, \bar{f})\}_{\geq 1}$. Moreover, assume that at time t, all the robots except for r_h and r_t crash, while the two remaining robots remain alive forever. So, from any $t' > t$, the two survival robots will be activated in alternate rounds, will take the same snapshot σ, and will always execute $\Psi(\mathbb{A}_{NOP}, \mathcal{F})(\sigma) = (\mathbf{O}, \not\emptyset, \bar{f})$, thus never detecting the fault. Contradiction achieved. □

The following theorem states the impossibility of distinguishing a not-yet-activated robot from a failed one in the SSYNCH mode.

Theorem 37. *A reliable FD procedure for a model $M \in \mathcal{M}$ under SSYNCH cannot exist.*

Proof. By contradiction, let $\mathcal{F}$ be a reliable FD procedure under $\mathcal{LUMI}^{\mathtt{S}}$ (i.e., the strongest SSYNCH model). Let $\mathcal{R} = \{r_0, \ldots, r_{n-1}\}$ be a swarm under $\mathcal{LUMI}^{\mathtt{S}}$. Suppose all the robots use the same coordinate system Ξ. Suppose the swarm works under a SSYNCH activation scheduler $\mathfrak{A}$ and a crash activation scheduler $\mathfrak{c}$ so that r_0 is the only robot to crash, and it crashes at time t_1. Formally, $\mathfrak{c}(r_0) = t_1$ and $\mathfrak{c}(r_{i\neq 0}) = \infty$. We assume w.l.o.g. that r_1 is the first robot to correctly detect the fault at time $t_2 > t_1$ by running $\mathcal{F}(\sigma)$ where σ is the snapshot of r_1 at time t_2. Moreover, let $r_0 \in \mathfrak{A}(t_1)$ and $r_0 \notin \mathfrak{A}(t')$ for any $t_1 < t' \leq t_2$. Now consider the schedulers $\mathfrak{A}'$ and $\mathfrak{c}'$ so that

$$\mathfrak{A}' = \mathfrak{A} \text{ except for: } \mathfrak{A}'(t_1) = \mathfrak{A}(t_1) \setminus \{r_0\}$$
$$\mathfrak{c}'(r_i) = \infty \; \forall i$$

Under $\mathfrak{A}' \setminus \mathfrak{c}'$, the robots that execute their LCM are the same as under $\mathfrak{A} \setminus \mathfrak{c}$ until t_2, so the evolution of the swarm is the same. Specifically, r_1 is activated at time t_2, takes the same snapshot σ, and executes $\mathcal{F}(\sigma)$, which still detects the fault. However, no crash has occurred. Contradiction achieved. □

The following problem witnesses the impossibility of having a reliable FD procedure under $\mathcal{FSTA}^{Y} \in \mathcal{M}$.

Problem 15 (Flip-Flop)**.** Consider two robots p, q lying on a point O and a third robot r lying on a distinct point x. Robot r must perform these two movements perpetually: reach the symmetric point x' w.r.t. O, and then come back to x.

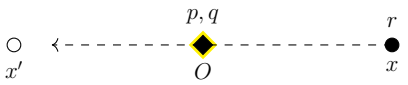


Figure 4.5: Flip-Flop problem.

Theorem 38. *A reliable FD procedure for a model $\mathcal{FSTA}^{Y} \in \mathcal{M}$ cannot exist.*

Proof. By contradiction, let $\mathcal{F}$ be a reliable FD procedure using a palette $\mathfrak{L}_{\mathcal{F}} = \{f_0, \ldots, f_{k-1}\}$ of $k \geq 1$ colors for an $\mathcal{FSTA}$ model. We show that there exists a problem for which $\mathcal{F}$ is not reliable. Consider the problem Flip-Flop that can be trivially solved under any fault-free $\mathcal{FSTA}$ model without the use of colors: let $\mathbb{A}$ be a solving algorithm with $\mathfrak{L} = \{\not\emptyset\}$. It is self-evident that the crash of a robot that is required to be still by the primary problem cannot be detected without assuming communication. In fact, r can never detect whether a robot in O has crashed. However, this impossibility is not limited to the static robots: we prove that p cannot reliably detect the fault of r (the same impossibility holds for q).

Let us denote the following coordinates: $\mathbf{O} = (0, 0)$, $\mathbf{1} = (1, 0)$, and $-\mathbf{1} = (-1, 0)$. Let Ξ^+ be a coordinate system which makes a robot perceives O in position $\mathbf{O}$ and x in position $\mathbf{1}$ (and thus x' in position $-\mathbf{1}$), and let Ξ^- be a coordinate system under which O is in position $\mathbf{O}$ and x in $-\mathbf{1}$ (and thus x' in $\mathbf{1}$). Let $\mathfrak{A}$ be an activation scheduler (any under SSYNCH, FSYNCH, RROBIN) for p, q, r. Consider the swarm under the combined scheduler $\mathfrak{A} \setminus \mathfrak{C}$ where $\mathfrak{C}$ is a suppression scheduler that crashes no robot. Suppose p is activated at times $t_0, t_1, t_2, \ldots$, and suppose that its coordinate system is Ξ^+ when r lies on x and Ξ^- when r lies on x'. In other words, p always perceives r in position $\mathbf{1}$. Since $\mathcal{F}$ is reliable and no robot has crashed, p will always obtain CORRECT by executing $\mathcal{F}$ at times $t_0, t_1, \ldots$. In particular, let $\sigma_0, \sigma_1, \ldots, \sigma_h$ be the set of the snapshots taken with some periodicity by p along its activations $t_0, t_1, \ldots$, where $0 \leq h < k$ and where σ_i has this form[7]: $\langle(\mathbf{O}, \not\emptyset, f_i), (\mathbf{O}, \not\emptyset, \not\emptyset), (\mathbf{1}, \not\emptyset, \not\emptyset)\rangle$.

Now, consider the same swarm under the combined scheduler $\mathfrak{A} \setminus \mathfrak{C}'$ where $\mathfrak{C}'$ only crashes r at time t_0. W.l.o.g., suppose that r lies on x at t_0, and suppose that from t_0 onward p will always have Ξ^+ as coordinate system, thus perceiving r in position $\mathbf{1}$. Thus p will be activated at times $t_0, t_1, t_2, \ldots$ and it will perform $\mathcal{F}(\sigma_0), \mathcal{F}(\sigma_1), \ldots \mathcal{F}(\sigma_h)$ at the same times and periodicity as under $\mathfrak{A} \setminus \mathfrak{C}$, thus never detecting the fault of r. Contradiction achieved. □

As a consequence, we inherit the same impossibility for $\mathcal{OBLOT}$.

Corollary 10. *A reliable FD procedure for a model $\mathcal{OBLOT}^Y \in \mathcal{M}$ cannot exist.*

4.6 Conclusions

In this chapter, we have investigated the ability of correct robots to detect or identify the presence of crashed robots in fault-prone swarms. We have defined the problems associated with this purpose, namely Fault Detection (FD) and Fault Identification (FI), and we have studied the (im)possibility of solving FD or FI under 12 robot models ($\mathcal{OBLOT}$, $\mathcal{FSTA}$, $\mathcal{FCOM}$, $\mathcal{LUMI}$, combined with FSYNCH, RROBIN, SSYNCH). We have formally defined two properties (reliability and punctuality) that an FD or FI procedure should satisfy, and we have soon proved that a reliable FI procedure cannot exist under the strong model $\mathcal{LUMI}^{\mathtt{F}}$. So, we have provided three reliable (or even punctual) FD procedures for the three models $\mathcal{LUMI}^{\mathtt{F}}$, $\mathcal{FCOM}^{\mathtt{F}}$, and $\mathcal{LUMI}^{\mathtt{RR}}$; for the other 9 models, we have proved that it is impossible to design a reliable FD procedure.

In this vein, future works should investigate the (im)possibility of developing reliable FD or FI procedures for models not considered in our work (e.g., operating under *k-bounded* activation schedulers), or finding weaker properties (w.r.t. reliability and punctuality) under which it is possible to solve FD or FI. Furthermore, this

[7] For the sake of conciseness, we here omit the notation for multiplicities.

preliminary study about FD and FI paves the way for subsequent work studying *fault recovery* distributed algorithms, i.e., where correct robots must fix the faulty ones.

Part II

Algorithm Design

Chapter 5

$O(\log n)$-time Uniform Circle Formation

5.1 Introduction

As seen in Section 1.3.4, `Pattern Formation` [Bos+21; SS96; SY99; YS10] is of great importance and interest for swarms of autonomous mobile robots; unsurprisingly, it represents the most studied class of problems for the LCM model. Alongside points (i.e., the `Gathering` problem), circles and regular polygons are the patterns that have received particular attention. The `Circle Formation` problem—requiring the swarm to stop into a *circle configuration*, i.e., where all robots lie on the same (non-degenerate) circle—has been investigated for various models, especially in the early stages of LCM research [Dat+13; DK02; DP08]. Thereafter, several algorithms have been designed to solve the "uniform" variant. The `Uniform Circle Formation` (`UCF`) problem [DK02; DP07a; FMP18; Fel+22; Flo+17; MC20; Vig19] requires robots to arrange on the vertices of a regular n-gon, n being the cardinality of the swarm. Generally, an algorithm for `UCF` is decomposed into two steps [DK02; FMP23a; FMP18; Fel+22; Flo+17]: firstly, a `Circle Formation` solution is provided, then the algorithm solves the `Uniform Transformation` sub-problem [DK02], which asks robots from a circle configuration to distribute on the circle equally.

The well-known geometric properties of a regular polygon (e.g., agreement on both origin and unit distance, equally-spaced distribution, full symmetry) make the `UCF` solution a fundamental algorithmic primitive for a distributed mobile system. From a practical perspective, these properties make a regular layout convenient for a swarm, as it facilitates their communications, visibility, and computations. As a matter of fact, every robot is equidistant from its neighbors and has the same view of the system: this guarantees a fair load balancing and communication, where there are no evident differences in the energy spent in sending messages [FMP23b].

Related work

`UCF` has been investigated under different combinations of robot features in order to identify the minimal sets of assumptions for its solution (see [Vig19] for a survey).

The first studies focus on the $\mathcal{OBLOT}$ standard model, which assumes transparent and non-myopic robots (i.e., with complete visibility): in the context of `UCF`, this means that, at any activation, each robot is aware of the cardinality of the swarm n, and thus on the n-gon to be formed. The algorithms in [DK02] make disoriented $\mathcal{OBLOT}^{\texttt{SSYNCH}}$ robots, initially being on distinct locations, form a circle configuration, and then converge toward a uniform distribution, though without guaranteeing deterministic termination. Note that the assumption of starting from distinct positions (and never

[0]The content of this chapter refers to the work [FMP23a].

Algorithm	Time (in epochs)	Synchronization mode
[FMP18]	$O(1)$	FSYNCH
[FMP23b; Fel+22]	$O(1)$	SSYNCH
[FMP23b; Fel+22]	$O(n)$	ASYNCH
this chapter, [FMP23a]	$O(\log n)$	

TABLE 5.1: Existing UCF deterministic solutions under $\mathcal{LUMI}^*_{\mathcal{O}}$.

creating collisions) is necessary to solve UCF under such a model (two co-located robots might always be activated simultaneously, thus remaining co-located indefinitely). Subsequent works provide improved solutions for the same model [CMN04; DS08; DP07b; Kat05]: e.g., [CMN04] does not rely on the expensive computations of Voronoi diagrams, while [DP07a] proposes a deterministic algorithm for Uniform Circle Formation with $n \geq 5$ robots, n being prime. The case of $n \leq 5$, which includes Square Formation, can be solved using ad-hoc algorithms [DP08; MV16]. The state-of-the-art result for the $\mathcal{OBLOT}$ model with complete visibility is due to [Flo+17], which shows that UCF can be solved under ASYNCH starting from any arbitrary configuration of $n > 5$ robots, without any additional assumption (i.e., chirality, rigidity, etc.). However, the study in [Flo+17] focuses on the computability of UCF and not on its runtime complexity.. We mention other works that consider additional objectives or robots under the non-standard model: notably, [BM18] minimizes the maximum distance traveled by a robot, while [MC18; MC20] solve the problem for fat and/or myopic robots.

Of course, visibility plays a fundamental role in determining whether a problem can be solved, the algorithmic strategies required for its solution, and the resulting computational complexity. The first work studying UCF under the *opaque* setting is [FMP18]. As in [Flo+17], robots are assumed to be non-myopic, completely disoriented, and collision-intolerant. Yet, to overcome the great constraint of obstructed visibility, the $\mathcal{LUMI}^{\texttt{FSYNCH}}_{\mathcal{O}}$ model with rigid movements has been considered first[1]. Specifically, [FMP18] provides the first UCF solution in the opaque setting using $O(1)$ time[2] and $O(1)$ colors. In the subsequent work [FMP23b; Fel+22], the authors solve the problem under the same base model, but considering the three main modes FSYNCH, SSYNCH, and ASYNCH, all using $O(1)$ colors; however, in terms of time, the ASYNCH algorithm requires $O(n)$ epochs, while the synchronous algorithms solve the problem in $O(1)$ time. Although they maintain the same asymptotic complexity, the new FSYNCH solution in [FMP23b; Fel+22] improves the algorithm presented in [FMP18] since it requires three instead of six rounds in the worst case. See Table 5.1 for a summary of the existing algorithms for UCF. So far, solving UCF for such an opaque model, but without memory and/or communication means (e.g., $\mathcal{OBLOT}$), is still an open question.

The algorithms in [FMP23b; FMP18; Fel+22] solve UCF by splitting it into three sub-problems:

1. Convex Hull Formation. Starting from the initial configuration where all the robots are OFF-colored and on distinct points, robots use the algorithms in [Sha+16] (for FSYNCH and SSYNCH) and [SVT21] (for ASYNCH) to arrange themselves on the vertices of a convex hull, thus achieving mutual visibility in the

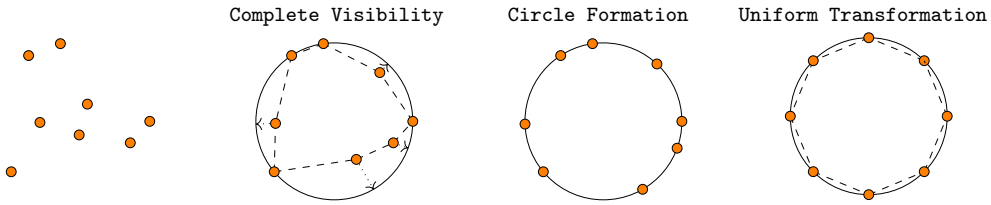
[1] We remind that $X^Y_{\mathcal{O}}$ denotes the opaque version of the model X^Y.
[2] Remember that time is measured in epochs, or in rounds (in the FSYNCH mode).

swarm. Note that both the algorithms in [SVT21; Sha+16] work in $O(1)$ time and colors.

2. `Circle Formation`. Robots are easily made to move radially to reach their *smallest enclosing circle* (SEC). Even in this case, this phase can be done in $O(1)$ time and colors, even under `ASYNCH`.

3. `Uniform Transformation`. Original strategies are proposed to make robots uniformly arrange themselves on the same circle, using $O(1)$ colors. Yet, it is precisely this third phase that is $O(n)$-time in [FMP23b; Fel+22] for the `ASYNCH` mode.

Table 5.2 depicts the configurations obtained by executing these three phases.



Initial configuration	Convex configuration	Circle configuration	Regular configuration
$\mathcal{C}_{init}$	$\mathcal{C}_{convex}$	$\mathcal{C}_{circle}$	$\mathcal{C}_{regular}$

TABLE 5.2: Phases for solving `UCF`.

The objective here is to provide a better solution in terms of time for `UCF` under the `ASYNCH` mode. Note that, being `Arbitrary Pattern Formation` a macro-problem which encompasses `UCF`, at first glance, the solution in [VST22] might be suitable for our purpose, since it solves `Arbitrary Pattern Formation` in $O(\log n)$ epochs using $O(1)$ colors under `ASYNCH`. However, such a solution is not compatible with our assumptions; in fact, it assumes that the target n-gon positions are given to robots as input a priori. Hence, the need of an ad hoc strategy, where the target configuration is not an input to the swarm.

Contribution

This chapter provides an improved solution for the `Uniform Transformation` sub-problem in the `ASYNCH` mode, using a number of epochs which is *logarithmic* in the number of robots. As a corollary, this provides a $O(\log n)$-time algorithm for `UCF` under the $\mathcal{LUMI}_{\mathcal{O}}^{\texttt{ASYNCH}}$ model, considering disoriented and collision-intolerant robots.

The first phase of our solution aims to split the circle configuration into uniform sectors and select some robots that will play fundamental roles in the later phases. A key tool for this essential phase is the formalization of geometric properties of swarm configurations through *circular strings* [CF02; HN16; LS62; Shi81]. Note that strings and their lexicographical order have been widely adopted in algorithm design for the LCM model [CP02; Das+15; DP07a; DP07b; EP07] to analyze the configurations of robots, to provide robots with orientation and sorting, and to solve the `Leader Election` primitive. Moreover, specific patterns of strings have been defined to formalize particular geometric patterns: in [DP07a], the authors solve `UCF` for asymmetric configurations by using the *Lyndon words* to elect a leader robot among the swarm. In [DP07b], the authors introduced the *Swing words* to formalize specific configurations and solve `UCF`. Aware of this theoretical parallelism between strings

and robot configurations, this chapter presents a general, exhaustive, and rigorous formalization of any arbitrary circle configuration through circular strings and an appropriate formal method to check the correctness of the results. For this purpose, several theorems and lemmas are presented in order to prove our strategies and outline the significant properties of robot configurations. We hope this treatment serves as a foundation for future work, providing a starting point for the development of other algorithms.

5.2 Model

We present in detail the model of robots under which we study the `Uniform Circle Formation` problem. We consider a swarm of n robots which, as usual, share the core features presented in Section 1.2.1 (namely, punctiform entities acting in the Euclidean plane, anonymous, indistinguishable, homogeneous, deterministic, and autonomous). Moreover, we fix some other features.

- we consider the $\mathcal{LUMI}^{\texttt{ASYNCH}}$ model, where robots have a $O(1)$-size palette of colors;
- robots are completely disoriented (i.e., no agreement on any component of their local coordinate systems);
- movements are *rigid* (Rig), thus no adversary can stop the robots' movements;
- robots are *collision intolerant*, thus they start from multiplicity-free configurations and they have to avoid collisions[3];
- robots are non-myopic (i.e., their visibility does not depend on the distance), but *opaque* (so, we consider $\mathcal{LUMI}_{\mathcal{O}}^{\texttt{ASYNCH}}$), thus they suffer from obstructed visibility in case of collinearities.

5.3 Configuration analysis

We will use the following notations to denote different configuration types for a swarm of $n \geq 3$ robots (refer to Table 5.2).

- $\mathcal{C}_{init}$ denotes an initial configuration where the robots are `OFF`-colored and on distinct points of the plane;
- $\mathcal{C}_{convex}$ denotes a multiplicity-free configuration where the robots form a convex polygon (*convex configuration*);
- $\mathcal{C}_{circle}$ denotes a multiplicity-free configuration where the robots lie on the same circle (*circle configuration*);
- $\mathcal{C}_{biangular}$ denotes a circle configuration where two non-zero angles $a \neq b$ exist such that the sequence of central angles of adjacent robots alternate a and b;
- $\mathcal{C}_{regular}$ denotes a configuration where the robots form a non-degenerate regular n-gon (*regular configuration*).

5.3.1 Circular configurations

Let $r_0, \ldots, r_{m-1}$ be m distinct robots lying ordered on a curve (e.g., line, circle) according to a fixed orientation. We say that they are *consecutive* if they appear in sequence by traveling along the curve (without looping, if the curve is closed). We say they are *adjacent* if, for every $0 \leq i < m-1$, no other robot lies on the arc $\overset{\frown}{r_i r_{i+1}}$.

[3]We refer the reader to the definition of collision in Section 1.2.7.

Given a regular n-gon, we refer to its *base angle* as the angle $\frac{2\pi}{n}$. Unless otherwise stated, given two non-antipodal[4] points a and b on a circle, we refer to the arc $\overset{\frown}{ab}$ as the *minor arc* joining a and b.

Consider a circle configuration $\mathcal{C}_{circle}$, and let us call Cir the circle where the robots lie. Given a listing $r_0, \ldots, r_{n-1}$ of n adjacent robots on Cir, centered in O, we define the sequence $\alpha_0 \cdots \alpha_{n-1}$ as the corresponding *angle-string*, where $\alpha_i = \angle r_i O r_{(i+1) \bmod n}$ is the central angle. Thus, we can study the geometric properties of a circle disposition by considering the circular strings formed by the angle-strings. From this viewpoint, spotting special geometric properties amounts to analyzing some properties on the circular strings.

Circular strings

Let $x = x_0 \cdots x_{l-1}$ be a *linear string* on an alphabet Σ (i.e., set of *symbols*). From now on, we simply call it *string*. We denote the *length* of x by $|x| = l$, the *empty string* by ε, and the reverse string of x by $x^R = x_{l-1} \cdots x_0$. We denote with Σ^* the set of all the strings with symbols in Σ, while $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$. A *factor* of x is ε or any string $x_i x_{i+1} \cdots x_j$ such that $0 \leq i \leq j \leq l-1$. We define the *k-shift* of x the string $\sigma_k(x) = x_k \cdots x_{l-1} x_0 \cdots x_{k-1}$, where $0 \leq k \leq l-1$. A string x is a *palindrome* if $x = x^R$. A *mirrored* string is a palindrome with an even length. We say that x is *symmetric* if *(i)* x is a palindrome, or *(ii)* $x = ayby^R$ where $y \in \Sigma^+, a, b \in \Sigma$. A string x is a *power string* whenever there exists a factor y of x, such that $y \neq \varepsilon$, $y \neq x$ and $x = y^k$ for an integer k. On the contrary, x is a *base string* if it is not a power string.

Given a linear string $x = x_0 \cdots x_{l-1}$, the corresponding *circular string* is the multiset $\boldsymbol{\sigma}(x) = \{\!\{\sigma_0(x), \ldots, \sigma_{l-1}(x)\}\!\}$. A circular string is *aperiodic* if it contains only base strings. A circular string is said to be *symmetric* if it contains a symmetric string.

Symmetries and circular strings

For simplicity, we use $\mathcal{C}$ in place of $\mathcal{C}_{circle}$. Let us consider the listing of adjacent robots $r_0, \ldots, r_{n-1}$ on Cir with the corresponding angle-string $\alpha = \alpha_0 \cdots \alpha_{n-1}$. Note that an angle-string is a string on the alphabet $\Sigma_{2\pi} = (0, 2\pi)$.

From each robot r_i, two angle-strings start: $\sigma_i(\alpha)$ (clockwise) and its reverse $\sigma_{n-i}(\alpha^R)$ (counterclockwise). We denote these relations with the following notation: $r_i \curvearrowright \sigma_i(\alpha)$ and $r_i \curvearrowleft \sigma_{n-i}(\alpha^R)$. We call *configuration strings* of $\mathcal{C}$ the circular string $\boldsymbol{\sigma}_{\mathcal{C}} = \boldsymbol{\sigma}(\alpha)$ and $\boldsymbol{\sigma}^R_{\mathcal{C}} = \boldsymbol{\sigma}(\alpha^R)$. According to the properties of $\boldsymbol{\sigma}_{\mathcal{C}}$ (clearly the same holds for $\boldsymbol{\sigma}^R_{\mathcal{C}}$), the configuration $\mathcal{C}$ comes in three different cases of *geometric* symmetry:

- in case of asymmetry, $\boldsymbol{\sigma}_{\mathcal{C}}$ is aperiodic and not symmetric;
- in case of symmetry with one axis, $\boldsymbol{\sigma}_{\mathcal{C}}$ is symmetric. In particular, three sub-cases exist according to the number of *axis robots* (i.e., robots lying on the endpoints of the symmetry axis):
  - one axis robot (odd n), $\boldsymbol{\sigma}_{\mathcal{C}}$ contains a palindrome in the form xax^R, with $a \in \Sigma_{2\pi}$, $x \in \Sigma^+_{2\pi}$;
  - two axis robots (even n), $\boldsymbol{\sigma}_{\mathcal{C}}$ contains two mirrored strings in the form xx^R and $x^R x$, with $x \in \Sigma^+_{2\pi}$;
  - zero axis robots (even n), $\boldsymbol{\sigma}_{\mathcal{C}}$ contains two symmetric strings in the form $axbx^R$ and $bx^R ax$, with $a, b \in \Sigma_{2\pi}$, $x \in \Sigma^+_{2\pi}$.

  In all the above three sub-cases, no other symmetric strings are contained in $\boldsymbol{\sigma}_{\mathcal{C}}$;

[4] Two points on a circle are antipodal if they lie on the distinct endpoints of the same diameter.

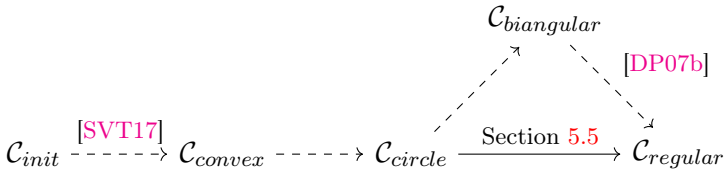
- in case of rotational symmetry, all the strings in $\boldsymbol{\sigma}_{\mathcal{C}}$ are power strings in the form x^k where x is a base string. If $x = ab$ with $a \neq b \in \Sigma_{2\pi}$, then $\mathcal{C}$ is *biangular*.

5.4 Problem and sub-problems

We provide a formal definition of the problem under study.

Definition 12 (UCF)**.** *Given an initial configuration $\mathcal{C}_{init}$ where $n \geq 3$ robots are* OFF*-colored and on distinct points of the plane, the* Uniform Circle Formation *problem requires the swarm to stop in a configuration $\mathcal{C}_{regular}$ where robots are arranged on the vertices of a regular (and non-degenerate) n-gon. Formally, the robots have to form the pattern $\Pi = \{(\cos(\frac{2j\pi}{n}), \sin(\frac{2j\pi}{n}))\ \ \forall\, 0 \leq j \leq n-1\}$.*

As already mentioned (see Table 5.2), we solve UCF by splitting it into sub-problems, each one transforming a configuration from one type to another one. We outline the preliminary phases (dashed lines in the diagram) that precede our Uniform Transformation solution (solid line in the diagram), and complete the proposed UCF algorithm.



5.4.1 Convex hull formation

Starting from $\mathcal{C}_{init}$, robots exploit the Complete Visibility solution in [SVT17] (which solves Convex Hull Formation), to arrange robots on the vertices of a convex polygon, forming $\mathcal{C}_{convex}$. Indeed, $\mathcal{C}_{convex}$ ensures mutual visibility to all the robots in the swarm, which now know which n-gon they have to form. This algorithm works in $O(1)$ time and colors.

5.4.2 Circle formation

Let us call $SEC(\mathcal{C}_{convex})$ as Cir, i.e., the smallest circle enclosing the points in $\mathcal{C}_{convex}$. The robots already on Cir (which are at least two if antipodal or three otherwise) will set their color to onSec, and stay still. Instead, each internal robot moves along its radial trajectories to reach Cir, thus forming $\mathcal{C}_{circle}$. As explained in [PS24], such a simple procedure safely (i.e., without collisions) transforms $\mathcal{C}_{convex}$ into $\mathcal{C}_{circle}$ using $O(1)$ time and colors, even in ASYNCH. Notably, since onSec robots remain visible to the internal robots even in the presence of asynchronous movements, they will maintain the target circle Cir throughout this phase. W.l.o.g., we assume that, once formed $\mathcal{C}_{circle}$, all robots reset to OFF.

5.4.3 Biangular case

If the formed $\mathcal{C}_{circle}$ is a biangular configuration, we can borrow the solution in [DP07b] to achieve the target regular configuration. As stated in [DP07b], $\mathcal{C}_{biangular}$ has these properties (refer to Figure 5.1):

- Let P be the n-gon formed by the robots in $\mathcal{C}_{biangular}$. Then, $n = 2h$ for some $h \geq 2$.

- P has h axes of symmetry passing through the midpoint of its edges.
- P has h *short edges* (related to the angles a) and h *long edges* (related to the angles b), in an alternate order.
- The intersection points of the lines on which the short edges lie form a regular h-gon. We denote it as $IL(P)$. Let O be the center of $IL(P)$.
- Within $IL(P)$, we can construct the regular n-gon *associated* to $IL(P)$ in this way: for each edge $\overline{xy}$ of $IL(P)$, place two points x', y' centered in the edge (i.e., so that $dist(x, x') = dist(y, y')$), and so that $\angle x'Oy' = \frac{2\pi}{n}$. The set of all the points so constructed forms the associated regular n-gon, denoted as P'.
- in $\mathcal{C}_{biangular}$, robots lie on alternated edges of P'.

In the strategy developed in [DP07b], P' becomes the target regular n-gon on which robots must arrange themselves. Notably, we first make robots set their color to `biangular`, thus indicate the strategy to be adopted; then we make them slide on the edges of P' until they stop at its vertices. Indeed, these sliding movements are collision-free and do not create collinearities. Note that, even considering `ASYNCH`, the target n-gon P' remains invariant and robots can always compute it. However, as evident in Figure 5.1, we highlight that P' is not contained in Cir. We anticipate that in Chapter 6, we will provide an improved solution for `UCF` by which robots form the target regular polygon without moving outside $SEC(\mathcal{C}_{init})$.

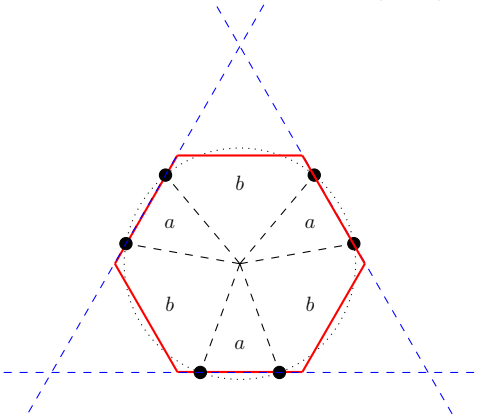


FIGURE 5.1: Six robots in $\mathcal{C}_{biangular}$ forming the hexagon P. The blue equilateral triangle is $IL(P)$. The red hexagon is the target regular hexagon P'. Robots slide on the edges of $IL(P)$.

5.5 The `Uniform Transformation` algorithm

This algorithm transforms $\mathcal{C}_{circle}$ (when it is not biangular) in $\mathcal{C}_{regular}$, making robots uniformly arrange on Cir.

5.5.1 Notion and techniques

Regular tuple. As we will see, starting from $\mathcal{C}_{circle}$, our algorithm uniquely determines the target regular n-gon, say Π', inscribed in Cir and so that the robots must arrange on its vertices[5]. Our algorithm consists of two initial phases where triples or 4-tuples of robots will move to some vertices of Π' and set their lights with special

[5]Using the notation in Section 1.3.4, we have that $\Pi' \in \mathscr{P}(\Pi)$, where Π is the regular n-gon defined in Definition 12.

colors [6] or the `placeholders` (`pivot` and `angle`) to communicate "we are the reference robots". In particular, we call *regular tuple* the tuple of consecutive robots in the form (`angle`, `pivot`, `angle`) or (`angle`, `pivot`, `pivot`, `angle`), lying on adjacent vertices of Π'. Accordingly, it holds that, if $(r_1, \ldots, r_k)$ is a regular tuple on Cir centered in O (with $k \in \{3, 4\}$), then $\angle r_i O r_{i+1} = \frac{2\pi}{n}$ for $1 \leq i < k$.

Uniform sectors. Regular tuples are also set to partition Cir into *uniform sectors*, i.e., circular sectors of the same central angle, containing the same number of robots. Let τ, τ' be two regular tuples such that a, a' (resp. b, b') are the points where the `angle` robots of τ (resp. τ') lie on, such that a, a', b, b' are consecutive. Let us consider the two disjoint arcs $\overset{\frown}{ab'}$ and $\overset{\frown}{ba'}$. If in at least one of the two arcs, no regular tuple is formed (or must be formed), we say that τ and τ' are adjacent. Given two adjacent regular tuples τ, τ' defined as above, let m_a (resp. m_b) be the middle point on the minor arc aa' (resp. bb'). We call the chord $\overline{m_a m_b}$ as *splitting chord* of τ, τ'. Given a splitting chord ρ on Cir, we call *performance arc* of ρ, denoted as $\overset{\frown}{\rho}$, the minor arc cut by ρ. The sector of Cir defined by ρ is a uniform sector. We call *performance area* of ρ the region of the plane φ_ρ such that *(i)* it contains the performance arc $\overset{\frown}{\rho}$ and *(ii)* it is delimited by ρ and the two distinct straight lines, perpendicular to ρ, each one passing through one of the two endpoints of ρ. Figure 5.2 shows the explained elements. Note that, if ρ is a diameter, it defines two opposite performance arcs/areas.

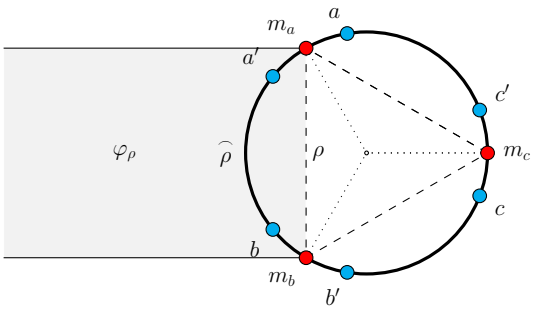


FIGURE 5.2: Cir is partitioned into three uniform sectors (delimited by dotted lines) defined by three regular triples in the form (`angle`, `pivot`, `angle`). ρ is a splitting chord (dashed line), $\overset{\frown}{\rho}$ is its performance arc, and φ_ρ is its performance area (gray region).

Algorithm overview. We list the main phases used in our algorithm.

- At first, we split Cir into uniform sectors whose performance arcs are of equal length and robot cardinality, such that all the robots on each arc (or on each half-arc) are *oriented*, i.e., they agree on a common clockwise direction. In this part, we use circular strings to formalize the geometric properties of the configurations, leading to an interesting parallelism between Cir splitting and string factorization

- To fix the performance arcs, some robots per arc are elected and made to move in specific positions on Cir, around each arc endpoint. These reference robots

[6] We use `this font` to denote the exact color of a robot (e.g., `guard, guard_c, pivot` ...), whereas we indicate the role of a robot by the normal text (e.g., pivot, guard). Multiple colors can be used for the same role (e.g., a guard robot can assume the colors `guard, guard_c, moving_guard` ...). For the sake of clarity, we often prefer `this font` to refer to the specific robots.

will form the regular tuples which will be used by a robot r to *(i)* reconstruct the original Cir where the regular n-gon Π' must be formed, *(ii)* recompute its base angle and so the global number of robots in the system, *(iii)* determine the performance area where r has to move, and *(iv)* detect the group of robots (on the same performance area of r) with which r has to collaborate.

- Performance areas partition the swarm into groups of robots. Each group will act independently of the others, in its own original performance area. This allows the next phases to be executed in parallel by each group.

- Within each performance area, robots equally distribute themselves on the performance arc by executing the same routine. This routine includes a loop that iterates a sequence of nine phases, where each phase is fulfilled within a constant number of epochs. Since the sequence of phases is iterated an amount of time which is *logarithmic* in the number of robots acting in the performance area, we obtain an exponential decrease in the running time, when compared to existing algorithms for `UCF` [FMP23b; Fel+22]. Such a routine can be adapted into a stand-alone algorithm that uniformly arranges robots along a given arc.

- The time-complexity improvement from $O(n)$ [FMP23b; Fel+22] to $O(\log n)$ is given by the "pairing technique" used in the phase loop: robots pair themselves and use their mutual distance to encode all the needed information to reach their target destinations.

5.5.2 Asynchronism

The main challenge in the `ASYNCH` mode with opaque and collision-intolerant robots is the management of moving robots, which can *(i)* hide other robots or *(ii)* be hidden by other robots.

To address issue *(i)*, our algorithm makes a robot color itself as `moving_<x>` before starting to move, where `<x>` will be the color it must set at its next activation without doing anything else. A phase of our algorithm is said to be *fully parallelized* if robots can safely execute their Move step (update their color or position) even if they see other `moving_*` robots. Instead, given a region ω on $\mathbb{R}^2$, we say that a phase for a subset of robots S is ω-*synchronized* if it requires any robot $r \in S$ to skip its Move step (i.e., update neither its color nor position) if r sees a `moving_*` colored robot in ω, different from itself. The region of synchronization of a robot can be the whole $\mathbb{R}^2$ plane or its performance area, according to the phase to be executed.

For the issue *(ii)*, our algorithm guarantees no collinearity with moving robots creating ambiguous snapshots.

Let us show the algorithm phase by phase. For the sake of brevity, our explanation will omit the details of the `moving_*` colors, assuming their need and logic as understood. Refer to Section 5.6 for the correctness proofs.

5.5.3 Phase Split

For simplicity, we refer to $\mathcal{C}_{circle}$ as $\mathcal{C}$. In the first phase, some robots have to set their lights to fix the splitting chords. Moreover, in a particular configuration, at most one robot is required to travel. Starting from the configuration $\mathcal{C}$, our strategy is to select the splitting chords which split Cir into performance arcs with the same length and with the same number of robots (possibly with just one robot of difference). The method to select such arcs depends on the geometric symmetry degree of $\mathcal{C}$.

Asymmetry. In this case, every robot can elect the lexicographically smallest string y over $\boldsymbol{\sigma}_{\mathcal{C}} \cup \boldsymbol{\sigma}_{\mathcal{C}}^R$ and so the robot p from which y starts (i.e., either $p \curvearrowright y$ or $p \curvearrowleft y$), such that the diameter passing through p splits Cir into two halves with the same robot cardinality (except for just one exceeding robot at most). Note that such a diameter always exists (see Theorem 1 in [Fel+22], which has been reported in Lemma 33 in Section 5.6). In this case, the selection of y is always taken unambiguously (see Lemma 36 in Section 5.6.3). The diameter passing through p is elected as the splitting chord for $\mathcal{C}$.

In this phase, p sets its light as `pivot` and fixes the splitting chord. If n is even, on the other endpoint e of the splitting chord, a robot is elected to move to e where it will assume the color `pivot_a` once reactivated. For this purpose, we elect the robot already on e, if it exists, otherwise we elect the closest robot to e belonging to the most populated half-Cir.

Symmetry with just one axis. In this case, robots elect the diameter on the axis of symmetry as the splitting chord. The elected splitting chord, say d, can pass through 0, 1, or 2 opposite robots. In this phase, the robots lying on d become pivots and set their light as `pivot`. If no robot lies on d, we have to color just two robots, symmetric to d, with the color `placeholder`, to univocally fix the splitting chord (in fact, given two distinct points on a circle, there exists a unique axis of symmetry for the two points which does not pass through them). To establish an unambiguous method to elect the symmetric placeholders, we can elect the farthest robots to d (in case of equal distance, we choose according to the lexicographic order of the half-Cir angle-string).

Rotational symmetry. In this case, all strings in $\sigma(\mathcal{C})$ are power strings. Let $\alpha = \alpha_0 \cdots \alpha_{n-1}$ be an angle-string in $\sigma(\mathcal{C})$, and let β be the base string such that $\beta^k = \alpha$ (the configuration $\mathcal{C}$ can be divided into $k > 1$ identical sectors, each being the $\frac{2\pi}{k}$-rotation of the previous one). We call $\boldsymbol{\sigma}(\beta)$ and $\boldsymbol{\sigma}(\beta^R)$ the *sector circular strings* of the configuration. We note that they are *aperiodic*. Consider the ordered list of the adjacent robots on Cir $r_0, \ldots, r_{n-1}$ which forms α. Let $P_i = \{r_j \text{ s.t. } j \equiv i \bmod \frac{n}{k}\}$ be the *class of symmetry* which contains k robots sharing the same position in the k different sectors. We can observe that, for each robot $r \in P_i$, $r \curvearrowright \sigma_i(\alpha)$ and $r \curvearrowleft \sigma_i(\alpha)^R$. Our strategy now selects one or two classes of symmetry among $P_0, \ldots, P_{\frac{n}{k}-1}$ according to a particular property of their associated angle-strings. Observe that

Observation 6. Since each angle-string $\sigma_i(\alpha)$ (resp. $\sigma_i(\alpha)^R$) is the k-power of $\sigma_i(\beta)$ (resp. $\sigma_i(\beta)^R$), we can analyze the properties of the angle-strings just taking into account their factors $\sigma_i(\beta)$ and $\sigma_i(\beta)^R$.

We say that P_i *reads* $\sigma_i(\beta)$ and $\sigma_i(\beta)^R$ (formally $P_i \curvearrowright \sigma_i(\beta)$ and $P_i \curvearrowleft \sigma_i(\beta)^R$) since the robots in P_i are the starting points for these strings (in opposite directions).

According to this, our algorithm unambiguously chooses *just one* string, say $\tilde{\gamma}$, within $\boldsymbol{\sigma}(\beta) \cup \boldsymbol{\sigma}(\beta^R)$, which is read exactly from *one or two* classes of symmetry. In particular, let Γ_1 and Γ_2 be the sets of strings in $\boldsymbol{\sigma}(\beta) \cup \boldsymbol{\sigma}(\beta^R)$, such that Γ_1 (resp. Γ_2) contains the strings read by just one class (resp. 2 classes) of symmetry. In particular, we can have two scenarios, according to the size of Γ_1:

- if $\Gamma_1 \neq \varnothing$, we unambiguously select a string $\tilde{\gamma}$ from Γ_1 (e.g., the lexicographically smallest one) and the class of symmetry, say P_i, which reads $\tilde{\gamma}$;

- if $\Gamma_1 = \varnothing$, we properly[7] select a string $\tilde{\gamma}$ from Γ_2 and the two classes of symmetry, say P_i and P_j, which read $\tilde{\gamma}$.

We call P_i and, if it exists, P_j *eligible classes of symmetry*. Note that we cannot have more than two classes of symmetry reading the same string in $\boldsymbol{\sigma}(\beta)$ and $\boldsymbol{\sigma}(\beta^R)$, otherwise the sectors would contain sub-sectors in turn, i.e., β would not be a base string (see Lemma 37 in Section 5.6.3 for the proof). Other properties about robot classes of symmetry (in particular about Γ_1, Γ_2, and so the presence of one or two eligible classes of symmetry) are explained in Section 5.6.3. For example, note that, if $|\beta|$ is odd or $\boldsymbol{\sigma}(\beta)$ contains a mirrored string, then Γ_1 contains at least a string, i.e., we can elect a unique eligible class (see Theorems 41 and 42 in Section 5.6.3 for the proofs). Let us now show how Cir is split in both scenarios (see Figures 5.3a and 5.3b).

One eligible class of symmetry. Each robot computes $\boldsymbol{\sigma}(\beta)$ and $\boldsymbol{\sigma}(\beta^R)$, and selects $\tilde{\gamma}$ and P_i as explained above. The k robots in P_i form the pivots (which will not move for the whole algorithm) by setting their color as `pivot`. The chords joining pivots belonging to adjacent sectors will be the splitting chords.

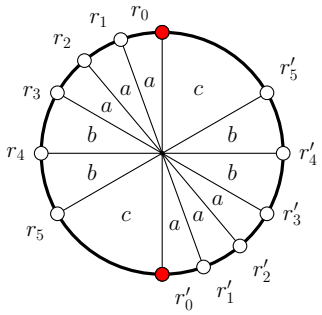
Two eligible classes of symmetry. Each robot computes $\boldsymbol{\sigma}(\beta)$ and $\boldsymbol{\sigma}(\beta^R)$ and selects properly $\tilde{\gamma}$, P_i, and P_j. The $2k$ robots in $P_i \cup P_j$ play the role of placeholders and set their color as `placeholder`. At the end of this phase, there will be $2k$ placeholders, two for each sector. Moreover, each placeholder forms two different angle-strings with its adjacent[8] placeholders. Hence, it is important to share a common method to pair adjacent placeholders in k disjoint pairs; such pairs will be used in the next phase to define the splitting chords and so delimit the k performance areas. In fact, the chords passing through the middle points of the selected pairs of placeholders will be the splitting chords. Between the two different ways to pair adjacent placeholders, robots can always choose a proper and unambiguous method to form the pairs (and so the splitting chords) so that, in the next phase, the regular tuples can be formed safely. Note that for each pair of adjacent placeholders, an axis of symmetry passes between them (see Theorem 43 in Section 5.6.3). Moreover, no robot lies on the endpoint of these axes of symmetry (see Theorem 44 in Section 5.6.3).

5.5.4 Phase Regular Tuples

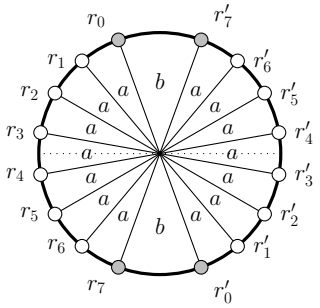
This phase requires some robots to change color and move for completing the regular tuples. The positions of these tuples are fixed by the presence of the `pivots` or the `placeholders`, which have been set in the previous phase. In particular, assume the current configuration presents k splitting chords $\rho_1, \ldots, \rho_k$ defined by `pivots` and `placeholders`. From now on, no robot will cross the splitting chords: each robot remains in its original performance area. Consider a splitting chord ρ, its two endpoints e_0, e_1, and its performance area φ_ρ. In this phase, one or two robots per endpoint e_i are elected on $\widehat{\rho}$, and move to complete the regular tuple around e_i in the form (`angle`, `pivot`, `angle`) or (`angle`, `pivot`, `pivot`, `angle`). The first case arises when the pivot already lies on e_i, so one robot on $\widehat{\rho}$ elects itself as an angle robot, moves in a position a on $\widehat{\rho}$ where it will set its light as `angle` for forming the base angle $\frac{2\pi}{n}$ with the pivot. Note that a second angle robot will do the same in the adjacent performance area, completing the regular triple centered on e_i. The second case arises when no

[7] In order to guarantee complete visibility during regular tuple setting.

[8] Two placeholders p and r are adjacent if no other placeholder lies on the arc $\widehat{pr}$.



(A) Pivot coloring in a 2-rotation configuration with 6 classes of symmetry P_i, $i \in \{0, \ldots, 5\}$. Here, $\tilde{\gamma} = a^3b^2c$ starts from just one eligible class $P_0 = \{r_0, r'_0\}$. The splitting chord is the diameter joining the two pivots.



(B) Placeholder coloring in a 2-rotation configuration with 8 classes of symmetry P_i, $i \in \{0, \ldots, 7\}$. Here, $\tilde{\gamma} = a^7b$ starts from two eligible classes $P_0 = \{r_0, r'_0\}$ and $P_7 = \{r_7, r'_7\}$. The dotted diameter is elected as the splitting chord.

FIGURE 5.3: *Cir* splitting in the rotation case.

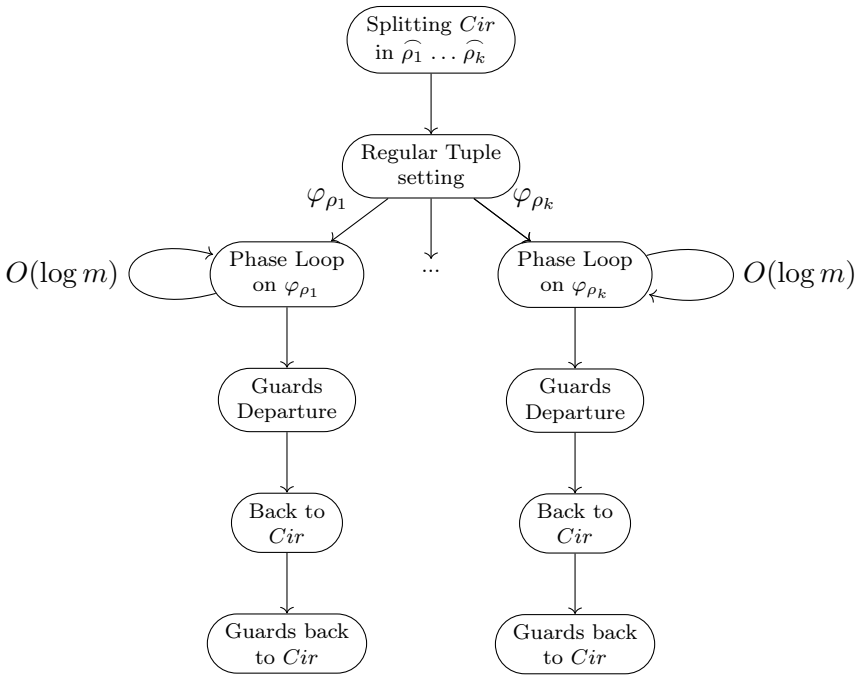
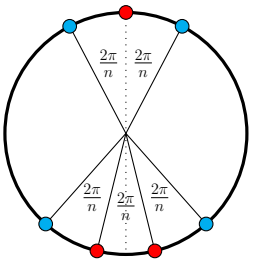


FIGURE 5.4: Diagram depicting the main phases of our `Uniform Transformation` algorithm. Each branch can be executed in parallel in the related performance area φ_{ρ_i}.

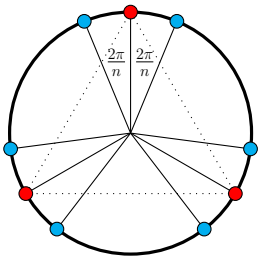
robot lies on e_i, and ρ is held by *(i)* the pairs of placeholders or *(ii)* the pivot on the opposite endpoint e_{1-i} (cases of asymmetry and symmetry with odd n). In this case, two robots on $\widehat{\rho}$ elect themselves as pivot and angle, must set their light properly, and move to complete the 4-tuple centered on e_i. Figure 5.5 shows the regular tuple setting for different configuration types.

This phase is φ_ρ-synchronized, i.e., a robot r in φ_ρ skips its Move step only if r *(i)* is not `moving_*` colored and *(ii)* sees a `moving_*` robot belonging to φ_ρ. In fact, other moving robots do not prevent r from executing its Move step. Note that there always exists an unambiguous strategy used by the swarm to *(i)* elect the robots which

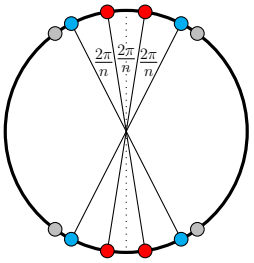
will become angle robots and, possibly, pivots, *(ii)* make them move without colliding and always guaranteeing complete visibility to the swarm, and *(ii)* completing the k regular tuples in a constant number of epochs, potentially in parallel (see Theorem 40 in Section 5.6.3). Eventually, the `pivot_a` robot (asymmetry case) turns its light into `pivot`, aligning its color with all other pivots. Once all the regular tuples have been set, all `pivots` and `angle` robots will do nothing. Placeholders can turn off their lights since their role is no longer needed.



(A) Regular tuple setting in case of symmetry with one axis. The splitting chord lies on the axis of symmetry.



(B) Regular tuple setting in case of 3-rotation (with one eligible class). The splitting chords join the three `pivots`.



(C) Regular tuple setting in case of 2-rotation (with two eligible classes), around the pairs of `placeholders`.

FIGURE 5.5: Regular tuple setting with `pivot` (here *red*), `angle` (*cyan*), and, possibly, `placeholder` (*lightgray*) robots.

5.5.5 Phase loop

Let φ_ρ be a performance area cut by the splitting chord ρ. Let $\widehat{\rho}$ be its performance arc on Cir, and let β_ρ be the angle-string related to the robots on $\widehat{\rho}$. If β_ρ is not a palindrome, we can establish an "upper" endpoint of $\widehat{\rho}$ according to the lexicographical orientation given by β_ρ. Otherwise, each half-arc of $\widehat{\rho}$ defines its nearest endpoint of ρ as its upper part.

Let m be the number of robots on $\widehat{\rho}$ which are not `pivots` or `angle` robots. If $m < 4$, then the robots can be uniformly arranged along $\widehat{\rho}$ by directly moving the target vertices following the sequential order(s) given by β_ρ.

Otherwise, the m robots execute a sequence of phases in loop, which is iterated until all the m robots have been moved to two particular lines external to Cir, and parallel to ρ. As we will see, at each iteration, half of the robots on $\widehat{\rho}$ leave the arc and move towards these lines, thus leading to logarithmic behavior for the whole algorithm. Note that no robot invades another performance area and the target polygon vertex of each robot is located on its performance arc.

Let k be the number of performance areas on Cir. The k groups of robots perform the same sequence of phases independently and possibly in parallel. However, some critical phases of the loop are φ_ρ-synchronized. Now, let us explain in detail each phase of the loop (see Algorithm 8 for the pseudo-code of the whole phase loop).

Phase TBLG. In this φ_ρ -synchronized phase[9], just 1 or 2 robots have to move on $\widehat{\rho}$. Let g be the middle point on $\widehat{\rho}$. We have to set 3 (if m is odd) or 2 (if m is even) guard robots around g, on $\widehat{\rho}$, in this way:

[9]The phase name derives from the robot roles: top, bottom, loner, and guard.

Algorithm 8: Pseudo-code of the *Phase Loop* on the performance area φ_ρ.

Input: A performance arc $\overset{\frown}{\rho}$ where robots lie on.
$m \leftarrow$ robots on $\overset{\frown}{\rho}$ which are not `pivot` or `angle`;
while $m \geq 4$ **do**
  ***Phase TBLG*:**
    The m robots color themselves as `top`, `bottom`, `loner`, `guard`, and `guard_c` in case;
    Guards arrange themselves as in Figures 5.6a and 5.6b;
  ***Phase Loner Pairing*:**
    The uppermost `loner`s move to pair themselves (see Figure 5.6b);
  ***Phase Δ Setting*:**
    if guards do not see any `projection` robot in φ_ρ **then**
      Outer guards approach each other to fix the distance Δ;
  ***Phase Pair Approaching*:**
    Each `top` robot approaches its `bottom` robot to form a distance δ;
  ***Phase Top robots to h*:**
    Each `top` robot travels towards line h, perpendicularly;
  ***Phase Green Light On*:**
    Guards color themselves as `guard_green` or `guard_c_green`;
  ***Phase Top Robots to l*:**
    `top` robots on h recompute Cir and their target position on it;
    `top` robots move to the projection of their target position on line l, setting their color as `projection`;
  ***Phase Gap Fix*:**
    if the number of `bottom` robots is odd **then**
      A `bottom` robot travels to l on the missing projection, setting its color as `projection`;
  ***Phase Green Light Off*:**
    Guards set their color as `guard_end` or `guard_c_end`;
  $m \leftarrow$ robots on $\overset{\frown}{\rho}$ which are not `pivot`, `angle`, or guards;
The remaining robots on $\overset{\frown}{\rho}$, which are not `pivot`s, `angle`s or guards, reach l;
Output: Performance area φ_ρ such that all robots lie on l (`projection` robots) or on $\overset{\frown}{\rho}$ (`pivot`s, `angle`s, guards).

- if m is odd, the robot closest to g (in case of two equally distant robots, the robot in the upper part of the arc is elected) moves to g and sets its color as `guard_c`. We call this guard the *central guard*[10];
- the two robots closest to g (besides the one which has to head to g when m is odd) must set themselves with the `guard` color. If both are located in the same half-arc, the farthest from g must travel in the opposite half-arc, at a distance d from g which is a fraction of the minimum distance between any two robots on $\overset{\frown}{\rho}$ (this measure assures no collisions can occur). These two guards are called *outer guards*.

Note that no robots lie between two guards (except the central guard if m is odd). All the other robots in $\overset{\frown}{\rho}$, except for the `pivot`s, the `angle`s, and the guards, will color themselves according to the following roles. For each half-arc split by g, the robots pair themselves starting from the upper part of the half-arc (where the regular tuple lies): each pair is composed of the `top` robot and the `bottom` robot (in each `top-bottom` pair, the `top` robot is the closest robot to the related `pivot`). Each `top-bottom` pair

[10] If β_ρ is a mirrored string, then the guard is already on g.

must not have other robots between them. If, for example, there is an odd number of robots between the pivot and the related angle robot, the robot closest to the angle robot remains unpaired. The unpaired robots color themselves as loner. The final configuration is shown in Figure 5.6.

(A) TBLG setting with three guards. Indexing of the top-bottom pairs.

(B) TBLG disposition with two guards, and loner pairing.

FIGURE 5.6: Setting of top (here *violet*), bottom (here *orange*), loner (here *gray*), and guard robots. The top-bottom pairs are highlighted through arcs.

Phase LONER PAIRING. In the previous phase, at most three loners per half-arc of $\widehat{\rho}$ are created: one between the endpoint of $\widehat{\rho}$ and the pivot of the half-arc (if the pivot does not lie on ρ), one between the pivot and the angle robot, and the last one between the angle robot and its closest guard. In this phase, if a half-arc presents more than one loner, the two uppermost loners (according to the upper part of each half-arc) pair themselves: the uppermost loner stays still and lights itself as top, while the second-uppermost one moves towards a position on $\widehat{\rho}$ in order be down with respect to the new top robot, at a relative distance from it (e.g., a fraction of the minimum distance between two robots on $\widehat{\rho}$). If just two loners remain (one per half-arc) and there is an unambiguous method to elect one of them, the elected loner will become a top robot while the second loner will approach it (using the same strategy as before) and become its bottom (see Figure 5.6b). This phase aims to reduce the number of loners, maximizing the number of top-bottom pairs.
Note that this phase is φ_ρ-synchronized. At the end of this phase, at most two loner robots (one per each half-arc of $\widehat{\rho}$) can exist.

Phase Δ SETTING. In this φ_ρ -synchronized phase, the outer guards g_1, g_2 have to approach symmetrically around g (the middle point of $\widehat{\rho}$). Let η be the minimum distance between two robots on $\widehat{\rho}$. If χ is the tangent line to Cir passing through g, let μ be the length of the shortest projection of a segment $\overline{gg_i}$ on χ, for $i \in \{1, 2\}$. Let t be the position of the vertex of the target n-gon Π' which is closest to g, but not lying on g. Let ζ be the length of the projection of the segment $\overline{gt}$ on χ. Then, each outer guard g_i moves to a position g'_i on $\widehat{\rho}$ such that g'_1 is symmetric to g'_2 with respect to g, and such that the distance between g'_1 and g'_2 is $\Delta = \min\{\eta, 2\mu, 2\zeta\}$. This measure assures *(i)* no guards turn away from g and risk colliding with other robots on the arc, and *(ii)* no guards will collide against other robots that will be moved out of Cir in the next phases. The measure Δ will be used as a shared unit distance for the other robots on φ_ρ.

Phase Pair Approaching. In this fully parallelized phase, every `top` robot r approaches its `bottom` robot s by traveling straight towards a point on $\widehat{\rho}$ so that the new distance between r and s is δ, defined as

$$\delta = \frac{\Delta}{\langle n, m', d', i, t, c, z \rangle}$$

where

- Δ is fixed by the outer guards,
- n is the total number of robots (fixed by the base angle in the regular tuple),
- m' is the original number of robots on the arc (`pivots`, `angle`s, and guards excluded),
- d' is the index of the current iteration of the loop (so, $d' = 1, \ldots, \lceil \log_2 m' \rceil$),
- i is the incremental index of the `top-bottom` pair starting from each endpoint of $\widehat{\rho}$ (see Figure 5.6a), such that the pairs closest to each endpoint have index 1 (so, $i = 1, \ldots, \lceil \frac{m'}{4} \rceil$),
- t is the number of `top` robots in the performance area,
- $c \in \{0, 1\}$ is a flag such that if $c = 0$ then the target robot vertex is on the current half-arc where r already sits, otherwise the target vertex of r is on the other half-arc of $\widehat{\rho}$,
- $z \in \{0, 1, 2\}$ is a flag such that, if $\widehat{\rho_{1/2}}$ is the half-arc of $\widehat{\rho}$ where the target vertex for r is located, then
  - $z = 0$ if the endpoints of ρ are either both covered by `pivots` or no `pivot` lies on the endpoints;
  - otherwise, $z = 1$ if the external endpoint of $\widehat{\rho_{1/2}}$ is covered by a `pivot`[11];
  - otherwise, $z = 2$ (i.e., if no `pivot` lies on the external endpoint of $\widehat{\rho_{1/2}}$).
- $\langle\ \rangle : \mathbb{N}^7 \to \mathbb{N}$ is the Cantor tuple function[12].

Note that each robot on Cir has complete visibility of all the robots on or inside Cir since `top` robots do not create collinearity with them during their movements. Note also that the setting of Δ guarantees that all `top` robots have to approach (and not turn away) the related `bottom` robots to form the distance δ, avoiding collisions with other robots on the arc. Moreover, at the end of this phase, the distance between each `top` and `bottom` robot of the same pair is always smaller than the distance between the `top` and the `bottom` robot belonging to different pairs.

Let us show how to compute d'. At each loop iteration, all the `top` robots are moved out of $\widehat{\rho}$, along a specific line; namely, half of the robots on $\widehat{\rho}$ leave Cir at each iteration. So each robot r on $\widehat{\rho}$ must compute the "degree of splitting" d in this way: it obtains n (from the base angle), the number k of performance areas (from the number of regular tuples), and so the number m' (where $m' < m < \frac{n}{k}$) of the original robots in its performance arc which are not `pivots`, `angle` robots or guards. Then, r counts the current q robots (`pivot`, `angle`, and guard robots excluded) in its performance arc, and computes $d = \lceil \log_2 m' \rceil - \lceil \log_2 q \rceil$, which corresponds to the index of the last iteration of the loop. Now, r knows it is starting the d'-th iteration of the loop, where $d' = d + 1$.

[11] The external endpoint of $\widehat{\rho_{1/2}}$ is the endpoint farthest from the guards.
[12] $\langle\ \rangle$ can be any isomorphism between $\mathbb{N}^7$ and $\mathbb{N}$.

Phase Top robots to h. Let R be the length of the radius of Cir. Let h be the straight line parallel to the splitting chord ρ, external to Cir, and at distance R from g (i.e., the middle point on $\overset{\frown}{\rho}$ between the outer guards). The activated `top` robots on $\overset{\frown}{\rho}$ can compute h, and move to this line perpendicularly, in a fully parallelized scheme. Note that moving robots do not obstruct other `top` robots from computing the next action: in fact, all the needed information to act properly is located on their performance arc.

Phase Green Light On. The guards can check if all the `top` robots have moved on h. If this is the case, the outer (central, resp.) guards color themselves as `guard_green` (`guard_c_green`, resp.) to indicate to robots on h that they can proceed with the next phase.

Phase Top Robots to l. Let r be an activated `top` robot on h. It can recompute Cir (in fact, it can see at least two guards and at least a `bottom` robot), its original position on Cir, and so the distance δ from its related `bottom` robot. Then, r decodes δ and recomputes the values of n, m', t, c, z, the current loop index d', and its pair index i.

Let l be the straight line parallel to h, external to Cir, at distance R (resp. $2R$) from h (resp. Cir). Again, let Π' be the target regular n-gon to be built on Cir. Suppose m is even and let $v_1 \dots v_{\frac{m}{2}} v'_{\frac{m}{2}} \dots v'_1$ be the list of adjacent vertices of Π' on $\overset{\frown}{\rho}$ which are not already occupied by `pivots` and `angle` robots, such that they are equally distributed on the two distinct performance half-arcs. Note that the odd case is equivalent, with $v_1 \dots v_{\lfloor \frac{m}{2} \rfloor} w v'_{\lfloor \frac{m}{2} \rfloor} \dots v'_1$ as list of the vertices[13], where $w = v_{\lceil \frac{m}{2} \rceil} = v'_{\lceil \frac{m}{2} \rceil}$. Let us show how r can compute its vertex index. Thanks to the flag c, r can understand if its target vertex on Π' is on its original half-arc or on the other half-arc. This information is essential since two robots with the same index i can lie on the same half-arc: to avoid collisions, they have to know which half-arc the index refers to. Also, the flag z is needed to compute the right position of the target vertex of r. Let $v_1 v_2 \dots v_{\frac{m}{2}}$ be the ordered list of the missing vertices on the target half-arc and let $u_1 u_2 \dots u_{\frac{m}{2}}$ be their projections on l. So, r chooses the vertex v_k where

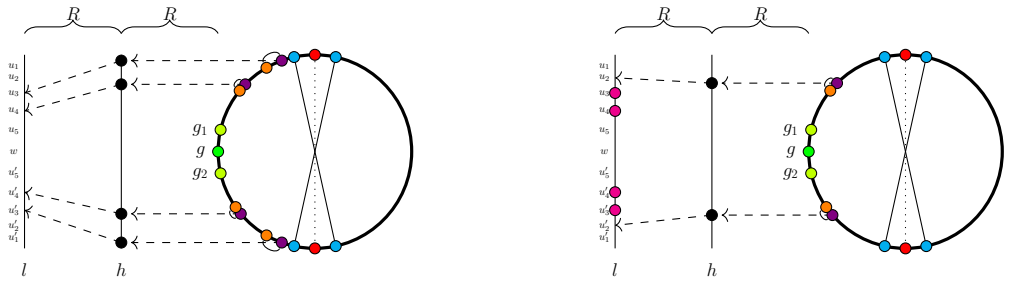
$$k = \left\lfloor \frac{m}{2} \right\rfloor - \sum_{j=2}^{d'} \left\lfloor \frac{m}{2^j} \right\rfloor - \left\lceil \frac{t}{2} \right\rceil + i - 1$$

and travels to the vertex projection u_k setting its light as `projection`. Note that this phase is fully parallelized: in fact, r can compute l and δ even though all other robots are moving toward l, since all the needed information is located "behind" r (i.e., on Cir).

By following this scheme, at each loop iteration, `top` robots dispose themselves on l symmetrically with respect to the central positions $u_{\frac{m}{2}}$ and $u'_{\frac{m}{2}}$, by covering the most internal free-robot projections and by leaving the two or three central positions which will be intended for the guards (see Figures 5.7a and 5.7b).

Phase Gap Fix. This phase is executed just when on $\overset{\frown}{\rho}$ there is an odd number of `bottom` robots. In this case, after the previous phase, there exists a projection, e.g., u_k on l where a robot lays, whereas its symmetric position u'_k is empty. From this

[13] As we will explain later, the vertex w is intended for the central guard.



(A) Loop iteration $d' = 1$.

(B) Loop iteration $d' = 2$.

FIGURE 5.7: Migration of `top` robots from $\widehat{\rho}$ to h, and from h to l (after the guards have set the green light). The central positions u_5, w, u'_5 will be covered by the guards.

configuration, a `bottom` robot can be easily elected[14] to cover the gap on l. Thus, it heads straight to u'_k and sets its color as `projection`. Note that, in no other cases, a symmetry gap can exist on l.

Phase GREEN LIGHT OFF. The guard robots can check if no `top` robots still lie on h, and if, possibly, the symmetry gap on l has been covered. If so, the outer (central, resp.) guard robots set their light as `guard_end` (`guard_c_end`, resp.).

Loop. At this point, the algorithm re-executes the previous sequence of phases $O(\log_2 m)$ times, until all the robots on $\widehat{\rho}$ (which are not `pivots`, `angle`, or guard robots) displace themselves on the line l. Note that, in the TBLG setting, the guards do not change their positions after the first iteration, and the fixed Δ still remains the minimum distance between two robots on the same half-arc, before the *pair approaching* phase.

5.5.6 Phase GUARDS DEPARTURE

At this point, the guards are the only robots on $\widehat{\rho}$ together with `pivots` and `angle` robots. Let R be the length of the radius of Cir. Of course, the guards can compute it. In this phase, each guard moves perpendicularly to l, and reaches a straight line f parallel to l and at distance R (resp. $3R$) from l (resp. from Cir). They maintain their original colors `guard` and `guard_c` once on f. Note that their trajectories do not cross any `projection` robots on l thanks to how Δ has been chosen.

5.5.7 Phase BACK TO Cir

When all the `projection` robots on l see no more robots on Cir (`pivots` and `angle` robots excluded), they can go back to Cir traveling perpendicularly with respect to the line l they are placed on. Once on Cir, they change their color in `sec` (see Figure 5.8).

Note that this phase is fully parallelized since it is always possible for a robot r on l to reconstruct the original Cir since:

- r can see at least 3 robots on it, or
- if moving robots hide Cir, r looks at the guards: it reconstructs f and l, and so the radius length R of Cir and the position of its center.

[14] In fact, we can elect the closest robot to the guards that belongs to the half-arc with more `bottom` robots.

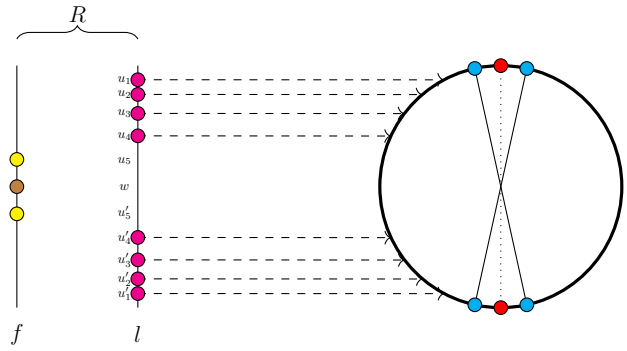


FIGURE 5.8: Robots migration from l to their target vertex on $\widehat{\rho}$.

5.5.8 Phase GUARDS TO Cir

Lastly, the guards reach the missing vertices in $\widehat{\rho}$, completing the section of the regular polygon. This phase is φ_ρ -synchronized since moving guards can prevent the other guards on f to compute the needed information to act properly.

5.5.9 Putting it all together

The algorithm presented in the previous sections allows us to state the contribution of this chapter:

Theorem 39. *Starting from $\mathcal{C}_{init}$ where n `OFF`-colored robots lie on distinct positions, a swarm can solve `Uniform Circle Formation` in $O(1)$ epochs, under $\mathcal{LUMI}^{\texttt{ASYNCH}}_{\mathcal{O}}$, with $O(1)$ colors. The solving algorithm requires rigid movements, avoids collisions, and assumes no orientation agreement among robots.*

5.6 Proofs

In this section, all the lemmas and theorems used in the chapter are stated and proved.

5.6.1 Collision-free trajectories

Lemma 32. *Let $w \in \{0,1\}^+$ be a Boolean string composed of as many 0s as 1s (i.e.,* balanced*). Let us factorize w into the minimal-length non-null factors such that each factor is balanced. Then, this factorization is unique, and each factor cannot be a palindrome.*

Proof. The factorization procedure starts searching for the minimal-length non-null balanced prefix of w. Indeed, such a prefix, say x, exists, and in the worst case, it corresponds to w itself. If $w = xy$ with $y \neq \varepsilon$, it is evident that y is balanced. Thus, we can proceed by searching for the next prefix of y, until only ε remains. Being this procedure finite and deterministic, the factorization is unique.

Assume a factor x resulting from this factorization is a palindrome (thus, $x = yy^R$, being $|x|$ even). Then, both y and y^R would be balanced, thus contradicting the hypotheses on x. □

Theorem 40. *Let us assume a circle configuration where all robots lie on the Cir. Let $\widehat{a}$ be an arc of Cir, and let $T = \{t_1, \ldots, t_h\}$ be h distinct robot-free (target) points of $\widehat{a}$, no one lying on the middle point of $\widehat{a}$. Assume at least h robots lie on distinct points of $\widehat{a} \setminus T$. Then, there exists an unambiguous way to elect h robots on $\widehat{a}$ and*

make them reach the h targets without colliding, and always guaranteeing complete visibility on the whole configuration.

Proof. Firstly, we need an unambiguous way to elect h robots that will reach the target points. For this purpose, we select the h robots that are closer to the target points. In the case of two robots equidistant from the same target point, we can use the geometry of the configuration (the distance from the closest endpoint of $\widehat{a}$) to elect one of them. Let $w = w_1 \dots w_{2h}$ be the Boolean string that represents the ordered displacement of the elected robots and the target points on $\widehat{a}$ such that: $w_i = 0$ symbolizes a target point, whereas $w_i = 1$ symbolizes a robot (e.g., $w = 11100010$ in Figure 5.9, following the clockwise direction). Let us factorize w in the shortest factors such that each factor has the same number of 0 and 1. Indeed, this factorization is unique, and each factor cannot be a palindrome (see Lemma 32). Let w' be a factor of w according to the above factorization. We indicate with 1_i (resp. 0_i) the i-th 1 (resp. 0) in such a factor, where $1 \leq i \leq |\frac{w'}{2}|$. So, robots in that factor move following the following a*sequential scheme*:

---

for $i = |\frac{w'}{2}| \dots 1$ **do**
  if no robot in $\widehat{a}$ is moving **then**
    robot symbolized by 1_i heads to the target point symbolized by 0_i;

This scheme assures that no moving robot creates collinearities with other robots in the swarm, thus guaranteeing complete visibility. This is easily provable by induction on $|\frac{w'}{2}|$. Moreover, if h is constant, this strategy accomplishes the task in $O(1)$ epochs. □

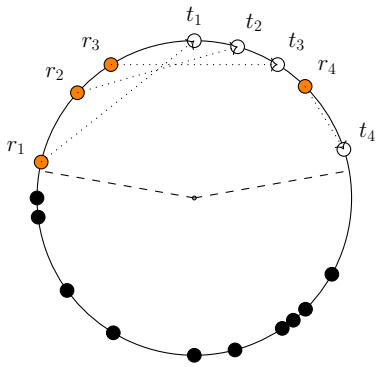


FIGURE 5.9: Trajectories of four robots assuring complete visibility to all the swarm. If a robot is moving, no other robot on the same arc moves.

5.6.2 Properties on circle configurations

Lemma 33 ([FMP23b])**.** *Let us consider a circle configuration where all n robots lie on the same circle Cir.*

- *For* odd *n, there exists a diameter passing through a robot and dividing Cir into two half-circles, each having $\frac{n-1}{2}$ robots.*
- *For* even *n, there exists a diameter dividing Cir into two halves such that (i) either the diameter passes through just one robot and the half-circles have $\frac{n}{2}$ and*

$\frac{n}{2} - 1$ robots, or (ii) the diameter passes through two antipodal robots and the half-circles have $\frac{n}{2} - 1$ robots each.

Proof. Let A, B be two distinct points on Cir, and let D_A, D_B be the respective antipodal points on Cir. We define the *overlap arcs* of A and B as the two equal-length arcs $\overset{\frown}{AD_B}$ and $\overset{\frown}{BD_A}$. Note that overlap arcs are null iff A, B are antipodal. Let us assume n is odd, and reason by induction on n. For $n = 1$, the property follows straightforwardly. Now, assume the property true for an odd $n > 1$, and let us prove the property for $n + 2$. Let us remove two robots, say r_1 and r_2, having minimal overlap arcs θ and θ', so that n robots are left on Cir. So, by the induction hypothesis, there exists a diameter d passing through a robot and dividing Cir into two halves, each having $\frac{n-1}{2}$ robots. Notice that the only way for d to leave r_1 and r_2 in the same half-Cir is to pass through the overlap arcs θ and θ'. Yet, since θ and θ' are minimal, no other robot can sit on these overlap arcs (except for r_1 and r_2). So, since d passes through a robot, it clearly cannot leave r_1 and r_2 on the same half-Cir. Thus, the result follows. For even n, the proof is similar. □

5.6.3 Properties on circular strings

In this section, for the sake of completeness, we provide some results on circular strings, even if they are probably known. These results are used to study and analyze geometric properties of a configuration $\mathcal{C}$ where the robots lie on the same circle Cir. Specifically, we aim to figure out some properties on Γ_1, Γ_2 (i.e., on the presence of one or two eligible classes in the rotation case) thanks to the string properties of the sectors circular string $\sigma(\beta), \sigma(\beta^R)$ of $\mathcal{C}$.

Lemma 34 (Lyndon and Schützenberger, [LS62])**.** *Let x and y be two non-empty strings. If $xy = yx$, then there exist a string z and two integers n and m, such that $x = z^n$ and $y = z^m$.*

Proof. Let us proceed by induction on $|y| - |x|$, assuming w.l.o.g. that $|x| \leq |y|$.
If $|x| = |y|$ then the fact holds: in fact $x = y$, thus $z = x = y$ and $m = n = 1$.
Otherwise, since $yx = xy$, y starts with a prefix that is equal to x. So $y = xu \Rightarrow xux = xxu \Rightarrow ux = xu$. By induction hypothesis, there exist z, m, n such that $u = z^m$, $x = z^n$, then $y = z^{n+m}$. □

Lemma 35. *Let x and y be two non-empty strings. If xx^Ryy^R is a palindrome, then there exists a string z and two integers n and m, such that $x = z^n$ and $y = z^m$.*

Proof. Let us proceed by induction on $|y| - |x|$, assuming w.l.o.g. that $|x| \leq |y|$.
If $|x| = |y|$ then the fact holds: in fact $x = y$, thus $z = x = y$ and $m = n = 1$.
Let us assume that $|x| < |y|$. Hence, since xx^Ryy^R is a palindrome, y^R ends with a suffix which is equal to x^R. So

$$y^R = u^R x^R \Rightarrow xx^R xuu^R x^R \text{ is a palindrome}$$

By simplifying, we have that $x^R xuu^R$ is a palindrome as well. By induction hypothesis, there exist z, m, n such that $u = z^m$, $x = z^n$, then $y = z^{n+m}$.

□

Lemma 36. *Let $\boldsymbol{\sigma}(w)$ be an aperiodic circular string, where $w = w_0 \dots w_{n-1}$. Then $\sigma_i(w) \neq \sigma_j(w)$ for each $i \neq j$.*

Proof. The proof follows directly from Lemma 34. In fact, let us assume that there exist two different indices such that $\sigma_i(w) = \sigma_j(w)$. Let us call them $w' = \sigma_i(w)$ and $w'' = \sigma_j(w)$, and w.l.o.g. let us assume $i < j$. Let z be the factor $w_i \dots w_{j-1}$, let x be the factor $w_j \dots w_{n-1}$, and let y be the factor $w_0 \dots w_{i-1}$, so that $w' = zxy$, and $w'' = xyz$. By Lemma 34, we can conclude that $\boldsymbol{\sigma}(w)$ contains a power string, contradicting the hypothesis. □

Lemma 37. *Given the two sector circular strings $\boldsymbol{\sigma}(\beta), \boldsymbol{\sigma}(\beta^R)$ of a configuration $\mathcal{C}$, there exist at most two classes of robots that read the same string γ. In particular, in case of two classes P_a and P_b, they read γ in opposite directions (clockwise or counterclockwise).*

Proof. Let us proceed by contradiction and let us suppose there is a third class P_c which reads the same string γ as P_a in the same direction. Let $a \in P_a$, $b \in P_b$, and $c \in P_c$ be three robots belonging to the same sector. Let $w' = xyz$ be the string read by a and let $w'' = yzx$ be the string read by c (x is the string between a and c). Since $w' = w''$, then we can conclude that $xyz = yzx$ (for the sake of simplicity, $xs = sx$). By Lemma 34, this means that w' and w'' are the power of the same factor v, which is in contrast with the initial hypothesis (a sector circular string must be aperiodic). □

Theorem 41. *Let $\boldsymbol{\sigma}(\beta), \boldsymbol{\sigma}(\beta^R)$ be the two sector circular strings of a configuration $\mathcal{C}$. If $|\beta| = q$ is odd, then Γ_1 contains at least one string. So, there exists always a unique eligible class of symmetry.*

Proof. The proof follows by the fact that *(i)* the same string in $\boldsymbol{\sigma}(\beta) \cup \boldsymbol{\sigma}(\beta^R)$ can be read by at most 2 classes (by Lemma 37) and *(ii)*, in the worst case, $q-1$ (even) classes read the same string two by two. So, at least one class reads a unique string. □

Lemma 38. *Let $\boldsymbol{\sigma}(\beta), \boldsymbol{\sigma}(\beta^R)$ be the two sector circular strings of a configuration $\mathcal{C}$, where $|\beta| = q$ is even. If $\boldsymbol{\sigma}(\beta)$ contains a mirrored string, then it contains exactly two different mirrored strings, whose rotation distance is $\frac{q}{2}$.*

Proof. By hypothesis, $\boldsymbol{\sigma}(\beta)$ is aperiodic (by definition of sector circular string) and contains a string $w' = xx^R$ (and its related x^Rx). Obviously, $x^Rx = \sigma_{\frac{q}{2}}(xx^R)$. Note that $xx^R \neq x^Rx$. In fact, if $xx^R = x^Rx$, for Lemma 34 it follows that xx^R is a power string, contradicting our hypothesis ($\boldsymbol{\sigma}(\beta)$ is aperiodic). Let us suppose that $\boldsymbol{\sigma}(\beta)$ contains another pair of palindromes, say $w'' = yy^R$ (and y^Ry), at a rotation distance $k < \frac{q}{2}$ from the first pair. Let us assume that $x = zs$ where $|z| = k$. Thus, we have that $w' = zss^Rz^R$ and $w'' = ss^Rz^Rz$. By Lemma 35, we can conclude that $w'' = v^n$ (for some not trivial v and n), again contradicting our hypothesis about $\boldsymbol{\sigma}(\beta)$. □

Theorem 42. *Let $\boldsymbol{\sigma}(\beta), \boldsymbol{\sigma}(\beta^R)$ be the two sector circular strings of a configuration $\mathcal{C}$, where $|\beta| = q$ is even. If $\boldsymbol{\sigma}(\beta)$ contains a mirrored string, then Γ_1 contains at least one string. So, there always exists a unique eligible class of symmetry.*

Proof. According to Lemma 38, there exist just 2 different palindromes in $\boldsymbol{\sigma}(\beta)$, say xx^R and x^Rx, which are read by two different classes of symmetry, say P_i and P_j. In particular, P_i reads xx^R in both directions, and P_j reads x^Rx in both directions. The fact that a third class P_k reads xx^R is excluded by Lemma 35. In fact, if the same palindrome string is read from two different classes, whose distance is less than $\frac{q}{2}$, then xx^R is in the form zz^Ryy^R, contradicting the hypothesis. Since $x \neq x^R$ (otherwise $\boldsymbol{\sigma}(\beta)$ would not be aperiodic), P_i (resp. P_j) is the unique class reading xx^R (resp. x^Rx). So, Γ_1 contains at least xx^R and x^Rx. □

Theorem 43. *Let $\boldsymbol{\sigma}(\beta), \boldsymbol{\sigma}(\beta^R)$ be the two sector circular strings of a configuration $\mathcal{C}$, where P_i and P_j are the two classes of placeholders. For each pair of adjacent placeholders $r_i \in P_i$ and $r_j \in P_j$ (no other placeholder lies on the arc $\widehat{p_i p_j}$), an axis of symmetry passes between them.*

Proof. By hypothesis, r_i and r_j read the same angle string in opposite directions. Let us suppose that the string distance between r_i and r_j is $k < |\beta|$. Thus, r_i reads $w' = x_1 \dots x_k y$ and r_j reads $w'' = x_k \dots x_1 y^R$. Since $w' = w''$, we can conclude that $x_k \dots x_1 = x_1 \dots x_k$ and $y^R = y$ (i.e., $x_1 \dots x_k$ and y are palindrome). Thus, between the arc delimited by r_i and r_j (in both directions), there is an axis of symmetry. □

Theorem 44. *Let $\boldsymbol{\sigma}(\beta), \boldsymbol{\sigma}(\beta^R)$ be the two sector circular strings of a configuration $\mathcal{C}$, where P_i and P_j are the two eligible classes of placeholders. Then the axis of symmetry passing through two adjacent placeholders r_i, r_j does not pass across any robot on the arc $\widehat{r_i r_j}$.*

Proof. Since r_i and r_j are placeholders, they read the same angle-string w. Let us assume by contradiction that the axis passes through a robot and let xx^R be the prefix of w which is common in r_i and r_j (which is a mirrored string, since it can be read in opposite directions and its length is even). So, r_i reads the angle-string $xx^R y$, whereas r_j reads $xx^R y^R$. Since $|\beta|$ and $|xx^R|$ are both even (otherwise we would have just one eligible class, by Theorem 41), then $|y|$ is even too. Since $xx^R y = xx^R y^R$, we have that y is a mirrored string too. Let vv^R be a factorization for y. Thus, $\boldsymbol{\sigma}(\beta)$ turns out to contain a rotation of w which is a mirrored string ($x^R vv^R x$). By Theorem 42, since $\boldsymbol{\sigma}(\beta)$ contains a mirrored string, then there is a unique eligible class of symmetry. The assumption that the axis passes through a robot must be wrong with these hypotheses. □

5.7 Conclusions

This chapter shows a $O(\log n)$-time solution for `Uniform Circle Formation` under the $\mathcal{LUMI}_{\mathcal{O}}^{\texttt{ASYNCH}}$ model, avoiding collisions and without any orientation agreement (chirality, etc.). In particular, we provide an original $O(\log n)$-time solution for the `Uniform Transformation` sub-problem, thus improving the existing $O(n)$-time solution in literature [FMP23b]. Note that the `ASYNCH` algorithm in [FMP23b] derives from an adaptation of a $O(1)$-time algorithm designed for synchronized robots. However, in the opaque setting, the transition from synchronism to asynchronism can require a fully-parallelizable algorithm to be executed within a one-by-one scheme, where each robot must wait until no other robot is moving before traveling.

In our new approach, ad-hoc strategies have been designed for making asynchronous-opaque robots achieve a partial, but remarkable, parallelization degree. Noteworthily, we cite the *pairing technique* which makes robots pair themselves and use their mutual distance to encode a list of data. Such a technique may be of independent interest in other contexts.

As we will soon demonstrate in the next chapter, it is indeed possible to achieve full parallelization under this model for solving `UCF`.

Chapter 6

Optimal Uniform Circle Formation

6.1 Introduction

The previous chapter has presented a $O(\log n)$-time (and $O(1)$-color) algorithm [FMP23a], which solves `Uniform Circle Formation` under $\mathcal{LUMI}_{\mathcal{O}}^{\texttt{ASYNCH}}$ (thus, by opaque robots). A subsequent work [PS24] considers the same model and develops a framework to solve `UCF` that provides a deterministic $O(x)$-time algorithm using $O(n^{1/2^x})$ colors under `ASYNCH`. Setting x to a (sufficiently large) constant k, we have a $O(1)$-time solution while reducing the colors to $O(n^\epsilon)$, with $\epsilon = \frac{1}{2^k} > 0$. Setting $x = \Theta(\log\log n)$, we have a $O(\log\log n)$-time solution which uses $O(1)$ colors.

In this chapter, we continue the investigation for solving `UCF` under $\mathcal{LUMI}_{\mathcal{O}}^{\texttt{ASYNCH}}$, and we provide an optimal solution (both in time and color). Compared to the result in [PS24], which is either optimal in time with $O(n^\epsilon)$ colors or optimal in colors with $O(\log\log n)$ time, the algorithm proposed here is simultaneously optimal on both time and number of colors.

Besides being optimal in time and colors, the proposed algorithm minimizes the *performance SEC*, i.e., the smallest circle containing all the points of the plane the robots touch during the execution of the algorithm. This objective is driven by a quite recent research direction that aims to solve classic distributed problems by optimizing some spatial metrics (e.g., the maximum distance traveled by a robot [BM18; Col+20]). Minimizing the performance SEC means forcing the swarm to act within the closed circular area delimited by $SEC(\mathcal{C}_{init})$, where $\mathcal{C}_{init}$ is the initial configuration of the swarm. This constraint may represent a realistic requirement in critical scenarios (e.g., lack of space or no guarantee about the safety of the space around robots).

Table 6.1 summarizes our result by comparing it with the existing `UCF` solutions.

Algorithm	Time	Colors	Performance SEC	Synchronization mode
[FMP18]	$O(1)$	$O(1)$	Not minimized	`FSYNCH`
[FMP23b; Fel+22]	$O(1)$	$O(1)$	Not minimized	`SSYNCH`
[FMP23b; Fel+22]	$O(n)$	$O(1)$	Not minimized	`ASYNCH`
[FMP23a]	$O(\log n)$	$O(1)$		
Generic [PS24]	$O(x)$	$O\left(n^{1/2^x}\right)$		
OptTime [PS24]	$O(1)$	$O(n^\epsilon)$		
OptColor [PS24]	$O(\log\log n)$	$O(1)$		
OptTime&Color [FPS24a; FPS24b]	$O(1)$	$O(1)$	**Minimized**	

TABLE 6.1: Existing `UCF` deterministic solutions for $n \geq 1$ luminous-opaque robots, avoiding collisions. $x \in [1, O(\log\log n)]$. $\epsilon = \frac{1}{2^k}$, with $k > 0$ is the chosen constant.

[0]The content of this chapter refers to the work [FPS24a; FPS24b].

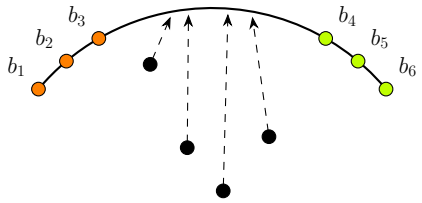


FIGURE 6.1: BDCP on a circular arc, where b_1, b_2, b_3 (b_4, b_5, b_6, resp.) are the *left* (*right*, resp.) beacons.

6.2 Preliminaries

We consider the same model as in Chapter 5, thus $\mathcal{LUMI}_{\mathcal{O}}^{\texttt{ASYNCH}}$, where robots are completely disoriented (no orientation agreement), with rigid movements, and collision-intolerant. Given a swarm of n robots, we consider the same problem `UCF` of forming a regular n-gon.

6.2.1 Challenges and techniques

The main challenge is to make robots exploit parallelism (thus achieving a $O(1)$ runtime) even in conditions of asynchrony and obstructed visibility, having only a $O(1)$-size palette of colors and avoiding collisions among robots. For this purpose, the key techniques adopted along our algorithm include the arrangement of robots along an (inner) circle in a symmetric pattern, the BEACON DIRECTED CURVE POSITIONING (BDCP) procedure [SVT17], and a novel *rank encoding* technique (existing encoding techniques in [BLT23; FMP23a] do not fit our requirements) which instructs each robot about the index of the target vertex it must reach in the final configuration.

Beacon Directed Curve Positioning. BDCP is a procedure developed in [SVT17] that uses $2k$ robots as *beacons* to define a k-point curve, and then guide other m robots (external to the curve all on one side) from their initial positions to their final positions on the curve, along non-colliding trajectories (see Figure 6.1). A k-point curve Ω is a curve on $\mathbb{R}^2$ where k is the minimal number of points necessary to define Ω. Beacons must lie on Ω so that the target positions of the m robots are located between two groups of k beacons (*left* and *right* beacons). In [SVT17], it was shown that relocating the m robots on Ω takes $O(\log k)$ epochs in `ASYNCH`, using $O(1)$ colors. We borrow BDCP to relocate robots on line segments ($k = 2$) and circular arcs ($k = 3$) using $2k$ beacons in $O(\log k) = O(1)$ epochs. Algorithm 9 summarizes the input assumptions and the resulting output of the BDCP procedure [SVT17].

6.2.2 Sub-problems to be optimized

We split the problem into the same sub-problems as in Table 5.2. The main objective in this chapter is to provide an optimal algorithm for the sub-problem `Uniform Transformation`, which transforms $\mathcal{C}_{circle}$ into $\mathcal{C}_{regular}$. In fact, as highlighted in Chapter 5, `Uniform Transformation` is the only sub-problem comprising `UCF` for which there does not exist a $O(1)$-time/color solution.

Moreover, as mentioned above, our second objective is to provide a `UCF` algorithm which makes robots perform their actions within $SEC(\mathcal{C}_{init})$. For this purpose, the algorithm in [SVT17] that transforms $\mathcal{C}_{init}$ into $\mathcal{C}_{convex}$ aligns with our aim. Even the subsequent transformation from $\mathcal{C}_{convex}$ to $\mathcal{C}_{circle}$, which makes robots move radially till $SEC(\mathcal{C}_{convex})$, serves our purpose.

Algorithm 9: BEACON DIRECTED CURVE POSITIONING [SVT17]

Input Assumptions:

1. Ω, a k-point curve, with $k \in \{2, 3\}$;
2. $2k$ *beacons* lie on Ω (k *left* beacons and k *right* beacons);
3. m *waiting* robots $r_1, \ldots, r_m$ s.t.
   (a) they are external to Ω belonging to the same half-plane ($k = 2$) or the convex side ($k = 3$);
   (b) they can see all the $2k$ original beacons;
   (c) they have to reach Ω
   - in some points between the original *left* beacons and the *right* beacons;
   - traveling along disjoint paths $\pi_1, \ldots, \pi_m$;
   - s.t. their paths intersect Ω only once;
   - s.t. if a waiting robot r_i sees $\geq k$ beacons, then it can compute its path π_i;

   (d) a waiting robot becomes a new beacon once it reaches Ω.

Result: All the waiting robots reach Ω without colliding in $O(\log k)$ epochs.

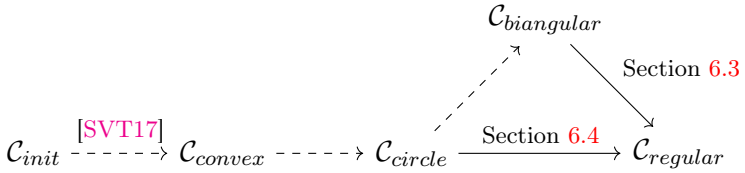


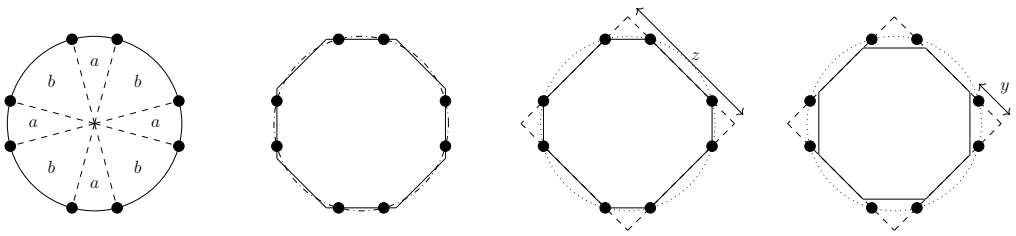
FIGURE 6.2: Phases of the UCF solution. Straight lines represent the phases for which a new strategy is provided in this chapter.

However, as seen in Section 5.4.3, the transformation from $\mathcal{C}_{biangular}$ to $\mathcal{C}_{regular}$ using the strategy in [DP07b] makes robots move out from $SEC(\mathcal{C}_{biangular})$. Thus, in Section 6.3, we first provide a variant algorithm to [DP07b] that copes with the biangular configuration.

Then, in Section 6.4, we provide our algorithm transforming $\mathcal{C}_{circle}$, when it is not biangular, into a regular configuration $\mathcal{C}_{regular}$.

Figure 6.2 depicts the two UCF phases for which this chapter provides an optimized solution.

6.3 Spatial optimization for the biangular case



(A) $\mathcal{C}_{biangular}$ (B) Regular n-gon P' [DP07b] (C) Exogenous Polygon EP (D) Regular n-gon P''

FIGURE 6.3: Arrangement of $\mathcal{C}_{biangular}$ in a regular n-gon.

We here propose a variant of the algorithm in [DP07b] for transforming $\mathcal{C}_{biangular}$ into $\mathcal{C}_{regular}$ that suits our objective (i.e., make robots remain within the closed area

delimited by $SEC(\mathcal{C}_{biangular})$). As in Section 5.4.3, we instruct robots to set their color to `biangular` to agree on the strategy to be performed. We know that in $\mathcal{C}_{biangular}$, n is even. Let $n = 2k$ and let P be the n-gon formed by the robots. The strategy developed in [DP07b] was to spot the target regular n-gon P' which encloses P, such that robots lie on alternative edges of P' (Figure 6.3b), and make robots slide on the edges of P' until they stop at its vertices. However, this strategy leads the robots to move outside $SEC(\mathcal{C}_{biangular})$ to reach the vertices of P'. Since our objective is also to minimize the performance SEC, we propose a target n-gon P'' that is inscribed inside P and has its vertices on the larger edge of P. Robots are made to move along the larger edges of P towards the vertices of P''. We now show that, as in [DP07b], our strategy also works under `ASYNCH`. Given P, we form a regular k-gon by extending the larger edges of P (see Figure 6.3c) and we call it the *exogenous polygon* (EP for short). As long as the robots move along the larger edges of P, the EP also remains the same. For an EP, there exists a unique regular $2k$-gon inscribed in it (see Figure 6.3d). This property guarantees that the target polygon P'' (i.e., the regular $2k$-gon inscribed in EP) is maintained even when some robots are in transit, and so it can be correctly computed even when robots no longer form P. In particular, the vertices of P'' are located at a distance y from the vertex of the EP, where z is the edge length of the EP. The value of y is given by

$$y = \frac{z}{2 + \sqrt{2 - 2\cos\left(\frac{(k-2)\pi}{k}\right)}}$$

After reaching a vertex of P'', the robots assume color `regular`, and terminate.

6.4 `Uniform Transformation` for the non-biangular case

Let $\mathcal{C}_{circle}$ be a circular configuration, which is neither non-biangular, nor regular (otherwise, no action is needed). Assume all the robots have the same color, w.l.o.g., `OFF`. Let $Cir = SEC(\mathcal{C}_{circle})$, and let O be its center: this will be the circle on which the robots will be arranged uniformly. Our `Uniform Transformation` algorithm ensures each robot moves within the close circular area delimited by Cir.

$\mathcal{C}_{circle}$ is transformed into $\mathcal{C}_{regular}$ through a sequence of four phases: Figure 6.4 summarizes the different output configurations and the phases involved in this transformation.

The first phase, Phase Split, partitions Cir into k *uniform sectors* by setting k regular tuples, as done at the beginning of the $O(\log n)$-time algorithm in Chapter 5. However, some variations are applied w.r.t. the previous algorithm.

After the setting of the uniform sectors, robots will move within their own sector to uniformly arrange on Cir. If the number of robots inside each uniform sector is too small (less than 12), Phase Seq Match is used to arrange them sequentially on the regular polygon[1]. Otherwise, the group of robots within each uniform sector executes three phases to eventually arrange themselves uniformly along the corresponding arc.

6.4.1 `ASYNCH` and obstructed visibility

To synchronize robots' movements under the `ASYNCH` mode, we adopt two strategies, one already used in Section 5.5.2 (i.e., the use of `moving_*` colors to signal moving robots).

[1] As similarly done in Chapter 5, for less than 4 robots within each uniform sector.

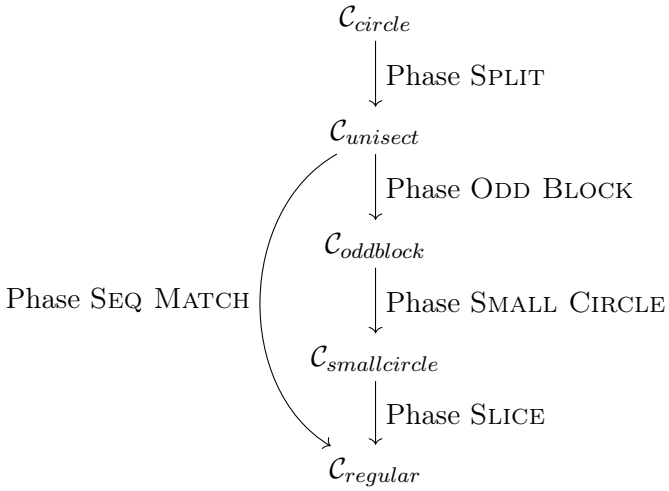


Figure 6.4: Phases and configurations in the `Uniform Transformation` algorithm in Section 6.4.

The second strategy becomes necessary in this algorithm due to spatial constraints: unlike what happens in Chapter 5, here robots must stay within $SEC(\mathcal{C}_{init})$. This spatial constraint indeed causes several visibility obstructions in the opaque setting. Instead, in the $O(\log n)$-algorithm, robots are properly instructed to temporarily move out of Cir, thus ensuring them the visibility of some fundamental elements needed for the computation (e.g., the regular tuples, the guards, etc.). Here, to tackle the visibility issues imposed by staying within $SEC(\mathcal{C}_{init})$, we need a proper way to encode information and to synchronize the movements in order to make robots execute their actions in constant time, even under `ASYNCH`.

We here describe the two synchronization strategies:

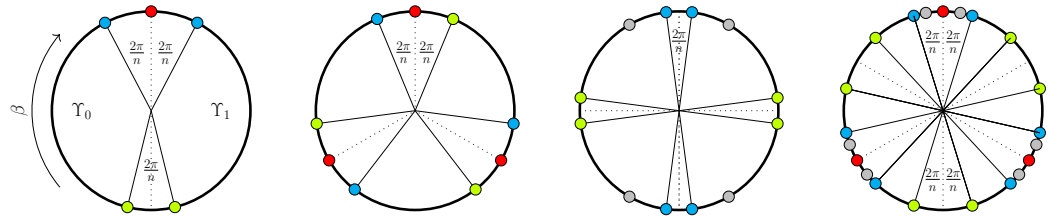
- **Election**: in some points of our algorithm, a group of robots is elected to perform a movement, in parallel. Being under `ASYNCH`, we need to ensure that all robots can correctly elect themselves even if part of the elected group is already moving. For this reason, our algorithm requires *all* the elected robots in the group to synchronize themselves by setting their colors as `pre_<color>` before starting moving, where `<color>` will be the color they have to set once stopped and reactivated.
- **Movement**: before performing a non-null movement, a robot colors itself as `moving_<color>` before starting moving, where `<color>` will be the color it has to set once stopped and reactivated. This strategy guarantees robots recognize moving robots in `ASYNCH` (those not hidden by other robots), thus avoiding ambiguous situations.

Note that we will often omit the details of these two strategies while explaining the algorithm, assuming their rationale is understood.

In the following sections, we describe the phases involved in our algorithm. The correctness proofs are all collected in Section 6.5.

6.4.2 Phase Split

Phase Split partitions $\mathcal{C}_{circle}$ into $k \geq 2$ uniform sectors $\Upsilon_0, \ldots, \Upsilon_{k-1}$ with a strategy similar to what happened in the $O(\log n)$-time algorithm. For a uniform sector Υ_i, we



(A) 2 symmetric uniform sectors. The sector string β defines the chirality of both Υ_0, Υ_1.

(B) 3 uniform sectors.

(C) 4 uniform sectors.

(D) 6 uniform sectors.

FIGURE 6.5: $\mathcal{C}_{unisect}$ where Cir is split into uniform sectors (here delimited by dotted lines) using the light color: `pivot` (here *red*), `left` (here *cyan*), `right` (here *lime*), and `placeholder` (here *lightgray*). All the other robots have been omitted.

define its arc as $\widehat{\Upsilon_i}$. Each uniform sector will be divided into arcs of equal length, and the corresponding endpoints will be called *uniform positions*: such positions become the target n-gon vertices on which robots must arrange.

We here describe the similarities and the differences w.r.t. Chapter 5:

- Our objective is to obtain uniform sectors such that *(i)* they have the same arc length and *(ii)* they contain the same number of robots, and *(iii)* they are *chiral* (i.e. the robots are arranged in an asymmetric pattern along the arc of each Υ_i). Note that in Chapter 5, it may happen that a uniform sector Υ is not chiral.

- As in Chapter 5, the uniform sectors are delimited by regular tuples (i.e., tuples of robots arranged on uniform positions on Cir). In particular, a regular tuple is set so that it is centered on the endpoints of a $\widehat{\Upsilon_i}$. However, here regular tuples are in different forms: they are formed by robots with colors $(x, \texttt{pivot}, y)$ or (x, y) where $x, y \in \{\texttt{left}, \texttt{right}\}$. Colors `left` and `right` are used to fix the chirality of each uniform sector, thus on each $\widehat{\Upsilon_i}$ there must be exactly one `left` robot and exactly one `right` robot.

- As in Section 5.5.3, Cir is split according to the properties of the circular angle-strings. After the splitting of Cir, regular tuples must be set. We however adopt the following changes w.r.t. Section 5.5.4:

  - a regular tuple in the form $(\texttt{angle}, \texttt{pivot}, \texttt{angle})$ now becomes $(x, \texttt{pivot}, y)$
  - a regular tuple in the form $(\texttt{angle}, \texttt{pivot}, \texttt{pivot}, \texttt{angle})$ now becomes (x, y)

  such that $x, y \in \{\texttt{left}, \texttt{right}\}$ are properly set to fix the chirality of the uniform sectors. The chirality of Υ is defined by comparing the lexicographical order of its sector strings β (clockwise direction) and β^R (counterclockwise direction).

- In the $O(\log n)$-time algorithm, it may happen that a uniform sector Υ is not chiral. This occurs when the angle-string β of that sector is a palindrome. In this case, a new regular tuple must be set to split Υ into two sectors: indeed, the two half-sectors are chiral (otherwise, by Lemma 35, β will not be a base string). Thus, if $\beta = xx^R$ ($\beta = xax^R$, resp.), then a regular tuple in the form $(x, \texttt{pivot}, x)$ ((x, x), resp.) must be set centered in the midpoint of $\widehat{\Upsilon}$, with $x \in \{\texttt{left}, \texttt{right}\}$.

Let $\mathcal{C}_{unisect}$ be the configuration obtained at the end of Phase Split (see Figure 6.5). In $\mathcal{C}_{unisect}$, all the n robots lie on Cir, and they are partitioned in $k \geq 2$ uniform sectors $\Upsilon_0, \ldots, \Upsilon_{k-1}$. If a robot lies on the boundary of a $\widehat{\Upsilon_i}$, then it is `pivot`-colored. Each Υ_i contains one `left` robot and one `right` robot: they set the chirality of Υ_i. All the robots that are not {`pivot`, `left`, `right`}-colored change their color to `split`. Note that all the {`pivot`, `left`, `right`}-colored robots will not move anymore: they already lie on uniform positions.

From now on, no robot will move out of its original uniform sector, and the subsequent phases will run in parallel inside each uniform sector. Refer to Section 5.6 for the correctness proofs of Phase Split, which establish the following:

Lemma 39 (Split). *Starting from a non-biangular $\mathcal{C}_{circle}$, $\mathcal{C}_{unisect}$ is reached in $O(1)$ epochs using $O(1)$ colors under `ASYNCH`, avoiding collisions, guaranteeing that robots always operate within $SEC(\mathcal{C}_{circle})$. Robots are evenly partitioned in uniform sectors, whose chirality is set by `right`- and `left`-colored robots.*

6.4.3 Phase Odd Block

We build an *odd block* inside each uniform sector Υ_i. An odd block for Υ_i is a circular sector contained in Υ_i, having the same origin and radius as Υ_i. Firstly, we depict the target configuration achieved at the end of this phase.

Odd block. An odd block of size $2l+1$, for an integer l, is a circular sector of Cir whose central angle measures $\frac{4l\pi}{n}$. When l is known, we will refer to the odd block as l-block. An l-block contains $2l+1$ uniform positions along its arc, which we denote as Γ. Let $U_1, U_2, \ldots, U_{2l+1}$ be the uniform positions of this block, where U_1 and U_{2l+1} lie on the endpoints of Γ. The block contains two robots on U_1 and U_{2l+1}. We will refer to these two robots as the *left* and the *right guard*. A robot, called the *median robot*, is located at position U_{l+1}. Let $\mathcal{L}$ be the block chord $\overline{U_1 U_{2l+1}}$. At the end of Phase Odd Block, the guards and the median robot are colored as `blockL` (left), `blockR` (right), and `median`, respectively. These robots do not move anymore. Instead, all the other $2l-2$ robots will have moved to distinct points of $\mathcal{L}$, with the `chord` color. Through the subsequent phases, the `chord` robots will uniformly arrange themselves on Γ, in order to cover the unoccupied uniform positions $U_2, \ldots, U_l, U_{l+2}, \ldots, U_{2l}$. The guards of the block fix the *block chirality*, which is local and is the same as the chirality of the related uniform sector.

Given any Υ_i in $\mathcal{C}_{unisect}$, Phase Odd Block works in three stages.

- **Stage 1.** Defining the guards of the odd block within Υ_i by adding a `padding` robot if necessary.
- **Stage 2.** Moving the robots located between the boundaries of Υ_i and U_1 (resp. U_{2l+1}) to the block chord $\mathcal{L}$.
- **Stage 3.** Moving the robots from Γ to the block chord $\mathcal{L}$.

Let Υ_i be a uniform sector in each $\mathcal{C}_{unisect}$, and let q be the number of robots in Υ_i (except for those on its boundaries). Let $U_1, \ldots, U_q$ be the uniform positions along the arc of Υ_i, such that the `left` robot lies on U_1 whereas the `right` robot lies on U_q. All the robots involved in the following stages belong to Υ_i. Robots in the various uniform sectors synchronize themselves through the `pre_<color>` colors before starting to move.

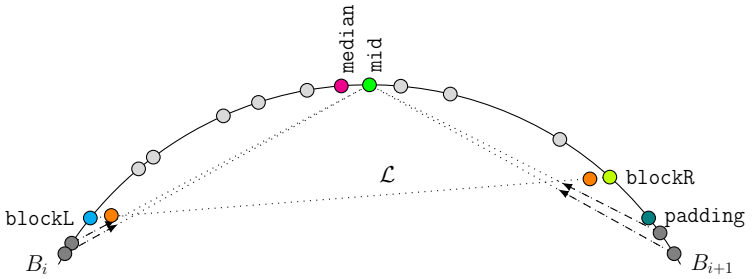


FIGURE 6.6: Odd block if q is even. Migration of `out_chord` robots to $\mathcal{L}$ using BDCP during Stage 2 of Phase ODD BLOCK.

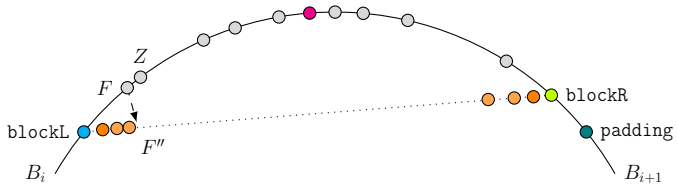


FIGURE 6.7: Stage 3 of Phase ODD BLOCK: `in_chord` robot in position F reaches $\mathcal{L}$ without colliding.

Stage 1. If q is even, the `split` robot closest to U_{q-1} sets its color as `pre_blockR` and, after the synchronization with all the other `pre_blockR` in the other uniform sectors, it reaches U_{q-1} where $\angle U_{q-1}OU_q = \frac{2\pi}{n}$, and it changes its color to `blockR`. The `right` robot turns `padding`. The `padding` robot will no longer move. The `left` robot turns `blockL`. If q is odd, then the robots with color `left` and `right` turn `blockL` and `blockR`, respectively. In both cases, `blockL` and `blockR` robots act as the left and right guards for each odd block. The block chord $\mathcal{L}$ is defined as the chord joining the guards, and the block arc Γ is the arc cut by $\mathcal{L}$. Then, the `split` robot closest to the midpoint of Γ elects itself as `pre_median` and then moves to the midpoint, changing its color to `median`.

Stage 2. Let us consider the l-odd block built within Υ_i. Let B_i and B_{i+1} be the boundaries of Υ_i, and let U_1 and U_{2l+1} be the boundaries of Γ (i.e. where the guards lie on). All the `split` robots which lie on Γ change their color to `in_chord`, otherwise they change into `out_chord` (i.e. if they lie on $\widehat{B_iU_1}$ and on $\widehat{U_{2l+1}B_{i+1}}$). This stage aims to make all the `out_chord` robots migrate to $\mathcal{L}$. If no robot with color `out_chord` exists in Υ_i, then Stage 3 begins directly. Let M be the middle point of $\widehat{\Upsilon_i}$ (i.e., the arc cut by Υ_i. If the `median` robot does not lie on M, then the `split` robot closest to M elects itself as `pre_mid` and then it moves to M changing its color to `mid`. Otherwise, the `median` robot sets its color to `mid`. Notice that, for $q \geq 12$, M always lies on Γ. Now, we use BDCP to make `out_chord` robots migrate to $\mathcal{L}$. Firstly, we properly select two `out_chord` robots so that they will become the beacons on $\mathcal{L}$ together with the guards on U_1 and U_{2l+1}. The two selected `out_chord` robots reach $\mathcal{L}$ in positions so that they play the role of the left and the right beacon, and they do not create collinearities with the `mid` robot. Once on $\mathcal{L}$, they turn `beacon` (see Figure 6.6). By BDCP, each `out_chord` robot r located at E heads to $\mathcal{L}$ to the point intersecting $\overline{EM}$ and $\mathcal{L}$. Once r reaches $\mathcal{L}$, it changes its color to `chord`. Note that the setting of

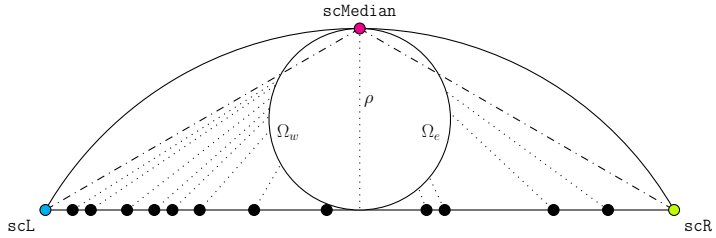


FIGURE 6.8: An odd-block at Stage 1 of Phase SMALL CIRCLE. All the `chord` robots (here *black*) lie on $\mathcal{L}$ and will reach Ω traveling along the dotted trajectories. The two dot-dashed lines delimit the *safe arcs* on Ω_e and Ω_w.

the `mid` robot at the midpoint of the arc Υ_i was necessary to make r select the `mid` robot belonging to its own uniform sector Υ_i (in fact, no ambiguity arises since the `mid` robot of Υ_i is always closer to r than the other `mid` robots belonging to $\Upsilon_{j\neq i}$). Stage 2 ends when no more `out_chord` robots are located in Υ_i.

Stage 3. The `in_chord` robots migrate from Γ to $\mathcal{L}$ by BDCP. From now on, the `mid` robot is no longer necessary, so it changes its color to `in_chord` if the `median` exists on the block arc. Otherwise, the `mid` robot resets its color to `median`. At the end of Stage 2, there are `beacon`- and `chord`-colored robots on $\mathcal{L}$. Such robots can act as beacons for fixing the chord $\mathcal{L}$ for the `in_chord` robots. Let r be a `in_chord` robot located at F on Γ. Let F' be the projection of F on $\mathcal{L}$, i.e., $\overline{FF'}$ is perpendicular to $\mathcal{L}$. Let Z be the robot location such that the projection Z' on $\mathcal{L}$ is the closest to F'. If a robot already occupies F', then the revised destination is computed as F'' where $dist(F', F'') = \frac{dist(F',Z')}{3}$ (see Figure 6.7). Once r reaches $\mathcal{L}$, it changes its color to `chord`. Note that, if there were no robots colored `out_chord` in Stage 2, then the closest `in_chord` colored robots to U_1 and U_{2l+1} move to their projection to act as beacons at the beginning of Stage 3. This ensures that all robots reach $\mathcal{L}$ in $O(1)$ time by implementing BDCP. See Section 6.5.1 for the correctness proofs of Phase ODD BLOCK, which establish the following:

Lemma 40 (Odd Block). *Starting from $\mathcal{C}_{unisect}$, $\mathcal{C}_{oddblock}$ is reached in $O(1)$ epochs using $O(1)$ colors under `ASYNCH`, avoiding collisions, guaranteeing robots always operate within $SEC(\mathcal{C}_{unisect})$. In each odd block, the guards are colored as `blockL` and `blockR`, the median robot is colored as `median`, whereas all the other robots are located on the block chord $\mathcal{L}$ with color `chord`.*

6.4.4 Phase SMALL CIRCLE

Consider an l-block in $\mathcal{C}_{oddblock}$. W.l.o.g., let the block chirality be clockwise. First, the left guard, the median robot, and the right guard change their color to `scL`, `scMedian`, and `scR`, respectively. Let ρ be the line segment connecting U_{l+1} to the projection of U_{l+1} on the block chord $\mathcal{L}$. We define the *small circle* Ω as the circle such that ρ is its diameter (i.e. U_{l+1} lies on Ω and $\mathcal{L}$ is its tangent). We say that ρ splits Ω into two halves Ω_w (left) and Ω_e (right). We will refer to ρ as the *median diameter*. See Figure 6.8.

Phase SMALL CIRCLE proceeds in three stages:

- **Stage 1 – Robots reach Ω:** Each `chord` robot moves along the line joining its position and the `scMedian` robot, until it reaches Ω.

- **Stage 2 – All robots move to Ω_e:** The robots on Ω_w migrate to Ω_e.
- **Stage 3 – Robots balance on Ω:** The robots on Ω_e split into two equal groups, and one group returns to Ω_w forming a reflective symmetric configuration on Ω.

Stage 1. Let r be a `chord` robot on $\mathcal{L}$. It always sees the `scMedian` robot of its block. Thanks to the presence of the other robots on $\mathcal{L}$ (`chord`-, `scR`- or `scL`-colored), r reconstructs the supporting line of $\mathcal{L}$, and so ρ and Ω. Then, r reaches Ω traveling along the trajectory connecting its own position to the `scMedian` robot (see Figure 6.8), and it turns `smallcircle`. Consider the line connecting the left (right, resp.) guard of the block with the median robot such that it splits Ω_w (Ω_e, resp.) into two arcs. Indeed, all the `smallcircle` robots on Ω lie on the lower arcs of Ω_w and Ω_e. We call such arcs the *safe arcs* of Ω (see Figure 6.8). When the `scR` (`scL`, resp.) robot does not see any `chord` or `moving_smallcircle` (i.e., the moving robots towards Ω before becoming `smallcircle`) robots on its half-block, then it sets its color as `scR_complete` (`scL_complete`, resp.). When a `smallcircle` robot on Ω_e (Ω_w, resp.) sees the `scR_complete` (`scL_complete`, resp.) robot on its half-block, it sets its colors as `smallcircle_complete`. These color updates are needed to synchronize all robots on Ω before proceeding with the next stage. See Lemma 50 in Section 6.5.2 for the correctness proof of Stage 1.

Stage 2. At this point, all the robots on $\mathcal{L}$ have reached the safe arcs of Ω. Specifically, we have two groups of `smallcircle_complete` robots on Ω_e and Ω_w, possibly with different cardinalities. We assume all the `smallcircle_complete` robots on Ω_e (Ω_w, resp.) set their color as `smallcircle_east` (`smallcircle_west`, resp.). If the robots on Ω_e and Ω_w lie in a reflective symmetry, they do nothing else in this stage. Otherwise, let r be a `smallcircle_west` robot on Ω_w. If a `smallcircle_east` robot lies on Ω_e, symmetrical with respect to r, then r moves to another position x on Ω_w such that *(i)* r does not collide with other robots or change its rank in the displacement of robots, *(ii)* the symmetric position of x on Ω_e is robot-free, and *(iii)* r does not leave the safe arc of Ω_w. When all the `smallcircle_west` robots have properly shifted their positions (if necessary), then they use BDCP to migrate on their projection on Ω_e. After the migration, we assume that all the robots on Ω_e (except for the median one) are colored as `smallcircle_east`. See Lemma 51 in Section 6.5.2 for the correctness proof of Stage 2.

Stage 3. The `smallcircle_east` robots update their color alternating two colors, `west` and `east`, so that the closest robot to the median robot is colored as `east`. After that, each `west` robot heads to the projection on Ω_w of its upper-adjacent `east` robot (see Figure 6.9). See Lemma 52 in Section 6.5.2 for the correctness proof of Stage 3. The proofs in Section 6.5.2 establish the following:

Lemma 41 (Small Circle). *Starting from $\mathcal{C}_{oddblock}$, $\mathcal{C}_{smallcircle}$ is reached in $O(1)$ epochs using $O(1)$ colors in the* `ASYNCH` *setting, avoiding collisions, guaranteeing robots operate within $SEC(\mathcal{C}_{oddblock})$. In $\mathcal{C}_{smallcircle}$, the robots in each odd block (guards and median robot excluded) are arranged on the same* small circle Ω *in a mirror-symmetric pattern where the line of symmetry lies on the median diameter ρ.*

6.4.5 Phase Slice

Consider an l-block in $\mathcal{C}_{smallcirle}$ (see Figure 6.10). First, the guards and the median robots turn `sliceL`, `sliceR` and `sliceMedian`, respectively. Let Q be the center of

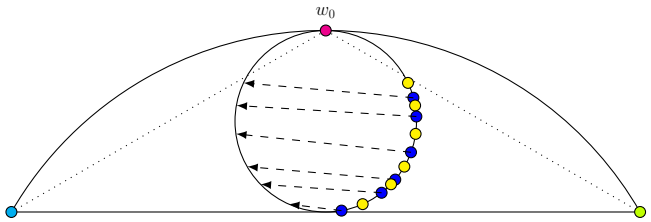


FIGURE 6.9: Stage 3 of Phase SLICE: `west` robots reach Ω_e.

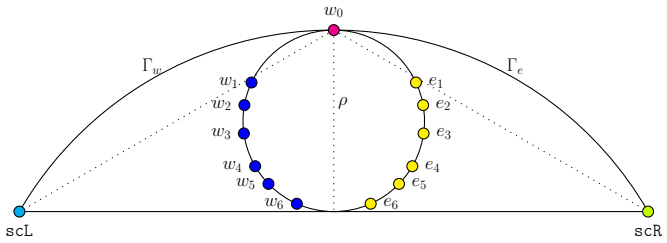


FIGURE 6.10: $\mathcal{C}_{smallcirle}$ at the beginning of Phase SLICE.

Ω. Let $e_1, \ldots, e_m$ (resp., $w_1, \ldots, w_m$) be the ordered sequences of `east` (resp., `west`) robots which lie on Ω_e (resp., Ω_w) starting from the closest to the median robot. Note that e_j and w_j lie on symmetric positions in the two half-circles, for each $1 \leq j \leq m$. We say that the median robot splits Γ into the right arc Γ_e and the left arc Γ_w. Phase SLICE aims at moving the robots from Ω to Γ to the vertices of the target regular n-gon. Let us present the three main stages that constitute the entire phase:

- **Stage 1 – Rank encoding:** Each w_j encodes its rank j by reaching a specific position on Ω_w.
- **Stage 2 – Moving towards the right target vertices:** Each e_j reaches its uniform position on Γ_e.
- **Stage 3 – Moving towards the left target vertices:** Each w_j reaches its uniform position on Γ_w.

Stage 1. Let us denote with w_0 and w_{m+1} the two endpoints of Ω_w (the median robot lying on w_0). Let $\alpha_j = \angle w_j Q w_{j+1}$, for each $0 \leq j \leq m$. At the beginning of this stage, robot w_1 computes $\delta = \min\{\alpha_j\}_{0 \leq j \leq m}$, moves to a position w_1' on Ω_w so that $\angle w_0 Q w_1' = \delta$ setting its color as `angle` (see Figure 6.11a). In this way, the `angle` robot fixes the amplitude of the slices both Ω_e and Ω_w will be split throughout this phase. After this setting, each `west` robot splits Ω_w into δ-slices, starting from the point w_0 (see Figure 6.11b). Let $\eta_0, \ldots, \eta_{\lfloor \frac{\pi}{\delta} \rfloor - 1}$ be the ordered sequence of such δ-slices, starting from the median robot. We ignore the possible remaining slice of amplitude $< \delta$ that can exist between $\eta_{\lfloor \frac{\pi}{\delta} \rfloor - 1}$ and w_{m+1}. Indeed, the arc of each δ-slice contains at most two robots: in the case of exactly two robots, both of them must lie on the endpoints of the slice arc. Let w_j be a `west` robot and let η_k be the slice it lies on (if w_j lies on the endpoint between two slices, then it chooses the slice with the smaller rank k). Then, w_j moves to a new position w_j' on Ω_w such that $\angle w_0 O w_j' = k\delta + \frac{j\delta}{m+1}$. This strategy always guarantees w_j' is strictly contained in η_k, thus ensuring these

(A) Setting of δ. (B) δ-slices of Ω_w.

FIGURE 6.11: Steps of Stage 1 in Phase SLICE.

movements never create collisions or collinearities among `west` robots. See Lemma 53 in Section 6.5.3 for the correctness proof of Stage 1.

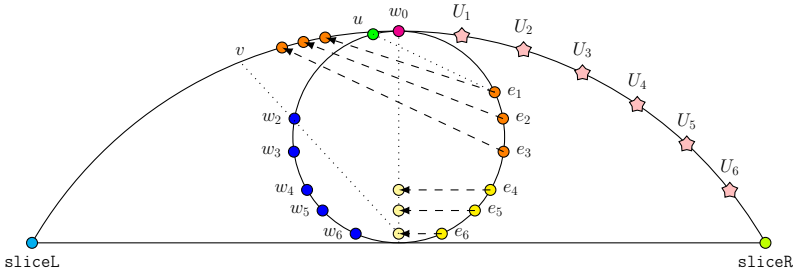
Stage 2. In this stage, each `east` robot reaches its uniform position on Γ_e in two steps. Let $e_1, \ldots, e_m$ be the sequence of the `east` robots and let $U_1, \ldots, U_m$ be the sequence of the uniform positions on Γ_e so that U_j is intended for e_j. In the first step, robots e_1, e_2 and e_3 turn their color as `beacon`. Afterward, all the other `east` robots $\{e_j\}_{4 \leq j \leq m}$ on Γ_e migrate towards ρ. Specifically, each `east` robot e_j directly heads to the projection e'_j of its position on Ω_e along ρ, setting its color as `east_diameter`. This migration is accomplished to make robots be aligned and so to avoid collisions and collinearities which might not allow a complete parallelism in the second step. Finally, the three beacons e_1, e_2 and e_3 migrate on Γ_w in order to fix Γ and so to help the `east_diameter` robots to uniformly arrange themselves on the arc Γ_e in the second step. The beacons must safely migrate to some positions that are visible to the robots on ρ. Let u be the intersection point between Γ_w and the line connecting e_1 (the first `beacon`) with w_1 (the `angle` robot). Let v be the intersection point between Γ_w and the line connecting e'_m with w_2. We call $\widehat{uv}$ the *safe arc* of Γ_w. So, e_1, e_2 and e_3 reach some deterministic and internal points of the safe arc $\widehat{uv}$ (see Figure 6.12a).

In the second step, all the `east_diameter` robots on ρ reach their uniform position. Let e be an `east_diameter` robot on ρ. To compute its uniform position, e needs to recover the arc Γ_e and its rank j. The rank j can be obtained by slicing Ω_w (resp., Ω_e) in $\eta_0, \ldots, \eta_{\lfloor \frac{\pi}{\delta} \rfloor - 1}$ (resp., $\eta'_0, \ldots, \eta'_{\lfloor \frac{\pi}{\delta} \rfloor - 1}$). Let η'_k be the slice where e originally laid on (before its migration on ρ). Since e can see all the `west` robots, it is sufficient for e to compute the rank j of the `west` robot located on η_k. Upon obtaining its rank j and the arc Γ_e (thanks to the presence of the beacons fixing Γ), e heads to U_j setting its color as `regular` (see Figure 6.12b). Finally, the beacons e_1, e_2, e_3 reach the missing uniform positions U_1, U_2, U_3.

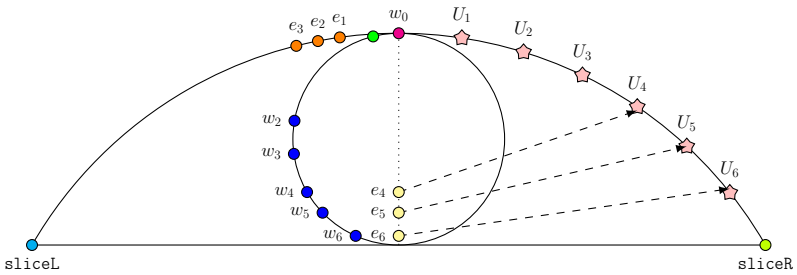
Stage 3. Each `west` robot reaches its uniform position on Γ_w in two steps. Let $w_2, \ldots, w_m$ be the sequence of the `west` robots and let $U'_2, \ldots, U'_m$ be the sequence of the uniform positions on Γ_w so that U'_j is intended for w_j. Recall that robot w_1 updated its color to `angle` in Stage 1: w_1 is destined for U'_1.

In the first step, the `angle` robot moves perpendicularly to ρ and reaches its symmetric position on Ω_e. Then, w_2 reaches a position on Ω_e to form an angle $\frac{\delta}{m}$ with `angle`, setting its color as `anglem`. After this setting, all the `west` robots migrate from Ω_w to their projection on ρ in constant time by implementing BDCP (w_0, w_3, w_{m-1}, w_m as beacons) and change their color in `west_diameter`.

In the second step, all the `west_diameter` robots along ρ compute and reach their uniform position. Let w be a `west_diameter` robot on ρ. It needs Γ_w and its rank to



(A) Step 1: `east_diameter` robots on ρ and beacons on the safe arc $\widehat{uv}$.



(B) Step 2: the `east_diameter` robots reach their uniform positions.

FIGURE 6.12: Stage 2, starting with $m = 6$ `east` robots.

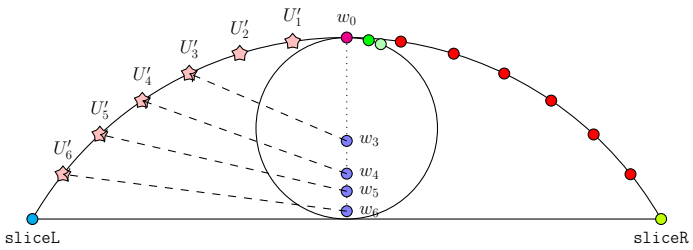


FIGURE 6.13: Stage 3, Step 2: `west_diameter` robots reach their uniform positions.

compute its target uniform position. To obtain its rank, w must exploit the slicing technique of Ω to decode its rank. So, w recomputes Ω_w and Ω_e through the presence of the `angle` and `anglem` robots, and the line where ρ lies. Then, w splits Ω_w in δ-slices and determines the slice η_k where it originally lay on. Then, w decodes its rank j inverting the formula $\angle w_0 O w' = k\delta + \frac{j\delta}{m+1}$, where w' is the projection of w on Ω_w. Recall that w_0 is the position of the median robot. Note that w can determine m through the angle formed by the `angle` and `anglem` robots. After having obtained its rank j and the arc Γ_w (through the presence of the `sliceR` robot and the `regular` robots on Γ_e), w heads towards U'_j and sets its color as `regular` (see Figure 6.13). Lastly, the `angle` and `anglem` robots on Ω_e reach U'_1 and U'_2 setting their color as `regular`. At the end of Phase SLICE, all the uniform positions of the block are occupied by a robot, and no other robot lies on Ω (except for the median one).

Since Phase SLICE is repeated for each block of Cir, the target n-gon is achieved. Eventually, all robots turn into `regular`.

Lemma 42 (Slice). *Starting from $\mathcal{C}_{smallcircle}$, $\mathcal{C}_{regular}$ is reached in $O(1)$ epochs using $O(1)$ colors under `ASYNCH`, avoiding collisions, guaranteeing that robots always operate*

within $SEC(\mathcal{C}_{smallcircle})$. Robots form a regular n-gon and terminate.

6.4.6 Phase Seq Match

Let $\mathcal{C}_{unisect}$ be the output configuration of Phase Split, where Cir is split into k uniform sectors $\{\Upsilon_i\}_{0\leq i\leq k-1}$. Phase Seq Match is executed starting from $\mathcal{C}_{unisect}$ when the number of robots q in each uniform sector Υ_i (except for its boundaries) is less than 12 (i.e., relatively small compared to the number of robots involved along the other phases of the algorithm). Remember that, at the end of Phase Split, each Υ_i contains a `left` and a `right` robot at U_1 and U_q, and possibly a `pivot` robot at its boundaries. We assume all the robots that are not `left`-, `right`- or `pivot`-colored set their color as `unmatched`. Now, the closest `unmatched` robot to the related U_2 of each uniform sector sets its color as `pre_matched`. After the setting of the `pre_matched` in each of the k uniform sectors, these robots reach the corresponding uniform position U_2 and set their color as `matched`. This routine is repeated until each uniform position U_j for $j = 2, \ldots, q-1$ is covered by a `matched` robot. Since the number of movements is upper-bounded by a constant, the whole phase is performed in constant time. Eventually, all the robots change their color to `regular`, solving `UCF`.

Lemma 43 (Seq Match). *Starting from $\mathcal{C}_{unisect}$, $\mathcal{C}_{regular}$ is reached in $O(1)$ epochs using $O(1)$ colors under `ASYNCH`, avoiding collisions, guaranteeing that robots always operate within $SEC(\mathcal{C}_{unisect})$. Robots form a regular n-gon and terminate.*

6.4.7 Putting it all together

We have shown all the phases involved in Figure 6.4 and composing our algorithm, whose pseudocode is listed in Algorithm 10. By combining Lemmas 39 to 43, we obtain the following theorem, which summarizes our contribution.

Theorem 45 (Uniform Circle Formation). *Starting from $\mathcal{C}_{init}$ where n `OFF`-colored robots lie on distinct positions, a swarm can solve `Uniform Circle Formation` in $O(1)$ epochs, under $\mathcal{LUMI}^{\texttt{ASYNCH}}_{\mathcal{O}}$, with $O(1)$ colors. The solving algorithm requires rigid movements, avoids collisions, assumes no orientation agreement among robots, and guarantees robots always perform within $SEC(\mathcal{C}_{init})$.*

6.5 Proofs

6.5.1 Proofs for Phase Odd Block

We show here all the correctness proofs for Phase Odd Block (Section 6.4.3).

Observation 7. Consider a circle configuration where the robots lie on Cir. If a robot r on position X has to reach a new position Y on Cir traveling along the chord $\overline{XY}$, then r can create a collinearity $\overline{arb}$ with just two robots a, b at a given time during its movement. Indeed, a and b must lie on the two distinct arcs cut by the chord $\overline{XY}$, each on a different arc.

Lemma 44. *Consider a configuration $\mathcal{C}_{unisect}$ with k uniform sectors. Let r be a `pre_x` robot (being `x` $\in$ {`mid, median, blockR`}) lying on the arc of a sector Υ_i. Within Υ_i, no robot is moving. Then, r can always determine the boundaries of all the uniform sectors and the cardinality of the swarm n even if at most one robot `moving_x` is moving within any other sector $\Upsilon_{j\neq i}$ to reach a new position on Cir.*

Algorithm 10: `UCF` pseudocode

```
Complete Visibility
    Input: C_init with n off robots in distinct positions on ℝ²
    Use [SVT17] to arrange robots into a convex pattern;
    Output: C_convex
Circle Formation
    Input: C_convex
    Cir ← SEC(C_convex);
    Make all robots move to Cir radially;
    Output: C_circle
Uniform Transformation
    Input: C_circle
    if C_circle is C_regular then
        Final Result: C_regular
    else if C_circle is C_bingular then
        Robots slide along the edges of the exogenous polygon [Section 6.3];
        Final Result: C_regular
    else
        C_circle is C_periodic;
        Phase Split [Section 6.4.2]
            Input: C_periodic
            Split Cir into k uniform sectors ϒ_0, ..., ϒ_{k-1};
            Set the chirality of each ϒ_i through the left and right robots;
            Output: C_unisect
        q ← number of robots in each ϒ_i;
        if q ≥ 12 then
            Phase Odd Block [Section 6.4.3]
                Input: C_unisect
                In each ϒ_i form an odd-block with the left and right guards;
                Use BDCP to make robots on ϒ_i migrate on the block chord ℒ;
                Output: C_oddblock
            Phase Small Circle [Section 6.4.4]
                Input: C_oddblock
                In each odd-block, spot the inscribed small circle Ω;
                Robots migrate from ℒ to Ω;
                Robots on Ω equally distribute on the two halves Ω_e and Ω_w;
                Output: C_smallcircle
            Phase Slice [Section 6.4.5]
                Input: C_smallcircle
                Robots on Ω_w encode their rank;
                Robots on Ω_e and Ω_w reach their target uniform positions;
                Final Result: C_regular
        else
            Phase Sequential Match [Section 6.4.6]
                Input: C_unisect
                Robots in each ϒ_i reach their target uniform position using a sequential
                  scheme;
                Final Result: C_regular
```

Proof. Since no robot moves in Υ_i, r can see at least a `left` and a `right` robot. If $\mathcal{C}_{unisect}$ is also defined by `pivot` robots, then r can see all of them (in fact, no robot moves out from its original uniform sector, and so hides `pivot` robots from r). If there are `pivot` robots, r can determine n by the central angle $\frac{2\pi}{n}$ between the `pivot` and the related `left` or `right` robot, and the positions of all the uniform sectors.

If r cannot see any `pivot` robots, then no robots lie on the boundaries of the uniform sectors (Figure 6.5d). In this case, r can spot the pair of `left-right` robots belonging to Υ_i. In fact, r selects the closest `left-right` pair which cuts a sector

where no robot is moving. Following the `left-right` orientation of Υ_i, let us consider the adjacent uniform sector Υ_{i+1}, which is located on the right w.r.t. Υ_i. Let g be the closest `left` or `right` robot that r sees following the right direction. Of course, g must belong to Υ_{i+1}. In fact, since within Υ_{i+1} at most one `moving_x` robot is moving, at least one {`left`, `right`}-colored robot cannot be hidden from r (by Observation 7). If g is `right`, then r can determine the boundary between Υ_i and Υ_{i+1} as the middle point on Cir between g and the position of the `right` robot belonging to Υ_i. Otherwise, it means that the `right` robot of Υ_{i+1} is hidden by a moving robot which is visible to r. Thus, r can compute the exact position of the hidden `right` robot and so determine the boundary between Υ_i and Υ_{i+1}. The same strategy is applied searching for the `left` robot in Υ_{i-1}. Since the uniform sectors are equal in amplitude and number of robots, r can easily reconstruct the boundaries of all the other sectors and so the number of robots in the swarm. □

Corollary 11. *In Phase Odd Block the setting of the `blockR`, `median`, and `mid` takes $O(1)$ epochs.*

Proof. By Lemma 44, we know that each `pre_x` robot r (`x` $\in$ {`mid, median, blockR`}) computes its target position (where it will assume the color `x` once stopped there and reactivated) even if other `moving_x` robots are moving to reach their target positions in the other uniform sectors. It follows that the time taken for setting the `x` robots is independent of the number of uniform sectors. □

Lemma 45. *Given an odd block built inside a uniform sector Υ_i, the midpoint M of the arc of Υ_i lies on the block arc Γ.*

Proof. Let U_1 and U_{2l+1} be the uniform positions where the guards of the block lie on (i.e. they are the endpoints of Γ), and let B_i and B_{i+1} be the boundaries of Υ_i. To prove the statement, we need to prove that $\angle U_1OU_{2l+1} \geq \frac{\angle B_iOB_{i+1}}{2}$ (where O is the center of Cir), i.e. the amplitude of the odd block is always not smaller than the amplitude of half Υ_i. By construction we know that $\angle B_iOU_1 \leq \frac{2\pi}{n}$ (for the left guard), while $\angle U_{2l+1}OB_{i+1} \leq \frac{2\pi}{n} + \frac{2\pi}{n}$ (for the right guard and for the padding robot). Thus, $\angle U_1OU_{2l+1} \geq \angle B_iOB_{i+1} - \frac{6\pi}{n}$. To prove that $\angle U_1OU_{2l+1} \geq \frac{\angle B_iOB_{i+1}}{2}$, it is sufficient to verify that

$$\angle B_iOB_{i+1} - \frac{6\pi}{n} \stackrel{?}{\geq} \frac{\angle B_iOB_{i+1}}{2}$$

$$\angle B_iOB_{i+1} \stackrel{?}{\geq} \frac{12\pi}{n} = 6 \cdot \frac{2\pi}{n}.$$

Since each Υ contains $q \geq 12$ robots (lying not on its boundaries), we know that $\angle B_iOB_{i+1} \geq 12 \cdot \frac{2\pi}{n}$, thus proving our statement. □

Observation 8. Two chords of the same circle can *(i)* have no points in common, *(ii)* intersect in only one point, or *(iii)* be coincident.

Lemma 46. *During Stage 2 of Phase Odd Block, the two `out_chord` robots which will become beacons on $\mathcal{L}$ are unambiguously selected.*

Proof. Let us show how to select the left beacon b (a complement strategy holds for the selection of the right beacon). If there exists an `out_chord` robot on $\overset{\frown}{B_iU_1}$, then the closest `out_chord` robot to U_1 is elected as b. In this case, b changes its color to `pre_beacon` and moves to the intersection of the line segments $\overline{DM}$ and $\mathcal{L}$, where D was the position of b. Otherwise, b is the farthest robot to U_1 selected from the other `out_chord` robots on the arc of Υ_i.

In this case, b reaches a position on $\mathcal{L}$ so that it can play the role of the left beacon and it does not create collinearities with the `mid` robot. Once on $\mathcal{L}$, b sets its color as `beacon`. An analogous procedure is repeated to choose the right beacon close to U_{2l+1}. □

Lemma 47. *During Stage 2 of Phase* Odd Block*, all the* `out_chord` *robots can reach* $\mathcal{L}$ *in* $O(1)$ *epochs using* BDCP*.*

Proof. We verify that all the input assumptions listed in Algorithm 9 hold to correctly implement BDCP in $O(1)$ time.

1. $\mathcal{L}$ is the target 2-curve;
2. 4 beacons are originally set to fix $\mathcal{L}$ (the guards and the two *ad hoc* beacons);
3. the `out_chord` robots are the waiting robots which have to reach $\mathcal{L}$, so that
   (a) they lie on the arcs $\overset{\frown}{B_iU_1}$ and $\overset{\frown}{U_{2l+1}B_{i+1}}$ (except for their endpoints), so they are external to $\mathcal{L}$, belonging to the same half-plane delimited by $\mathcal{L}$;
   (b) at the beginning of Stage 2, no robot is moving in Υ_i, and all robots of the uniform sector lie on the arc of Υ_i or on $\mathcal{L}$. So, any `out_chord` robot can see the 4 original beacons on $\mathcal{L}$.
   (c) they have to reach $\mathcal{L}$
   - by construction, the two internal beacons lie on $\mathcal{L}$ so that the target points of each `out_chord` robots are located between them;
   - each `out_chord` r robot must reach $\mathcal{L}$ traveling along the trajectory $\overline{rM}$ where M is the position of the `mid` robot. Of course, the trajectories do not intersect each other;
   - such trajectories intersect $\mathcal{L}$ in just one point (see Observation 8);
   - by construction, the beacons have been placed on $\mathcal{L}$ so that they do not create collinearities with the `mid` robot and the `out_chord` robots. This guarantees that any `out_chord` robot can see the `mid` robot at the beginning of Stage 2. Moreover, throughout the whole execution of BDCP, `out_chord` robots travel along distinct concurrent trajectories that are generated by the origin (the `mid` robot), they never create collinearities with the `mid` robot. As follows, any `out_chord` robot can reconstruct its path towards $\mathcal{L}$. For the same reason, the moving robots never collide since each `out_chord` trajectory does not intersect with any other trajectory.
   (d) each `out_chord` robot sets its color as `chord` once on $\mathcal{L}$, playing the role of a new beacon fixing the chord $\mathcal{L}$.

All these assumptions guarantee all `out_chord` robots reach $\mathcal{L}$ in $O(\log 2) = O(1)$ epochs without colliding. □

Lemma 48. *During Phase 3 of Phase* Odd Block*, all the* `in_chord` *robots can reach* $\mathcal{L}$ *in* $O(1)$ *epochs using* BDCP*.*

Proof. The proof is similar to the proof of Lemma 47, so we report just the differences. The presence of the original 4 beacons, plus the `chord` robots, fix the target curve $\mathcal{L}$. If no such beacons exist, then the robots closest to U_1 and U_{2l+1} move to the $\mathcal{L}$ to act as beacons. By construction, the beacons have been set in order to be

on the left and on the right of the target positions of the `in_chord` robots on $\mathcal{L}$. Let r be a `in_chord` which sees at least 2 robots on $\mathcal{L}$, and let F' be its projection on $\mathcal{L}$. Let Z be the position of the closest `in_chord` robot (still on Γ or in transit towards $\mathcal{L}$) and let Z' be its projection on $\mathcal{L}$. Then r always can compute its target position F'' on $\mathcal{L}$ and so its path towards $\mathcal{L}$ ($F'' = F'$ if no robot lies on F', otherwise $dist(F', F'') = \frac{dist(F',Z')}{3}$). According to this rule, two adjacent `in_chord` robots will always compute two paths without colliding. The other assumptions hold for the same reasons as in Lemma 47. □

6.5.2 Proofs for Phase Small Circle

We show here all the correctness proofs for each step in Phase Small Circle (Section 6.4.4).

Lemma 49. *When all the robots have migrated from $\mathcal{L}$ to Ω, then they can always see the right guard (if they lie on Ω_e) or the left guard (if they lie on Ω_w).*

Proof. Let l be the line passing through the median robot and the right guard. This line splits Ω_e into two arcs: the upper one (which we call *the unsafe* arc) and the lower one (which we call *the safe* arc). According to Stage 1 of Phase Small Circle, all the robots on Ω_e are located in the safe arc cut by line l (see Figure 6.10). Let l' be any other line passing through the right guard and a point of Ω_e (different from the median robot). If l' intersects the unsafe arc, then it intersects it into two points. Otherwise, l' intersects the safe arc into only one point. This means that no robot on Ω_e creates collinearity with another robot on Ω_e and the right guard. The same holds considering the left guard and Ω_w. □

Lemma 50. *During Stage 1 of Phase Small Circle, the following statements hold:*

1. *each `chord` robot on $\mathcal{L}$ can see the `scMedian` robot even when other robots are moving to Ω. Moreover, the movements at Stage 1 are collision-less.*
2. *If the right guard (left guard, resp.) sets its color as `scR_complete` (`scL_complete`, resp.), no `chord` or `moving_smallcircle` robot exists in the right (left, resp.) half-block.*

Proof. We prove each statement:

1. At the beginning of Stage 1, all the robots (except for the `scMedian` one) are located on the block chord $\mathcal{L}$ (and not inside the block), so no robot obstructs the visibility of the `chord` robot. Moreover, since the `chord` robots travel along distinct concurrent trajectories that are generated by the origin (the `scMedian` robot), they never create collinearities with the `scMedian` robot. For the same reason, the moving robots never collide.
2. Let us prove for Ω_e (the same holds for Ω_w). By Lemma 49, robots on Ω_e do not create collinearities with the right guard. Moreover, a `moving_smallcircle` robot cannot be hidden by other robots on Ω_e from the right guard. So, if a `moving_smallcircle` robot exists, the right guard can see it. At the same time, if at least a `chord` robot exists on $\mathcal{L}$, then the right guard can see.

□

Lemma 51. *During Stage 2 of Phase Small Circle, the following statements hold:*

1. *Each* `smallcircle_complete` *robot on* Ω *can correctly detect if it belongs on* Ω_e *or* Ω_w *and so set its color as* `smallcircle_east` *or* `smallcircle_west`*, respectively.*

2. *The position shift of the* `smallcircle_west` *robots can be executed without colliding and in* $O(1)$ *time.*

3. *The* `smallcircle_west` *robots migrate from* Ω_w *to* Ω_e *in* $O(1)$ *using* BDCP*.*

Proof. We prove each statement:

1. By Lemma 49, all the `smallcircle_complete` robots on Ω_e (resp. Ω_w) see the related guard. Thus, they can detect if they belong to the right/left side of Ω, and update their color properly.

2. The new positions of `smallcircle_west` robots can be easily computed so that each robot shifts on Ω_w of a little distance downward so that it neither collides with nor goes beyond the adjacent robot. Such movements always guarantee complete visibility among the robots within Ω. Note that a `smallcircle_west` robot can always recompute correctly Ω even when the other robots are shifting; in fact, there are always at least three robots still on Ω: the robot itself, the median robot, and at least one `smallcircle_east` robot on Ω_e (otherwise, no shift is needed).

3. We verify that all the input assumptions listed in Algorithm 9 hold to correctly implement BDCP in $O(1)$ time.

   (a) Ω_e is the target 3-curve;

   (b) 6 robots (selected among the `smallcircle_east` or `smallcircle_west` robots) can be set on Ω_e so that they play the role of the left and right beacons (using a proper color). This setting can be achieved in constant time;

   (c) the `smallcircle_west` robots are the waiting robots which have to reach Ω_e. We can guarantee that:

      i. being on Ω_w, they are external to Ω_e, and they lie on the convex region delimited by Ω_e;

      ii. at the beginning of this step, no robot is moving within Ω, so the waiting robot on Ω_w can see all the robots on Ω_e;

      iii. they have to reach Ω_e

         - to their projections on Ω_e. Being the beacons originally the "leftmost" and the "rightmost" robots on Ω_e, the robots on Ω_w will migrate between the two groups of beacons;
         - being parallel, the robots' trajectories do not intersect each other;
         - such trajectories are chords connecting Ω_w to Ω_e, thus they intersect Ω_e in just one point;
         - if a `smallcircle_west` robot on Ω_w can reconstruct Ω, then it can reconstruct its trajectory (looking at the left guard, a `smallcircle_west` robot can detect the position of ρ and so its trajectory towards Ω_e);

      iv. each `smallcircle_west` robot becomes a new beacon (setting its color as `smallcircle_east`) once on Ω_e.

   All these assumptions guarantee all `smallcircle_west` robots reach Ω_e in $O(\log 3) = O(1)$ epochs without colliding.

□

Lemma 52. *During Stage 3 of Phase* Small Circle*, the following statements hold:*

1. *Each* `west` *robot on* Ω_e *correctly detects its upper-adjacent* `east` *robot, even if other robots are moving.*
2. *Each* `west` *robot on* Ω_e *correctly and safely reaches its target position on* Ω_w*.*

Proof. We prove each statement:

1. Let w be a `west` robot on Ω_e. Thanks to the presence of the right guard (always visible to w), w understands it has to migrate on the other side. Moreover, w can always see its two adjacent `east` robots even when other robots are migrating to Ω_w (the first robot on Ω_e has an `east` robot and the median robot as adjacent). So w recomputes Ω. Now, let us suppose the median robot is hidden from w. Thus, w must detect which of its two `east` adjacent robots is the upper-adjacent one. From the right guard, two chords can be elected as $\mathcal{L}$. However, by the presence of other robots of other blocks, w can always detect which is the correct block chord, and so it can establish the upper-down orientation of the block.
2. Again, let w be a `west` robot on Ω_e. As proved above, w computes Ω by the presence of at least three robots on it, and ρ by the presence of the right guard. Once obtained Ω_w and once selected its upper-adjacent robot u, w heads to the projection of u on Ω_w. All the trajectories are disjoint and so no collision can occur.

□

6.5.3 Proofs for Phase Slice

We show here all the correctness proofs for each step in Phase Slice (Section 6.4.5).

Lemma 53. *During Stage 1 of Phase* Slice*, the following statements hold:*

1. *At the beginning of Stage 1, each slice arc on* Ω_w *contains at most two robots: in the case of exactly two robots, both of them must lie on the endpoints of the slice arc;*
2. *Each slice is chosen by at most one* `west` *robot where it will encode its rank;*
3. *During the movements in Stage 1, each* `west` *robot* w_j *will move to a new point* w'_j *which is strictly contained in its original slice;*
4. *At the end of Stage 1, each slice arc of* Ω_w *contains at most one robot, except for the slice* η_0 *which contains two robots (the* `sliceMedian` *robot and the* `angle` *one) on its endpoints.*
5. *Movements at Stage 1 never create collisions or collinearities among robots on* Ω*.*

Proof. We prove each statement:

1. By construction, each slice has an amplitude of δ, where δ is the minimum angle between any two robots on the circle Ω_w. Let η be a δ-slice of Ω_w. If three robots lie on the arc of η, then they define a smaller angle than δ (contradiction). The same contradiction results in the case of two robots where one of them does not lie on the endpoint of the η arc. Note that the remaining slice of amplitude $< \delta$ which can exist between the last δ-slice and the lower endpoint of Ω_w does not contain any robot (otherwise δ would not be the minimum angle).

2. If a slice does not contain any robots, then it will be chosen by no robots. If a slice contains just one `west` robot, then it will choose its slice to encode its rank. If a slice η_k contain two `west` robots, say w_j, w_{j+1}, one per each arc endpoint, then w_j chooses η_{k-1} whereas w_{j+1} chooses η_k. Since η_{k-1} contains either just w_j or both w_{j-1} and w_j (by Item 1), η_{k-1} will be chosen just by w_j (in fact, if w_{j-1} is not `angle`, then it will choose η_{k-2}).

3. Let η_k be the chosen slice for the rank encoding of w_j (i.e. the slice where w_j lies on, possibly on one of its endpoints). Since w_j reaches a new point w'_j such that $\angle w_0 O w'_j = k\delta + \frac{j\delta}{m+1}$, it is sufficient to prove that w'_j belongs to η_k (endpoints excluded). Such a property is easily proved since $\frac{j\delta}{m+1} < \delta$ (in fact, $j < m+1$) and since $j \neq 0$ (`west` robots have rank $j = 2, \ldots, m$).

4. By Item 1, each `west` robot travels in a new position which is strictly contained in its chosen slice. By Item 2, each slice is chosen by at most one robot. These two results ensure each slice arc of Ω_w contains at most one robot. To conclude, η_0 is the only slice in Ω_w containing two robots at the end of Stage 1.

5. Movements of `west` robots are performed in separated slices, thus never creating collisions with the other robots. Let w_j be a `west` robot which has to travel in the slice η_k to reach the point w'_j. This robot creates collinearity with another robot on Ω only if both the arcs cut by the trajectory $\overline{w_j w'_j}$ contain at least a robot. However, since in the minor arc $\overset{\frown}{w_j w'_j}$ (i.e. the arc strictly contained in η_k) no robots are contained except for w_j itself, no collinearities can occur.

□

Lemma 54. *During Stage 2, the following statements hold:*

1. **[Step 1]** *The migration of the* `east` *robots* $\{e_j\}_{4 \leq j \leq m}$ *on* ρ *is always possible in one epoch without collisions.*

2. **[Step 1]** *The beacons* e_1, e_2 *and* e_3 *correctly and safely migrate on the safe arc* $\overset{\frown}{uv}$.

3. **[Step 2]** *An* `east_diameter` *robot on* ρ *can recompute the slices on* Ω_e *and* Ω_w.

4. **[Step 2]** *The rank* j *computed by an* `east_diameter` *robot on* ρ *coincides with its original rank on* Ω_e, *for* $2 \leq j \leq m$.

5. **[Step 2]** *Every* `east_diameter` *robot on* ρ *can see the three beacons.*

6. **[Step 2]** *An* `east_diameter` *robot* e *on* ρ *can instantly recompute its uniform position* U_j *on* Γ_e.

7. **[Step 2]** *Beacons* e_1, e_2 *and* e_3 *on* Γ_w *safely reach their target positions on* Γ_e *in constant time.*

Proof. We prove each statement:

1. Let e be an `east` robot on Ω_e at the beginning of Stage 2 (after the setting of the beacons on Ω_e). At most $m-4$ `moving_east_diameter` robots are moving to ρ, thus possibly obstructing the visibility of e. However, since each moving robot can hide only one other robot on Ω from e, and since there are at least $m+5$ robots on Ω (included e), then e can always reconstruct Ω. By Lemma 49,

e can see `sliceR` and so it can recompute the supporting line of *chord*, and so ρ. Thus, e directly travels perpendicularly towards ρ, without waiting. Since all the trajectories are perpendicular to the same line, the migration of the `east` robots is accomplished without colliding.

2. Let b be a `beacon` robot on Ω_e which sees all the `east_diameter` robots on ρ. Thanks to the presence of the `west` robots, e can recompute Ω, and so it can establish whether a `beacon` is still moving or if the configuration is static. For the sake of simplicity, we make our beacons reach Γ_w following a sequential scheme: thus, firstly e_3 reaches a deterministic point on $\widehat{uv}$, while e_1 is the last beacon to move. As soon as b sees a static configuration (within its block) and understands it is its turn, it can easily compute the safe arc $\widehat{uv}$ and move to a deterministic position on $\widehat{uv}$, thus avoiding collisions with the other robots. Since the number of beacons for a block is constant, their migration takes $O(1)$ time.

3. Since at Stage 2 no robot moves within the west half-circle given by Ω_w, an `east_diameter` robot e on ρ can always see all the `west` robots and the `angle` one, thus allowing it to determine Ω and the angle δ. Moreover, e can easily recompute ρ by the presence of the adjacent `east_diameter` or `sliceMedian` robot. So, e can split Ω_w and Ω_e in δ-slices and determine the slice η'_k and η_k which contain its projection on Ω.

4. Thanks to Lemma 53, each `west` robot w_j (except for w_1 which became the `angle` robot) has moved in its own original slice during Stage 1, thus maintaining the one-to-one rank association with the `east` side.

5. See Figure 6.12a. The beacons lie on the safe arc $\widehat{uv}$, which was defined by the two lines: the first one which connects the initial position of e_1 and w_1 (the `angle` robot), and the second one which connects the position of e_m and w_2. Let ρ' be the segment of ρ cut by such lines. By construction, all the `east_diameter` lie on ρ' (e_m lies on its lower endpoint). Let x be an internal point of the safe arc $\widehat{uv}$ and let y be a point of ρ'. Then, the line connecting x with y crosses the arc $\widehat{w_1w_2}$ (endpoints excluded). Since no robot lies within the arc $\widehat{w_1w_2}$, any robot on $\widehat{uv}$ can see any robot on ρ'.

6. The robot e can always recompute its rank j (according to the previous Items), the circle Ω, and the diameter ρ. Thanks to Item 5, e can see the three beacons on $\widehat{uv}$ and so it can reconstruct Γ_e. Since e has complete visibility of Ω_w, it counts the m `west` robots and so it can define the positions of all the uniform positions on Γ_e. Thus, it computes its target position U_j.

7. See Figure 6.12b. Let $b \in \{e_1, e_2, e_3\}$ be a `beacon` robot on Γ_w that sees no `east_diameter` robots or moving robots within the block. Note that, in this situation, b has complete visibility of Γ_e except for at most one robot which can be hidden by the `angle` robot. So, following a sequential scheme ($b = e_3, e_2, e_1$), b recomputes Γ_e, and the angle of each uniform arc $\widehat{U_jU_{j+1}}$ and so its target position. Thus, b reaches its target uniform position and sets its color as `regular`. In particular, if `angle` lies on the trajectory between b and its target position U_j, then b reaches U_j in two moves: firstly, it stops in a position on ρ which is visible to the other `beacon` on Γ_w, secondly, it reaches U_j. Note that at most one beacon has to split its trajectories into two steps (in fact, `angle` can be collinear with just one e_j and the related U_j). So, this task is accomplished in constant time.

□

Lemma 55. *During Stage 3, the following statements hold:*

1. **[Step 1]** *Robots w_1 and w_2 can safely and correctly travel to Ω_e fixing the angle δ and the cardinality m.*
2. **[Step 1]** *If a* `west` *robot on Ω_w sees a* `west_diameter` *or a* `sliceMedian` *robot, then it is always able to determine whether such robots belong to its block or not.*
3. **[Step 1]** *The* `west` *robots w_3, w_{m-1}, w_m migrate to ρ in constant time in order to play the role of the beacons (together with w_0, the median robot) for the other* `west` *robots;*
4. **[Step 2]** *A* `west_diameter` *robot w on ρ can correctly and instantly compute Ω_e and Ω_w and its original slice η_k. So, it correctly decodes its rank j.*
5. **[Step 2]** *A* `west_diameter` *robot w on ρ can correctly spot Γ_w and its target uniform position U'_j.*
6. **[Step 2]** *The* `angle` *and* `anglem` *robots reach U'_1 and U'_2 without colliding.*

Proof. We prove each statement:

1. The `angle` robot (w_1) can detect Stage 2 has ended since it can see all the uniform positions on Γ_e are covered by `regular` robots. So, it can safely migrate on its projection on Ω_e, maintaining the angle δ. Once w_1 has switched half-circle, w_2 enjoys complete visibility of all the $m+1$ robots on Ω and so it can correctly migrate on Ω_w to fix the angle $\frac{\delta}{m}$.
2. Let us suppose that a `west` robot w on Ω_w sees some robots (`west_diameter` or `sliceMedian`) of another median diameter ρ', and it cannot see the robots on its ρ (due to the movements of the other robots). This situation never creates ambiguous snapshots since w can always understand ρ' does not belong to its block. In fact, the moving robots in the block of w can never hide all the `regular` robots on Γ_e: the visibility of w can be obstructed by at most $m-6$ robots which are moving inside Ω (where the 6 still robots are: w itself, the `angle` and `anglem` robots, and the three beacons w_3, w_{m-1}, w_m). So w can see at least 6 `regular` robots on its block. Such `regular` robots are sufficient to prevent w from selecting the wrong median diameter.
3. Before this step, w_0 (the median robot) is the only robot on ρ. Robots w_3 (as the "left" beacon) and w_{m-1}, w_m (as "right" beacons) can easily and safely elect themselves and reach their projections on ρ, by following a sequential scheme (i.e., in constant time). Traveling along parallel trajectories, they do not collide.
4. Let w be a `west_diameter` robot on ρ. It can always recompute the supporting line of ρ (w can always see another `west_diameter` robot or the `sliceMedian` robot). Moreover, it can always see the robots `angle` and `anglem`. Thus, w can recompute Ω (the center of Ω is the intersection point between the bisector of `angle-anglem` and the supporting line of ρ) and its projection w' on Ω_w. Then, w splits Ω_e and Ω_w in δ-slices (δ being fixed by the `angle` robot) and computes the slice η_k where w' lies on (i.e. where w originally lay before the migration on ρ).

   After having recomputed η_k, w obtains its original rank j by inverting the formula $\angle w_0 O w' = k\delta + \frac{j\delta}{m+1}$, where m is fixed by `angle` and `anglem`.

5. See Figure 6.13. Thanks to Item 4, we know that w can compute Ω_e, Ω_w, and its rank j, even when other robots are moving to reach their uniform positions on Γ_w. Moreover, w can see at least three robots on Γ_e (one of them is `sliceR`) and so it recomputes Γ_w. The presence of the `angle` and `anglem` robots allows w to recompute m and so the positions of each uniform position. Thus, it can easily spot the position of its target U'_j.

6. For the sake of simplicity, we make `angle` and `anglem` move sequentially. Let a be the `angle` or `anglem` robot. When a sees no `west_diameter` robots, a easily recomputes the (possible) missing uniform positions U'_1 or U'_2 (by the presence of the `sliceMedian` robot and the robots on $\Gamma_{\{e,w\}}$). Then, it heads to its U'_j without colliding with any other robots (due to the absence of robots on ρ and Ω_w, except for the `sliceMedian` robot).

□

Lemma 56. *During Stage 3 of Phase* Slice, *all the* `west` *robots on* Ω_w *can reach* ρ *in* $O(1)$ *epochs by implementing* BDCP.

Proof. We verify that all the input assumptions listed in Algorithm 9 hold to correctly implement BDCP in $O(1)$ time.

1. ρ is the target 2-curve;

2. 4 beacons (w_3, w_{m-1}, w_m together with the median robot w_0) are originally set to fix ρ (see Lemma 55, Item 3);

3. the `west` robots are the waiting robots which have to reach ρ, so that

   (a) since they start from Ω_w (excluded its endpoints), they are external to ρ, belonging to the same half-plane delimited by ρ;

   (b) at the beginning of this step, no robot is moving within Ω, so each `west` robot can see all the robots on ρ;

   (c) they have to reach ρ

   - to their projections on ρ. Being the beacons originally the "leftmost" and the "rightmost" `west` robots on Ω_w, the `west` robots will migrate between the two groups of beacons;
   - being parallel, the `west` robots' trajectories do not intersect each other;
   - such trajectories are perpendicular to ρ, so they intersect each other in just one point;
   - if a `west` robot on Ω_w can see its target line ρ, then it can reconstruct its trajectory (the projection on ρ);

   (d) each `west` robot sets its color as `west_diameter` once on ρ, playing the role of a new beacon fixing the chord ρ.

All these assumptions guarantee all `west` robots reach ρ in $O(\log 2) = O(1)$ epochs without colliding. □

6.5.4 Proofs for Phase Seq Match

We show here all the correctness proofs for Phase Seq Match (Section 6.4.6).

Lemma 57. *Given $\mathcal{C}_{unisect}$ with $q < 12$ robots in each uniform sector (boundaries excluded), a* `pre_matched` *robot can compute and reach its target uniform position U_j even if other* `moving_matched` *robots are moving and hiding* `right` *or* `left` *robots.*

Proof. The proof is similar to the one for Lemma 44. In particular, a `pre_matched` robot can always spot the `left` and `right` robots of its uniform sector Υ_i. Using the same strategy as Lemma 44, it recomputes the boundaries of the uniform sectors and the central angle $\frac{2\pi}{n}$ formed by two consecutive uniform positions (occupied by `left`, `right`, or `pivot` robots). Since a `pre_matched` robot has complete visibility of its uniform sector (in fact, no robot is moving within it), it counts the `matched` robots in Υ_i and, so, it recomputes the current index j of the uniform position to be matched. Thus, it heads to the uniform position U_j of Υ_i and sets its color as `matched`. □

6.6 Conclusions

In this chapter, we have presented a $O(1)$-time $O(1)$-color deterministic algorithm for the `Uniform Circle Formation` (`UCF`) problem under `ASYNCH` in the *luminous-opaque* robot model. Our algorithm minimizes the *performance SEC*, i.e., the smallest circular area touched by the swarm during the execution of the algorithm. The state-of-the-art solution [PS24] was optimal in either time or color complexity but not in both, and the performance SEC was not minimized.

Although our solution optimizes the *color complexity*, we have not focused on minimizing the *exact number* of colors used along the whole algorithm. Indeed, the size of our palette can be significantly reduced by reusing some colors in multiple steps along the algorithm. However, for the sake of clarity, instead of saving on the number of colors, we have used colors with specific, consistent, and meaningful names to help the reader understand their purposes and follow the algorithm steps. Moreover, minimizing the palette size would result in proving that no ambiguities, deadlocks, or wrong computations can ever happen along the algorithm. Trustingly, we leave this investigation for further work.

Future work might investigate whether a $O(1)$-time solution can be obtained for `UCF` considering oblivious robots (i.e., without lights) and/or non-rigid movements (i.e., robots may be stopped before reaching their destination).

Chapter 7

Universal Dancing

7.1 Introduction

Chapters 5 and 6 have studied the `Uniform Circle Formation` problem (`UCF`), a special case of the more general `Pattern Formation` problem that requires a swarm to form a given pattern Π and terminate (see Section 1.3.4). In `UCF`, the given Π contains the vertices of a regular n-gon. Indeed, `Pattern Formation` and `UCF` may represent primitive tasks to be repeatedly achieved in a more complex problem. A typical example of such more complex problems is `Flocking`, requiring the robots to construct a pattern and then maintain it while moving [Can+16; GP04; SIW11; Yan+11; Yan+09]. A more recent and less explored example is the `Dancing` problem, as nicely named in [Das+14], requiring the swarm to form a given ordered sequence of patterns (aka *choreography*) [Das+20; Das+15]. A notable real-world case of swarm choreography was shown during the 2020 Olympic Opening Ceremony, where a swarm of thousands of drones combined to display an artistic performance in the sky of Tokyo. While it is true that such a technological spectacle has required an engineering effort, it is evident that it could not have been achieved without an accurate algorithmic study. In that case, the drones had to combine in a sequence of patterns (e.g., the Olympic rings), dancing in perfect synchrony, and avoiding collisions. Of course, besides entertainment and performance, making swarms form a sequence of patterns may be an important task in other contexts (e.g., surveillance, exploration, shape reconfiguration in response to an event, and communication using patterns as codes).

Solving `Dancing` may appear as an iteration of `Pattern Formation` algorithms. However, as is intuitive, the combination of algorithms in a more complex solution does not come without a cost. Sometimes, ad hoc algorithmic strategies are necessary to combine existing solutions to obtain the desired result. This chapter focuses on both the techniques to effectively "iterate" `Pattern Formation` solutions to solve the `Dancing` problem, and, more importantly, on defining the most general class of feasible choreographies that can be performed by a swarm. Hence, the name `Universal Dancing`.

Related work

Several factors determine which patterns can be formed by a swarm, including the robot model, the initial configuration from which the swarm starts the algorithm, and the pattern type. Much literature has focused on characterizing the patterns formable under a given model (especially, its X^Y), and, more importantly, which patterns are formable from any initial configuration (aka `Arbitrary Pattern Formation`) [BT16; CDSN19a; CSN21b; DPV10; Flo+99; Flo+08; Pre19; VST22; YY14b]). Trivially, as mentioned in Section 1.3.4, for a swarm $\mathcal{R}$ to form a pattern Π, the *cardinality*

[0]The content of this chapter refers to the works [Fel+25a; Fel+25b].

condition must hold, i.e., the number of robots must match the number of vertices in Π. In the following, we will consider it as implicitly assumed.

Most of the algorithmic research has however relied on a variety of non-trivial assumptions, such as: restricting the type of allowed initial configurations (e.g., the robots occupy distinct initial locations); imposing restrictions on the number of robots or on the vertices of the pattern (e.g., the size of the swarm is a prime number, or the pattern contains distinct vertices); requiring specific symmetry relationship between the initial configuration and the pattern; the robots have some agreement on coordinate systems or on chirality; the movements of the robots are rigid; one of the robots is visibly different from all the others (i.e., there is a *leader*), etc. (see, e.g., [CDSN19a; CDSN19b; CSN21b; Flo+17; Flo+08; SY99; YS10]).

The most general problem in this class, `Universal Pattern Formation` (`UPF`), requires the robots to form *any* arbitrary pattern given in input, starting from *any* arbitrary initial configuration, regardless of the number of robots, and of the number of distinct vertices in the pattern, under just the cardinality condition. As for the power of the robots, the only requirement is strong multiplicity detection (i.e., robots can detect the exact number of robots occupying the same point), which is required by the nature of the problem. This problem is generally unsolvable in $\mathcal{OBLOT}^{\texttt{FSYNCH}}$ even with a leader, full agreement on the coordinate systems, and rigid movements. However, it has recently been shown that, except for point formation, `Universal Pattern Formation` is solvable in $\mathcal{OBLOT}^{\texttt{SEQ}}$ without any additional assumption [Flo+25]. This implies that `UPF`, including point formation, is solvable under $\mathcal{LUMI}^{\texttt{SEQ}}$. This result brings to light the strong computational power that robots have under the sequential schedulers in regard to pattern formation problems.

The transition from forming a single pattern to performing a choreography has been first investigated in [Das+15], where the authors present a solution for `Dancing` under $\mathcal{OBLOT}^{\texttt{SSYNCH}}$, assuming chirality agreement and rigid movements. However, the possibility to perform a choreography $\mathcal{S} = (\Pi_0, \ldots, \Pi_{q-1})^x$ is subject to several constraints:

- on the *initial configuration* $\mathcal{C}(0)$: the n robots must start from distinct locations and their positions must have the same symmetricity[1] as the patterns of $\mathcal{S}$;
- on the *patterns* Π_i: they must be all distinct, with n distinct vertices, and they must have the same degree of symmetry [2];
- on the *structure of the choreography*: $\mathcal{S}$ must be periodic, i.e., $x = \infty$.

Under these inevitable constraints, the algorithm in [Das+15] allows $\mathcal{OBLOT}$ robots to perform a choreography employing a peculiar technique, based on the related distances of selected robots, to collectively memorize the index of the next pattern to be formed.

The impact that the limited memory and communication means provided by $\mathcal{LUMI}$ has on the feasibility of `Dancing` was studied next [Das+20]. It was shown that, under $\mathcal{LUMI}$ and assuming a global chirality, the feasible sequences are subject to fewer constraints, even considering `ASYNCH` and non-rigid movements; more precisely, a swarm can perform a periodic choreography containing pattern repetitions and contractions/expansions, starting from any configuration with all robots at distinct locations, subject to a weaker conditions of symmetry on the patterns and initial configuration.

[1] Symmetricity of a pattern Π is the maximum number of equiangular sectors in which $SEC(\Pi)$ can be split, so that each one is a rotation of the others [SY99].

[2] Conditions on the symmetry of the patterns and of the initial configuration derive from the fact that symmetric robots can be activated simultaneously under `SSYNCH` (and, thus, `ASYNCH`) and they may perform a symmetric movement, thus never "breaking" the original symmetry.

The results of [Das+20; Das+15] have brought to light an important fact in the passage from `Pattern Formation` to `Dancing`. Let $Feas$ be the set of feasible (i.e., formable) patterns under a given set of assumptions on the robot model and initial configuration. Given a set $S \subseteq Feas$ of feasible patterns, it does not mean that any periodic sequence of patterns chosen from S is a feasible choreography under the same assumptions. In other words, *a sequence of feasible patterns is not necessarily a feasible sequence.*

Contribution

As mentioned, `Universal Pattern Formation` is solvable in $\mathcal{LUMI}^{\texttt{SEQ}}$ without any additional assumption [Flo+25]; that is, luminous-sequential robots can form any pattern.

In this chapter, we show that *a sequence of arbitrary patterns is performable* in $\mathcal{LUMI}^{\texttt{SEQ}}$, independently of any property (e.g., symmetry, multiplicities, shape, etc.) of the patterns or of the initial configuration, whether the sequence is periodic or not. In other words, the `SEQ` mode enables a swarm of $\mathcal{LUMI}$ robots to solve what we shall call the `Universal Dancing` problem.

Our proof is constructive: we develop an algorithm, analyze its properties, and establish its correctness. The main ingredient of our solution is the implementation of a distributed colored counter, which allows them to keep track of the current pattern being formed in the choreography: this new technique exploits the peculiar nature of `SEQ` to implement Gray code through robots' lights. The formation of each individual pattern is based on a novel technique that makes the robots match with their respective target points, exploiting properties of Dyck words on the Boolean alphabet. Our solution tolerates non-rigid movements. Yet, despite non-rigidity, our solution guarantees an interesting spatial property which provides homogeneity to the choreography performance: the smallest circles enclosing the patterns of the choreography (except for patterns with only two or three points) formed by the swarm are concentric.

We highlight that, unlike previous solutions [Das+20; Das+15], our algorithm does not assume chirality agreement among the robots, and it allows a swarm to perform choreographies of finite length (in addition to the periodic ones). However, we prove that, to be performable under $\mathcal{LUMI}$, the length/period of the choreographies is bounded by the compositions of n (i.e., the number of robots) into the number of colors available to the robots. Refer to Table 7.1 for a comparison of the existing algorithms for `Dancing`. Our results clearly show the strong computational power that `SEQ` robots have in regard to pattern formation problems, confirming and extending to the $\mathcal{LUMI}$ model the evidence found for $\mathcal{OBLOT}$ in [Flo+25].

7.2 Preliminaries

We here present in detail the model of robots under which we study the `Universal Dancing` problem, and all the elements required for its formal definition. Some notations have already been introduced in Section 1.3.4; however, for the sake of clarity and consistency, we recall here the essential ones.

7.2.1 Model

We consider a swarm of n robots which share the core features presented in Section 1.2.1. Moreover, we make the following assumptions:

Algorithm	Model	Constraints on $\mathcal{S} = (\Pi_0, \dots, \Pi_{q-1})^x$	Constraints on $\mathcal{C}(0)$
[Das+15]	$\mathcal{OBLOT}^{\texttt{SSYNCH}}$ rigid, global chirality	$x = \infty$ (periodic), $\Pi_i \neq \Pi_j$ for any $i \neq j$, Π_i no multiplicities for any i, all Π_i have the same symmetricity	distinct positions, $\mathcal{C}(0)$ same symmetricity of Π_i
[Das+20]	$\mathcal{LUMI}^{\texttt{ASYNCH}}$ non-rigid, global chirality	$x = \infty$ (periodic), weaker constraints on patterns symmetricity w.r.t. [Das+15]	distinct positions, weaker constraints on the symmetricity of $\mathcal{C}(0)$ w.r.t. [Das+15]
this chapter (`Universal`)	$\mathcal{LUMI}^{\texttt{SEQ}}$ with $k \geq 4$ colors non-rigid, no global chirality	$q \leq \binom{n+k-7}{k-4}$	none

TABLE 7.1: Comparison of the existing algorithms for `Dancing`.

- we consider the $\mathcal{LUMI}^{\texttt{SEQ}}$ model;
- robots are completely disoriented (i.e., no agreement on any component of their local coordinate systems);
- robots suffer from *variable disorientation* (VARDIS). For simplicity, we assume the EGO hypothesis, i.e., all local coordinate systems are egocentric;
- movements are *non-rigid* (NONRIG), thus robots can be stopped after having traveled at least a $\delta > 0$ fixed distance, but unknown to robots;
- robots can form multiplicities and have strong multiplicity detection;
- robots have complete visibility of the swarm.

7.2.2 (Sequences of) Patterns

Patterns and vertices. Let us consider an absolute[3] coordinate system $\hat{\Xi}$ on the Euclidean plane $\mathbb{R}^2$; unless otherwise specified, we consider the positions of the points of the plane according to $\hat{\Xi}$. Let $\Pi = \{\!\{v_1, \dots, v_n\}\!\}$ a pattern of n vertices, so that $v_i \in \mathbb{R}^2$. We denote with $shape(\Pi)$ the set of the unique positions in Π. A pattern Π can belong to one of the four classes: $\Pi \in$ `Point` if $shape(\Pi)$ is a singleton; $\Pi \in$ `TwoPoints` if $|shape(\Pi)| = 2$; $\Pi \in$ `ThreePoints` if $|shape(\Pi)| = 3$; otherwise, $\Pi \in$ `NPoints`. `NLine` is the special subclass of `NPoints` where the positions of Π are aligned.

We remind that $\mathscr{P}(\Pi)$ denotes the set of patterns similar to Π, i.e., all the patterns obtained from Π by any composition of non-degenerate similarity transformations (uniform scaling with a non-zero scale factor, translation, rotation, and reflection). It will be convenient to define a pattern Π using the set notation, so that $\Pi = \{(v_1, \#_1), \dots, (v_m, \#_m)\}$, where $\#_i \in \mathbb{N}_{>0}$ indicates the multiplicity of the relative vertex and where $\sum_{j=1}^{m} \#_j = n$. We refer to $(v_i, \#_i)$ as a *multivertex* of Π.

Configurations and SEC. Let $\mathcal{R}$ be a swarm of robots, starting from an initial configuration $\mathcal{C}(0)$ where all the lights are set to `OFF`; no other assumptions are made on $\mathcal{C}(0)$. Given a configuration $\mathcal{C}$, we denote with $\mathcal{C}_{|\mathbb{R}^2}$ the multiset containing only the positions in $\mathcal{C}$. We say that a multivertex $(v_i, \#_i)$ is *saturated* if exactly $\#_i$ robots lie on v_i in $\mathcal{C}$; on the contrary, we say that it is *unsaturated* (*oversaturated*, resp.) if it is covered by fewer than (more than, resp.) $\#_i$ robots. We say that $\mathcal{R}$ forms a pattern Π at time t if $\mathcal{C}_{|\mathbb{R}^2} \in \mathscr{P}(\Pi)$.

[3] Unknown by the robots.

Given a pattern Π, we indicate with $SEC(\Pi)$ the *smallest enclosing circle* of Π and with $\omega(\Pi)$ the center of $SEC(\Pi)$. We remind that the SEC of a set of points is unique, and at least three (or two antipodal) points of the set lie on its perimeter. With a slight abuse of notation, given a configuration $\mathcal{C}$, we indicate with $SEC(\mathcal{C})$ and $\omega(\mathcal{C})$ the smallest circle enclosing the points in $\mathcal{C}_{|\mathbb{R}^2}$ and its center, respectively.

Choreographies. We define a *sequence of patterns*, also known as *choreography*, as $\mathcal{S} = (\Pi_0, \dots, \Pi_{q-1})^x$ where each Π_i is a pattern and $x \in \{1, \infty\}$. It must be $\Pi_{i+1} \notin \mathscr{P}(\Pi_i)$; if $\mathcal{S}$ is *periodic* (i.e., if $x = \infty$), then it must also hold that $\Pi_0 \notin \mathscr{P}(\Pi_{q-1})$. If $\mathcal{S}$ is periodic, we assume that the sequence $\Pi_0, \dots, \Pi_{q-1}$ cannot be written as $(\Pi_0, \dots, \Pi_{h-1})^{\frac{q}{h}}$ for any $h < q$. We say that q is the *length* (*period*, resp.) of $\mathcal{S}$ if $x = 1$ ($x = \infty$, resp.).

We say that $\mathcal{R}$ *performs* $\mathcal{S}$ if, for any $j = 1, \dots, x$, there exists a series of increasing finite times $\{t_{i+(j-1)q}\}_{i=0,\dots,q-1,j=1,\dots,x}$ such that the swarm forms Π_i at time $t_{i+(j-1)q}$. If $x = 1$, then the swarm must remain still after time t_{q-1} (i.e., after having formed the last pattern of the choreography). Obviously, a choreography $\mathcal{S}$ must satisfy the *cardinality condition* which states that each pattern of $\mathcal{S}$ must have n vertices to be performed by an n-swarm $\mathcal{R}$.

7.2.3 The `Dancing` problems

Given a swarm $\mathcal{R}$, the general `Dancing` problem requires $\mathcal{R}$ to perform any *feasible* choreography. A choreography is feasible if it satisfies some necessary conditions so that a swarm of n robots can perform it; indeed, the cardinality condition is one of them. Moreover, some constraints may be satisfied by the initial configuration of $\mathcal{R}$ so that it can perform a feasible choreography. Formally, we can define a `Dancing` problem for a swarm $\mathcal{R}$ of n robots as the tuple $\mathfrak{D} = \langle \Sigma(n), \mathcal{I}(n), \phi \rangle$, where

- $\Sigma(n)$ defines the *alphabet of patterns*, i.e., the set of patterns with n vertices a feasible choreography can be composed of;
- $\mathcal{I}(n)$ defines the set of all initial configurations from which $\mathcal{R}$ can start a feasible choreography;
- ϕ is a predicate that must be satisfied by all the feasible choreographies (for example, it can limit their length, the presence of repetitions, the periodicity, etc.).

Thus, $\mathcal{S} = (\Pi_0, \dots, \Pi_{q-1})^x$ is a *feasible* choreography for $\mathfrak{D}$ if $\Pi_i \in \Sigma(n)$ for any $i \in \{0, \dots, q-1\}$ and $\mathcal{S}$ satisfies ϕ. Let $Feas(\mathfrak{D})$ be the set of all feasible choreographies for $\mathfrak{D}$. Note that, for $\mathfrak{D}$ to be well defined, each pattern $\Pi \in \Sigma(n)$ must appear in at least one choreography in $Feas(\mathfrak{D})$; in other words, ϕ cannot exclude a pattern of $\Sigma(n)$ from all the feasible choreographies. An algorithm $\mathbb{A}$ solves $\mathfrak{D}$ if, for any swarm $\mathcal{R}$ starting from a configuration in $\mathcal{I}(n)$, it makes $\mathcal{R}$ perform any feasible choreography in $Feas(\mathfrak{D})$.

A `Dancing` problem $\langle \Sigma(n), \mathcal{I}(n), \phi \rangle$ is said to be `Universal` if $\Sigma(n) = (\mathbb{R}^2)^n$ (i.e., any pattern with n vertices), $\mathcal{I}(n) = (\mathbb{R}^2 \times \{\texttt{OFF}\})^n$ is the set of all possible initial configurations where robots are `OFF`-colored, and ϕ can only bound the length/period of the choreographies. As we will see, we solve `Universal Dancing` under a specific bound ϕ. We here prove that if we drop such a bound, i.e., if ϕ is a tautology $\top$, then the problem is unsolved under our model. Formally:

Theorem 46. *The $\langle (\mathbb{R}^2)^n, (\mathbb{R}^2 \times \{\texttt{OFF}\})^n, \top \rangle$ problem cannot be solved under $\mathcal{LUMI}^{\texttt{SEQ}}$ even if robots have rigid movements and share a global coordinate system.*

Proof. By contradiction, let $\mathbb{A}$ be an algorithm for $\langle (\mathbb{R}^2)^n, (\mathbb{R}^2 \times \{\texttt{OFF}\})^n, \top \rangle$ under $\mathcal{LUMI}^{\texttt{SEQ}}$ with $k > 1$ colors. Suppose $\mathcal{R}$ is a swarm of n robots, and let $z = \binom{n+k-1}{k-1}$ be the number of color-compositions of the swarm. Let $\mathcal{S} = (\mathtt{P}_1, \Pi_1, \ldots, \mathtt{P}_z, \Pi_z, \mathtt{P}_{z+1}, \Pi_{z+1})$ be a feasible choreography for $\mathcal{R}$, such that $\mathtt{P}_i \in \texttt{Point}$ for any $1 \leq i \leq z+1$ and $\Pi_i \neq \Pi_j$ for any $1 \leq i \neq j \leq z+1$. In other words, $\mathcal{R}$ is asked to perform a choreography that requires the swarm to gather into a point before forming each pattern of the series $\Pi_1, \ldots \Pi_{z+1}$. Following the definition of choreography, we know that $\Pi_i \notin \texttt{Point}$. Consider an execution of $\mathbb{A}$ under a *round-robin* activation scheduler $\mathfrak{A}$. Let t_b be the first time when the swarm forms a point $\mathtt{P}_b$ where robots have the same color-composition as while forming a previous point $\mathtt{P}_a$, with $a < b$. Note that t_b exists since the maximum number of k-colorings for a swarm of n robots is z. Let t_a be the time when $\mathtt{P}_a$ was formed under $\mathfrak{A}$. W.l.o.g. and for simplicity, let us assume that the same colors in $\mathtt{P}_a$ are assigned to the same robots in $\mathtt{P}_b$.

Consider the $\texttt{SEQ}$ scheduler $\mathfrak{A}' = \mathfrak{A}(0) \ldots \mathfrak{A}(t_a) \left(\mathfrak{A}(t_{a+1}) \ldots \mathfrak{A}(t_b)\right)^{\infty}$ such that it equals $\mathfrak{A}$ until t_b and then activates the same sequence of robots as starting from t_{a+1}. It is easy to prove that $\mathfrak{A}'$ guarantees the fairness condition too: in fact, all the robots are activated at least once during the time $[t_{a+1}, t_b]$ in $\mathfrak{A}$. This is true since, in that time, the swarm passes from a $\texttt{Point}$ pattern (i.e., $\mathtt{P}_a$), to a non-point pattern (i.e., Π_a), and lastly it gathers again into a point (i.e., $\mathtt{P}_b$): to perform this sequence of patterns, at least one epoch is needed.

So, if we consider the swarm under $\mathfrak{A}'$, then, after time t_b, the robots will perform the same actions as in the interval $[t_{a+1}, t_b]$ in a loop, since they will take the same snapshots as in that time frame. This results in the swarm never terminating. Contradiction achieved. □

7.3 Techniques

Our algorithm uses a palette $\mathfrak{L} = \{\ell_1, \ldots, \ell_{k-3}, \mathsf{L}_1, \mathsf{L}_2, \mathsf{L}_3\}$ of $k \geq 4$ colors which allows a n-size swarm $\mathcal{R}$ to perform any feasible choreography $\mathcal{S} = (\Pi_0, \ldots, \Pi_{q-1})^x$ with $x \in \{1, \infty\}$ that satisfies $q \leq \binom{n+k-7}{k-4}$. The colors L_1, L_2, and L_3 are destined for three robots called *leaders*. All the other $n-3$ robots, called *non-leaders*, will assume colors in $\{\ell_1, \ldots, \ell_{k-3}\}$. Using those colors, the non-leaders implement a counter which, thanks to Gray Code (Section 7.3.1), encodes the index i of the pattern $\Pi_i \in \mathcal{S}$ to be formed. Then, the leaders have the role to set the univocal position of the *actual pattern* $\Pi \in \mathscr{P}(\Pi_i)$ on whose vertices robots must arrange themselves. For this purpose, robots need to agree on a global ranking of the multivertices of Π_i (Section 7.3.2) which allows them to elect some vertices. Exploiting this ranking, the leaders choose some vertices of Π and arrange themselves on a triangle, called *chiral angle* (Section 7.3.3), that provides a global orientation to $\mathcal{R}$ and the position of Π. Once Π is fixed, the non-leaders must move to their designated pattern points, for which we employ a Dyck word–based matching (Section 7.3.4).

We now describe in detail the main concepts and techniques adopted by our algorithm, which combine both the power of $\mathcal{LUMI}^{\texttt{SEQ}}$ and some concepts adapted from other fields.

7.3.1 Colored counter and composition Gray codes

One of the main benefits of the sequential schedulers is the possibility of implementing a *distributed counter* through the robots' lights, by exploiting the Gray code. The size of the counter (i.e., the maximum value that can be counted) increases as the number of available colors and the number of robots in the swarm increase. Then, in

Section 7.4, we will explain how our algorithm uses this technique to keep track of the ongoing pattern that is being formed, and to reset the counter in case of periodic choreographies. Thus, the counter size defines a tight bound on the length (if finite) /period (if periodic) q of the choreographies performable by a swarm of n robots with k colors according to our algorithm.

The $n-3$ non-leader robots will use their lights, colored using all the colors in $\{\ell_1, \ldots, \ell_{k-3}\}$. We assume that such colors are totally ordered: w.l.o.g., let $\ell_1 < \cdots < \ell_{k-3}$, and we assume that $\ell_1 = \texttt{OFF}$, i.e., the color each robot is initialized. We implement a counter whose value iterates over the range of integers $\{0, \ldots, z-1\}$, where $z = \binom{n+k-7}{k-4}$ corresponds to the number of weak compositions of $n-3$ into $k-3$ parts[4]. For the sake of simplicity, in the remainder of this section, we will use n and k instead of $n-3$ and $k-3$, respectively, thus illustrating the counter technique for a general set of n robots using $k \geq 2$ colors[5] $\{\ell_1, \ldots, \ell_k\}$.

Given a configuration $\mathcal{C}$, we define the *composition vector* (or simply, *counter*) as the tuple $X = (x_1, x_2, \ldots, x_k)$, where x_i denotes the number of robots in $\mathcal{C}$ with color ℓ_i. Let $\mathfrak{X}(n,k)$ be the set of all $z = \binom{n+k-1}{k-1}$ composition vectors made with n robots and k colors. If there exists a univocal method to order all the elements in $\mathfrak{X}(n,k)$ so that $X_0 < \cdots < X_{z-1}$, then the robots can encode the composition vector X_a with the integer a, which will be the current value of the swarm counter. Another desirable property of the order $X_0 < \cdots < X_{z-1}$ is that any two consequent vectors $X_a = (x_1, \ldots, x_k)$ and $X_{a+1} = (x'_1, \ldots, x'_k)$ (with $a < z-1$) differ at exactly two indexes: in particular, there exist $i \neq j \in \{1, \ldots, k\}$ such that $x_i = x'_i - 1$ while $x_j = x'_j + 1$. This means that if the swarm forms a composition vector X_a and wants to increment the counter by 1, it is sufficient for a ℓ_i-colored robot to turn its light into ℓ_j, thus updating the composition vector into X_{a+1}.

To implement the order of $\mathfrak{X}(n,k)$ that fits our purposes, we use *Gray code* for compositions as described by Klingsberg [Kli82]. In that work, the authors present a fast algorithm to generate the Gray codes[6] to represent the compositions of n into k parts. In particular, the algorithm produces the ordered list $(n,k) = \langle X_0, \ldots, X_{z-1} \rangle$ of all the Gray codes of all the $z = \binom{n+k-1}{k-1}$ compositions. We illustrate the recursive method to generate the list of gray codes (j,k) for any values $j \geq 0$ and $k \geq 2$. The reverse of the list is denoted by $-(j,k)$. For the base case, $k = 2$, we have $(j,2) = \langle X_0, \ldots, X_j \rangle$ where $X_a = (j-a, a)$. For an arbitrary $k > 2$, the gray code can be obtained by recursion as follows,

$$(j, k+1) = \oplus_{l=0}^{j} (-1)^l [(j-l, k) \otimes \{l\}]$$

where $\oplus$ is the concatenation operator, and $\otimes$ is the Cartesian product of the sets. From the results of [Kli82], we know that this Gray code construction follows the same property: it begins with the composition vector $X_0 = (n, 0, \ldots, 0)$ and ends with $X_{z-1} = (0, \ldots, 0, n)$. See Figure 7.1 for some examples.

7.3.2 Multivertices ranking

We now describe how robots obtain an unambiguous rank of the multivertices of a pattern $\Pi = \{(v_1, \#_1), \ldots, (v_m, \#_m)\} \in \texttt{NPoints}$ to be formed with n vertices (and m multivertices). Refer to Figure 7.2 for an example. To lighten the notation, we

[4] A weak composition of n into k parts is an ordered representation of n as the sum of k non-negative integers.

[5] The case $k = 1$ is trivial.

[6] Gray code (patented in 1953 by Frank Gray [Fra53]) is a binary encoding method with the additional property that the representations of two consecutive integers differ by only one bit.

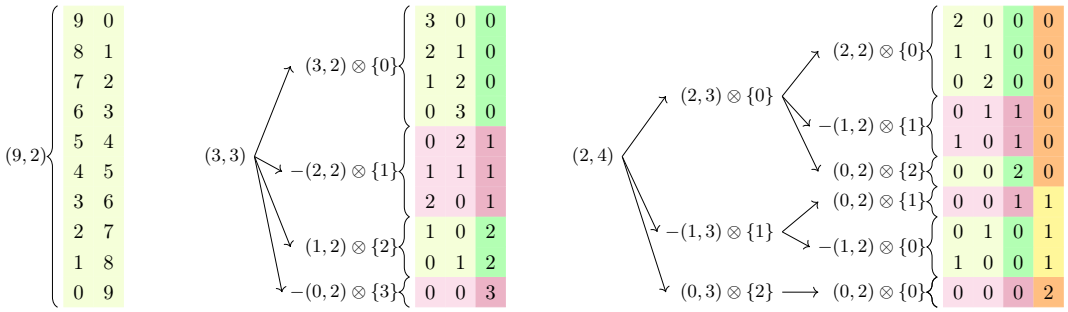


FIGURE 7.1: Gray codes for $(9,2)$, $(3,3)$, and $(2,4)$. They allow a group of 9, 3 or 2 robots to implement a counter from 0 to 9 by using 2, 3, or 4 colors, respectively.

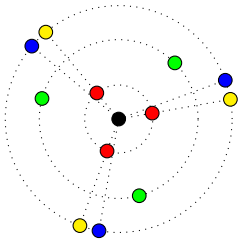


FIGURE 7.2: Ranking of the multivertices. Assuming each multivertex has cardinality 1, the classes of multivertices are ordered in this way: $[v_{yellow}] < [v_{blue}] < [v_{green}] < [v_{red}] < [v_{black}]$.

often indicate a multivertex $(v_i, \#_i)$ as only v_i. Let Ω be the $SEC(\Pi)$ whose center is O. Let us suppose, w.l.o.g., that the positions of the multivertices in Π are given considering a coordinate system whose origin and unit distance are $O = (0,0)$ and $radius(\Omega) = 1$, respectively. Let Π_O denote Π without the multivertex lying in O, if it exists, and let $m' = |shape(\Pi_O)| \in \{m, m-1\}$.

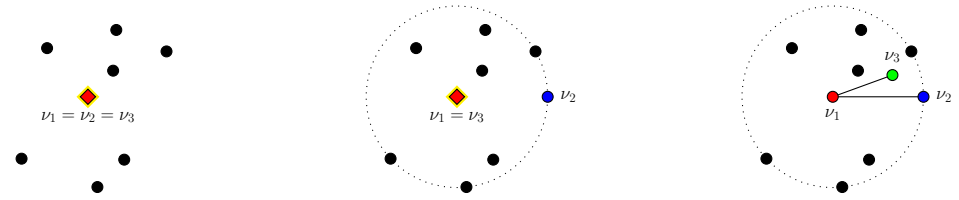
For any multivertex v_i, we define the following elements:

- $\Psi(v_i)$ as the radial projection of v_i on Ω, if $v_i \in \Pi_O$ (i.e., the intersection between the radius where v_i lies and Ω). Let $\{\psi_1, \ldots, \psi_{m''}\}$ be the set of all the radial projections, where $m'' \leq m'$, assuming w.l.o.g. that $[\psi_1, \ldots, \psi_{m''}]$ is the cyclic order[7] according to the clockwise orientation of Ξ.
- $\theta_i = \angle \psi_j O \psi_{j+1}$ and $\hat{\theta}_i = \angle \psi_j O \psi_{j-1}$ where $\psi_j = \Psi(v_i)$. If v_i lies on O, we set $\theta_i = \hat{\theta}_i = 0$.
- $\mu(v_i) = (\rho(v_i), \theta_i, \#_i)$ and $\hat{\mu}(v_i) = (\rho(v_i), \hat{\theta}_i, \#_i)$ where $\rho(v_i) = 1 - dist(v_i, O)$.

The multivertices of Π belong to concentric circles of Ω. Then, it is possible to unambiguously define two cyclic orders on Π_O, using the radial projections and the distances from O: one clockwise according to Ξ, and the other counterclockwise. Let $\mathcal{A} = [v_{a_1}, \ldots, v_{a_{m'}}]$ be the clockwise one, and let $\mathcal{B} = [v_{b_1}, \ldots, v_{b_{m'}}]$ be the counterclockwise[8]. Starting from each $v_{a_i} \in \mathcal{A}$ we define the string $CW(v_{a_i}) = \mu(v_{a_i})\mu(v_{a_{i+1}}) \ldots \mu(v_{a_{i-1}})$ (clockwise); from each $v_{b_i} \in \mathcal{B}$ we define the string $CCW(v_{b_i}) = \hat{\mu}(v_{b_i})\hat{\mu}(v_{b_{i+1}}) \ldots \hat{\mu}(v_{b_{i-1}})$ (counterclockwise). Let $\gamma(v_i) = \min\{CW(v_i), CCW(v_i)\}$, where the total order $\leq$ of the strings is given by considering the three numerical orders (distances, angles, and

[7] We here consider indexes in the circular range $[1, \cdots, m'']$.

[8] $a_1, \ldots, a_{m'}$ and $b_1, \ldots, b_{m'}$ represent two permutations of the m' indexes of the multivertices of Π_O.



(A) $\Pi_i \in$ `Point`. (B) $\Pi_i \in$ `TwoPoints`. (C) $\Pi_i \in$ (`NPoints` $\setminus$ `NLine`).

FIGURE 7.3: Chiral angles and supporting circles in different scenarios where $\mathcal{C}_{|\mathbb{R}^2} \in$ `NPoints`.

multiplicities) in the triples[9]. If v_i lies on O, we set $\gamma(v_i) = (\rho(v_i), \theta_i, \#_i)$ which is $(1, 0, \#_i)$ by definition (in this case, string $\gamma(v_i)$ is the concatenation of just one $\mu(\cdot)$). Formally, given two triples $(\rho, \theta, \#)$ and $(\rho', \theta', \#')$ from the domain $[0, 1]\times[0, 2\pi)\times[1, n]$, we define the following relation:

$$(\rho, \theta, \#) \leq (\rho', \theta', \#') \iff \begin{cases} \rho < \rho' & \text{or} \\ \rho = \rho' \wedge \theta < \theta' & \text{or} \\ \rho = \rho' \wedge \theta = \theta' \wedge \# \leq \#' \end{cases}$$

Thus, the multivertices of Π can be partitioned in equivalence classes $\{[v_1]_\equiv, \ldots, [v_s]_\equiv\}$ so that $1 \leq s \leq m$ and $v_j \in [v_i]_\equiv$ iff $\gamma(v_j) = \gamma(v_i)$. Indeed, it is possible to define a total strict order (*ranking*) among the classes $[v_{\iota_1}]_\equiv < \cdots < [v_{\iota_s}]_\equiv$ which is obtained[10] by considering the corresponding strings $\gamma(v_{\iota_j})$ for any $1 \leq j \leq s$. Note that, according to the definition of $\leq$, the multivertices in $[v_{\iota_1}]_\equiv$ lie on Ω (in fact, they have the minimal value of $\rho(\cdot)$). Moreover, if a multivertex lies on $\omega(\Pi)$, then $[v_{\iota_s}]_\equiv$ is a singleton containing only such a multivertex.

Let $\Pi' \in \mathscr{P}(\Pi)$ be a similar pattern to Π, and let $\tau : \Pi \to \Pi'$ be the related similarity transformation. It is obvious that, if $\Pi/_\equiv = \{[v_1], \ldots, [v_s]\}$, then $\Pi'/_\equiv = \{[\tau(v_1)], \ldots, [\tau(v_s)]\}$. Moreover, the ranking in Π is preserved in Π', i.e., $[\tau(v_{\iota_1})]_\equiv < \cdots < [\tau(v_{\iota_s})]_\equiv$.

7.3.3 Chiral angle

Let $\mathcal{C}$ be a configuration from which the swarm must start the formation of a pattern $\Pi_i \in \mathcal{S}$. Being totally disoriented, the robots need to univocally establish the position, orientation, and scale of the *actual pattern* $\Pi \in \mathscr{P}(\Pi_i)$ they have to arrange on. Thus, before forming Π_i, our algorithm includes a preliminary phase where the leaders L_1, L_2, and L_3, place themselves on three points ν_1, ν_2, ν_3 respectively. The formed triangle $\triangle\nu_1\nu_2\nu_3$ (possibly degenerate) is called *chiral angle*, and provides the swarm a partial (if the triangle is degenerate) or total (if the triangle is not degenerate) global coordinate system of the plane. Notably, ν_1 represents its origin; if $\nu_1 \neq \nu_2$, then ν_2 represents the coordinate $(1, 0)$ (thus fixing the unit distance and the x-axis); if ν_3 is not aligned with ν_1 and ν_2, then ν_3 represents a y-positive coordinate (thus fixing the y-axis, and thus the clockwise orientation of the plane).

Once L_1 and L_2 have fixed ν_1 and ν_2 for Π_i, all the robots can construct the circle Ω_i whose center is ν_1 and whose radius is $dist(\nu_1, \nu_2)$. We call Ω_i as the *supporting circle* of Π_i. If $\Pi_i \in$ (`Point` $\cup$ `NPoints`), then the swarm must form Π so that $\Omega_i = SEC(\Pi)$

[9] Indeed, any $v_i \in \Pi_O$ corresponds to some $v_{a_j} \in \mathcal{A}$ and some $v_{b_h} \in \mathcal{B}$.
[10] $(\iota_1, \ldots, \iota_s)$ represents a permutation of $\{1, \ldots, s\}$.

(trivially, it means that if $\Pi_i \in \texttt{Point}$, then Ω_i will be a degenerate circle). Otherwise, if $\Pi_i \in (\texttt{TwoPoints} \cup \texttt{ThreePoints})$, then the swarm must form Π so that $\overline{\nu_1\nu_2}$ is the (longest) edge of Π. Note that, in all the cases, all the vertices of Π are contained within Ω_i.

Note that our strategy for constructing the supporting circles is to guarantee a spatial homogeneity in the $\mathcal{S}$ performance: notably, all the supporting circles are concentric. In fact, regardless of Π_i, ν_1 is static at the same point of the plane, which corresponds to $\omega(\mathcal{C})$ if $\mathcal{C}_{|\mathbb{R}^2} \in \texttt{NPoints}$, otherwise ν_1 corresponds to the position of L_1, denoted by $pos(\mathsf{L}_1)$ (thus, L_1 does not move since it already lies on ν_1). Moreover, given any subsequence $\Pi_a, \ldots, \Pi_b$ of $\mathcal{S}$, if $\Pi_i \notin \texttt{Point}$ for any $i \in [a, b]$, then $\Omega_a, \ldots, \Omega_b$ are equal. We now explain how to compute ν_2 and ν_3 for the different cases, and the actual pattern $\Pi \in \mathscr{P}(\Pi_i)$ to be formed.

If $\Pi_i \in \texttt{Point}$, the chiral angle (and thus Ω_i) degenerates on the same point $\nu_1 = \nu_2 = \nu_3$, which is the vertex at which the swarm will gather (see Figure 7.3a). Formally, $\Pi = \{(\nu_1, n)\}$.

If $\Pi_i \in \texttt{TwoPoints}$, then $\nu_3 = \nu_1$, while ν_2 is a different point: the swarm will arrange itself on these two vertices with the respective multiplicities. The point ν_2 is computed in this way: if L_2 does not lie on ν_1, then its position becomes the new ν_2 for Π_i (see Figure 7.3b); otherwise, the position $(1, 0)$ according to the local coordinate system of L_2 becomes the new ν_2. Formally, if $\Pi_i = \{(v_1, \#_1), (v_2, \#_2)\}$ with $\#_1 \geq \#_2$, then $\Pi = \{(\nu_1, \#_1), (\nu_2, \#_2)\}$.

If $\Pi_i \in \texttt{ThreePoints}$, then ν_1, ν_2, ν_3 are three distinct points forming the target triangle. Point ν_2 is computed as before, and it fixes the longest edge $\overline{\nu_1\nu_2}$ of the target pattern Π. Once L_1 and L_2 have fixed ν_1 and ν_2, then L_3 chooses the position[11] of ν_3, so that $\triangle\nu_1\nu_2\nu_3 \in \mathscr{P}(\Pi_i)$ and $dist(\nu_1, \nu_2) \geq dist(\nu_1, \nu_3) \geq dist(\nu_2, \nu_3)$. Formally, if $\Pi_i = \{(v_1, \#_1), (v_2, \#_2), (v_3, \#_3)\}$ with $dist(v_1, v_2) \geq dist(v_1, v_3) \geq dist(v_2, v_3)$, $\#_1 \geq \#_2$, and with $\#_1 \geq \#_3$ if $dist(v_1, v_3) = dist(v_1, v_2)$, then $\Pi = \{(\nu_1, \#_1), (\nu_2, \#_2), (\nu_3, \#_3)\}$.

If $\Pi_i \in \texttt{NPoints}$, then ν_1, ν_2, ν_3 are three distinct points forming a triangle with $dist(\nu_1, \nu_2) \geq dist(\nu_1, \nu_3)$. Even in this case, ν_2 is computed as for the $\texttt{TwoPoints}$ case. The swarm must form Π so that $SEC(\Pi) = \Omega_i$ (which is defined by ν_1 and ν_2), so that ν_2 corresponds to one vertex of Π. The precise position of Π within Ω_i will be given by L_3 once set on ν_3. Let us describe how L_3 computes ν_3. Assume that the ranking of the multivertices of Π_i is $[v_{\iota_1}]_{\equiv} < \cdots < [v_{\iota_s}]_{\equiv}$. Then ν_2 will be one multivertex of Π corresponding to one multivertex of Π_i in $[v_{\iota_1}]$. As seen in Section 7.3.2, we know that the vertices in the minimal class $[v_{\iota_1}]$ lie on $SEC(\Pi_i)$. So, L_3 chooses one vertex in $[v_{\iota_1}]$, say w.l.o.g. v_{ι_1}, and then it computes a similarity transformation τ such that $\tau(v_{\iota_1}) = \nu_2$. If $\Pi_i \in \texttt{NLine}$, then L_3 chooses the multivertex closest to ν_2 but not lying on ν_1, and moves there. Otherwise, L_3 chooses the multivertex closest to ν_2 but not collinear with ν_1 and ν_2: if multiple multivertices are eligible, then L_3 chooses one belonging to the lowest class in the ranking (see Figure 7.3c). This strategy allows all robots to unambiguously reconstruct $\tau(\Pi_i) = \Pi$. Table 7.2 summarizes how $\triangle\nu_1\nu_2\nu_3$ is computed.

7.3.4 Robot-vertex matching with Dyck words

Our algorithm adopts a technique to match each robot with its target vertex by exploiting the Dyck words on the Boolean alphabet $\{0, 1\}$. Dyck words are well-parenthesized strings over an alphabet of brackets [CS63; Gin66]; in our case, 0 is the opening bracket, while 1 is the closing one.

[11] Note in fact that different positions of ν_3 can be found in order to form the target triangle.

Π_i	$\triangle\nu_1\nu_2\nu_3$	$\Pi \in \mathscr{P}(\Pi_i)$ to be formed
$\in$ Point $= \{(v_1, n)\}$	$\nu_1 = \nu_2 = \nu_3 = \begin{cases} \omega(\mathcal{C}) & \mathcal{C}_{\lvert\mathbb{R}^2} \in$ NPoints $pos(\mathsf{L}_1)$ otherwise	$\Pi = \{(\nu_1, n)\}$
$\in$ TwoPoints $= \{(v_1, \#_1), (v_2, \#_2)\}$ $\#_1 \geq \#_2$	$\nu_1 = \nu_3 = \begin{cases} \omega(\mathcal{C}) & \mathcal{C}_{\lvert\mathbb{R}^2} \in$ NPoints $pos(\mathsf{L}_1)$ otherwise $\nu_2 = \begin{cases} (0,1)$ of $\mathsf{L}_2 & \mathcal{C}_{\lvert\mathbb{R}^2} \in$ Point $pos(\mathsf{L}_2)$ otherwise	$\Pi = \{(\nu_1, \#_1), (\nu_2, \#_2)\}$
$\in$ ThreePoints $= \{(v_1, \#_1), (v_2, \#_2), (v_3, \#_3)\}$ $dist(v_1, v_2) \geq dist(v_1, v_3) \geq dist(v_2, v_3)$ with $\#_1 \geq \#_2$, $\#_1 \geq \#_3$ if $dist(v_1, v_3) = dist(v_1, v_2)$	$\nu_1 = \begin{cases} \omega(\mathcal{C}) & \mathcal{C}_{\lvert\mathbb{R}^2} \in$ NPoints $pos(\mathsf{L}_1)$ otherwise $\nu_3 =$ position of v_3 for L_3 $\nu_2 = \begin{cases} (0,1)$ of $\mathsf{L}_2 & \mathcal{C}_{\lvert\mathbb{R}^2} \in$ Point $pos(\mathsf{L}_2)$ otherwise	$\Pi = \{(\nu_1, \#_1), (\nu_2, \#_2), (\nu_3, \#_3)\}$ $dist(\nu_1, \nu_2) \geq dist(\nu_1, \nu_3) \geq dist(\nu_2, \nu_3)$
$\in$ NPoints $= \{(v_1, \#_1), \ldots, (v_m, \#_m)\}$ $[v_{\iota_1}]_{\equiv} < \cdots < [v_{\iota_s}]_{\equiv}$	$\nu_1 = \begin{cases} \omega(\mathcal{C}) & \mathcal{C}_{\lvert\mathbb{R}^2} \in$ NPoints $pos(\mathsf{L}_1)$ otherwise $\nu_2 = \begin{cases} (0,1)$ of $\mathsf{L}_2 & \mathcal{C}_{\lvert\mathbb{R}^2} \in$ Point $pos(\mathsf{L}_2)$ otherwise $\nu_3 = \begin{cases} v \neq \nu_1$ closest to ν_2 & $\Pi_i \in$ NLine $v \neq \nu_1$ closest to ν_2, not aligned with ν_1, ν_2 otherwise	$\Pi = \{(\tau(v_1), \#_1), \ldots, (\tau(v_m), \#_m)\}$ $[\tau(v_{\iota_1})]_{\equiv} < \cdots < [\tau(v_{\iota_s})]_{\equiv}$ s.t. $\nu_2 \in [\tau(v_{\iota_1})]_{\equiv}$ $\nu_3 \in [\tau(v)]_{\equiv}$

TABLE 7.2: Chiral angle $\triangle\nu_1\nu_2\nu_3$ and Π in each different scenarios.

Given a Boolean string x, $|x|$ denotes its length, and $|x|_0$ ($|x|_1$, resp.) denotes the number of 0s (1s, resp.) contained in x. A Boolean string x is *balanced* if $|x|_0 = |x|_1$. We remind that ε denotes the empty string (i.e., without symbols). For any $i \in \{0, \ldots, |x|\}$, we indicate with $\delta_x(i) = |y|_0 - |y|_1$ where y is a prefix of x of length i.

We consider the Dyck language $\mathcal{L}_{\mathsf{Dyck}} = \{x \in \{0,1\}^* \text{ s.t. } |x|_0 = |x|_1 \wedge (\delta_x(i) \geq 0 \; \forall i \in \{0, \ldots, |x|\})\}$. In other terms, x is a Dyck word if it is balanced and any prefix of x must not have more 1s than 0s. Equivalently, $x \in \mathcal{L}_{\mathsf{Dyck}}$ if *(i)* $x = \varepsilon$, or *(ii)* $x = 0y1z$ for some $y, z \in \mathcal{L}_{\mathsf{Dyck}}$ where the 0 and 1 that envelops y are a pair of *matched brackets*. If $x = x_1 \cdots x_m$ is a Dyck word, then $x_b = 1$ is matched with $x_a = 0$ where $a = \max\{1 \leq i < b \text{ s.t. } x_i = 0 \wedge \delta_x(i) - 1 = \delta_x(b)\}$. Condition *(ii)* implies that $\mathcal{L}_{\mathsf{Dyck}}$ is closed under concatenation.

Given a balanced string x, we say that x is *minimal* if it cannot be non-trivially factorized into multiple balanced strings. E.g., 110100 is minimal, while 010011 is not minimal since it can be factorized into 01 and 0011.

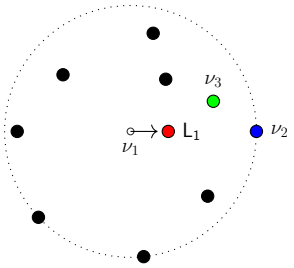
7.4 Algorithm

We present our algorithm solving `Universal Dancing` for a swarm $\mathcal{R}$ of $n \geq 3$ robots that satisfies all the hypotheses presented in Section 7.2. In particular, we exhibit an algorithm with $k \geq 4$ colors which allows $\mathcal{R}$ to perform any feasible choreography $\mathcal{S} = (\Pi_0, \ldots, \Pi_{q-1})^x$ with $x \in \{1, \infty\}$ that satisfies $q \leq \binom{n+k-7}{k-4}$.

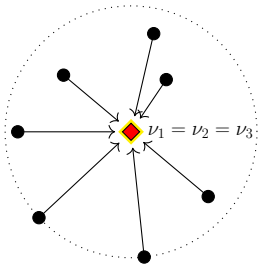
The core of our algorithm consists of three phases, namely Phase 1, Phase 2, and Phase 3, which are repeated cyclically until $\mathcal{S}$ ends. For each $\Pi_i \in \mathcal{S}$, Phase 1 aims at setting the chiral angle; Phase 2 aims at making robots form Π_i; Phase 3 aims at updating the counter value. If the $\mathcal{S}$ is periodic, then the algorithm execution never ends. Before entering the loop of the three phases, we need an initial phase Phase 0 to set up the three leader robots. See Figure 7.4 for the pseudo-code of the algorithm and a simple performance illustration.

7.4.1 Phase 0 - Election of leaders

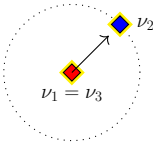
This phase starts from the initial configuration $\mathcal{C}$ (where all the robots are `OFF`-colored) and terminates when the three leaders have elected themselves by setting the dedicated colors. No robot moves during Phase 0. If $\mathcal{C}_{\lvert\mathbb{R}^2} \notin$ `ThreePoints`, the first activated



(A) L_1 moves from ν_1 to cover the last vertex of $\Pi \in \mathscr{P}(\Pi_0)$.



(B) The chiral angle degenerate in ν_1 where all the non-leaders gather to form a `Point`.



(C) ν_1 never changes. L_2 now fixes a different ν_2, and thus a new supporting circle for Π_2.

FIGURE 7.4: Performance of the choreography (Π_0, Π_1, Π_2), where $\Pi_0 \in (\texttt{NPoints} \setminus \texttt{NLine})$, $\Pi_1 \in \texttt{Point}$, and $\Pi_2 \in \texttt{TwoPoints}$. The supporting circle changes after the formation of Π_1.

`OFF` robot lying on $SEC(\mathcal{C})$ sets its color to L_2. Otherwise, an `OFF` robot lying on one of the longest edges of the triangle sets to L_2. Let us see which `OFF` robot becomes L_1: if $\mathcal{C}_{|\mathbb{R}^2} \in \texttt{Point}$, the next activated `OFF` robot sets its color to L_1. If $\mathcal{C}_{|\mathbb{R}^2} \in (\texttt{TwoPoints} \cup \texttt{ThreePoints})$, the `OFF` robot lying on the other endpoint of the (longest) edge of $\mathcal{C}_{|\mathbb{R}^2}$ sets to L_1, so that $pos(\mathsf{L}_1) \neq pos(\mathsf{L}_2)$. Otherwise, the `OFF` robot closest to $\omega(\mathcal{C})$ on a distinct position from $pos(\mathsf{L}_2)$ becomes L_1. Note that, in this phase, L_2 always lies on $SEC(\mathcal{C})$, while $pos(\mathsf{L}_1) = pos(\mathsf{L}_2)$ iff $\mathcal{C}_{|\mathbb{R}^2} \in \texttt{Point}$.

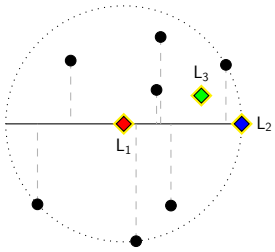
Lastly, the next activated `OFF` robot turns its color to L_3, thus completing the three leaders' setting. When no ambiguity arises, we denote with L_1, L_2, and L_3 both the colors and the related leader robots. All the other $n-3$ non-leaders are `OFF`, thus the counter value is 0: now the loop can start with the formation of Π_0 (i.e., the first pattern of the choreography).

7.4.2 Phase 1 - Chiral angle setup

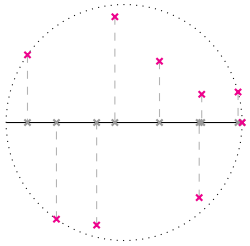
Let $\mathcal{C}$ be the current configuration. Let the current counter vector be $X_i = (x_1, \ldots, x_{k-3})$, where x_j represents the number of robots of color $\ell_j \in \mathfrak{L} \setminus \{\mathsf{L}_1, \mathsf{L}_2, \mathsf{L}_3\}$, for $j \in \{1, 2, 3\}$. This means that i is the counter's value and that Π_i is the ongoing pattern to be formed in the choreography. Phase 1 starts when Π_i is not formed by the swarm and the three leaders are not located on the chiral angle related to $\mathcal{C}$ and Π_i. In this phase, the three leaders L_1, L_2, L_3 compute the chiral angle $\triangle\nu_1\nu_2\nu_3$ as explained in Section 7.3.3, and sequentially reach the relative vertices. Specifically, L_1 moves to $\omega(\mathcal{C})$ if $\mathcal{C}_{|\mathbb{R}^2} \in \texttt{NPoints}$, otherwise it stays still since it is already on ν_1. Then, L_2 calculates the position of ν_2 for Π_i, and moves to ν_2. Note that, if $\mathcal{C}_{|\mathbb{R}^2} \notin \texttt{Point}$, $\nu_2 = pos(\mathsf{L}_2)$; otherwise, all the robots lies in $(0, 0)$ (since all have an egocentric coordinate system) and L_2 is required to move in a distinct position, conventionally to $(1, 0)$ (thus, the request is satisfied even if L_2 is stopped before reaching its point $(1, 0)$). Lastly, L_3 calculates and moves to ν_3, completing the chiral angle $\triangle\nu_1\nu_2\nu_3$ for Π_i. No other robot moves in this phase.

7.4.3 Phase 2 - Pattern formation

This phase starts when the leaders have formed the chiral angle for Π_i and ends as soon as the swarm forms the pattern $\Pi \in \mathscr{P}(\Pi_i)$ which is unequivocally defined by the chiral angle. Let r be an activated non-leader robot. Thanks to the counter's value of the swarm, r detects the pattern Π_i to be formed. If Π_i is not formed and the triangle $\triangle\mathsf{L}_1\mathsf{L}_2\mathsf{L}_3$ constitutes a valid chiral angle for Π_i, then r knows it is in Phase 2. Thus, r



(A) Non-leader robots and their projections on D.



(B) Projections of the target vertices for the non-leaders.

FIGURE 7.5: Phase 2, case $\Pi \in (\texttt{NPoints} \setminus \texttt{NLine})$.

detects the actual pattern Π on whose vertices robots must arrange themselves. Let us distinguish the two cases:

Case $\Pi \notin \texttt{NPoints}$

If $\Pi \in \texttt{Point}$ (thus, $\nu_1 = \nu_2 = \nu_3$), then r reaches the gathering point where the leaders lie. If $\Pi \in (\texttt{TwoPoints} \cup \texttt{ThreePoints})$, then if r lies on a (un)saturated multivertex of Π, it stays still. Otherwise, it reaches the closest unsaturated multivertex of Π. The leaders do not move, since they already lie on their target vertices.

Case $\Pi \in \texttt{NPoints}$

Let $\mathcal{C}$ be the current configuration, where L_1, L_2, and L_3 form a non-degenerate chiral angle. Let Ω be the supporting circle for Π, centered in L_1 and such that L_2 belongs to its boundary. We remind that, in this case, L_2 and L_3 already lie on their target vertices of Π by construction, thus they will not move during Phase 2. In this phase, all non-leaders reach their target vertices of Π, and subsequently, L_1 possibly moves to the last missing vertex left free. Note that L_1 already lies on its target vertex (and thus it will not move in this phase) only if ν_1 is a vertex of Π. If $\nu_1 \notin \Pi$, then, robots must properly agree on the target (multi)vertex to be left free for L_1: notably, they select the closest multivertex to L_1 (in case of equidistance, the choice is made following the orientation given by the chiral angle).

Let D be the *main diameter*, i.e., the diameter of Ω starting from L_2. Given a vertex v of Π, we indicate with $v_\perp$ its projection on D. We call the segment $\overline{vv_\perp}$ the *line projection* of v. Let $\Pi_\perp = \{\!\{v_\perp \text{ s.t. } v \in \Pi\}\!\}$. This case is handled in four sub-phases: *(i)* all non-leaders move to D perpendicularly, *(ii)* they shift along D to arrange on the vertices of $\Pi_\perp$ (except for the three vertices which are intended for the leaders), *(iii)* they move perpendicularly w.r.t. D to reach the corresponding target vertices of Π, and *(iv)* L_1 reaches the last missing vertex of Π.

Note that this strategy (i.e., projecting Π on D and making robots arrange on $\Pi_\perp$ before reaching Π) guarantees a constant bound on the maximum distance traveled by each robot so that the number of epochs remains independent of the number of robots, even under non-rigid movements. Referring to Figure 7.5, we explain the four sub-phases in detail:

Migration towards D. All the $m = n - 3$ non-leader robots reach D moving along a perpendicular trajectory (see Figure 7.5a)

Arrangement on D. Once all the m non-leaders have reached D, they compute $\Pi_\perp$ and remove from it the projections of the three vertices intended for the leaders.

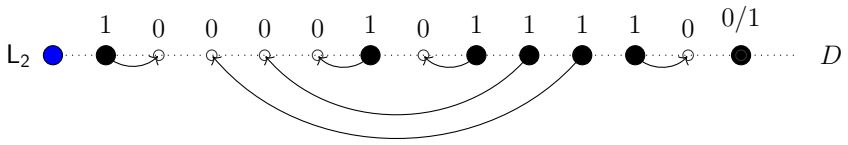


FIGURE 7.6: Here, $w = 100001011110$ is factorized in $x = 10$, $y = 00010111$, and $z = 10$. x^R, y, and z^R are Dyck words. The rightmost robot and the projection where it lies are not included in w.

In this sub-phase, the non-leaders move along D to reach the projection in $\Pi_\perp$ of their target vertex. They use the following matching algorithm to detect their target projection.

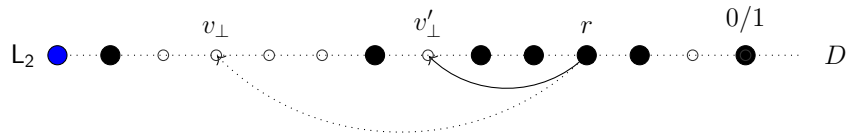
Let r be an activated non-leader on D. It computes the Boolean string $w \in \{0,1\}^{\leq 2m}$ that represents the ordered arrangement of non-leaders and projections along D, starting from the closest to L_2 (see Figure 7.6). In particular, each 1 represents a non-leader that does not lie on a projection, while each 0 represents a projection not covered by a non-leader. Note that multiplicities (multivertices, resp.) on D are treated by unrolling and representing them through factors of adjacent 1s (0s, resp.). Then, w can be factorized uniquely into minimal-length factors, such that they have the same number of 0s and 1s (see Lemma 61). Let $x = x_1, \ldots, x_h$ be such a factor of w. Then, x has this property: $x_1 = 1 - x_h$ (see Lemma 60). So, consider x if $x_1 = 0$, otherwise consider x^R (i.e., its reverse); for simplicity, we will refer to x in both cases. Thus, x is a Dyck word. So, if $x_i = 1$ corresponds to r, then r must reach the projection represented by the corresponding opening 0, say $v_\perp$.

Note that during the movement towards $v_\perp$, r may be stopped while it is crossing a (previously) uncovered projection, say $v'_\perp$ (see Figure 7.7a). By construction, at the next round, the corresponding 1 and 0 related to r and $v'_\perp$ are removed from w, and the matching algorithm can restart from the new arrangement string (see Figure 7.7b). Otherwise, it may happen that, during its trajectory towards $v_\perp$, r is stopped on a non-vertex point[12] (see Figure 7.8a). In this case, the new arrangement string and thus the new matching can change (see Figure 7.8b). However, the new matching always ensures that the new intended projection for r is in the same direction as $v_\perp$ (thus, r does not have to retrace its trajectory backwards).

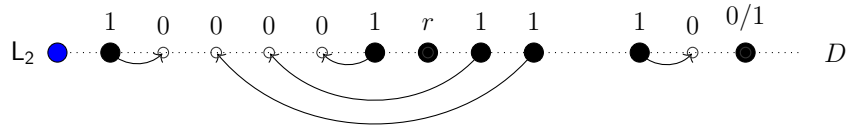
Back to the vertices. This sub-phase starts as soon as all the non-leader robots have arranged themselves on the vertex projections on D, and ends when all non-leader robots have reached the corresponding vertices traveling perpendicularly w.r.t. D.

Let us explain the invariant that each robot r can evaluate to understand that this sub-phase is in progress. Thanks to the counter (kept by the non-leaders) and the chiral-angle (kept by the leaders), r detects Π_i (the pattern of the choreography), the supporting circle Ω_i, and the actual pattern Π to be formed (so that $\Omega_i = SEC(\Pi)$). If Π is formed, then Phase 2 is terminated. Otherwise, for each vertex v of Π not intended for a leader, r checks that a distinct non-leader robot lies on $\overline{vv_\perp}$. Assume r is a non-leader robot lying on D, which has positively checked the invariant for this sub-phase. Then it unambiguously chooses the closest uncovered vertex[13] whose projection corresponds to its current position on D, and for which there does not yet exist a robot moving towards it. Note that multiple vertices (belonging to both the semi-circles cut by D) can have the same projections; in this case, r unambiguously

[12] Note that this case also comprehends the case when r stops on a multivertex (over)saturated.
[13] Indeed, it can belong to an unsaturated multivertex.

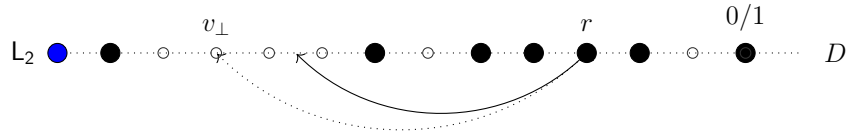


(A) $w = 100001011110$. Trajectory computed by r (dotted) vs. actual trajectory of r stopped by the adversary (solid).

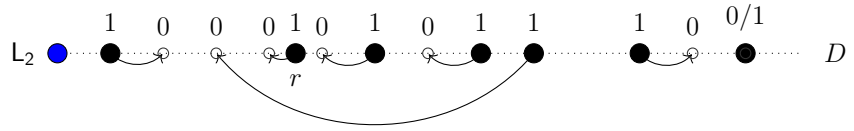


(B) New arrangement string $w = 1000011110$ and new robot-projection matching after the match of r with $v'_\perp$.

FIGURE 7.7: Before (a) and after (b) the non-rigid movement of r, which is stopped on a vertex (i.e., $v'_\perp$) different from the indented one (i.e., $v_\perp$).



(A) $w = 100001011110$. Trajectory computed by r (dotted) vs. actual trajectory of r stopped by the adversary in a non-vertex position (solid).



(B) New arrangement string $w = 100010101110$ and new robot-projection matching.

FIGURE 7.8: Before (a) and after (b) the non-rigid movement of r, which is stopped on a non-vertex point.

elects the target vertex by considering the clockwise orientation of the plane. After the vertex choice, r starts traveling perpendicularly to D towards the computed vertex.

Robot r may be stopped before reaching its vertex. If r is a non-leader robot not lying on D, then it evaluates the sub-phase invariant. Let h be the segment perpendicular to D which starts from D, contains r, and ends at Ω_i. Let $v_1, \ldots, v_s$ be the list of vertices along h so that $dist(v_i, D) \geq dist(v_{i+1}, D)$. Then r moves to v_j where $1 \leq j \leq s$ is the greatest index such that v_j is not covered and r lies on the segment $[v_j, v_{j_\perp})$.

L_1 towards its vertex. Once all the $m - 3$ robots have reached their target vertices, the last missing vertex, say w, must be covered by L_1. Let $\mathcal{C}$ be the current configuration: remind that L_1 lies on $\omega(\mathcal{C})$. By construction, w has been selected for L_1 among the closest to $\omega(\mathcal{C})$. If $\omega(\mathcal{C}) = w$, then L_1 does not move since it already covers the last vertex, and the pattern is formed. Otherwise, L_1 moves to w. Assume L_1 is stopped before reaching w, and let $\mathcal{C}'$ be the resulting configuration. Thus, any activated non-leader robot observes that L_1 does not lie on $\omega(\mathcal{C}')$ and that the swarm does not form Π: thus, it does nothing. As soon as L_1 is reactivated, it recomputes w and moves there.

7.4.4 Phase 3 - Counter update

Let $\mathcal{C}$ be the current configuration and i be the counter value, which points to the pattern Π_i of the choreography $\mathcal{S} = (\Pi_0, \ldots, \Pi_{q-1})^x$. This phase starts when $\mathcal{C}_{|\mathbb{R}^2} = \Pi$, where $\Pi \in \mathscr{P}(\Pi_i)$ is the pattern defined by the chiral angle (detectable even after the movement of $\mathsf{L_1}$). Let the current counter vector be $X_i = (x_1, \ldots, x_{k-3})$, where x_j represents the number of robots of color $\ell_j \in \mathfrak{L} \setminus \{\mathsf{L_1}, \mathsf{L_2}, \mathsf{L_3}\}$, for $j \in \{1, \ldots k-3\}$. We distinguish three actions:

- **Termination.** If $i = q-1$ and $x = 1$, then the swarm has terminated the choreography, and then it does not update the counter: this causes the swarm to freeze in Π_{q-1}.
- **Increment.** If $i < q-1$, then the swarm must increment the counter from i to $i+1$. Let X_{i+1} be the next vector for counter $i+1$. From the Gray code construction, there are exactly two differences between X_i and X_{i+1} corresponding to one decrement value and one increment value. Let a and b be the two indices that change between X_i and X_{i+1}. There must be x_a robots with color ℓ_a. When a robot with color ℓ_a gets activated and realizes that pattern Π_i is formed, it changes its color to ℓ_b. Now the counter vector corresponds to $i+1$, and all robots realize that the pattern Π_{i+1} is not formed. Phase 1 starts again with the movement of $\mathsf{L_1}$, $\mathsf{L_2}$, and $\mathsf{L_3}$ to form the chiral angle.
- **Resetting.** If $i = q-1$ and $x = \infty$, then the swarm must reset the counter to the initial value X_0, i.e., where all the non-leader robots are `OFF`-colored. However, since a swarm passing from X_{q-1} to X_0 under `SEQ` creates other counter vectors and thus may create unambiguous configurations, the swarm must mark this resetting phase through an unequivocal configuration. Specifically, we make the leader $\mathsf{L_3}$ set its color to $\mathsf{L_2}$; in this way, the presence of two $\mathsf{L_2}$ robots signals non-leader robots that they have to reset the counter. Once all non-leader robots have turned into `OFF`, one of the two $\mathsf{L_2}$ robots must turn into $\mathsf{L_3}$. If it is possible to establish which of the two $\mathsf{L_2}$ robots was the former $\mathsf{L_3}$ (e.g., if one $\mathsf{L_2}$ does not lie on $SEC(\mathcal{C})$), then leaders maintain their former roles.

7.4.5 Putting it all together

The whole algorithm is listed in Algorithm 11.

The following theorem summarizes our contribution. All the proofs are in Section 7.5.

Theorem 47. *The* `Universal Dancing` *problem* $\mathfrak{D}_U = ((\mathbb{R}^2)^n, (\mathbb{R}^2 \times \{\texttt{OFF}\})^n, \phi)$ *where* ϕ *only requires that any choreography has length/period* $q \leq \binom{n+k-7}{k-4}$ *for* $k \geq 4$*, can be solved under* $\mathcal{LUMI}^{\texttt{SEQ}}$ *by a swarm of* n *robots with* k *colors, even assuming non-rigid movements.*

7.5 Proofs

We here prove the correctness of our algorithm and we analyze its complexity in terms of runtime and colors.

7.5.1 Correctness of Phase 1

Lemma 58. *Given a pattern* Π_i *to be formed, the construction of the chiral angle* $\triangle\nu_1\nu_2\nu_3$ *as explained in Section 7.3.3 unequivocally defines one pattern in* $\mathscr{P}(\Pi_i)$.

Algorithm 11: `Universal Dancing` algorithm.

Input: A swarm $\mathcal{R} = \{r_1, \ldots, r_n\}$ which has to perform $\mathcal{S} = (\Pi_0, \ldots, \Pi_{q-1})^{x \in \{1,\infty\}}$
Phase 0
  Input: $\mathcal{C}$ initial configuration with n `OFF` robots on $\mathbb{R}^2$
  Color setting of the leaders: L_2, L_1, and L_3;
while true **do**
  $i \leftarrow$ value of counter;
  $\Pi_i \leftarrow$ pattern to be formed;
  Phase 1
    Input: $\mathcal{C}$ current configuration
    if $\mathcal{C}_{|\mathbb{R}^2} \notin \mathscr{P}(\Pi_i)$ **then**
      $\mathsf{L}_1, \mathsf{L}_2, \mathsf{L}_3$ set the chiral angle $\triangle\nu_1\nu_2\nu_3$ for Π_i;
  $\Pi \leftarrow$ pattern in $\mathscr{P}(\Pi_i)$ defined by $\triangle\nu_1\nu_2\nu_3$;
  Phase 2
    Input: $\mathcal{C}$ current configuration
    if $\mathcal{C}_{|\mathbb{R}^2} \neq \Pi$ and $\mathsf{L}_1, \mathsf{L}_2, \mathsf{L}_3$ form $\triangle\nu_1\nu_2\nu_3$ for Π_i **then**
      if $\Pi \notin$ `NPoints` **then**
        r reaches the closest unsaturated multivertex of Π;
      else
        $D \leftarrow$ main diameter;
        $\mathcal{R}_{NL} = \mathcal{R} \setminus \{\mathsf{L}_1, \mathsf{L}_2, \mathsf{L}_3\}$ non-leader robots;
        $\mathcal{R}_{NL}$ migrates on D perpendicularly;
        $\mathcal{R}_{NL}$ arranges on the vertex projections on D;
        $\mathcal{R}_{NL}$ reaches the target vertices moving perpendicularly to D;
        L_1 reaches the last missing vertex;
  Phase 3
    if $\mathcal{C}_{|\mathbb{R}^2} = \Pi$ **then**
      if $i = q - 1$ **then**
        if $x = \infty$ **then**
          L_3 sets its color to L_2;
          $\mathcal{R}_{NL}$ sets the counter to 0;
          resetting of L_3;
      else
        $\mathcal{R}_{NL}$ sets the counter to $i + 1$;

Proof. Let us prove the statement for any case:

- If $\Pi_i \in$ (`Point` $\cup$ `TwoPoints`), the proof follows straightforwardly from the definition of chiral angle. In fact, if $\Pi_i \in$ `Point`, then $\nu_1 = \nu_2 = \nu_3$ unequivocally defines the gathering point for the robots. If $\Pi_i \in$ `TwoPoints`, then the chiral angle corresponds to a degenerate triangle with $\nu_1 = \nu_3 \neq \nu_2$, thus these two points correspond to the gathering points. If the two multivertices of Π_i have different cardinalities, then ν_1 corresponds to the multivertex with the greatest cardinality.

- If $\Pi_i = \{(v_1, \#_1), (v_2, \#_2), (v_3, \#_3)\} \in$ `ThreePoints`, then $\triangle\nu_1\nu_2\nu_3 \in \mathscr{P}(\Pi_i)$ by construction. Let us now prove that the cardinalities of the multivertices of Π_i are unequivocally assigned to the points ν_1, ν_2, ν_3. Assume $dist(v_1, v_2) \geq dist(v_1, v_3) \geq dist(v_2, v_3)$, with $\#_1 \geq \#_2$, and with $\#_1 \geq \#_3$ if $dist(v_1, v_3) = dist(v_1, v_2)$. Note that such a ranking of the multivertices of Π_i is unambiguous. Then $\overline{\nu_1\nu_2}$ corresponds to the longest edge $\overline{v_1v_2}$ where the cardinality of ν_1

will not be less than the cardinality of ν_2. If Π_i is equilateral or isosceles, then the cardinality of ν_1 will not be less than the cardinality of ν_3. These constraints unequivocally define the pattern $\Pi = \{(\nu_1, \#_1), (\nu_2, \#_2), (\nu_3, \#_3)\}$ with $dist(\nu_1, \nu_2) \geq dist(\nu_1, \nu_3) \geq dist(\nu_2, \nu_3)$.

- If $\Pi_i = \{(v_1, \#_1), \ldots, (v_m, \#_m)\} \in \texttt{NPoints}$, then $\triangle\nu_1\nu_2\nu_3$ is a triangle with three distinct vertices by construction, with $dist(\nu_1, \nu_2) \geq dist(\nu_1, \nu_3)$, and such that $\{\nu_1, \nu_2, \nu_3\}$ are aligned if and only if $\Pi_i \in \texttt{NLine}$. We show that it defines a unique similarity transformation $\tau : \Pi_i \to \Pi$, where thus $\Pi \in \mathscr{P}(\Pi_i)$. Remember that ν_1 and ν_2 define the supporting circle for Π, i.e., its smallest enclosing circle, where ν_1 is the center and $\overline{\nu_1\nu_2}$ corresponds to its radius. Thus:
  - ν_1 fixes the *translation* component of τ.
  - By construction ν_2 must belong to the image under τ of (one of) the class $[v_{\iota_1}]_{\equiv}$ that lies on $SEC(\Pi_i)$. Hence $dist(\nu_1, \nu_2)$ uniquely fixes the non-zero *scale* of τ.
  - The leader $\mathsf{L_3}$ selects unambiguously a multivertex of Π_i and moves to it. If $\Pi_i \notin \texttt{NLine}$ then such a multivertex is chosen so that it is not aligned with $\mathsf{L_1}$ and $\mathsf{L_2}$. Thus, the oriented segment $\overline{\nu_1\nu_3}$ therefore fixes the *rotation* (and hence the chirality) of τ. If $\Pi_i \in \texttt{NLine}$, no rotation is needed.

  The three conditions above determine a unique similarity transformation τ and therefore a unique pattern $\Pi = \tau(\Pi_i)$. No other choice of Π is admissible without violating at least one of the selection rules (radius, multivertex class, or oriented order), so $\triangle\nu_1\nu_2\nu_3$ identifies exactly one representative of $\mathscr{P}(\Pi_i)$.

□

7.5.2 Correctness of Phase 2

Let $\Pi \in \mathscr{P}(\Pi_i)$ be the actual pattern to be formed during Phase 2.

Lemma 59. *During Phase 2,* Π *is invariant.*

Proof. By Lemma 58, Π is unequivocally defined by the chiral angle, which is in turn defined by the positions of the leaders $\mathsf{L_1}$, $\mathsf{L_2}$, $\mathsf{L_3}$. During Phase 2, the leaders do not move, except for $\mathsf{L_1}$ when it moves from $\omega(\mathcal{C})$ to reach its target vertex, if $\Pi_i \in \texttt{NPoints}$. Thus, if $\Pi_i \notin \texttt{NPoints}$, $\triangle\mathsf{L_1L_2L_3}$ defines the chiral angle. Otherwise, $\triangle\omega(\mathcal{C})\mathsf{L_2L_3}$ defines the chiral angle. This results in Π remaining invariant. □

Lemma 60 ([14])**.** *Let* $w = w_1 \ldots w_m$ *be a minimal Boolean balanced string with* $m \geq 1$*. Then,* $w_1 = 1 - w_m$*.*

Proof. By contradiction, assume $w_1 = w_m$. Assume $w_1 = w_m = 0$ (the proof for $w_1 = w_m = 1$ is similar). We know that w is a minimal balanced string; thus, for any x non-trivial prefix of w, we have that $|x|_0 > |x|_1$ (otherwise, w could be factorized into balanced strings). However, since $w_m = 0$, then $w_1 \ldots w_{m-1}$ is a non-trivial prefix of w containing more 1s than 0s. Contradiction achieved. □

Lemma 61. *A balanced Boolean string* $w \neq \varepsilon$ *can be factorized in an unambiguous way into minimal-length non-null factors so that each one is either a Dyck word or the reverse of a Dyck word.*

[14] For completeness, we prove the following properties on Boolean words, even if probably known.

Proof. Lemma 32 already proves the unique factorization of Boolean strings into minimal-length factors. Let us prove now that such factors are (the reverse of) Dyck words.

Let x be a factor of w, resulting from this process. Let us consider the case when x starts with 0 (i.e., the opening bracket). We have to prove that x is a Dyck word. We already know that it is balanced. We here prove that, for each non-trivial prefix of x, the number of 0s is $>$ than the number of 1s, which is a sufficient condition to state that x is a Dyck word. By contradiction, let us assume that instead there exists a non-trivial prefix of x for which the number of 0s is $\leq$ than the number of 1s. Thus, there is a non-trivial prefix of x which is balanced. This contradicts the hypothesis that x is a minimal-length non-null balanced prefix of w.

If x starts with 1 (i.e., the closing bracket), then we know that x^R starts with 0 by Lemma 60. Thus, applying the same argument, we state that x^R is a Dyck word. □

Lemma 62. *Let $x = x_1 \dots x_m$ be a Dyck word, and let $x_a = 0$ be matched with $x_b = 1$, with $1 \leq a < b \leq m$. Let $a+1 \leq c \leq b-1$. Let x' be the string obtained by moving x_b from the b-th position to the c-th position, and shifting all the other symbols, i.e.,*

$$x'_i = \begin{cases} x_i & \text{if } i < c \wedge i > b \\ x_b & \text{if } i = c \\ x_{i-1} & \text{if } c < i \leq b. \end{cases}$$

Then, x' is a Dyck word. Moreover, the matched 0 of $x'_c = 1$ is contained in the factor $x'_a \cdots x'_{c-1}$.

Proof. Indeed, x' is balanced. If $x' = x$, then trivially the property is true. Otherwise, we have to prove that $\delta_{x'}(i) \geq 0$ for any i. To lighten the notation, we use $\delta(\cdot)$ in place of $\delta_x(\cdot)$, and $\delta'(\cdot)$ in place of $\delta_{x'}(\cdot)$. We have that:

$$\delta'(i) = \begin{cases} \delta(i) & \text{if } i < c \wedge i > b \\ \delta(i-1) - 1 & \text{if } c \leq i \leq b \end{cases} \tag{7.1}$$

Note that, since x_a and x_b are matching brackets in x, then $\delta(a) = \delta(b) + 1 > 0$, and it holds that $\delta(i) > \delta(a) - 1$ for any $a < i < b$ (otherwise, x_a and x_b would not match). This proves that $\delta' \geq 0$ in all the cases.

We have now to prove that x'_c (which is the moved 1) is matched with a 0 in $x'_a \cdots x'_{c-1}$. By contradiction, assume that x'_c is matched with a 0 in the prefix $x'_1 \cdots x'_{a-1}$. Thus, $x'_a = 0$ is matched with a 1 in $x'_{a+1} \cdots x'_{c-1}$. However, since $x'_1 \cdots x'_{c-1} = x_1 \cdots x_{c-1}$, such a matching would have existed also in x. Contradiction achieved. □

Lemma 63. *Let r be a robot lying at a point $a \in D$ during Phase 2. Assume r is activated and has to reach its target projection $v \in D$. Assume r is stopped during its trajectory at position a' in the open segment $\overline{va}$ (i.e., $a' \in \overline{va} \setminus \{v, a\}$). Then, the (possibly new) target projection of r belongs to the closed segment $\overline{va'}$.*

Proof. If a' corresponds to a free vertex, then a' becomes the new target projection of r by construction (thus r must stay still). Otherwise, we have to prove that, according to the new arrangement string, robot r is assigned to a vertex (thus 0) in the substring corresponding to the segment $\overline{va'}$. Let $w = w_1 \cdots w_m$ be the arrangement string before the movement of r, which thus matches a with v, and let $w = y^{(1)} \cdots y^{(h)}$ be the factorization in minimal balanced strings, with $h \geq 1$. Let $y^{(i)}$ be the minimal balanced

factor that contains a and v. W.l.o.g., assume that $y^{(i)}$ is a Dyck word (otherwise, consider its reverse, by Lemma 61). Let y' be the new balanced factor, obtained after the movement of r to a' (note that the other factors are not affected by this movement). By Lemma 62, we know that y' is a Dyck word, and that the 1 corresponding to a' is matched to a 0 in the factor corresponding to $\overline{va'}$. □

Lemma 64. *No ambiguity arises if* L_1 *is stopped by the adversary while traveling during Phase 1 (towards the center of the configuration) or during Phase 2 (towards its target vertex).*

Proof. Suppose that L_1 is stopped by the adversary before reaching its target destination. Let $\mathcal{C}_{|\mathbb{R}^2} \in$ `NPoints` be this transitory configuration (i.e., where the swarm does not form Π_i and L_1 does not lie on $\omega(\mathcal{C})$). Note that a configuration with these properties can occur both during Phase 1 when the leaders are moving to form the chiral angle, and during Phase 2 when L_1 is moving to reach the last vertex of the pattern, say w.

Note also that, since the w for L_1 is chosen from the closest vertices of $\Pi \in \mathscr{P}(\Pi_i)$ to $\omega(\mathcal{C})$, then L_1 can form a multiplicity during its trajectory $\overline{w\omega(\mathcal{C})}$ only when it stops on w and w is a multivertex with cardinality > 1. Thus, when L_1 is stopped before reaching w (end of Phase 2) or before reaching $\omega(\mathcal{C})$ (beginning of Phase 1), it does not form multiplicities.

We now prove that the two situations are unambiguous. If any other robot (but L_1) is activated in $\mathcal{C}$, it will see that L_1 does not lie on $\omega(\mathcal{C})$, thus, it will stay still by construction. As soon as L_1 is reactivated, it computes the counter i, and it must understand if it was moving towards $\omega(\mathcal{C})$ or towards the last vertex of the similar pattern to Π_i. For this purpose, it computes the chiral angle $\triangle\nu_1\nu_2\nu_3$ where $\nu_1 = \omega(\mathcal{C})$, and thus the actual pattern Π to be formed. If the other robots of the swarm cover all the vertices of Π, and only one vertex w is uncovered, and the open line segment $\overline{w\mathsf{L}_1}$ is robot-free, then L_1 is aware that it must head to w (Phase 2). Otherwise, it heads to $\omega(\mathcal{C})$ in order to form the chiral angle for the new pattern (Phase 1). □

7.5.3 Runtime complexity

Runtime complexity is measured in epochs and, since we consider non-rigid movements, it depends on the non-rigidity parameter δ. We indicate with $\mathcal{T}(\cdot)$ the number of epochs taken to accomplish a given phase of the algorithm. We denote the radius of a circle with $\varrho(\cdot)$.

Lemma 65. *For every pattern* $\Pi_i \in \mathcal{S}$, $\mathcal{T}(Phase0) \leq 3$.

Proof. During Phase 0, no robot moves, and only three `OFF` robots must elect as leaders and change their colors properly. Thus, this phase lasts at most three epochs, assuming, for simplicity, that leaders elect themselves in sequence (i.e., L_2, L_1, and L_3). □

Lemma 66. *Let* $\mathcal{C}$ *be the configuration from which Phase 1 starts for a pattern* $\Pi_i \in \mathcal{S}$. *If* $\mathcal{C}_{|\mathbb{R}^2} \in$ `Point`, *let* $\mathcal{C}'$ *be the configuration obtained after the movement of* L_2. *Let* Ω *be* $SEC(\mathcal{C})$ *if* $\mathcal{C}_{|\mathbb{R}^2} \in$ `NPoints`, *otherwise let it be the circle centered in* $pos(\mathsf{L}_1)$ *and whose radius is* $dist(\mathsf{L}_1, \mathsf{L}_2)$ *if* $\mathcal{C}, \mathcal{C}' \in$ (`TwoPoints` $\cup$ `ThreePoints`). *Then,*

$$\mathcal{T}(Phase1) \leq \begin{cases} \frac{3\varrho(\Omega)}{\delta} & \text{if } \Pi_i \in \texttt{Point} \\ \frac{2\varrho(\Omega)}{\delta} & \text{if } \Pi_i \notin \texttt{Point}. \end{cases} \tag{7.2}$$

Proof. We remind that during Phase 1 the leaders $\mathsf{L_1}$, $\mathsf{L_2}$, and $\mathsf{L_3}$ move to form the chiral angle $\triangle\nu_1\nu_2\nu_3$ for defining the pattern $\Pi \in \mathscr{P}(\Pi_i)$ to be formed. By construction, ν_1 always corresponds to the center of Ω. Let us consider the four cases:

- If $\Pi_i \in \texttt{Point}$, then $\nu_1 = \nu_2 = \nu_3 = \omega(\Omega)$. Note that $\mathcal{C}_{|\mathbb{R}^2} \notin \texttt{Point}$, otherwise the pattern is already formed. Thus, the maximum traveled distance occurs when all the leaders must reach $\omega(\Omega)$, by traveling each one a distance $\varrho(\Omega)$: in total, this case takes $\frac{3\varrho(\Omega)}{\delta}$ epochs.
- If $\Pi_i \notin \texttt{Point}$, the worst case is when both $\mathsf{L_1}$ and $\mathsf{L_3}$ must reach ν_1 ν_3, each one traveling the maximum distance $\varrho(\Omega)$. Thus, the worst case takes $\frac{2\varrho(\Omega)}{\delta}$ epochs.

□

Lemma 67. *Let $\mathcal{C}$ be the configuration at the beginning of Phase 2, where $\triangle\nu_1\nu_2\nu_3$ is the chiral angle for Π_i. Let Ω be the circle having center in ν_1 and whose radius is $\max_{r\in\mathcal{R}}\{dist(\nu_1, r)\}$. Then,*

$$\mathcal{T}(Phase2) \leq \begin{cases} \frac{\varrho(\Omega)}{\delta} & \text{if } \Pi_i \in \texttt{Point} \\ \frac{2\varrho(\Omega)}{\delta} & \text{if } \Pi_i \in (\texttt{TwoPoints} \cup \texttt{ThreePoints}) \\ \frac{5\varrho(\Omega)}{\delta} & \text{if } \Pi_i \in \texttt{NPoints} \end{cases}$$

Proof. Note that Ω corresponds to the supporting circle of Π_i if and only if $\Pi_i \notin \texttt{Point}$. If $\Pi_i \in \texttt{Point}$, all the robots must gather on ν_1, thus in the worst case each robot travels a distance $\varrho(\Omega)$. If $\Pi_i \in (\texttt{TwoPoints} \cup \texttt{ThreePoints})$, the maximum distance traveled by a robot corresponds to the diameter of Ω. Otherwise ($\Pi_i \in \texttt{NPoints}$), the algorithm makes all non-leader robots arrange on the $n-3$ vertices of the pattern in three sub-phases, and eventually makes $\mathsf{L_1}$ reach the last missing vertex. In the three sub-phases, the non-leaders reach the D (maximum traveled distance $\varrho(\Omega)$), then they arrange on D covering the vertices' projections (maximum traveled distance $2\varrho(\Omega)$), and lastly they travel perpendicularly to reach their target vertices (maximum traveled distance $\varrho(\Omega)$). Eventually, $\mathsf{L_1}$ reaches the last vertex (maximum distance $\varrho(\Omega)$). Thus, if $\Pi_i \in \texttt{NPoints}$, the phase takes $\frac{5\varrho(\Omega)}{\delta}$ epochs. □

Lemma 68. $\mathcal{T}(Phase3) \leq 3$.

Proof. During Phase 3, no robot moves. Let $\mathcal{S} = (\Pi_0, \ldots, \Pi_{q-1})^x$ be the choreography the swarm is performing. Let $i \in \{0, \ldots, q-1\}$ be the index encoded by the swarm through the Gray code. If $x = 1$ or if $x = \infty$ and $i < q-1$, then this phase takes only one epoch. In fact, at most one robot has to change its color to update the counter value from i to $i+1$.

Otherwise, it takes one epoch for $\mathsf{L_3}$ to set to $\mathsf{L_2}$, one for all $n-3$ non-leaders to set to $\texttt{OFF}$, and the last one for the resetting of $\mathsf{L_3}$. □

7.5.4 Spatial homogeneity

Our strategy for constructing the supporting circles is to guarantee a sort of spatial homogeneity in the performance of $\mathcal{S} = (\Pi_0, \ldots, \Pi_{q-1})^x$. Notably,

Lemma 69. *Let $\Omega_0, \Omega_1, \Omega_2, \ldots$ be the sequence of supporting circles formed along the performance of $\mathcal{S}$, so that Ω_j is the supporting circle for $\Pi_{j \pmod q}$. Then,*

- *$\Omega_0, \Omega_1, \Omega_2, \ldots$ are all concentric;*

- *given any subsequence* $\Omega_a, \Omega_{a+1}, \ldots, \Omega_b$ *such that* $\Pi_{i \pmod q} \notin$ `Point` *for any* $i \in [a, b]$*, then* $\Omega_a, \Omega_{a+1}, \ldots, \Omega_b$ *are equal.*

Proof. Let us prove both the claims:

- Let ω be the center of Ω_0. If $\Pi_0 \in$ `NPoints`, then the pattern $\Pi \in \mathscr{P}(\Pi_0)$ will be formed so that $SEC(\Pi) = \Omega_0$, and once formed, L_1 will come back to ω to form the (the vertex ν_1 of the) chiral angle for Π_1. Instead, if $\Pi_0 \notin$ `NPoints`, L_1 will never move for the formation of the pattern Π_0, and $pos(\mathsf{L}_1)$ constitutes a vertex of Π_0. Thus, in both cases, the next supporting circle Ω_1 will be centered in ω as well. Recursively, the claim holds for the entire subsequence.

- By construction, the supporting circle of the pattern $\Pi_{a \pmod q} \notin$ `Point` is non-degenerate, and it is given by $pos(\mathsf{L}_1)$ (center) and $pos(\mathsf{L}_2)$ (radius) after the setting of the corresponding chiral angle. After the formation of $\Pi_{a \pmod q}$, the leaders must form the chiral angle (and thus the supporting circle) for $\Pi_{a+1 \pmod q}$. By the first claim, we know that Ω_{a+1} has the same center as Ω_a. Since L_2 only moves when the swarm must form a `Point`, then it results that the two circles have the same radius. Recursively, the claim holds for the entire subsequence.

□

7.5.5 Number of colors

The algorithm here proposed uses $k \geq 4$ colors: three are used by the leaders (i.e., $\mathsf{L}_1, \mathsf{L}_2, \mathsf{L}_3$) while the remaining $k' = k - 3$ colors are used by the $n - 3$ non-leader robots to implement the counter. Thus, exploiting the technique of Gray codes as explained in Section 7.3.1, the number of possible composition vectors is $\binom{n-3+k'-1}{k'-1} = \binom{n+k-7}{k-4}$. This allows to perform any choreography of lenght/period $q \leq \binom{n+k-7}{k-4}$.

7.6 Conclusions

This chapter has formally defined the `Universal Dancing` problem, and presented an algorithm to solve it, thus making a swarm of robots perform (periodic or finite) sequences of any type of pattern, starting from any initial configuration. For its solution, it was necessary to consider the $\mathcal{LUMI}^{\texttt{SEQ}}$ model, knowing that, under the `FSYNCH` mode, `Universal Dancing` is unsolvable. Notably, we have exhibited that the `SEQ` mode ensures the swarm to implement a distributed counter mechanism, which has been crucial for our algorithm design, and of independent interest for other contexts. Indeed, this result highlights the computational power of `SEQ` robots, concerning their ability to perform choreographies. We have proved that the number of robots and available colors define a bound on the length/period of the feasible choreographies. Moreover, our algorithm guarantees a sort of spatial homogeneity to the performed choreography, even considering completely disoriented robots and the non-rigid setting.

Two possible future research lines can follow: the first can define other classes of `Dancing` problems, and investigate under which assumptions they are solved. Secondly, further work may deeply investigate the distinctive power of robots under sequential schedulers for other problems.

Summary and Outlook

This thesis contributes to the ongoing investigation of distributed computing by mobile robots under the *Look-Compute-Move* model through a twofold approach: analyzing the computational power of different LCM (sub-)models (Part I) and designing algorithmic solutions to solve some problems for these systems (Part II).

In Part I, we examine how specific model features can affect the capability of robots in solving some tasks, thus comparing their computational power under different combinations of model assumptions. A general focus is given to the settings of robots' memory and communication means (i.e., $\mathcal{OBLOT}$, $\mathcal{FSTA}$, $\mathcal{FCOM}$, and $\mathcal{LUMI}$ base models), and on their synchronization mode (mainly `FSYNCH`, `SSYNCH`, and `ASYNCH`).

Then, Chapter 2 focuses on the visibility capability of robots, comparing models of transparent and opaque robots, where the latter ones suffer from obstructed visibility. Chapter 3 focuses on three modes of sequential schedulers (i.e., `SEQ`., `PERM`, and `RROBIN`). In both chapters, several open questions are reported and are worthy to be investigated: we mention the unknown computational relation between the models of opaque robots $\mathcal{LUMI}_{\mathcal{O}}^{\texttt{ASYNCH}}$ and $\mathcal{LUMI}_{\mathcal{O}}^{\texttt{SSYNCH}}$, and the possibility of dropping the assumption of variable disorientation in the relation between $\mathcal{FCOM}^{\texttt{PERM}}$ and $\mathcal{FCOM}^{\texttt{SEQ}}$.

Another research line could aim to unify and combine the model assumptions considered in the previous studies, and, for example, study the behavior of opaque robots under sequential schedulers. This cross-analysis would help to identify and isolate the specific factors that influence the computational power of different models.

Finally, Chapter 4 considers crash-prone models, and studies under which model assumptions the problems Fault Detection and Fault Identification can be reliably solved by a swarm. Being a novel research line, this chapter may pave the way to explore fault-recovery solutions in these models.

In Part II, we address the study on the computing capability of mobile robots by developing algorithmic solutions for two problems, namely `Uniform Circle Formation` and `Universal Dancing`, under specific models. Notably, we provide two solutions for `Uniform Circle Formation` under the $\mathcal{LUMI}_{\mathcal{O}}^{\texttt{ASYNCH}}$ model: the first one is a $O(\log n)$-time algorithm provided in Chapter 5, while the second one is a $O(1)$-time algorithm presented in Chapter 6. Indeed, some algorithmic strategies used in these chapters to enable asynchronous opaque robots to coordinate by using lights may offer insights into the open question about the computational difference or equivalence between $\mathcal{LUMI}_{\mathcal{O}}^{\texttt{ASYNCH}}$ and $\mathcal{LUMI}_{\mathcal{O}}^{\texttt{SSYNCH}}$.

In Chapter 7, the computational power of $\mathcal{LUMI}^{\texttt{SEQ}}$ robots, previously studied in Chapter 3, is further explored and exploited with respect to their ability to implement a distributed counter. This capability is fundamental for designing an algorithm that allows robots to form sequences of patterns, i.e., to perform choreographies, and its potential portability to other contexts also merits further investigation.

As observed, the two approaches undertaken in this thesis—the first one studying the computational power of selected models, the second one studying algorithmic solutions for selected problems—have several points of contact, which thus enable mutual contributions. In this context, further research could establish and explore these interactions in greater depth.

Thanks to

My deepest gratitude goes to Profs. Beatrice Palano and Carlo Mereghetti, who, throughout these years, have been much more than just academic supervisors. I am especially thankful for their trust and for introducing me to the fascinating field of mobile robots, long before it became my PhD research project. To them, I owe my love for theoretical computer science, and the appreciation for the beauty of formalism.

I am also sincerely grateful to Profs. Paola Flocchini and Nicola Santoro for welcoming me to Ottawa and for the opportunity to collaborate with their research group. Their guidance and enthusiasm showed me how much work lies behind a research direction, and how much still lies ahead.

A special thanks goes to Dr. Debasish Pattanayak, for his invaluable help on how to survive the Canadian winter and, above all, for the many fruitful discussions on algorithms and beyond.

I would also like to thank Prof. Gokarna Sharma, for reaching out while we were in Tokyo, for sharing ideas, and for proposing a successful research collaboration.

I am grateful to Prof. Masafumi Yamashita for our meetings in Ottawa and for his availability and passion for research. It was truly an honor to meet him and attend his explanations.

Among the colleagues I met in Ottawa, I would also like to thank Prof. Giuseppe Prencipe, with whom I had the pleasure of collaborating.

I am grateful to the former master's students, Stefano Clemente and Lucia Mambretti, whom I had the opportunity to co-supervise during their thesis projects, which led to co-authored publications. In particular, I thank Stefano for his trust and the engaging discussions. A special mention to Lucia: besides the incredible exchange of ideas, your presence and support have been of fundamental importance in these three years.

To Andre and Chiara, for always pretending not to take me too seriously. It has been the best support I could ever have. This work is the kite I promised.

Bibliography

[AGR25] Rusul J. Alsaedi, Joachim Gudmundsson, and André van Renssen. "Pattern formation for fat robots with memory". In: *Computational Geometry* 129 (2025), p. 102189.

[AP06] Noa Agmon and David Peleg. "Fault-Tolerant Gathering Algorithms for Autonomous Mobile Robots". In: *SIAM Journal on Computing* 36.1 (2006), pp. 56–82.

[Ash+19] Yotam Ashkenazi, Shlomi Dolev, Sayaka Kamei, Fukuhito Ooshita, and Koichi Wada. "Forgive & Forget: Self-Stabilizing Swarms in Spite of Byzantine Robots". In: *Proc. of 7th International Symposium on Computing and Networking (CANDAR)*. IEEE, 2019, pp. 188–194.

[ASY95] H. Ando, I. Suzuki, and M. Yamashita. "Formation and agreement problems for synchronous mobile robots with limited visibility". In: *Proc. of 10th International Symposium on Intelligent Control (ISIC)*. 1995, pp. 453–460.

[Aug+13] Cédric Auger, Zohir Bouzid, Pierre Courtieu, Sébastien Tixeuil, and Xavier Urbain. "Certified Impossibility Results for Byzantine-Tolerant Mobile Robots". In: *Proc. of 15th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*. Vol. 8255. LNCS. Springer, 2013, pp. 178–190.

[Bal+19] Thibaut Balabonski, Amélie Delga, Lionel Rieg, Sébastien Tixeuil, and Xavier Urbain. "Synchronous Gathering without Multiplicity Detection: a Certified Algorithm". In: *Theory of Computing Systems* 63.2 (2019), pp. 200–218.

[BCM18] Subhash Bhagat, Sruti Gan Chaudhuri, and Krishnendu Mukhopadhyaya. "Gathering of Opaque Robots in 3D Space". In: *Proc. of 19th International Conference on Distributed Computing and Networking (ICDCN)*. ACM, 2018, 2:1–2:10.

[BDL20] Quentin Bramas, Stéphane Devismes, and Pascal Lafourcade. "Infinite Grid Exploration by Disoriented Robots". In: *Proc. of 8th International Conference on Networked Systems (NETYS)*. Vol. 12129. LNCS. Springer, 2020, pp. 129–145.

[BDS08] Eduardo Mesa Barrameda, Shantanu Das, and Nicola Santoro. "Deployment of Asynchronous Robotic Sensors in Unknown Orthogonal Environments". In: *Algorithmic Aspects of Wireless Sensor Networks, 4th International Workshop ALGOSENSORS*. Vol. 5389. LNCS. Springer, 2008, pp. 125–140.

[BDS21] Kaustav Bose, Archak Das, and Buddhadeb Sau. "Pattern Formation by Robots with Inaccurate Movements". In: *Proc. of 25th International Conference on Principles of Distributed Systems (OPODIS)*. Vol. 217. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, 10:1–10:20.

[BLD23] Quentin Bramas, Pascal Lafourcade, and Stéphane Devismes. "Optimal exclusive perpetual grid exploration by luminous myopic opaque robots with common chirality". In: *Theoretical Computer Science* 977 (2023), p. 114162.

[BLT23] Quentin Bramas, Anissa Lamani, and Sébastien Tixeuil. "Stand up indulgent gathering". In: *Theoretical Computer Science* 939 (2023), pp. 63–77. ISSN: 0304-3975.

[BM18] Subhash Bhagat and Krishnendu Mukhopadhyaya. "Optimum Circle Formation by Autonomous Robots". In: *Advances in Intelligent Systems and Computing* 666 (2018), 153 – 165.

[BMR25] Subhash Bhagat, Krishnendu Mukhopadhyaya, and Rajarshi Ray. "The min-move mutual visibility problem for disoriented asynchronous robots". In: *Theoretical Computer Science* 1035 (2025), p. 115125.

[Bos+18] Kaustav Bose, Manash Kumar Kundu, Ranendu Adhikary, and Buddhadeb Sau. "Optimal Gathering by Asynchronous Oblivious Robots in Hypercubes". In: *Proc. of 14th International Symposium on Algorithms and Experiments for Wireless Sensor Networks (ALGOSENSORS)*. Vol. 11410. LNCS. Springer, 2018, pp. 102–117.

[Bos+19] Kaustav Bose, Ranendu Adhikary, Manash Kumar Kundu, and Buddhadeb Sau. "Positional Encoding by Robots with Non-rigid Movements". In: *Proc. of 26th International Colloquium of Structural Information and Communication Complexity (SIROCCO)*. Vol. 11639. Springer, 2019, pp. 94–108.

[Bos+21] Kaustav Bose, Manash Kumar Kundu, Ranendu Adhikary, and Buddhadeb Sau. "Arbitrary pattern formation by asynchronous opaque robots with lights". In: *Theoretical Computer Science* 849 (2021), pp. 138–158.

[Bra+24a] Quentin Bramas, Hirotsugu Kakugawa, Sayaka Kamei, Anissa Lamani, Fukuhito Ooshita, Masahiro Shibata, and Sébastien Tixeuil. "Stand-Up Indulgent Gathering on Lines for Myopic Luminous Robots". In: *Proc. of 38th International Conference on Advanced Information Networking and Applications (AINA)*. Vol. 200. Springer, 2024, pp. 110–121.

[Bra+24b] Quentin Bramas, Sayaka Kamei, Anissa Lamani, and Sébastien Tixeuil. "Stand-up indulgent gathering on lines". In: *Theoretical Computer Science* 1016 (2024), p. 114796.

[Bra+25] Quentin Bramas, Hirotsugu Kakugawa, Sayaka Kamei, Anissa Lamani, Fukuhito Ooshita, Masahiro Shibata, and Sébastien Tixeuil. "A Visibility vs. Memory Trade-Off for Stand-Up Indulgent Gathering on Lines". In: *International Colloquium On Structural Information and Communication Complexity (SIROCCO)*. Vol. 15671. LNCS. Springer, 2025, pp. 159–175.

[BT15] Quentin Bramas and Sébastien Tixeuil. "Wait-Free Gathering Without Chirality". In: *Proc. of 22nd International Colloquium On Structural Information and Communication Complexity (SIROCCO)*. Vol. 9439. LNCS. Springer, 2015, pp. 313–327.

[BT16] Quentin Bramas and Sébastien Tixeuil. "Probabilistic Asynchronous Arbitrary Pattern Formation (Short Paper)". In: *Proc. of 18th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*. Vol. 10083. LNCS. 2016, pp. 88–93.

[BT17] Quentin Bramas and Sébastien Tixeuil. "The Random Bit Complexity of Mobile Robots Scattering". In: *International Journal of Foundations of Computer Science* 28.2 (2017), pp. 111–117.

[Buc+21] Kevin Buchin, Paola Flocchini, Irina Kostitsyna, Tom Peters, Nicola Santoro, and Koichi Wada. "Autonomous Mobile Robots: Refining the Computational Landscape". In: *Proc. of 35th International Parallel and Distributed Processing Symposium Workshops (IPDPS)*. IEEE, 2021, pp. 576–585.

[Buc+25] Kevin Buchin, Paola Flocchini, Irina Kostitsyna, Tom Peters, Nicola Santoro, and Koichi Wada. "On the computational power of energy-constrained mobile robots". In: *Information and Computation* 303 (2025), p. 105280.

[Can+16] Davide Canepa, Xavier Dèfago, Taisuke Izumi, and Maria Potop-Butucaru. "Flocking with Oblivious Robots". In: *Proc. of 18th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*. 2016, pp. 94–108.

[Cas+22] Jannik Castenow, Jonas Harbig, Daniel Jung, Peter Kling, Till Knollmann, and Friedhelm Meyer auf der Heide. "A Unifying Approach to Efficient (Near)-Gathering of Disoriented Robots with Limited Visibility". In: *Proc. of 26th International Conference on Principles of Distributed Systems (OPODIS)*. Vol. 253. LIPIcs. 2022, 15:1–15:25.

[CDSN19a] Serafino Cicerone, Gabriele Di Stefano, and Alfredo Navarra. "Asynchronous Arbitrary Pattern Formation: The Effects of a Rigorous Approach". In: *Distributed Computing* 32.2 (2019), pp. 91–132.

[CDSN19b] Serafino Cicerone, Gabriele Di Stefano, and Alfredo Navarra. "Embedded Pattern Formation by Asynchronous Robots without Chirality". In: *Distributed Computing* 32.4 (2019), pp. 291–315.

[CF02] James D. Currie and D. Sean Fitzpatrick. "Circular Words Avoiding Patterns". In: *Proc. of 6th International Conference on Developments in Language Theory (DLT)*. Vol. 2450. LNCS. Springer, 2002, pp. 319–325.

[CF25] Stefano Clemente and Caterina Feletti. "Fault Detection and Identification by Autonomous Mobile Robots". In: *Proc. of 4th Symposium on Algorithmic Foundations of Dynamic Networks (SAND)*. Vol. 330. LIPIcs. 2025, 10:1—10:20.

[Cic+25] Serafino Cicerone, Alessia Di Fonso, Gabriele Di Stefano, Alfredo Navarra, and Francesco Piselli. "Mutual and total mutual visibility in hypercube-like graphs". In: *Applied Mathematics and Computation* 491 (2025), p. 129216.

[Cie+12] Mark Cieliebak, Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro. "Distributed Computing by Mobile Robots: Gathering". In: *SIAM Journal on Computing* 41.4 (2012), pp. 829–879.

[Cle+10] Julien Clement, Xavier Défago, Maria Gradinariu Potop-Butucaru, Taisuke Izumi, and Stephane Messika. "The cost of probabilistic agreement in oblivious robot networks". In: *Information Processing Letters* 110.11 (2010), pp. 431–438. ISSN: 0020-0190.

[Cle24] Stefano Clemente. "Faulty robots: a study of their computational power". MA thesis. Milan, Italy: Università degli Studi di Milano, 2024.

[CM15] Sruti Gan Chaudhuri and Krishnendu Mukhopadhyaya. "Leader Election and Gathering for Asynchronous Fat Robots without Common Chirality". In: *Journal of Discrete Algorithms* 33 (2015), pp. 171–192.

[CMN04] Ioannis Chatzigiannakis, Michael Markou, and Sotiris E. Nikoletseas. "Distributed Circle Formation for Anonymous Oblivious Robots". In: *Proc. of 3rd International Workshop on Experimental and Efficient Algorithms (WEA)*. Vol. 3059. LNCS. Springer, 2004, pp. 159–174.

[Col+20] Jared Ray Coleman, Evangelos Kranakis, Oscar Morales-Ponce, Jaroslav Opatrny, Jorge Urrutia, and Birgit Vogtenhuber. "Minimizing The Maximum Distance Traveled To Form Patterns With Systems of Mobile Robots". In: *Proc. of 32nd Canadian Conference on Computational Geometry (CCCG)*. 2020, pp. 73–79.

[Cor+11] Andreas Cord-Landwehr, Bastian Degener, Matthias Fischer, Martina Hüllmann, Barbara Kempkes, Alexander Klaas, Peter Kling, Sven Kurras, Marcus Märtens, Friedhelm Meyer auf der Heide, Christoph Raupach, Kamil Swierkot, Daniel Warner, Christoph Weddemann, and Daniel Wonisch. "Collisionless Gathering of Robots with an Extent". In: *Proc. of 37th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM)*. Vol. 6543. LNCS. Springer, 2011, pp. 178–189.

[Cou+16] Pierre Courtieu, Lionel Rieg, Sébastien Tixeuil, and Xavier Urbain. "Certified Universal Gathering in R^2 for Oblivious Mobile Robots". In: *Proc. of 30th International Symposium on Distributed Computing (DISC)*. 2016, pp. 187–200.

[CP02] Mark Cieliebak and Giuseppe Prencipe. "Gathering Autonomous Mobile Robots". In: *Proc. of 9th International Colloquium on Structural Information and Communication Complexity (SIROCCO)*. Vol. 13. Proceedings in Informatics. Carleton Scientific, 2002, pp. 57–72.

[CP07] Davide Canepa and Maria Gradinariu Potop-Butucaru. "Stabilizing Flocking Via Leader Election in Robot Networks". In: *Proc. of 9th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*. Vol. 4838. LNCS. Springer, 2007, pp. 52–66.

[CP08] Reuven Cohen and David Peleg. "Convergence of Autonomous Mobile Robots with Inaccurate Sensors and Movements". In: *SIAM Journal on Computing* 38.1 (2008), pp. 276–302.

[CS63] N. Chomsky and M.P. Schützenberger. "The Algebraic Theory of Context-Free Languages". In: *Computer Programming and Formal Systems*. Vol. 35. Studies in Logic and the Foundations of Mathematics. Elsevier, 1963, pp. 118–161.

[CSN15] Serafino Cicerone, Gabriele Di Stefano, and Alfredo Navarra. "MinMax-Distance Gathering on Given Meeting Points". In: *Proc. of 9th International Conference on Algorithms and Complexity (CIAC)*. Vol. 9079. LNCS. Springer, 2015, pp. 127–139.

[CSN21a] Serafino Cicerone, Gabriele Di Stefano, and Alfredo Navarra. "Gathering robots in graphs: The central role of synchronicity". In: *Theoretical Computer Science* 849 (2021), pp. 99–120.

[CSN21b] Serafino Cicerone, Gabriele Di Stefano, and Alfredo Navarra. "Solving the Pattern Formation by Mobile Robots With Chirality". In: *IEEE Access* 9 (2021), pp. 88177–88204.

[CT96] Tushar Deepak Chandra and Sam Toueg. "Unreliable Failure Detectors for Reliable Distributed Systems". In: *Journal of the ACM* 43.2 (1996), pp. 225–267.

[D'A+16] Gianlorenzo D'Angelo, Gabriele Di Stefano, Ralf Klasing, and Alfredo Navarra. "Gathering of robots on anonymous grids and trees without multiplicity detection". In: *Theoretical Computer Science* 610 (2016), pp. 158–168.

[Das+12] Shantanu Das, Paola Flocchini, Giuseppe Prencipe, Nicola Santoro, and Masafumi Yamashita. "The Power of Lights: Synchronizing Asynchronous Robots Using Visible Bits". In: *Proc. of 32nd International Conference on Distributed Computing Systems (ICDCS)*. IEEE Computer Society, 2012, pp. 506–515.

[Das+14] Shantanu Das, Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro. "Synchronized Dancing of Oblivious Chameleons". In: *Proc. of 7th International Conference on Fun with Algorithms (FUN)*. Vol. 8496. LNCS. Springer, 2014, pp. 113–124.

[Das+15] Shantanu Das, Paola Flocchini, Nicola Santoro, and Masafumi Yamashita. "Forming Sequences of Geometric Patterns with Oblivious Mobile Robots". In: *Distributed Computing* 28.2 (2015), pp. 131–145.

[Das+16] Shantanu Das, Paola Flocchini, Giuseppe Prencipe, Nicola Santoro, and Masafumi Yamashita. "Autonomous mobile robots with lights". In: *Theoretical Computer Science* 609 (2016), pp. 171–184.

[Das+19] Shantanu Das, Riccardo Focardi, Flaminia L. Luccio, Euripides Markou, and Marco Squarcina. "Gathering of robots in a ring with mobile faults". In: *Theoretical Computer Science* 764 (2019), pp. 42–60.

[Das+20] Shantanu Das, Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro. "Forming Sequences of Patterns With Luminous Robots". In: *IEEE Access* 8 (2020), pp. 90577–90597.

[Das+24] Archak Das, Satakshi Ghosh, Avisek Sharma, Pritam Goswami, and Buddhadeb Sau. "The Computational Landscape of Autonomous Mobile Robots: The Visibility Perspective". In: *Proc. of 20th International Conference on Distributed Computing and Intelligent Technology (ICDCIT)*. Vol. 14501. LNCS. Springer, 2024, pp. 85–100.

[Dat+13] Suparno Datta, Ayan Dutta, Sruti Gan Chaudhuri, and Krishnendu Mukhopadhyaya. "Circle Formation by Asynchronous Transparent Fat Robots". In: *Proc. of 9th International Conference on Distributed Computing and Internet Technology (ICDCIT)*. Vol. 7753. LNCS. Springer, 2013, pp. 195–207.

[D'E+18] Mattia D'Emidio, Gabriele Di Stefano, Daniele Frigioni, and Alfredo Navarra. "Characterizing the computational power of mobile robots on graphs and implications for the Euclidean plane". In: *Information and Computation* 263 (2018), pp. 57–74.

[Déf+06] Xavier Défago, Maria Gradinariu, Stéphane Messika, and Philippe Raipin Parvédy. "Fault-Tolerant and Self-stabilizing Mobile Robots Gathering". In: *Proc. of 20th International Symposium on Distributed Computing (DISC)*. Vol. 4167. LNCS. Springer, 2006, pp. 46–60.

[Déf+23] Xavier Défago, Adam Heriban, Sébastien Tixeuil, and Koichi Wada. "Using model checking to formally verify rendezvous algorithms for robots with lights in Euclidean space". In: *Robotics and Autonomous Systems* 163 (2023), p. 104378.

[DFN15] Mattia D'Emidio, Daniele Frigioni, and Alfredo Navarra. "Synchronous Robots vs Asynchronous Lights-Enhanced Robots on Graphs". In: *Electronic Notes in Theoretical Computer Science*. 16th Italian Conference on Theoretical Computer Science (ICTCS). Elsevier, 2015, pp. 169–180.

[DFN16] Mattia D'Emidio, Daniele Frigioni, and Alfredo Navarra. "Characterizing the Computational Power of Anonymous Mobile Robots". In: *Proc. of 36th International Conference on Distributed Computing Systems (ICDCS)*. IEEE Computer Society, 2016, pp. 293–302.

[Die+13] Yoann Dieudonné, Florence Levé, Franck Petit, and Vincent Villain. "Deterministic geoleader election in disoriented anonymous systems". In: *Theoretical Computer Science* 506 (2013), pp. 43–54.

[Die+19] Yoann Dieudonné, Shlomi Dolev, Franck Petit, and Michael Segal. "Explicit Communication Among Stigmergic Robots". In: *International Journal of Foundations of Computer Science* 30.2 (2019), pp. 315–332.

[Dij74] Edsger W. Dijkstra. "Self-stabilizing Systems in Spite of Distributed Control". In: *Communications of the ACM* 17.11 (1974), pp. 643–644.

[DK02] Xavier Défago and Akihiko Konagaya. "Circle formation for oblivious anonymous mobile robots with no common sense of orientation". In: *Proc. of 2nd ACM International Workshop on Principles of Mobile Computing (POMC)*. ACM, 2002, pp. 97–104.

[DM23] Bibhuti Das and Krishnendu Mukhopadhyaya. "Uniform k-Circle Formation by Fat Robots". In: *Proc. of 25th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*. Vol. 14310. LNCS. Springer, 2023, pp. 359–373.

[Dob+03] Stefan Dobrev, Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro. "Multiple Agents RendezVous in a Ring in Spite of a Black Hole". In: *Proc. of 7th International Conference on Principles of Distributed Systems (OPODIS)*. Vol. 3144. LNCS. Springer, 2003, pp. 34–46.

[DP07a] Yoann Dieudonné and Franck Petit. "Circle formation of weak robots and Lyndon words". In: *Information Processing Letters* 101.4 (2007), pp. 156–162.

[DP07b] Yoann Dieudonné and Franck Petit. "Swing Words to Make Circle Formation Quiescent". In: *Proc. of 14th International Colloquium on Structural Information and Communication Complexity (SIROCCO)*. Springer, 2007, pp. 166–179.

[DP08] Yoann Dieudonné and Franck Petit. "Squaring the Circle with Weak Mobile Robots". In: *19th International Symposium on Algorithms and Computation (ISAAC)*. Vol. 5369. LNCS. Springer, 2008, pp. 354–365.

[DP09] Yoann Dieudonné and Franck Petit. "Scatter of robots". In: *Parallel Processing Letters* 19.1 (2009), pp. 175–184.

[DP12] Yoann Dieudonné and Franck Petit. "Self-Stabilizing Gathering with Strong Multiplicity Detection". In: *Theoretical Computer Science* 428 (2012), pp. 47–57.

[DPBT19] Xavier Défago, Maria Potop-Butucaru, and Sébastien Tixeuil. "Fault-Tolerant Mobile Robots". In: *Chapter 10 of [FPS19a]*. Springer, 2019, pp. 234–251.

[DPP20] Xavier Défago, Maria Potop-Butucaru, and Philippe Raipin Parvédy. "Self-stabilizing gathering of mobile robots under crash or Byzantine faults". In: *Distributed Computing* 33.5 (2020), pp. 393–421.

[DPT13] Stéphane Devismes, Franck Petit, and Sébastien Tixeuil. "Optimal probabilistic ring exploration by semi-synchronous oblivious robots". In: *Theoretical Computer Science* 498 (2013), pp. 10–27. ISSN: 0304-3975.

[DPV10] Yoann Dieudonné, Franck Petit, and Vincent Villain. "Leader Election Problem versus Pattern Formation Problem". In: *Proc. of 24th International Symposium on Distributed Computing (DISC)*. Vol. 6343. 2010, pp. 267–281.

[DS08] Xavier Défago and Samia Souissi. "Non-Uniform Circle Formation Algorithm for Oblivious Mobile Robots with Convergence toward Uniformity". In: *Theoretical Computer Science* 396.1-3 (2008), pp. 97–112.

[DS19] Shantanu Das and Nicola Santoro. "Moving and Computing Models: Agents". In: *Chapter 2 of [FPS19a]*. Springer, 2019, pp. 15–34.

[EP07] Asaf Efrima and David Peleg. "Distributed Models and Algorithms for Mobile Robot Systems". In: *Proc. of 33rd International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM)*. Vol. 4362. LNCS. Springer, 2007, pp. 70–87.

[Fel+22] Caterina Feletti, Carlo Mereghetti, Beatrice Palano, and Priscilla Raucci. "Uniform Circle Formation for Fully Semi-, and Asynchronous Opaque Robots with Lights". In: *Proc. of 23rd Italian Conference on Theoretical Computer Science (ICTCS)*. Vol. 3284. CEUR-WS.org, 2022, pp. 207–221. URL: https://ceur-ws.org/Vol-3284/8511.pdf.

[Fel+24] Caterina Feletti, Lucia Mambretti, Carlo Mereghetti, and Beatrice Palano. "Computational Power of Opaque Robots". In: *Proc. of 3rd Symposium on Algorithmic Foundations of Dynamic Networks (SAND)*. Vol. 292. LIPIcs. 2024, 13:1–13:19.

[Fel+25a] Caterina Feletti, Paola Flocchini, Debasish Pattanayak, Giuseppe Prencipe, and Nicola Santoro. "Brief Announcement: Luminous Robots under Sequential Schedulers: Universal Dancing". In: *Proc. of 39th International Symposium on Distributed Computing (DISC)*. Vol. 356. LIPIcs. 2025, 56:1–56:7.

[Fel+25b] Caterina Feletti, Paola Flocchini, Debasish Pattanayak, Giuseppe Prencipe, and Nicola Santoro. *Universal Dancing by Luminous Robots under Sequential Schedulers*. 2025. arXiv: 2508.15484 [cs.DC].

[Fel+25c] Caterina Feletti, Lucia Mambretti, Carlo Mereghetti, and Beatrice Palano. "Computational power of autonomous robots: Transparency vs. opaqueness". In: *Theorerical Computer Science* 1036 (2025), p. 115153.

[FFS25] Caterina Feletti, Paola Flocchini, and Nicola Santoro. "On the Computational Power of Mobile Robots under Sequential Schedulers". In: *Proc. of 27th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*. Vol. 16350. LNCS. Springer Nature Switzerland, 2025.

[Flo+05] Paola Flocchini, Giuseppe Prencipe, Nicola Santoro, and Peter Widmayer. "Gathering of asynchronous robots with limited visibility". In: *Theoretical Computer Science* 337.1-3 (2005), pp. 147–168.

[Flo+08] Paola Flocchini, Giuseppe Prencipe, Nicola Santoro, and Peter Widmayer. "Arbitrary Pattern Formation by Asynchronous, Anonymous, Oblivious Robots". In: *Theoretical Computer Science* 407.1 (2008), pp. 412–447.

[Flo+17] Paola Flocchini, Giuseppe Prencipe, Nicola Santoro, and Giovanni Viglietta. "Distributed Computing by Mobile Robots: Uniform Circle Formation". In: *Distributed Computing* 30.6 (2017), pp. 413–457.

[Flo19] Paola Flocchini. "Gathering". In: *Chapter 4 of [FPS19a]*. Springer, 2019, pp. 63–82.

[Flo+23] Paola Flocchini, Nicola Santoro, Yuichi Sudo, and Koichi Wada. "On Asynchrony, Memory, and Communication: Separations and Landscapes". In: *Proc. of 27th International Conference on Principles of Distributed Systems (OPODIS)*. Vol. 286. LIPIcs. 2023, 28:1–28:23.

[Flo25] Paola Flocchini. "Distributed Computing by Mobile Robots: Exploring the Computational Landscape (Extended Abstract)". In: *Proc. of 50th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM)*. Vol. 15538. LNCS. Springer, 2025, pp. 3–7.

[Flo+25] Paola Flocchini, Alfredo Navarra, Debasish Pattanayak, Francesco Piselli, and Nicola Santoro. "Oblivious Robots Under Sequential Schedulers: Universal Pattern Formation". In: *Proc. 32nd International Colloquium of Structural Information and Communication Complexity (SIROCCO)*. Vol. 15671. LNCS. Springer, 2025, pp. 297–314.

[Flo+99] Paola Flocchini, Giuseppe Prencipe, Nicola Santoro, and Peter Widmayer. "Hard Tasks for Weak Robots: The Role of Common Knowledge in Pattern Formation by Autonomous Mobile Robots". In: *Proc. of 10th International Symposium on Algorithms and Computation (ISAAC)*. 1999, pp. 93–102.

[FMP18] Caterina Feletti, Carlo Mereghetti, and Beatrice Palano. "Uniform Circle Formation for Swarms of Opaque Robots with Lights". In: *Proc. of 20th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*. Vol. 11201. LNCS. Springer, 2018, pp. 317–332.

[FMP23a] Caterina Feletti, Carlo Mereghetti, and Beatrice Palano. "$O(\log n)$-Time Uniform Circle Formation for Asynchronous Opaque Luminous Robots". In: *Proc. of 27th International Conference on Principles of Distributed Systems (OPODIS)*. Vol. 286. LIPIcs. 2023, 5:1–5:21.

[FMP23b] Caterina Feletti, Carlo Mereghetti, and Beatrice Palano. "Uniform Circle Formation for Fully, Semi-, and Asynchronous Opaque Robots with Lights". In: *Applied Sciences* 13.13 (2023). ISSN: 2076-3417.

[FPS19a] Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro, eds. *Distributed Computing by Mobile Entities, Current Research in Moving and Computing*. Vol. 11340. LNCS. Springer, 2019. ISBN: 978-3-030-11071-0.

[FPS19b] Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro. "Moving and Computing Models: Robots". In: *Chapter 1 of [FPS19a]*. Springer, 2019, pp. 3–14.

[FPS24a] Caterina Feletti, Debasish Pattanayak, and Gokarna Sharma. "Brief Announcement: Optimal Uniform Circle Formation by Asynchronous Luminous Robots". In: *Proc. of 38th International Symposium on Distributed Computing (DISC)*. Vol. 319. LIPIcs. 2024, 46:1–46:7.

[FPS24b] Caterina Feletti, Debasish Pattanayak, and Gokarna Sharma. *Optimal Uniform Circle Formation by Asynchronous Luminous Robots*. 2024. arXiv: 2405.06617 [cs.DC].

[Fra53] Gray Frank. *Pulse code communication*. US Patent 2,632,058. 1953.

[FSW19] Paola Flocchini, Nicola Santoro, and Koichi Wada. "On Memory, Communication, and Synchronous Schedulers When Moving and Computing". In: *Proc. of 23rd International Conference on Principles of Distributed Systems (OPODIS)*. Vol. 153. LIPIcs. 2019, 25:1–25:17.

[Fuj+10] Nao Fujinaga, Hirotaka Ono, Shuji Kijima, and Masafumi Yamashita. "Pattern Formation through Optimum Matching by Oblivious CORDA Robots". In: *Proc. of 14th International Conference on Principles of Distributed Systems (OPODIS)*. Vol. 6490. LNCS. Springer, 2010, pp. 1–15.

[FW24] Fabian Frei and Koichi Wada. "Brief Announcement: Distinct Gathering Under Round Robin". In: *Proc. of 38th International Symposium on Distributed Computing (DISC)*. Vol. 319. LIPIcs. 2024, 48:1–48:8. ISBN: 978-3-95977-352-2.

[Gin66] Seymour Ginsburg. *The mathematical theory of context free languages*. McGraw-Hill Book Company, 1966.

[GP04] Vincenzo Gervasi and Giuseppe Prencipe. "Coordination without communication: the case of the flocking problem". In: *Discrete Applied Mathematics* 144.3 (2004), pp. 324–344.

[HMU06] John Edward Hopcroft, Rajeev Motwani, and Jeffrey Ullman. *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*. USA: Addison-Wesley Longman Publishing Co., Inc., 2006. ISBN: 0321455363.

[HN16] László Hegedüs and Benedek Nagy. "On periodic properties of circular words". In: *Discrete Mathematics* 339.3 (2016), pp. 1189–1197.

[Izu+18a] Taisuke Izumi, Daichi Kaino, Maria Gradinariu Potop-Butucaru, and Tixeuil Sébastien. "On time complexity for connectivity-preserving scattering of mobile robots". In: *Theoretical Computer Science* 738 (2018), pp. 42–52.

[Izu+18b] Taisuke Izumi, Daichi Kaino, Maria Gradinariu Potop-Butucaru, and Sébastien Tixeuil. "On time complexity for connectivity-preserving scattering of mobile robots". In: *Theoretical Computer Science* 738 (2018), pp. 42–52.

[Kam+19] Sayaka Kamei, Anissa Lamani, Fukuhito Ooshita, Sébastien Tixeuil, and Koichi Wada. "Gathering on Rings for Myopic Asynchronous Robots With Lights". In: *Proc. of 23rd International Conference on Principles of Distributed Systems (OPODIS)*. Vol. 153. LIPIcs. 2019, 27:1–27:17.

[Kat05] Branislav Katreniak. "Biangular Circle Formation by Asynchronous Mobile Robots". In: *Proc. of 12th International Colloquium on Structural Information and Communication Complexity (SIROCCO)*. Vol. 3499. LNCS. Springer, 2005, pp. 185–199.

[Kir+24] David G. Kirkpatrick, Irina Kostitsyna, Alfredo Navarra, Giuseppe Prencipe, and Nicola Santoro. "On the power of bounded asynchrony: convergence by autonomous robots with limited visibility". In: *Distributed Computing* 37.3 (2024), pp. 279–308.

[Kli82] Paul Klingsberg. "A Gray Code for Compositions". In: *Journal of Algorithms* 3.1 (Mar. 1982), pp. 41–44. ISSN: 0196-6774.

[KT24] Sayaka Kamei and Sébastien Tixeuil. "An Asynchronous Maximum Independent Set Algorithm By Myopic Luminous Robots On Grids". In: *The Computer Journal* 67.1 (2024), pp. 57–77.

[LHP21] Shouwei Li, Friedhelm Meyer auf der Heide, and Pavel Podlipyan. "The impact of the Gabriel subgraph of the visibility graph on the gathering of mobile autonomous robots". In: *Theoretical Computer Science* 852 (2021), pp. 29–40.

[LS62] R.C. Lyndon and M.P. Schützenberger. "The Equation $a^M = b^N c^P$ in a Free Group". In: *Michigan Mathematical Journal* 9 (1962), pp. 289–298.

[Lun+14] Giuseppe Antonio Di Luna, Paola Flocchini, Federico Poloni, Nicola Santoro, and Giovanni Viglietta. "The Mutual Visibility Problem for Oblivious Robots". In: *Proc. of 26th Canadian Conference on Computational Geometry (CCCG)*. Carleton University, Ottawa, Canada, 2014. URL: http://www.cccg.ca/proceedings/2014/papers/paper51.pdf.

[Lun+17] Giuseppe Antonio Di Luna, Paola Flocchini, Sruti Gan Chaudhuri, Federico Poloni, Nicola Santoro, and Giovanni Viglietta. "Mutual visibility by luminous robots without collisions". In: *Information and Computation* 254 (2017), pp. 392–418.

[Mam23] Lucia Mambretti. "Distributed systems of mobile robots: a model-driven systematic study". MA thesis. Milan, Italy: Università degli Studi di Milano, 2023.

[MC18] Moumita Mondal and Sruti Gan Chaudhuri. "Uniform circle formation by mobile robots". In: *Proc. of 19th International Conference on Distributed Computing and Networking (ICDCN)*. ACM, 2018, 20:1–20:2.

[MC20] Moumita Mondal and Sruti Gan Chaudhuri. "Uniform Circle Formation by Swarm Robots Under Limited Visibility". In: *Proc. of 16th International Conference Distributed Computing and Internet Technology (ICDCIT)*. Vol. 11969. LNCS. Springer, 2020, pp. 420–428.

[MC22] Moumita Mondal and Sruti Gan Chaudhuri. "Uniform Scattering of Robots on Alternate Nodes of a Grid". In: *Proc. of 23rd International Conference on Distributed Computing and Networking (ICDCN)*. ACM, 2022, pp. 254–259.

[MMJ20] Anisur Rahaman Molla, Kaushik Mondal, and William K. Moses Jr. "Efficient Dispersion on an Anonymous Ring in the Presence of Weak Byzantine Robots". In: *Proc. of 16th International Symposium on Algorithms and Experiments for Wireless Sensor Networks (ALGOSENSORS)*. Vol. 12503. LNCS. Springer, 2020, pp. 154–169.

[MV16] Marcello Mamino and Giovanni Viglietta. "Square Formation by Asynchronous Oblivious Robots". In: *Proc. of 28th Canadian Conference on Computational Geometry (CCCG)*. Simon Fraser University, Canada, 2016, pp. 1–6.

[NOI23] Shota Nagahama, Fukuhito Ooshita, and Michiko Inoue. "Ring exploration of myopic luminous robots with visibility more than one". In: *Information and Computation* 292 (2023), p. 105036.

[NP25] Alfredo Navarra and Francesco Piselli. "Oblivious Robots Under Round Robin: Gathering on Rings". In: *19th International Joint Conference on Theoretical Computer Science – Frontier of Algorithmic Wisdom (IJTCS-FAW)*. Vol. 15828. Springer, 2025, pp. 166–180.

[NSW21] Rikuo Nakai, Yuichi Sudo, and Koichi Wada. "Asynchronous Gathering Algorithms for Autonomous Mobile Robots with Lights". In: *Proc. of 23rd International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*. Vol. 13046. LNCS. Springer, 2021, pp. 410–424.

[Oos+24] Fukuhito Ooshita, Naoki Kitamura, Ryota Eguchi, Michiko Inoue, Hirotsugu Kakugawa, Sayaka Kamei, Masahiro Shibata, and Yuichi Sudo. "Crash-Tolerant Perpetual Exploration with Myopic Luminous Robots on Rings". In: *Proc. of 28th International Conference on Principles of Distributed Systems (OPODIS)*. Vol. 324. LIPIcs. 2024, 12:1–12:16.

[OWD23] Takashi Okumura, Koichi Wada, and Xavier Défago. "Optimal L-algorithms for rendezvous of asynchronous mobile robots with external-lights". In: *Theoretical Computer Science* 979 (2023), p. 114198.

[PAM21] Debasish Pattanayak, John Augustine, and Partha Sarathi Mandal. "Randomized Gathering of Asynchronous Mobile Robots". In: *Theoretical Computer Science* 858 (2021), pp. 64–80.

[PAS21] Pavan Poudel, Aisha Aljohani, and Gokarna Sharma. "Fault-tolerant complete visibility for asynchronous robots with lights under one-axis agreement". In: *Theoretical Computer Science* 850 (2021), pp. 116–134.

[Pat+19] Debasish Pattanayak, Kaushik Mondal, Ramesh H., and Partha Sarathi Mandal. "Gathering of Mobile Robots with Weak Multiplicity Detection in Presence of Crash-Faults". In: *Journal of Parallel and Distributed Computing* 123 (Jan. 2019), pp. 145–155.

[Pat+20] Debasish Pattanayak, Klaus-Tycho Foerster, Partha Sarathi Mandal, and Stefan Schmid. "Conic Formation in Presence of Faulty Robots". In: *Proc. of 16th International Symposium on Algorithms and Experiments for Wireless Sensor Networks (ALGOSENSORS)*. Vol. 12503. LNCS. Springer, 2020, pp. 170–185.

[Pot+19] Maria Potop-Butucaru, Nathalie Sznajder, Sébastien Tixeuil, and Xavier Urbain. "Formal Methods for Mobile Robots". In: *Chapter 12 of [FPS19a]*. Springer, 2019, pp. 278–313.

[Pre19] Giuseppe Prencipe. "Pattern Formation". In: *Chapter 3 of [FPS19a]*. Springer, 2019, pp. 37–62.

[PS24] Debasish Pattanayak and Gokarna Sharma. "Time-Color Tradeoff on Uniform Circle Formation by Asynchronous Robots". In: *Proc. of 38th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2024, pp. 987–997.

[San07] Nicola Santoro. *Design and analysis of distributed algorithms*. Wiley series on parallel and distributed computing. Wiley, 2007. ISBN: 978-0-471-71997-7.

[SDY06] Samia Souissi, Xavier Défago, and Masafumi Yamashita. "Gathering Asynchronous Mobile Robots with Inaccurate Compasses". In: *Proc. of 10th International Conference on Principles of Distributed Systems (OPODIS)*. Vol. 4305. LNCS. Springer, 2006, pp. 333–349.

[Sha+16] Gokarna Sharma, Ramachandran Vaidyanathan, Jerry L. Trahan, Costas Busch, and Suresh Rai. "Complete Visibility for Robots with Lights in O(1) Time". In: *Proc. of 18th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*. Vol. 10083. LNCS. 2016, pp. 327–345.

[Sha+25] Avisek Sharma, Satakshi Ghosh, Pritam Goswami, and Buddhadeb Sau. "Move-optimal arbitrary pattern formation by mobile robots on rectangular grid using near-optimal spatial area". In: *Theoretical Computer Science* 1038 (2025), p. 115179.

[Shi+21] Masahiro Shibata, Masaki Ohyabu, Yuichi Sudo, Junya Nakamura, Yonghwan Kim, and Yoshiaki Katayama. "Gathering of seven autonomous mobile robots on triangular grids". In: *Proc. of 35th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2021, pp. 566–575.

[Shi81] Yossi Shiloach. "Fast Canonization of Circular Strings". In: *Journal of Algorithms* 2.2 (1981), pp. 107–121.

[SIW11] Samia Souissi, Taisuke Izumi, and Koichi Wada. "Oracle-based flocking of mobile robots in crash-recovery model". In: *Theoretical Computer Science* 412.33 (2011), pp. 4350–4360.

[SS96] Kazuo Sugihara and Ichiro Suzuki. "Distributed algorithms for formation of geometric patterns with many mobile robots". In: *Journal of Field Robotics* 13.3 (1996), pp. 127–139.

[SVT17] Gokarna Sharma, Ramachandran Vaidyanathan, and Jerry L. Trahan. "Constant-Time Complete Visibility for Asynchronous Robots with Lights". In: *Proc. of 19th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*. Vol. 10616. LNCS. Springer, 2017, pp. 265–281.

[SVT21] Gokarna Sharma, Ramachandran Vaidyanathan, and Jerry L. Trahan. "Constant-Time Complete Visibility for Robots with Lights: The Asynchronous Case". In: *Algorithms* 14.2 (2021), p. 56.

[SY99] Ichiro Suzuki and Masafumi Yamashita. "Distributed Anonymous Mobile Robots: Formation of Geometric Patterns". In: *SIAM Journal on Computing* 28.4 (1999), pp. 1347–1363.

[SYD08] Samia Souissi, Yan Yang, and Xavier Défago. "Fault-Tolerant Flocking in a k-Bounded Asynchronous System". In: *Proc. of 12th International Conference on Principles of Distributed Systems (OPODIS)*. Vol. 5401. LNCS. Springer, 2008, pp. 145–163.

[TS07] Andrew S. Tanenbaum and Maarten van Steen. *Distributed systems - principles and paradigms, 2nd Edition*. Pearson Education, 2007. ISBN: 978-0-13-239227-3.

[Vig19] Giovanni Viglietta. "Uniform Circle Formation". In: *Chapter 5 of [FPS19a]*. Springer, 2019, pp. 83–108.

[VS06] Vandi Verma and Reid G. Simmons. "Scalable robot fault detection and identification". In: *Robotics and Autonomous Systems* 54.2 (2006), pp. 184–191.

[VST22] Ramachandran Vaidyanathan, Gokarna Sharma, and Jerry Trahan. "On Fast Pattern Formation by Autonomous Robots". In: *Information and Computation* 285 (2022), p. 104699. ISSN: 0890-5401.

[Yam+17] Yukiko Yamauchi, Taichi Uehara, Shuji Kijima, and Masafumi Yamashita. "Plane Formation by Synchronous Mobile Robots in the Three-Dimensional Euclidean Space". In: *Journal of the ACM* 64.3 (2017), 16:1–16:43.

[Yan+09] Yan Yang, Samia Souissi, Xavier Défago, and Makoto Takizawa. "Fault-Tolerant Flocking of Mobile Robots with Whole Formation Rotation". In: *Proc. of 23rd International Conference on Advanced Information Networking and Applications (AINA)*. IEEE Computer Society, 2009, pp. 830–837.

[Yan+11] Yan Yang, Samia Souissi, Xavier Défago, and Makoto Takizawa. "Fault-tolerant flocking for a group of autonomous mobile robots". In: *Journal of Systems and Software* 84.1 (2011), pp. 29–36.

[YS10] Masafumi Yamashita and Ichiro Suzuki. "Characterizing Geometric Patterns Formable by Oblivious Anonymous Mobile Robots". In: *Theoretical Computer Science* 411.26-28 (2010), pp. 2433–2453.

[YY14a] Yukiko Yamauchi and Masafumi Yamashita. "Randomized Pattern Formation Algorithm for Asynchronous Oblivious Mobile Robots". In: (2014), pp. 137–151.

[YY14b] Yukiko Yamauchi and Masafumi Yamashita. "Randomized Pattern Formation Algorithm for Asynchronous Oblivious Mobile Robots". In: *Proc. of 28th International Symposium on Distributed Computing (DISC)*. 2014, pp. 137–151.