Meeting in Harsh Conditions

Yoann Dieudonne

UNIVERSITY OF PICARDY JULES VERNE

DOCTORAL SCHOOL
OF
SCIENCES, TECHNOLOGY AND HEALTH

HABILITATION THESIS

Specialty : Computer Science

Defended by

Yoann Dieudonné

on December 4, 2023

Meeting in Harsh Conditions

prepared at the MIS Laboratory (UR UPJV 4290)

Jury :

Reviewers:	Pierre Fraigniaud	-	Director of research, CNRS
	David Peleg	-	Professor, Weizmann Institute of Science
	Sébastien Tixeuil	-	Professor, Sorbonne University
Examiners:	Ahmed El Hajjaji	-	Professor, University of Picardy Jules Verne
	Stéphane Devismes	-	Professor, University of Picardy Jules Verne

Preamble

Any researcher who undertakes to write an HDR manuscript is faced with the task of synthesizing their work and tying it together with a common thread, even if it means leaving out some parts for the sake of coherence. And, for my part, I have been no exception. My main area of research concerns mobile distributed systems and, although my work has covered a wide range of topics related to this domain, I chose, by virtue of the principle given above, to focus on the results I obtained for the rendezvous problem and some of its variants – the gathering problem and the treasure hunt problem. This choice was a natural one, as I believe these are the problems to which I have made my most significant contributions since my PhD. This decision was further strengthened by the fact that all of my publications related to rendezvous, gathering, and treasure hunting share a common denominator: they have all been studied under extreme assumptions, making the problems even more challenging, thus justifying the title of this manuscript.

As the style for an HDR is relatively free, I took the approach of presenting my results from a perspective emphasizing the underlying ideas and intuitions, while skipping the formal proofs and pseudocodes: the reader wishing to delve further into this matter will be simply referred to the publications in which all the demonstrations supporting my work have already been published. Actually, my goal, in addition to obtaining the HDR of course, is to provide *in a single manuscript* an intuitive understanding of all my work, in order to facilitate its reuse by other researchers as well as students. Indeed, I am firmly convinced that the appropriation of any algorithmic concept, and by extension its application to lead to new discoveries, cannot be done properly without becoming intimately familiar with the intuitions underlying it. To close this preamble, I then cannot resist reiterating this opinion, by quoting Henri Poincaré, who perfectly sums it up: “C’est avec la logique que nous prouvons, mais avec l’intuition que nous trouvons” (It is by logic that we prove, but by intuition that we discover).

Contents

Chapter 1

Introduction

Contents

1.1 Context

The study of cohorts of mobile robots cooperating with each other to accomplish a common task began to receive renewed attention in the early 2000s. One of the main reasons for this renewed interest was that, for a long time, the tendency had been to use a single robot to perform a given task. However, it must be recognized that such robots are complex, difficult to program and build, and therefore very expensive. This is why the trend has gradually shifted towards systems consisting of several independent and autonomous robots whose capabilities are very limited but which are easy to program, manufacture (because they are simpler and less expensive), and deploy. Under the terms "independent" and "autonomous", we precisely mean that the robots form a full *distributed system*, *i.e.,* a system in which the entities do not depend on a central controlling mechanism. Despite their limitations, these autonomous robots should be able to accomplish non-trivial tasks together. In fact, such systems are generally preferable to a single complex robot for many reasons. Among them, we can mention a faster execution speed, the ability to solve tasks that would otherwise be impossible for a single entity, or the fact that the system is likely to exhibit some form of resilience to malfunctions, *i.e.,* the deficiency of one or more entities does not necessarily lead to mission failure. In literature, this last point is generally referred to as *fault tolerance*.

Obviously, there is a flip side to this coin. It is that the use of multiple entities, instead of a single one, poses a new problem, which is that of coordination. How to communicate effectively? How to subdivide the work and divide it up? How can we ensure that each of the decisions taken locally contributes, in the end, to the achievement of the objectives at the global level? How to react if some robots break down? All these questions are examples of difficulties that can arise and are

direct consequences of the problem of coordination that is inherent to all distributed systems. Answering them is often far from trivial, and sometimes turns out to be a real challenge.

When I started my research in the field of Robotics, these problematics were essentially approached from an empirical point of view. Existing algorithms mainly used heuristics, and were then simply tested either through simulations or on a handful of real robots. Few or no formal proofs of validity (or even only of termination) of the solutions were proposed, let alone proofs of complexity. It is with the interest of complementarity at heart that studies then began to emerge under what we can call a *formal computational approach*. One of the common features of these formal computational studies was the focus on some elementary tasks such as leader election, pattern formation, meeting, etc. The task of leader election consists in making the robots elect exactly one of them as a leader (which can subsequently play the role of an orchestra conductor in order to obtain the central controlling mechanism that is initially lacking), while the task of pattern formation consists in making the robots arrange themselves so that their positions coincide with the points of a given pattern. As for the task of meeting, the goal, as the name suggests, is to make the robots gather at the same location. Basic only apparently, these tasks actually turn out to be both building blocks and essential prerequisites to accomplish much more complex missions. In other words, their feasibility (resp. unfeasibility) in a given context automatically leads to the feasibility (resp. unfeasibility), in the same context, of a whole class of problems. Hence, far from being a loss of generality, the focus on just a bunch of elementary but nonetheless fundamental tasks, helps bring about a better understanding of what can be achieved or not by a cohort of robots. Not only is this statement applicable to robots, but it also extends to other forms of mobile entities, such as drones, software agents (i.e, pieces of software navigating in a network), human-operated ground vehicles, and so on. This diverse range of entities, which encompasses robots, is frequently referred to as *agents*, a generic term that we will thus adopt throughout the remainder of this document.

Following the aforementioned formal approach, after my PhD, I decided to set my sights on the task of meeting. In the literature, this task has been extensively studied under three major variants that are known as *rendezvous*, *gathering* and *treasure hunting*. The rendezvous problem consists in making a team made of exactly two mobile agents eventually meet in a given environment. Gathering is a generalization of this problem to a situation where there are two or more agents in the team, while treasure hunting is a restriction to a situation where one of the two agents of the team (called the treasure) is assumed to always remain stationary.

Depending on the models and the considered assumptions, these variants may be more or less difficult to handle. In this respect, what constitutes the originality of our studies and also what links them together is that they have been carried out by assuming drastic conditions, thus transforming the problem of meeting into a real challenge. By "drastic conditions", we mean, for example, an erratic asynchronism between the agents, the possibility of having to face Byzantine faults (which are considered, in the field of fault tolerance, as the worst ones that can occur), the

absence of direct means of communication, or the fact that agents can be equipped with fuel tanks of small capacity. Of course, some conditions sometimes turned out to be too restrictive, and thus, when appropriate, we demonstrated results of impossibility. Nevertheless, and perhaps surprisingly, we were also able to establish many positive results, including algorithms with reasonable complexities.

Another common thread in all our work is the essentially deterministic approach we adopted to analyze the problem of meeting, which contrasts with the probabilistic approach based on well-known methods such as Monte Carlo, Las Vegas or Atlantic City. Although these methods may offer significant advantages in terms of speed or ease of design and analysis, they may not offer absolute guarantees in terms of complexity and/or accuracy of the resulting solution. By opting for a deterministic approach, we wanted to give the highest priority to these guarantees, keeping in mind that our solutions could potentially be used in turn in critical applications. Therefore, in this regard, whenever an algorithm is mentioned in the manuscript, it will always be assumed to be deterministic, unless explicitly stated otherwise.

A crucial question that arises, when we want to adopt a particular formalism to study mobile agents, is to decide how to model the environment in which they navigate. The two most commonly used possibilities in the literature are a network modeled as a graph, or a terrain modeled as a plane. In both cases, the agents can be considered as points. In the first case, they navigate in the graph by traversing its edges and visiting its nodes, while in the second case they simply move across the plane. Obviously, defining the environment is not enough in order to obtain a complete model. We also need to define some parameters such as the possible behavior of the agents, their features, their capacities, their total number, etc. Depending on the aspects we wish to analyze, there are many different ways of doing this, leading to a lot of distinct models, as it is the case in all of our contributions. Nevertheless, all of our contributions essentially derive from two root models, one for each of the two environments of navigation, *i.e.,* a graph-based root model, which we will call `GBM`, and a planar-based root model, which we will call `PBM`. Therefore, rather than providing a complete description of each model in which we contributed, it is more convenient to fully describe only the two root models. This way, when presenting a particular contribution, we can just detail the few changes/derivations that need to be made from `GBM` or `PBM` in order to obtain the model that is related to that contribution.

If nothing else is stated, solving the task of meeting in `GBM` (resp. `PBM`), or in a model deriving from it, will always mean, throughout this manuscript, getting the agents to share the same position (resp. node) at the same time and never move thereafter.

The rest of this chapter is made of two sections. The former of these sections gives a presentation of the two root models `GBM` and `PBM`, while the latter provides an overview of our contributions that have been done in models deriving from them and that will be subsequently addressed in more details in the next chapters.

1.2 Two root models

1.2.1 GBM

The present model, in the exact form given below, was first introduced in [Dessmark *et al.* 2006], but without calling it GBM as we do in our manuscript. In GBM, the network is viewed as a finite, connected, undirected graph $G = (V, E)$ within which, two agents are placed arbitrarily at two distinct nodes. Initially, the agents know neither the topology of the graph, nor an upper bound on its size, *i.e.,* its number of nodes, which is denoted by $|G|$. We assume that the agents can see each other only when they share the same node, which is of course not the case initially.

Two assumptions are made about the labeling of the two main components of the graph, namely nodes and edges. The first assumption is that nodes are anonymous, *i.e.,* they do not have any kind of labels or identifiers allowing them to be distinguished from one another. The second assumption is that edges incident to a node v are locally ordered with a fixed port numbering ranging from 0 to $\texttt{deg}(v) - 1$ where $\texttt{deg}(v)$ is the degree of v. Therefore, each edge has exactly two port numbers, one for each of both nodes it links. The port numbering does not have to be consistent: a given edge $\{u, v\} \in E$ may be the ith edge of u but the jth edge of v, where $i \neq j$. Given a node v of G and a port i at this node, $\texttt{succ}(v, i)$ will denote the node that is reached by taking port i from node v. Given two adjacent nodes v and v' of G, $\texttt{port}(v, v')$ will denote the port that must be taken from v to reach v'.

The two above labeling assumptions are not fortuitous. The primary motivation of the first one is that if each node can be identified by a label, the task of meeting becomes quite easy to solve, as it is then reducible to a simple graph exploration. Precisely, it is then enough for each agent to execute a breadth-first search and stop at the node having the smallest label. While the first assumption is made so as to avoid making the problem of meeting trivial, the second assumption is made in order to avoid making the problem impossible to solve. Indeed, in the absence of a way of distinguishing locally the edges incident to a node, it is impossible for an agent to simply designate which edge to take in order to exit its current node.

Time is discretized into an infinite sequence of instantaneous rounds. At the beginning, every agent is said to be dormant. While an agent is dormant, it does nothing. A dormant agent can be woken up only in two different ways: either by an adversary, or as soon as a non-dormant agent enters the starting node of the dormant agent. The adversary can choose to wake up only one agent or both (not necessarily in the same round). For consistency, we need to assume that at least one agent is woken up by the adversary. Initially, the adversary also assigns a positive integer, called label, to each agent so that the labels of the two agents are different. Each agent knows its label, but does not know *a priori* the label of the other agent. While the labels are different, the algorithm executed by the agents is the same.

When entering (resp. located at) a node v, a non-dormant agent learns the incoming port number (resp. can only see the degree of v and, if it also occupies v,

the other agent). In each round r, every non-dormant agent executes its algorithm, which instructs it either to stay idle at its current node or to exit by a specific port number. This decision is based only on the state of the (finite but not bounded in advance) memory of the agent, which is made of everything it has seen and learned since its wake-up (including its label). Once the instruction of leaving by a port p (resp. waiting) is completed, the agent is in round $r+1$ at node $\texttt{succ}(v,p)$ (resp. at node v). When traversing an edge, an agent is supposed to be "blind", *i.e.,* if both agents traverse the same edge simultaneously in different directions, they do not notice this fact.

Note that in the absence of a way of distinguishing the agents, as we do through the use of labels, the problem of rendezvous would not be solvable deterministically in some graphs. This is especially the case in a ring in which at each node the edge going clockwise has port number 0 and the edge going counterclockwise has port 1: if both agents are woken up in the same round, start from different nodes of this ring and have the same label (or equivalently, no label at all), then within each round they always act identically and thus always remain in a configuration in which they occupy distinct nodes.

1.2.2 PBM

Numerous variants of PBM can be found in the literature, but the model, in the exact form in which it is presented below, was first introduced in [Dieudonné & Pelc 2014b]. In PBM, the environment is modeled as a spatial universe, here assumed to be $\mathbb{R}^2$, in which two agents are placed at arbitrary but distinct positions. Each of them is provided with an algorithm, a clock, a compass showing the cardinal directions, and a unit of length. The algorithm as well as the clock rate is the same for both agents. It is also the case of the cardinal directions and the unit of length, which permit to define for each member of the team a local coordinate system whose the origin is its initial position.

Moreover, the agents have a *limited sensory radius* (also referred to as *radius of vision*), the value of which is denoted by ε, which allows them to sense (or, to see) all the surroundings at distance at most ε from their respective current locations. The value of ε is positive and is known a priori to the agents. On the other hand, they have no prior knowledge of the distance that separates their initial positions.

The model shares some similarities with GBM. First, the agents are also assigned arbitrary but pairwise distinct labels, corresponding to non-negative integers. As in GBM, this is done in order to avoid symmetries that would be impossible to break deterministically, and, although each agent knows its own label, it does not know *a priori* the label of the other. Second, we can also find the distinction between dormant and non-dormant agents. The way of becoming a non-dormant agent is analogous to that introduced in GBM: the two agents are initially dormant, and each of them can become non-dormant either by an adversary that has to wake up at least one agent (it may wake up the two agents but not necessarily at the same time), or when it gets within the range of vision of the other agent.

In this model, agents can execute instructions of two types: "go in direction *dir* at distance x" and "stay put for time t". When an agent moves, it travels at speed 1 and the execution of any of the above instructions can be interrupted by the agent as soon as it gets at distance ε from its fellow. Once the agents are able to see each other, they can easily solve rendezvous by moving towards the midpoint between their current positions. The instruction ordered by the algorithm depends only on the label of the executing agent and of its memory, on which we do not impose any restriction.

We close the presentation of PBM by noting that it is necessary to have $\varepsilon > 0$, because otherwise it is impossible to design an algorithm allowing the agents to get together, whatever their starting positions. Evidence of this can be easily provided when the adversary wakes up exactly one agent. In this case, the algorithm must be designed to construct a trajectory passing through each given point after a finite time. Such a trajectory is composed of a possibly infinite but countable sequence of segments $S_1, S_2, S_3, \ldots$ and the moving agent can find the other when traversing a segment S_i only if the sum of the slopes of the first i segments is equal to the slope between the starting points of the agents. However, since the number of possible slopes is uncountable, there is one for which the necessary condition will never be fulfilled, which proves the impossibility.

1.3 Overview of the contributions

As explained earlier, we will not discuss the task of meeting directly in GBM or PBM, but within models deriving from them.[1] In this section as well as in the rest of the manuscript, we sometimes refer to GBM and PBM, or their derivations, simply by their navigation environment. These metonyms are used in order to lighten the text and when it is clear from the context.

The quality of our meeting algorithms is often measured in terms of the time elapsed in the worst case between the wake-up of the first agent and the completion of the task: we then speak of *time complexity*. However, it may sometimes be more relevant to evaluate the efficiency of a meeting algorithm with respect to the worst-case total distance traveled by all agents or the worst-case number of edge traversals made by all agents, depending on the navigation environment. In such cases, we refer to this evaluation as the *cost complexity* of the algorithm.

Among all the possible derivations, one of the most basic but crucial one is that concerning the total number of mobile agents, as it influences the definition

[1] Aside from the task of meeting, I also studied other fundamental tasks such as *leader election*, *pattern formation*, *exploration*, etc., which, as explained in the preamble, will not be discussed in the present manuscript. However, in case the reader would also be interested by them, here are the references to these studies: in [Dieudonné & Petit 2007, Dieudonné & Petit 2008, Dieudonné *et al.* 2008a, Dieudonné *et al.* 2008b, Dieudonné & Petit 2009, Dieudonné *et al.* 2009a, Dieudonné *et al.* 2009b, Dieudonné *et al.* 2010b, Dieudonné *et al.* 2010c, Dieudonné *et al.* 2010a, Dieudonné *et al.* 2012a, Dieudonné & Pelc 2012b, Dieudonné & Pelc 2012a, Dieudonné *et al.* 2013a, Dieudonné & Pelc 2014a, Dieudonné & Pelc 2017, Dieudonné *et al.* 2019, Dieudonné & Pelc 2019].

itself of the variant of meeting. Playing with this number can bring us to consider three distinct variants, namely rendezvous, gathering and treasure hunt. In the rendezvous problem, the number of mobile agents is simply left unchanged at two, *i.e.,* as originally defined in GBM and PBM, while in the gathering problem (resp. treasure hunt problem) we consider that the team may be made of more than two mobile agents (resp. is exactly made of one mobile agent and one inert agent). Since we have obtained significant results for each of these three variants, it seems natural to dedicate one entire chapter to each of them.

We begin by presenting our contributions [Dieudonné & Pelc 2015, Dieudonné *et al.* 2015, Chalopin *et al.* 2016, Bouchard *et al.* 2018c, Bouchard *et al.* 2019] to the problem of rendezvous in Chapter 2 (preliminary versions of these papers appeared in [Dieudonné & Pelc 2013b, Dieudonné *et al.* 2013b, Chalopin *et al.* 2014, Bouchard *et al.* 2017]). For this problem, we worked both in GBM and PBM, but by weakening the level of synchrony between the agents. We will detail what we precisely mean by this in the next chapter, but roughly speaking, instead of assuming that the agents traverse edges in synchronous rounds or move in the plane at speed 1, we consider, in our four contributions, different scenarios where it is not the case. In particular, we consider a scenario where the adversary can initially assign possibly different speeds to each agent and an even harder scenario in which the adversary can arbitrarily vary the speed of each agent during each of its moves. In both cases, rendezvous gets significantly more complicated. Yet, in the first scenario, we give a rendezvous algorithm working in the plane [Dieudonné & Pelc 2015] (resp. in an arbitrary graph [Bouchard *et al.* 2018c]) with a time complexity that is polynomial in the initial distance D separating the agents (resp. in the size of the graph), in the logarithm of the smaller label, and in the inverse of the faster assigned speed. In the second scenario, we give an algorithm working in the plane [Bouchard *et al.* 2019] (resp. in an arbitrary graph [Dieudonné *et al.* 2015]) with a cost that is polynomial in D (resp. in the size of the graph) and in the logarithm of the smaller label. It should be however noted that the algorithm of [Dieudonné *et al.* 2015] only solves a relaxed version of rendezvous in which the meeting can occur at a node or inside an edge, as the original version allowing meetings only at nodes is impossible to solve in graphs with an adversary as powerful as that of the second scenario.

Orthogonally, we have also studied the problem of rendezvous in GBM, but by assuming that the agents are subject to *delay faults* [Chalopin *et al.* 2016]: if an agent incurs a fault in a given round, then it remains in the current node, whether it was supposed to move or not. If it was supposed to move and the fault happened, the agent is aware of it. In [Chalopin *et al.* 2016], we consider three scenarios of fault distribution: random (independently in each round and for each agent with constant probability $0 < p < 1$), unbounded adversarial (the adversary can delay an agent for an arbitrary finite number of consecutive rounds) and bounded adversarial (the adversary can delay an agent for at most c consecutive rounds, where c is unknown to the agents).

For random faults, we show an algorithm with a time complexity polynomial

in the size of the network, which achieves rendezvous with very high probability in arbitrary networks. By contrast, for unbounded adversarial faults, we provide a negative result by showing that rendezvous is not possible, even in the class of rings. For bounded adversarial faults, we give a rendezvous algorithm having a time (resp. cost) complexity polynomial in the size of the network, in the bound c and in the larger label (resp. in the logarithm of c and in the logarithm of the larger label).

In Chapter 3, we present our contributions [Dieudonné *et al.* 2014, Dieudonné & Pelc 2016, Bouchard *et al.* 2016, Bouchard *et al.* 2022, Bouchard *et al.* 2023b] to the problem of gathering (preliminary versions of these papers appeared in [Dieudonné *et al.* 2012b, Dieudonné & Pelc 2013a, Bouchard *et al.* 2015, Bouchard *et al.* 2018a, Bouchard *et al.* 2020a]). All of them have been made in models deriving only from `GBM` and, except in [Bouchard *et al.* 2023b], we always assume that in each round the agents sharing the same node can exchange all currently held information. Unless explicitly stated otherwise, in our work, solving the gathering problem always goes hand in hand with its detection: not only do the agents solve the gathering, but they also end up being aware of it.

We start in Chapter 3 with the paper [Dieudonné & Pelc 2016], in which we investigate the problem of gathering assuming that the agents are all anonymous. This particularly means that the agents are entirely identical and have no labels allowing to distinguish them. As pointed out at the end of the presentation of `GBM`, we then give rise to initial configurations from which the problem cannot be solved due to symmetry considerations. In [Dieudonné & Pelc 2016], we characterize completely the set Π of all the configurations from which gathering is possible and gave two universal gathering algorithms, *i.e.,* algorithms working from every configuration of Π. The first algorithm operates under the assumption that a common upper bound on the size of the network is known to all agents. In this case, our algorithm guarantees the existence of a round for any configuration of Π such that gathering is solved. If no upper bound on the size of the network is known, we show that a universal algorithm for gathering with detection does not exist. Hence, for this harder scenario, we construct a second universal gathering algorithm guaranteeing, for any configuration of Π, that all agents eventually get to one node and stop, although they can never determine if gathering is over. The time complexity of the first (resp. second) universal algorithms is polynomial in the known upper bound on the size of the network (resp. in the size of the network).

In [Dieudonné *et al.* 2014], we address the problem of gathering for a team of agents all having distinct labels (the label of any given agent can be seen only by those sharing the same node as it), but by assuming that up to f of them may be Byzantine, *i.e.,* prone to Byzantine faults. We consider in [Dieudonné *et al.* 2014] two types of Byzantine agent: (1) the strongly Byzantine agent, which can choose an arbitrary port when it moves, convey arbitrary information to other agents and forge any label, and (2) the weakly Byzantine agent, which can do the same, except changing its label. No matter the type, we cannot count on the Byzantine agents to cooperate. Hence, in such a context, we can just hope to gather only the good agents,

i.e., those that are not Byzantine. This naturally raises the following question: what is the minimum number of good agents that guarantees gathering of all of them? In the above paper, we bring exact answers to this question when coping with weakly Byzantine agents, and give approximate ones when coping with strongly Byzantine agents, both when an upper bound on the size of the network is known and when it is not. For weakly Byzantine agents, we show that any number of good agents permit to solve the problem when an upper bound on the graph size is known. If no such upper bound is known, then this minimum number is $f+2$. More precisely, we show a polynomial algorithm that gathers all good agents in an arbitrary network, provided that there are at least $f+2$ of them. We also provide a matching lower bound: we prove that if the number of good agents is at most $f+1$, then they are not able to solve the gathering problem in some networks.

For strongly Byzantine agents, we come up with negative and positive results too. On the negative side, we give a lower bound of $f+1$, when the graph size is known: we show that f good agents cannot gather in the presence of f Byzantine agents even in a ring of known size. When no upper bound on the graph size is known, the lower bound of $f+2$ that holds for weakly Byzantine agents, is obviously also true for strongly Byzantine agents. On the positive side, we give gathering algorithms for at least $2f+1$ good agents when an upper bound on the graph size is known, and for at least $4f+2$ good agents when no such upper bound is known. In [Bouchard *et al.* 2016], we significantly improve on these positive results: when an upper bound on the size of the network is known (resp. no upper bound on the size of the network is known), we provide an algorithm working with at least a number of good agents that perfectly matches the lower bound of $f+1$ (resp. $f+2$) identified in [Dieudonné *et al.* 2014].

Unfortunately, the algorithms dealing with strongly Byzantine agents in the above two papers, all have the major disadvantage of having a time complexity that is exponential in the size of the network and the largest label of a good agent. To circumvent this problem, we propose in [Bouchard *et al.* 2022] to make a concession on the proportion of strongly Byzantine agents within the team, in order to get an algorithm offering a significantly lower complexity. Precisely, in this paper, we design a gathering algorithm working in all graphs of size at most n in time polynomial in n and the logarithm of the smallest label ℓ of a good agent, provided the agents are in a *strong team*, *i.e.,* a team where the good agents are at least some quadratic polynomial in f. Our algorithm requires that the agents are initially provided with a common input that can be coded in $O(\log\log\log n)$ bits. However, we prove that this size of input is asymptotically optimal to obtain a polynomial time complexity in n and ℓ.

As can be seen, we have essentially analyzed the task of *gathering* assuming that the agents at the same node can exchange information. In [Bouchard *et al.* 2023b], we ask if this ability of talking is needed. The answer turns out to be no. In support of this, we describe in [Bouchard *et al.* 2023b] two algorithms that accomplish gathering in `PBM` with a team of agents having distinct labels, assuming that they are entirely silent and unable to see the labels of the others even when sharing the

same node. Actually, in each round, a non-dormant agent just sees how many agents are at the node that it currently occupies and the degree of this node. Our first algorithm assumes that agents know some upper bound on the size of the network, and works in time polynomial in this upper bound and in the length of the smallest label. Our second algorithm does not assume any knowledge about the network, but its complexity is at least exponential in the size of the network and in the labels of the agents. Its purpose is to show feasibility of gathering under this harsher scenario.

As a by-product of our techniques we obtain, in the same weak model, solutions to two fundamental problems, whether the agents initially know some upper bound on the size of the network or not. The first problem is that of leader election: one agent is elected as a leader, and all agents learn its identity. The second problem is that of gossiping: each agent has a message at the beginning, and all agents must end up learning all messages. This result about gossiping is perhaps our most surprising finding: agents devoid of any transmitting devices can solve the most general information exchange problem, as long as they can count the number of agents present at visited nodes.

In Chapter 4, we describe our contributions [Bouchard *et al.* 2020b, Bouchard *et al.* 2023a] to the last remaining variant of meeting, which is treasure hunt (preliminary versions of these papers appeared in [Bouchard *et al.* 2018b, Bouchard *et al.* 2021]). This problem involves a duo of agents that are initially separated by a distance at most D (unknown to them). In this problem, only one agent is authorized to move, while the other, referred to as the treasure, simply waits to be discovered by staying always idle. The fact of being woken up or not by the adversary becomes obsolete in these circumstances, because, in order for this problem to be meaningful, we necessarily need to suppose that the only mobile agent ends up starting the execution of its algorithm.

The paper [Bouchard *et al.* 2020b] investigates the problem in the plane, assuming that in the beginning and after each move the mobile agent gets only an approximated hint consisting of a positive angle smaller than 2π whose vertex is at the current position of the agent and within which the treasure is contained. The main question that is then raised is how these hints permit the mobile agent to lower the length of its trajectory before finding the treasure. It is well known that, without any hint, the optimal cost complexity is $\Theta(D^2)$. We show that if all angles given as hints are at most π, then the cost can be lowered to $O(D)$, which is asymptotically optimal. If all angles are at most β, where $\beta < 2\pi$ is a constant unknown to the agent, then the cost is at most $O(D^{2-\varepsilon})$, for some $\varepsilon > 0$. For both these positive results, we present algorithms achieving the above cost complexities. In the case where angles given as hints can be arbitrary, smaller than 2π, we show that cost complexity $\Theta(D^2)$ cannot be beaten.

Finally, in [Bouchard *et al.* 2023a], we bring one of our most significant results to the community by contradicting a conjecture made more than two decades ago by Awerbuch, Betke, Rivest and Singh in [Awerbuch *et al.* 1999].

Precisely, Awerbuch, Betke, Rivest and Singh considered treasure hunt in a

model corresponding to GBM, but by assuming that the nodes all have pairwise distinct labels and that the mobile agent has a fuel tank that can be replenished only at its starting node s. The size of this tank is $B = 2(1+\alpha)r$, for some positive real constant α, where r, called the radius of the graph, is the maximum distance from s to any other node. The tank imposes an important constraint as it forces the agent to make at most $\lfloor B \rfloor$ edge traversals before having to refuel at node s, or otherwise, the agent will be left with an empty tank and unable to move, preventing further exploration of the graph. Let $e(D)$ be the number of edges whose at least one endpoint is at distance less than D from s. Awerbuch, Betke, Rivest and Singh conjectured that it is impossible to find a treasure hidden in a node at distance at most D at cost nearly linear in $e(D)$. In [Bouchard *et al.* 2023a], we refute this conjecture by designing a treasure hunt algorithm having a cost complexity of $\mathcal{O}(e(D)\log D)$. We also observe that no treasure hunt algorithm can beat cost $\Theta(e(D))$ for all graphs, and thus our algorithm turns out to be almost optimal.

Chapter 2

Rendezvous

Contents

2.1 Introduction

Historically, the first mention of the rendezvous problem appeared in *The Strategy of Conflict* [Schelling 1960], which is a classic text in game theory and international relations. The book explores how individuals and nations interact in situations of conflict, and how they can use strategic thinking to achieve their goals. In his book, Schelling introduces the problem of rendezvous through a scenario in which two individuals need to coordinate their actions to meet at a specific time and place, without communicating with each other and agreeing beforehand. In particular, he uses the example of two individuals who need to meet in New York City. They each have a choice of two landmarks to meet at - Grand Central Station or the Empire State Building. In the absence of communication, the most likely outcome is that they will both choose the Empire State Building as their meeting place, because it is a more distinctive landmark.

The rendezvous problem allows Schelling to illustrate a concept that is dear to him, that of the "focal point" - a solution to which players tend to converge, even if it is not necessarily the most rational or optimal outcome. In the absence of prior coordination, individuals will often choose a solution that is socially salient, or that seems more obvious or recognizable. By understanding the concept of focal points, individuals can improve their chances of coordinating their actions and achieving their goals, even in situations where communication is not possible.

After Schelling's book, the problem of rendezvous has been extensively studied, resulting in a vast literature on the subject. This is partly due to the fact

that the problem can be applied to other fields beyond distributed algorithms, as we have seen with game theory, but also because, within the domain of distributed algorithms, which concerns us directly, there are many possible combinations for addressing the problem, such as playing on the navigation environment, using deterministic or randomized sequences of instructions, allowing agents to leave traces in visited locations or not, etc. Since we are specifically interested in deterministic scenarios where agents cannot leave any marks or indicators in the visited locations, we will essentially focus on studies related to these in the remainder of this introduction. However, for the curious reader wishing to consider the matter in greater depth, including aspects that we will discuss, a good starting point is to go through the following excellent surveys on rendezvous [Alpern 2002, Alpern & Gal 2003, Kranakis *et al.* 2006, Pelc 2019].

The literature on the deterministic scenarios in which we are concerned, can be roughly divided according to two ways of modeling the navigation environment - a graph or a plane.

Concerning rendezvous in graphs, many papers have explored the problem through model `GBM` (cf. Section 1.2.1) in which agents move in synchronous rounds. One such paper is [Dessmark *et al.* 2006], which provides a deterministic protocol that guarantees the meeting of the two agents after a number of rounds that is polynomial in the size n of the graph, the length ℓ of the shorter of the two labels and the delay τ between their wake-up times. As an open problem, the authors ask whether it is possible to obtain a polynomial solution to this problem depending only on the first two parameters. In [Kowalski & Malinowski 2008], a positive answer is brought to this question, by providing an algorithm ensuring rendezvous in time $\mathcal{O}(\log^3 \ell \;\; n^{15} \log^{12} \ell)$. This complexity is subsequently improved in [Ta-Shma & Zwick 2014] with an algorithm working in time $\mathcal{O}(n^5 \log l)$, which is currently the most time-efficient method for solving rendezvous in `GBM`. This must be contrasted with the best known lower bound of $\Omega(n \log \ell)$ proven in [Dessmark *et al.* 2006], which leaves a challenging gap to bridge.

While the three above algorithms ensure rendezvous in polynomial time, they also ensure it at polynomial cost. Indeed, a polynomial time always implies a polynomial cost in `GBM`, because each agent can make at most one edge traversal per round. The reciprocal is not true, as the agents can have very long waiting periods, sometimes interrupted by a movement. Thus, these parameters of cost and time are not always necessarily linked to each other. This is highlighted in [Miller & Pelc 2016] where the authors study the tradeoffs between these parameters for rendezvous in `GBM`. In parallel with this, some efforts have been dedicated to analyzing the impact on time complexity of rendezvous when in every round the agents are provided with some pieces of information by making a query to some device or some oracle [Das *et al.* 2014, Miller & Pelc 2015]. Along with the work aiming at optimizing the parameters of time and/or cost of rendezvous, some other work examines the amount of required memory to solve the problem, with two anonymous agents, *e.g.,* [Fraigniaud & Pelc 2008, Fraigniaud & Pelc 2013] for tree networks and in [Czyzowicz *et al.* 2012a] for general networks.

On the fault-tolerant side, a self-stabilizing rendezvous algorithm is provided in [Ooshita *et al.* 2017]. Self-stabilization is an elegant concept of fault tolerance in distributed systems, introduced by E. W. Dijkstra in [Dijkstra 1974]. Roughly speaking, an algorithm is said to be self-stabilizing for a problem P if it solves P starting from any arbitrary configuration (which permits, in particular, to tolerate any kind of transient faults such as temporal memory corruption). Hence, in the context of rendezvous in arbitrary graphs, the algorithm of [Ooshita *et al.* 2017] guarantees the meeting of the two agents regardless of their initial memory states and starting nodes. However, the time complexity of their algorithm cannot be bounded.

Some studies have been also dedicated to the scenario in which the agents move asynchronously in a network [Marco *et al.* 2006, Czyzowicz *et al.* 2012b], *i.e.,* as in `GBM`, but assuming that the agents' speed may vary, controlled by the adversary. In [Marco *et al.* 2006], the authors investigate the cost of rendezvous for both infinite and finite graphs. In the former case, the graph is reduced to the (infinite) line and polynomial bounds are given depending on whether the agents know the initial distance between them or not. In the latter case (finite graphs), similar bounds are given for ring shaped networks. They also propose a rendezvous algorithm for an arbitrary graph, provided that the agents initially know an upper bound on the size of the graph. This assumption is subsequently removed in [Czyzowicz *et al.* 2012b]. However, in both [Marco *et al.* 2006] and [Czyzowicz *et al.* 2012b], the cost of rendezvous in an arbitrary graph is exponential in its size and in the larger of the labels. Other work has been made on asynchronous rendezvous (in the wider context of gathering) in networks [Cicerone *et al.* 2019] but by assuming a specific topology for the underlying graph (*e.g.,* a ring, a tree or a grid) and/or assuming the agents are able to see the whole network.

Concerning rendezvous in the plane, we also find the same dichotomy of synchronicity *vs.* asynchronicity. The synchronous case is introduced in [Suzuki & Yamashita 1999]. In [Izumi *et al.* 2012], rendezvous in the plane is studied for oblivious agents equipped with unreliable compasses under synchronous and asynchronous models. Asynchronous rendezvous, as well as asynchronous gathering, is also studied in various settings in [Flocchini *et al.* 2005, Cohen & Peleg 2005, Cohen & Peleg 2008, Cieliebak *et al.* 2012, Pagli *et al.* 2015]. However, the common feature of all these papers is that the agents can observe all the positions of the other agents or at least the global graph of visibility is always connected (*i.e.,* the team cannot be split into two groups so that no agent of the first group can detect at least one agent of the second group).

The problem of asynchronous rendezvous in the plane, where the agents have a limited range of vision and cannot see each other initially, is addressed in [Collins *et al.* 2010, Bampas *et al.* 2010, Czyzowicz *et al.* 2012b]. In the first two papers, algorithms having a cost polynomial in the initial distance D separating the agents are proposed. However, they use a powerful assumption that each agent knows its starting position in a global system of coordinates. In [Czyzowicz *et al.* 2012b], the authors give a solution that does not rely on such

an assumption, but that has the major disadvantage of having a cost exponential in D and in the larger of the labels.

What did we do about the problem of rendezvous? When examining our literature review on deterministic rendezvous where agents cannot leave any marks or indicators in the visited locations, two significant observations can be made. Firstly, there is no algorithm that can solve efficiently asynchronous rendezvous in graphs or in the plane without making strong assumptions related to positioning (*i.e.,* being able to position itself either in a global and common coordinate system or relative to the position of the other). Secondly, fault-tolerant algorithms dedicated specifically to rendezvous in arbitrary graphs have not been much studied.[1]

Actually, these gaps correspond to places where there was originally some "darkness" and where we made our substantial contributions on rendezvous [Dieudonné & Pelc 2015, Dieudonné *et al.* 2015, Chalopin *et al.* 2016, Bouchard *et al.* 2018c, Bouchard *et al.* 2019]. This is further developed in the following two sections.

2.2 Asynchronous rendezvous

This section is made of two subsections. The first subsection is dedicated to the description of our work on asynchronous rendezvous in the plane, published in [Dieudonné & Pelc 2015, Bouchard *et al.* 2019], while the second subsection is dedicated to the description of our work on asynchronous rendezvous in graphs, published in [Dieudonné *et al.* 2015, Bouchard *et al.* 2018c].

2.2.1 In the plane

In [Dieudonné & Pelc 2015], we show a rendezvous algorithm working in `PBM`, but by assuming that the agents are initially assigned, by the adversary, possibly different speeds: each agent knows its speed but does not know the speed of the other. Hence, each time an agent moves, it does so at its assigned velocity. Under these settings, our algorithm accomplishes rendezvous in time polynomial in the initial distance separating the agents, in the logarithm of the smaller label and in the inverse of the faster assigned speed. As for its cost, it is polynomial in the first two parameters and does not depend on the third.

In [Bouchard *et al.* 2019], we show a rendezvous algorithm working in the plane in an even more difficult scenario. Precisely, each time an agent chooses to move from a position a to a position b, the adversary can arbitrarily vary the speed of the

[1] It should be noted that the problem of meeting may often lead to quick results of possibility or impossibility in fault-prone environments when exactly two agents are involved, while it is no longer the case with more than two agents. This may be especially true with crash or Byzantine faults. As explained in Chapter 3, that is certainly why, most of the studies assuming the occurrence of faults have been conducted in the more general context of gathering, rather than specifically focusing on rendezvous. Of course, there are also types of faults for which the problem of rendezvous cannot be easily swept aside, notably (and fortunately) the delay faults that underlie our contribution [Chalopin *et al.* 2016] that is presented in Section 2.3 ☺.

agent, stop it and even move it back and forth as long as the trajectory of the agent is continuous, does not leave the segment $[a, b]$, and ends at b. Despite this harsh scenario of asynchrony, our algorithm guarantees rendezvous at a cost polynomial in the initial distance separating the agents and in the logarithm of the smaller label. Note that speaking of time complexity would not have any sense here, as the time of meeting remains at the discretion of the adversary, which can delay it for an arbitrarily long but finite period (without, however, being able to prevent it indefinitely).

Below, we detail only the intuitions behind the algorithm of [Bouchard *et al.* 2019] coping with full asynchrony, because it can be proven that, in the scenario where agents are initially assigned arbitrary velocities, it has a time complexity and a cost complexity that are roughly comparable to those of the algorithm of [Dieudonné & Pelc 2015].

Informal description in a nutshell

Instead of studying directly the problem of asynchronous rendezvous in the plane, we study the problem of asynchronous rendezvous in a slightly different scenario. Precisely, we consider the problem in `GBM` but by assuming:

1. The graph is an infinite square grid in which every node u is adjacent to 4 nodes located North, East, South, and West and reachable from node u by taking ports 0, 1, 2 and 3 respectively.

2. The agents do not move in synchronous rounds. When an agent chooses to exit a node u by some port p, the adversary can arbitrarily vary the speed of the agent, stop it and even move it back and forth as long as the trajectory of the agent is continuous, does not leave the edge linking u and $\texttt{succ}(u, p)$, and ends at $\texttt{succ}(u, p)$.

3. Rendezvous inside an edge is allowed. If two agents meet inside an edge, they can detect this event and stop.

Note that, while the *route* of an agent, *i.e.,* the sequence (with possible repetitions) of the traversed edges, is controlled by the algorithm, its *walk*, *i.e.,* its way of moving on the edges, is controlled by the adversary.

We proceed to this switch of scenarios, because that of the infinite square grid is easier to handle and because we know from [Czyzowicz *et al.* 2012b] that we have the following implication: if there is an algorithm solving the problem of asynchronous rendezvous in the infinite square grid at a cost polynomial in the logarithm of the smaller label and in the initial distance (in the Manhattan metric) separating the agents, then there is an algorithm solving the problem of asynchronous rendezvous in the plane at a cost polynomial in the same parameters (but with the initial distance expressed in the Euclidean metric). Hence, we focus in the sequel on the ideas allowing to design an algorithm satisfying the first part of the implication.

It is well known that solving rendezvous deterministically is impossible in some symmetric graphs (including our square grid) unless both agents are given distinct identifiers, called labels. We use them to break the symmetry, *i.e.,* in our context, to make the agents follow different routes. The idea is to make each agent "read" the binary representation of its bit, one bit at a time from the most to the least significant bits, and for each bit it reads, follow a route depending on the read bit. Our algorithm ensures rendezvous during some of the periods when they follow different routes, *i.e.,* when the two agents process two different bits.

Furthermore, to design the routes that both agents will follow, our approach would require to know an upper bound on two parameters, namely the initial distance between the agents and the length (of the binary representation) of the shortest label. As we suppose that the agents have no knowledge of these parameters, they both perform successive "assumptions", in the sequel called *phases*, in order to find out such an upper bound. Roughly speaking, each agent attempts to estimate such an upper bound by successive tests, and for each of these tests, acts as if the upper bound estimate is correct. Both agents first perform phase 0. When phase i does not lead to rendezvous, they perform phase $i+1$, and so on. More precisely, within phase i, the route of each agent is built in such a way that it ensures rendezvous if 2^i is a good upper bound on the parameters of the problem. Hence, in our approach two requirements are needed: both agents are assumed (1) to process two different bits (*i.e.,* 0 and 1) almost concurrently and (2) to perform phase $i = \alpha$ almost at the same time—where α is the smallest integer such that the two aforementioned parameters are upper bounded by 2^α.

However, in order to meet these requirements, we have to face two major issues. First, since the adversary can vary both agents' speeds, the idea described above does not prevent the adversary from making the agents always process the same type of bit at the same time. Moreover, the route cost depends on the phase number, and thus, if an agent were performing some phase i, where i is exponential in the initial distance and in the length of the binary representation of the smaller label, then our algorithm would not be polynomial. To tackle these two issues, we use a mechanism that prevents the adversary from making one agent execute the algorithm arbitrarily faster than the other without meeting. Each of both these issues is circumvented via a specific "synchronization mechanism". Roughly speaking, one of both mechanisms ensures that the agents start phase α at almost the same time, while the other makes the agents read and process, within phase α, the bits of the binary representation of their labels at nearly the same speed. This is particularly where our feat of strength is: orchestrating in a subtle manner these synchronizations in a fully asynchronous context while ensuring a polynomial cost. Now that we have described the very high-level idea of our algorithm, let us give more details.

Under the hood.

The approach described above allows us to solve rendezvous when there exists an

index for which the binary representations of both labels differ. However, this is not always the case, especially when one binary representation is a prefix of the other one (e.g., 100 and 1000). Hence, instead of considering its own label, each agent will consider a transformed label. The transformation borrowed from [Dessmark *et al.* 2006] will guarantee the existence of the desired difference over the new labels, while preserving asymptotically the same lengths for their binary representations. In the rest of this description, we assume for convenience that the initial Manhattan distance D separating the agents is at least the length of the shortest binary representation of the two transformed labels (the complementary case adds an unnecessary level of complexity to understand the intuition).

As mentioned previously, our solution works in phases numbered $0, 1, 2, \ldots$ During phase i, the agent supposes that the initial distance D is at most 2^i and processes one by one the first 2^i bits of its transformed label. In the case where 2^i is greater than the binary representation of its transformed label, the agent will consider that each of the last "missing" bits is 0. When processing a bit, the agent executes a particular route that depends on the bit value and the phase number. The route related to bit 0 and the route related to bit 1 are obviously different and are designed in such a way that if both these routes are executed almost simultaneously by two agents within a phase corresponding to a correct upper bound, then rendezvous occurs by the time any of them has been completed.

In the light of this, if we denote by α the smallest integer such that $2^\alpha \geq D$, it turns out that an ideal situation would be that the agents concurrently start phase α and process the bits at quite the same rate within this phase. Indeed, we would then obtain the occurrence of rendezvous by the time the agents complete the process of the λth bit of their transformed label in phase α, where λ is the smallest index for which the binary representations of their transformed labels differ. However, getting such an ideal situation in presence of a fully asynchronous adversary appears to be really challenging. This is where the two synchronization mechanisms briefly mentioned above come into the picture.

If the agents start phase α approximately at the same time, the first synchronization mechanism permits to force the adversary to make the agents process their respective bits at similar speed within phase α, as otherwise rendezvous would occur prematurely during this phase before the process by any agent of the λth bit. This constraint is imposed on the adversary by dividing each bit process into some predefined steps and by ensuring that after each step s of the kth bit process, for any $k \leq 2^\alpha$, an agent follows a specific route that forces the other agent to complete the step s of its kth bit process. This route, on which the first synchronization is based, is constructed by relying on a simple principle that enables one agent to "push" the other. The principle is as follows: if an agent performs a given route X included in a given area $\mathcal{S}$ of the basic grid, then the other agent can force it to finish route X by covering $\mathcal{S}$ as many times as there are edge traversals in X. More precisely, each covering of $\mathcal{S}$ permits to traverse all the edges of X at least once: so, in each covering, the agent executing X must complete at least one edge traversal or rendezvous occurs. Hence, by leaving aside the few extra technicalities involved

in implementing the first synchronization around this basic principle, one of the major difficulties we have to face lies in the setting up of the second synchronization mechanism guaranteeing that the agents start phase α at roughly the same time. At first glance, it might be tempting to use an analogous principle to the one used for dealing with the first synchronization. Indeed, if an agent a_1 follows a route covering r times an area $\mathcal{Y}$ of the grid, such that $\mathcal{Y}$ is where the first $\alpha-1$ phases of an agent a_2 take place and r is the maximal number of edge traversals an agent can make during these phases, then agent a_1 pushes agent a_2 to complete its first $\alpha-1$ phases and to start phase α. Nevertheless, a strict application of this principle to the case of the second synchronization directly leads to an algorithm having a cost that is superpolynomial in D and the length of the smaller label, due to a cumulative effect that does not appear for the case of the first synchronization. As a consequence, to force an agent to start its phase α, the second synchronization mechanism does not depend on the kind of route described above, but on a much more complicated route that permits an agent to "*push*" the second one. This works by considering the "pattern" that is drawn on the grid by the second agent rather than just the number of edges that are traversed. This is the most tricky part of our algorithm, one of the main ideas of which relies in particular on the fact that some routes made of an arbitrarily large sequence of edge traversals can be pushed at a relatively low cost by some other routes that are of comparatively small length, provided they are judiciously chosen. Let us illustrate this point through the following example. Consider an agent a_1 following from a node v_1 an arbitrarily large sequence of X_i, in which each X_i corresponds to either $A\overline{A}$ or $B\overline{B}$, where A and B are any routes ($\overline{A}$ and $\overline{B}$ corresponding to their respective backtrack, *i.e.,* the sequence of edge traversals followed in the reverse order). An agent a_2 starting from an initial node v_2 located at a distance at most d from v_1 can force agent a_1 to finish its sequence of X_i (or otherwise rendezvous occurs), regardless of the number of X_i, simply by executing $A\overline{A}B\overline{B}$ from each node at distance at most d from v_2. To support this claim, let us suppose by contradiction that it does not hold. At some point, agent a_2 necessarily follows $A\overline{A}B\overline{B}$ from v_1. However, note that if one agent starts following $A\overline{A}$ (resp. $B\overline{B}$) from node v_1 while the other is following $A\overline{A}$ (resp. $B\overline{B}$) from node v_1, then the agents meet. Indeed, this implies that the more ahead agent eventually follows $\overline{A}$ (resp. $\overline{B}$) from a node v_3 to v_1 while the other is following A (resp. B) from v_1 to v_3, which leads to rendezvous. Hence, when agent a_2 starts following $B\overline{B}$ from node v_1, agent a_1 is following $A\overline{A}$, and is not in v_1, so that it has at least started the first edge traversal of $A\overline{A}$. This means that when agent a_2 finishes following $A\overline{A}$ from v_1, a_1 is following $A\overline{A}$, which implies, using the same arguments as before, that they meet before either of them completes this route. Hence, in this example, agent a_2 can force a_1 to complete an arbitrarily large sequence of edge traversals with a single and simple route. Actually, our second synchronization mechanism implements this idea. This was the most complicated thing to set up, as each part of routes in every phase had to be orchestrated very carefully to permit, in the end, this low cost synchronization while still ensuring rendezvous. However, it is through this way of moving that we finally get the desired polynomial cost.

While the cost is polynomial, we made no attempt at optimizing it. The exponent of the polynomial being quite large (it is made of three digits), there seems to be room to make significant improvements. Consequently, a natural open problem is to find out the optimal cost to solve the task of rendezvous in the plane under the full asynchrony. This would be all the more important as in turn we could compare this optimal cost with the cost of solving the same task with agents that can position themselves in a global system of coordinates (as indicated in Section 2.1, the almost optimal cost for this case is given in [Collins *et al.* 2010]) in order to determine whether the use of such a system (*e.g.,* GPS) is finally relevant to minimizing the traveled distance.

2.2.2 In graphs

In [Dieudonné *et al.* 2015], we give a rendezvous algorithm working in `GBM` but by weakening drastically the level of synchrony. More precisely, the agents no longer move in synchronous rounds: when an agent is at a node u, its algorithm selects the port p by which it will exit, but the way of moving on the edge connecting u to $\texttt{succ}(u,p)$ is completely under the control of the adversary. As in the scenario of the asynchronous rendezvous in the infinite square grid presented in the previous subsection, the adversary can arbitrarily vary the speed of the agent during its edge traversal from u to $\texttt{succ}(u,p)$, stop it and even move it back and forth as long as the trajectory of the agent remains continuous, does not leave the edge, and ends at $\texttt{succ}(u,p)$. With such a powerful adversary, the task becomes so complex that even in the case of a simple two-node graph, we cannot design an algorithm guaranteeing the agents will meet at a node. That is why, we need here to relax the rendezvous requirement by allowing the agents to meet at a node or inside an edge. In particular, when the agents cross each other inside an edge, they can detect this event and stop. Although meetings inside an edge are allowed, we want to avoid the agents to meet accidentally when traversing two distinct edges that cross each other. Hence, we assume in [Dieudonné *et al.* 2015] that the agents move in an embedding of the underlying graph in the three-dimensional Euclidean space, with nodes of the graph being points of the space and edges being pairwise disjoint line segments joining them.

In this context, our algorithm of [Dieudonné *et al.* 2015] solves the problem of rendezvous at a cost polynomial in the size of the network and in the logarithm of the smaller label. The only previous algorithm solving the same asynchronous rendezvous problem [Czyzowicz *et al.* 2012b] is exponential in the size of the network and in the larger label. Thus, with our result, we decrease the cost exponentially in the size of the network and doubly exponentially in the labels of agents.

Besides, we show that our rendezvous algorithm can be used as a building block to solve fundamental problems, involving two or more agents, for which there were previously no solution in the asynchronous setting described above. Among them are the following problems: *team size*, in which every agent has to find the total number of agents, *leader election*, in which all agents have to output the label of a

single agent, *perfect renaming* in which all agents have to adopt new different labels from the set $\{1,\ldots,k\}$, where k is the number of agents, and *gossiping*, in which each agent has initially a piece of information (value) and all agents have to output all the values. Using our rendezvous algorithm, we solve all these problems at cost polynomial in the size of the network and in the logarithm of the smallest label among the participating agents.

To illustrate one of the key principles that is at work in our asynchronous rendezvous algorithm, consider an ideal situation in which the agents are labeled 0 and 1 and initially know an upper bound n on the size of the network. Let `Explo`(n) be a procedure that consists in executing an *effective part* followed by a *backtrack part*. The effective part allows the executing agent to visit every node of any graph of size at most n by making a polynomial number $p(n)$ of edge traversals, while the backtrack part makes the agent return to its starting node by traversing in the reverse order the entire sequence of edges it has traversed during the effective part (some edges may be traversed several times). (The effective part of procedure `Explo`(n) can be derived from the result of [Reingold 2008], which is based on *Universal eXploration Sequences*, better known by the acronym, UXS.[2])

In our ideal situation, rendezvous can be ensured at polynomial cost in n, by asking the agent to execute the following instructions.

- If the agent has label 0, then it applies $(2p(n)+1)^2$ times procedure `Explo`(n) from its starting node u.
- If the agent has label 1, then it applies `Explo`(n) from each node w of the graph. This is done by executing procedure `Explo`(n) but interrupting it at the beginning and after each edge traversal to just execute `Explo`(n) from the current node.

Rendezvous must indeed occur by the time an agent terminates to process its instructions. It is the case, if the agent with label 1 finishes to follow the round-trip trajectory induced by the execution of `Explo`(n) from node u before the agent having label 0 finishes its instructions, because this latter agent only repeats the same round-trip trajectory from node u: roughly speaking, it means the agent with label 1 ends up "catching" the agent with label 0. Otherwise, it is also the case as the number of edge traversals made by the agent with label 1 is smaller than the number of times the agent with label 0 traverses entirely the network: roughly speaking, it means the agent with label 0 ends up "catching" the agent with label 1.

Of course, the agents are not in such an ideal situation. However, keep in mind that the binary representations of their (distinct) labels are composed of 0 and 1. Hence, using similar strategies to those introduced in Section 2.2.1 for asynchronous rendezvous in the plane (*e.g.,* acting in phases in which we make assumptions on the graph size and the smaller label, transforming the labels and letting, within each phase, the agents process the resulting bits one by one, etc.), we can recreate

[2]We detail what is behind such sequences in the introduction of Chapter 4.

approximately the same ideal conditions at a cost polynomial in the graph size and in the logarithm of the smaller label.

Although, they share some similarities, it should be stressed that our asynchronous rendezvous algorithm in the infinite square grid of [Bouchard *et al.* 2019] (leading to a solution in the plane) and our asynchronous rendezvous algorithm of [Dieudonné *et al.* 2015] in arbitrary finite graphs are not comparable and that one cannot be used to design the other. First, the algorithm of [Dieudonné *et al.* 2015] does not have a cost polynomial in the initial distance separating the agents and in the logarithm of the smaller label. Actually, ensuring rendezvous at this cost is even impossible in an arbitrary graph, as witnessed by the case of the clique with two agents labeled 0 and 1: the adversary can hold one agent at a node and make the other agent traverse $\Theta(n)$ edges before rendezvous, in spite of the initial distance 1. Second, the validity of the strategy outlined above, which is based on procedure `Explo`(n), closely relies on the fact that both agents must evolve in the same *finite* graph, which is clearly not the case in [Bouchard *et al.* 2019]. In particular, even when considering the task of rendezvous in an infinite oriented grid, the natural attempt consisting in making each agent apply the algorithm from [Dieudonné *et al.* 2015] within bounded grids of increasing size and centered in its initial position, does not permit to claim that rendezvous ends up occurring. Indeed, the bounded grid considered by one agent is never exactly the same as the bounded grid considered by the other one (although they may partially overlap), and thus the agents never evolve in the same finite graph which is a necessary condition to ensure the validity of the solution of [Dieudonné *et al.* 2015] and by extension of this natural attempt. Finally, in [Bouchard *et al.* 2019], the agents have a common orientation, which is completely absent in [Dieudonné *et al.* 2015].

In [Bouchard *et al.* 2018c], we analyze the question of whether we can avoid relaxing the requirement of rendezvous and ensure a meeting at a node in a scenario with slightly less asynchrony. More precisely, we assume that the agents have a common measure of time but for each agent X and for each edge e of the graph, the adversary initially assigns a positive real $t(X, e)$. During the execution of an algorithm, agent X can wait at the currently visited node for a time of its choice, or it can choose a port to traverse the corresponding edge e. In the latter case, agent X traverses this edge in time $t(X, e)$, getting to the other end of the edge after this time. In this more lenient scenario, we give in [Bouchard *et al.* 2018c] an algorithm achieving rendezvous (at a node) and having a time complexity that is polynomial in the size n of the network, the logarithm ℓ of the smaller label and the maximum τ of all values $t(X, e)$ assigned by the adversary, over all edges e of the graph, and over all agents X.

It is natural to ask if it is possible to construct a rendezvous algorithm whose time depends on n, ℓ and τ', where τ' is the *minimum* of all values $t(X, e)$ assigned by the adversary, over all edges e of the graph, and over all agents X. The answer is trivially negative, if time is counted, as we do, since the wake-up of the earlier agent. Indeed, suppose that there exists such an algorithm working in some time $F(n, \ell, \tau')$. The adversary assigns $t(X_1, e) > F(n, \ell, \tau')$, for the first edge e taken by

agent X_1, wakes up X_1 at some time t_0 and delays the wake-up of the second agent until time $t_0 + t(X_1, e)$. Rendezvous cannot happen before time $t_0 + t(X_1, e)$, which is a contradiction.

It turns out that the answer is also negative in the easier case where time would be counted since the wake-up of the agent that is woken up later.

To be convinced of this, consider even a simplified situation, where the adversary assigns to the agent having the smaller (resp. larger) label the same speed t_1 (resp. t_2) for all edges. In other words, each of the agents has a constant speed, and thus, $\tau = \max(t_1, t_2)$ and $\tau' = \min(t_1, t_2)$. Call the agent for which the constant speed is larger the *slower* agent, and the other – the *faster* agent.

We now show that for any rendezvous algorithm $\mathcal{A}$, there exists a behavior of the adversary for which this algorithm takes time at least τ since the wake-up of the later agent.

Denote by β (resp. by γ) the waiting time imposed by $\mathcal{A}$ between the wake-up of the faster (resp. slower) agent and the time when it starts the first edge traversal. Let $d = |\beta - \gamma|$.

If $\beta \geq \gamma$, the adversary wakes up the faster agent at some time t_0 and wakes up the slower agent at time $t_0 + d$. Both agents start traversing their first edge at the same time $t_0 + \beta$ and cannot meet before time $t_0 + \beta + \tau$. Since both agents were awake at time $t_0 + \beta$, our claim follows.

If $\beta < \gamma$, the adversary wakes up the slower agent at some time t_0 and wakes up the faster agent at time $t_0 + d$. Both agents start traversing their first edge at the same time $t_0 + \gamma$ and cannot meet before time $t_0 + \gamma + \tau$. Since both agents were awake at time $t_0 + \gamma$, our claim follows. This concludes the justification that it is impossible to guarantee rendezvous in time depending on n, ℓ and τ', even if time is counted since the wake-up of the later agent.

2.3 Rendezvous with delay faults in graphs

In [Chalopin *et al.* 2016], we consider rendezvous in `GBM` but by assuming that agents are subject to *delay faults*: if an agent incurs a fault in a given round, it remains in the current node, regardless of its decision. When a fault happens in a round r, the agent is aware of it if, and only if, it planned to move in round r.

Under these circumstances, three scenarios of fault distribution are analyzed: random (independently in each round and for each agent with constant probability $0 < p < 1$), unbounded adversarial (the adversary can delay an agent for an arbitrary finite number of consecutive rounds) and bounded adversarial (the adversary can delay an agent for at most c consecutive rounds, where c is unknown to the agents). We present our results in the following three subsections.

2.3.1 Random faults

For random faults, we give an algorithm with a time complexity polynomial in the size of the network, which achieves rendezvous with very high probability in

arbitrary networks (the algorithm is deterministic, but, due to the stochastic nature of faults, the estimate of its time can be only given with some probability).

The high-level idea behind this rendezvous algorithm relies on the well-known fact that two random walks operating in an n-node network, without faults, meet after polynomial time with probability $1 - n^{-c}$, for some positive constant c. This follows, e.g., from the fact that the cover time of a random walk is polynomial in any graph and from [Aldous 1991]. By suitably changing the polynomial, the probability can be raised to very high, i.e., the error probability decreased to inverse-exponential in n. The problem with applying this fact is that it concerns random walks, i.e., an algorithm in which the agents have access to random bits, while we want a deterministic algorithm working with very high probability in the presence of random independent delay faults. Thus, the issue is how to simulate a random walk by a deterministic algorithm in this faulty environment. Actually, this can be done by harvesting randomness from the random independent faults occurring in the network.

To do so, our rendezvous algorithm makes use of a procedure called *unbiased random bit production* (`URBP`). Let u be a position of the executing agent and let v be an adjacent node. The procedure works in two rounds as follows. In each round, the agent attempts a move along the edge $\{u, v\}$. In the case where exactly one of these two attempts is successful, the procedure outputs bit 1 if the first attempt is successful and outputs bit 0 if it is the second attempt that is successful. Otherwise, the procedure outputs nothing. Clearly, a bit obtained in this way is unbiased (has probability $\frac{1}{2}$).

Let $Q(.)$ be a polynomial such that for every integer $n \geq 2$, two simultaneous random walks meet in any n-node network, with very high probability, after time at most $Q(n)$ rounds. Our rendezvous algorithm based on random walks works in epochs $n = 2, 3, 4 \ldots$. Each epoch is divided into two parts: *bit preparation* and *execution*. The bit preparation part of the nth epoch consists of repeating procedure `URBP` $nk \cdot Q(n)$ times, where $k = \lceil \log n \rceil$. By Chernoff bound, for a sufficiently large constant q depending on fault probability p, repeating procedure `URBP` $qk \cdot Q(n)$ times is enough in order to produce $k \cdot Q(n)$ random independent unbiased bits with very high probability. Hence, for n large enough (i.e. $n \geq q$), at least $k \cdot Q(n)$ such bits are obtained with very high probability at the end of the bit preparation part of the nth epoch.

The execution part of the nth epoch is as follows. In a given round of this epoch, some prefix of the sequence of $k \cdot Q(n)$ bits prepared in the bit preparation part is already *used*. The agent, currently located at a node of degree d takes the first unused r bits, where $r = \lceil \log d \rceil$. If this string of bits is the binary representation of an integer $i < d$, then the agent attempts a move by port i. Otherwise, the agent stays at the current node. The string of r bits is added to the prefix of used bits. The execution part of the nth epoch lasts $Q(n)$ rounds. If the total string of prepared bits is of length at least $k \cdot Q(n)$, there are enough bits to execute $Q(n)$ rounds of the execution phase, regardless of the degrees of the nodes. If there are not enough bits to perform the execution part, the agent waits inert at the node at which it ran

out of the bits to perform the rest of the random walk, until $Q(n)$ rounds of the execution part have elapsed. If rendezvous does not occur by the end of the nth epoch, the agent starts the $(n+1)$th epoch.

It follows from the above description that the execution part of the nth epoch is a random walk in which the agent, currently located at a node of degree d takes each port with equal probability $(1-p)/2^{\lceil \log d \rceil}$ and stays inert with probability $1-d(1-p)/2^{\lceil \log d \rceil}$. Let m be the maximum value between the graph size and the constant q (note that since q and the graph size are unknown to the agents, so is value m). If the later agent is not woken up before the earlier agent finishes its mth epoch, then the meeting occurs with high probability at the starting node of the later agent before the earlier agent starts its $(m+1)$th epoch. Otherwise, when the later agent wakes up, the earlier agent executes a round of the k-th epoch for some $k \leq m$. Hence, it can be shown that there exists a polynomial $K(m)$ for which by the time the earlier agent executes the last round of the $K(m)$th epoch, both agents performed simultaneously random walks during at least $Q(m)$ consecutive rounds, with very high probability. Consequently, in all cases, our algorithm ensures rendezvous with very high probability at a time polynomial in m, and thus at a time polynomial in the graph size.

2.3.2 Unbounded adversarial faults

As mentioned earlier, we also consider in [Chalopin *et al.* 2016], a scenario in which the agents can face unbounded adversarial faults: the adversary can delay each of the agents for any finite number of consecutive rounds. Under this harsh fault scenario, it turns out that feasibility of rendezvous is usually not guaranteed, even for quite simple graphs. For the sake of intuition, we provide below a sketch of the arguments allowing to prove this impossibility result.

Consider the family $\mathcal{F}$ of oriented rings of even size: a ring is said to be oriented if at each node the edge going clockwise has port number 0 and the edge going counterclockwise has port 1. Also consider a solo execution[3] of a (supposed) rendezvous algorithm $\mathcal{A}$ by an agent A in a ring $G \in \mathcal{F}$. In each round r of its solo execution, agent A can either decide to stay at the current node or try to traverse an edge by choosing some port. This decision depends on the label of the agent and its history before round r, i.e., on the entire knowledge it has in this round. Since the port labeling is homogeneous, the agent does not learn anything about the graph during navigation: it can differentiate neither G from any ring $G' \in \mathcal{F}$, nor any two nodes in G (recall that we *do not* assume knowledge of any upper bound on the size of the graph). It follows that the history of the agent before round r can be coded by the sequence of previous round numbers in which the agent made a move. (The agent made a move in a previous round if, and only if, it tried to traverse an edge and the adversary allowed the move by not imposing a fault in this round.) Hence, if the algorithm, the agent's label and the adversary's behavior are fixed, the agent's

[3] A solo execution of an algorithm by some agent is an execution in which this agent is alone in the graph.

history, in any round, is necessarily the same in any graph of $\mathcal{F}$ and for any starting node.

Now, consider the decisions of agent A starting in some round t and before its next move. We say that the agent *attacks* since round t, if, from round t on, it tries an edge traversal in each round until it makes the next move. Note that, within a given attack, the edge which the attacking agent attempts to traverse, is not always necessarily the same. We say that the agent does not attack after round t, if there does not exist a round $t' > t$ in which the agent starts attacking. In other words it means that, after round t, the agent decides to stay at the current node in rounds with arbitrarily large numbers, interleaved with possible attempts at a move, all of which can be prevented by an adversary.

It is important to note that an adversary must eventually allow an agent that attacks to make a next move, but it is capable of preventing an agent that never attacks after a given round from making any further move by imposing faults in all rounds in which the agent tries an edge traversal.

Using the above observations and explanations, we can show there exists a behavior of the adversary inducing an infinite increasing sequence $X = \tau_1, \tau_2, \ldots, \tau_i, \ldots$ of positive integers, such that, for every integer $1 \leq \lambda \leq 3$, one of the following conditions is respected in every solo execution, by an agent with label λ, of Algorithm $\mathcal{A}$ in every graph of $\mathcal{F}$:

- **Condition 1:** the agent attacks only a finite number $k \geq 0$ of times, in which case the agent either does not move if $k = 0$, or moves exactly in rounds $\tau_1, \tau_2, \ldots, \tau_k$ otherwise;
- **Condition 2:** the agent attacks infinitely many times, in which case it moves exactly in rounds τ_i, for all positive integers i.

These two conditions can then be used to show contradictions in two complementary cases.

The first case is when there exist two positive integers $\lambda_1 \neq \lambda_2$ that are most equal to 3 such that the solo executions, induced by X, of agents with labels λ_1 and λ_2 in any ring of $\mathcal{F}$ respect Condition 1. In this case, if these agents are placed at antipodal nodes in an oriented ring of $\mathcal{F}$ having a sufficiently large size, then the adversary, acting against each of them as in their respective solo execution induced by X, can make the agents stop forever before the total number of their moves is equal to half the size of the ring. Hence, the agents never meet, which is a contradiction.

The second case is when there exist two positive integers $\lambda_1 \neq \lambda_2$ that are most equal to 3 such that the solo executions, induced by X, of agents with labels λ_1 and λ_2 in any ring of $\mathcal{F}$ respect Condition 2. If the agents with these labels start simultaneously at an odd distance in a ring of $\mathcal{F}$, then both agents can be forced by the adversary to move in exactly the same rounds, i.e., the rounds of sequence X and they remain at an odd distance forever (since an even ring is bipartite, the parity of their distance never changes), which is again a contradiction.

2.3.3 Bounded adversarial faults

The last kind of fault that is considered in [Chalopin *et al.* 2016] corresponds to what we call bounded adversarial faults. Here, the adversary can delay each of the agents for at most c consecutive rounds, where c is a positive integer, called the *fault bound*. In this scenario, we show an algorithm solving rendezvous in any arbitrary networks, which does not require the agents to know initially the fault bound c.

In order to grasp some of the basic ideas of this algorithm, let us just focus here on a simpler situation in which this bound c is initially known to all agents. In this situation, given any synchronous rendezvous algorithm $\mathcal{A}$ working for arbitrary networks with a time complexity T when there are no faults, it is possible to obtain an algorithm working for bounded adversarial faults and for arbitrary networks with time complexity $\mathcal{O}(cT)$. Let us explain how.

Consider the following algorithm $\mathcal{A}(c)$ working for bounded adversarial faults with parameter c. Each agent replaces each round r of the execution of Algorithm $\mathcal{A}$ (in the scenario without any faults) with a segment of $2c+1$ rounds (in the scenario with faults). If in round r of the execution of $\mathcal{A}$ the agent is idle, this round is replaced by $2c+1$ consecutive rounds in which the agent is idle. If in round r the agent leaves its current node by port p, this round is replaced by a segment of $2c+1$ rounds in each of which the agent makes an attempt to leave the current node v by port p until it succeeds, and in the remaining rounds of the segment it stays idle at the node adjacent to v that it has just entered.

Note that, if the later agent does not move before the earlier agent finishes processing its Tth segment, then the meeting necessarily occurs at the starting node of the later agent at most $(2c+1)T \in \mathcal{O}(cT)$ rounds after the wake-up of the earlier agent. Hence, we can assume in the rest of our explanations that the later agent makes at least a move before the earlier agent finishes processing its Tth segment. This permits in turn to make the associations given below.

We associate the first segment of the later agent with the (unique) segment of the earlier agent that it intersects in at least $c+1$ rounds. Let it be the ith segment of the earlier agent. We then associate the jth segment of the later agent with the $(j+i-1)$th segment of the earlier agent, for $j > 1$. Hence, regardless of the delay between starting rounds of the agents, corresponding segments intersect in at least $c+1$ rounds. If the agents met at node x in the jth round of the later agent, by applying Algorithm $\mathcal{A}$ in the scenario without faults, then, by applying $\mathcal{A}(c)$ in the scenario with faults, in the last $c+1$ rounds of its jth segment the later agent is at x and in the last $c+1$ rounds of its $(j+i-1)$th segment the earlier agent is at x. Since these segments intersect in at least $c+1$ rounds, there is a round common to both these segments in which both agents are at node x during the execution by the agents of Algorithm $\mathcal{A}(c)$, regardless of the actions of the adversary, which are permitted by the bounded adversarial fault scenario. This shows that Algorithm $\mathcal{A}(c)$ is correct and solves rendezvous in time $\mathcal{O}(cT)$ as each segment has length $2c+1$ and $i+j$ is necessarily at most equal to T. Moreover, the cost complexity of Algorithm $\mathcal{A}(c)$ is the same as that of Algorithm $\mathcal{A}$, because in each segment

corresponding to an idle round in the execution of Algorithm $\mathcal{A}$, the agent stays idle in the execution of Algorithm $\mathcal{A}(c)$ and in each segment corresponding to a round in which an agent traverses an edge in the execution of Algorithm $\mathcal{A}$, the agent makes exactly one traversal in the execution of Algorithm $\mathcal{A}(c)$. Consequently, since there are algorithms working in GBM with a time complexity polynomial in the size of the network and in the logarithm of the smaller label (see, *e.g.,* [Bouchard *et al.* 2018c]), we can guarantee for $\mathcal{A}(c)$ a cost (resp. time) complexity that is polynomial in the size of the network and in the logarithm of the smaller label (resp. polynomial in the size of the network, the bound c and in the logarithm of the smaller label).

Things become more complex when c is unknown. However, by using the above principles and additional algorithmic components allowing, under certain conditions, the agent to "guess" bound c, we show in [Chalopin *et al.* 2016] an algorithm solving rendezvous in presence of bounded adversarial faults at a cost polynomial in the network size and in the logarithms of c and of the larger label. By contrast, the time complexity of this algorithm is slightly worse as it is polynomial in the network size, in c and in the larger label. It remains open whether there exists a rendezvous algorithm with bounded adversarial faults, working for arbitrary graphs, whose both cost and time are polynomial in the size n of the graph, and polylogarithmic in the fault bound c and in the smaller label.

Chapter 3

Gathering

Contents

3.1 Introduction

The problem of gathering is a generalization of the rendezvous problem to a situation where there are two or more agents to bring together. Both these problems being inherently close to each other, most of the related work presented in the introduction of the previous chapter could also have had its place here. Thus, we will be naturally much less expansive in the introduction of the present chapter.

At first glance, one might be tempted to think that this generalization is not particularly intriguing: after all, would it not be easy to derive solutions for the gathering problem using algorithms initially dedicated to rendezvous?

While this may be true in lots of situations, there are situations where such derivations may not be that easy (or even possible). For instance, this may be the case when agents do not know beforehand the size of the team to which they belong, and do not have any material capacities allowing them to determine it directly. Indeed, this kind of situation raises the delicate question of termination detection, which consists in deciding if everyone is together. Answering this question is trivial for an agent in the context of rendezvous,[1] as the task is simply terminated

[1] Provided that it can at least detect in some way whether it is with another agent or not.

as soon as it is no longer alone, but it may become a real challenge, depending on the considered assumptions, in the more general context of gathering. In fact, in some cases, the termination detection cannot simply be achieved, as the agents may never be sure whether there are any other agents that would be somewhere else, far "away", also looking for their peers.

Another situation in which the generalization can become interesting is when we explore scenarios for which the problem of meeting is trivial or conversely impossible to solve with only two agents. An emblematic illustration of this kind of situation is given in the seminal paper [Suzuki & Yamashita 1999]. In this paper, the authors show a fundamental impossibility result in a well-known semi-synchronous model (a.k.a. SYm that stands for Suzuki and Yamashita's model) where the agents move in the plane, as in `PBM`, but are anonymous, have unlimited visibility and operate in synchronous cycles in which they are not necessarily always active. Precisely, in this model, they show it is impossible to gather deterministically two agents if they are oblivious (*i.e.,* remember nothing from the previous cycles) and devoid of a common orientation. On the other hand, in the same context, but with at least three agents all starting from distinct positions, there exists a deterministic algorithm solving the gathering problem (provided the agents have the ability to detect, at any point on the plane, whether there are more than one agent present or not). These results remain true even when adding some level of asynchrony among the agents [Cieliebak *et al.* 2012].

The case where there may be faulty agents is also an illustration where gathering often brings a special "flavor" to the problem of meeting that is lacking with rendezvous. Besides, the scale-up when considering numerous agents is obviously tied to the occurrence of faults among them, which makes important, if not essential, to be provided with studies dedicated to gathering in fault-prone environment. This is witnessed by the large number of studies on the subject, both in graphs and in the plane, *e.g.,* [Agmon & Peleg 2006, Défago *et al.* 2006, Dieudonné & Petit 2012, Bouzid *et al.* 2013, Park & Hutchinson 2017, Pelc 2018, Das *et al.* 2019, Pattanayak *et al.* 2019, Défago *et al.* 2020, Miller & Saha 2020, Bramas *et al.* 2023]. For more information about this matter, the curious reader is referred to the survey [Défago *et al.* 2019], which gives a good overview of the fault-tolerance literature (albeit mainly in plane environments) for gathering but also for some other fundamental tasks in mobile distributed systems such as leader election, scattering or flocking.

What did we do about the problem of gathering? As in most of the aforementioned studies, our research efforts focused on specific settings in which gathering becomes really much more interesting than just rendezvous. When we started to study this problem in the early 2010s, all deterministic gathering algorithms working in anonymous graphs required the agents to either see all the nodes (*e.g.,* [Klasing *et al.* 2008, Klasing *et al.* 2010, D'Angelo *et al.* 2012]) or have the capability to mark nodes using either whiteboards or pebbles/tokens (*e.g.,* [Yu & Yung 1996, Flocchini *et al.* 2004, Barrière *et al.* 2007, Chalopin *et al.* 2007]).

In the light of this, we then investigated the problem in arbitrary anonymous

graphs assuming the agents were devoid of both these capabilities, but also assuming additional constraints making the task of meeting even more hard to solve. Precisely, in [Dieudonné & Pelc 2016] (resp. [Dieudonné *et al.* 2014, Bouchard *et al.* 2016, Bouchard *et al.* 2022]), we studied the task of gathering in `GBM` for a team made of two or more agents under the constraint that they are all anonymous (resp. they all have pairwise distinct labels but are infiltrated by at most f Byzantine agents): the obtained results are described below in Section 3.2 (resp. Section 3.3). In all these studies, the agents sharing the same node are supposed to be able to exchange all currently held information. Is this ability really necessary? In [Bouchard *et al.* 2023b], we embraced this question and provided interesting answers that are detailed in Section 3.4.1.

In the rest of this chapter, when we speak of an algorithm solving the gathering problem, it will always be understood that it includes the termination detection by all the agents, through a declaration that the gathering is done, unless explicitly stated otherwise.

3.2 Anonymous gathering

In [Dieudonné & Pelc 2016], we study the problem of gathering anonymous agents in arbitrary graphs. Concretely, we consider model `GBM`, but under the assumptions that:

- The team is made of two or more agents that are all unlabeled and that all start from distinct nodes.
- The adversary wakes up some of the agents (at least one) in possibly different rounds.
- When several agents are at the same node in the same round, they can exchange all information they currently have.

An initial configuration of agents is called *gatherable* if there exists a deterministic algorithm, even dedicated to this particular configuration, that achieves meeting of all agents in one node. Which configurations are gatherable and how to gather all of them deterministically by the same algorithm?

In our paper, we give a complete solution to this problem in arbitrary networks. We characterize all gatherable configurations and give two algorithms. The first algorithm works under the assumption that a common upper bound n on the size of the network is known to all agents and guarantees gathering (with detection), in polynomial time in n, from any gatherable configuration. If no upper bound on the size of the network is known, we show that no algorithm can solve gathering with detection from all gatherable configurations. Hence, for this harder scenario, we construct a second gathering algorithm, which guarantees, for any gatherable configuration, that all agents eventually get to one node and stop, although they cannot tell if gathering is over. The time complexity of this second algorithm is

polynomial in the (unknown) size of the network. Further insights about these results are distilled in Sections 3.2.1 and 3.2.2.

It is worth noting that while gathering only two anonymous agents in the present context is a relatively easy task when feasible (cf. [Czyzowicz *et al.* 2012a]), our problem of gathering an unknown team of anonymous agents presents the following major difficulty. The asymmetry of the initial configuration because of which gathering is feasible, may be caused not only by non-similar locations of the agents in the graph (*e.g.,* when the degrees of the starting nodes are different), but also by their different situation *with respect to other agents*. Hence, a new algorithmic idea is needed in order to gather: agents must make decisions based on the memories of other agents met to date, in order to distinguish their future behavior. In the beginning, the memory of each agent is a blank slate and in the execution of the algorithm it records what the agent has seen in previous steps of the navigation and what it heard from other agents during meetings. Even a slight asymmetry occurring in a remote part of the graph must eventually influence the behavior of initially distant agents. Notice that agents in different initial situations may be unaware of this difference in early meetings, as the difference may be revealed only later on, after meeting other agents. Hence, for example, an agent may mistakenly "think" that two different agents that it met in different stages of the algorithm execution, are the same agent. Confusions due to this possibility are a significant challenge that is absent when we deal with only two anonymous agents.

3.2.1 Known upper bound on the size of the graph

The aim of this subsection is to give the characterization of the gatherable configurations as well as the high-level idea of our algorithm that ensures gathering (with detection) from all of them, when an upper bound on the size of the graph is known to the agents.

The condition that must be met by a configuration to be gatherable is based on two notions. The first notion is that of *view* (firstly introduced in [Yamashita & Kameda 1996]). Let v be a node of the underlying graph G. The view of node v is defined recursively. The view of depth 0 $T^0(v)$ of node v, consists of a single node called the root. The view of depth $k \geq 1$ of v, $T^k(v)$, is a port-labeled tree that is constructed by taking a node x_0, which will become the root of the resulting tree, and, for every neighbor v_i of v in G, connecting the tree $T^{k-1}(w)$ of root x_i to node x_0 with an edge between x_0 and x_i such that $\texttt{port}(x_0, x_i) = \texttt{port}(v, v_i)$ and $\texttt{port}(x_i, x_0) = \texttt{port}(v_i, v)$. Now, the view $T(v)$ of node v is the infinite rooted tree with labeled ports, such that its truncation to depth l is $T^l(v)$ for every integer $l \geq 0$. The second notion is that of *enhanced view* that is similar to that of view but reflecting the positions of agents in an initial configuration. Precisely, the *enhanced view* of node v is the couple $(T(v), f)$, where f is a binary valued function defined on the set of nodes of $T(v)$, such that $f(w) = 1$ (resp. $f(w) = 0$) if by following from node v in G, the same shortest sequence of ports that leads to w from the root of $T(v)$, we reach a node that is occupied (resp. is not occupied). Loosely speaking,

the enhanced view of a node of G corresponds to its view in which we additionally mark the nodes representing the occupied positions in G. By misuse of language, we will use the expression "view of an agent" (resp. "enhanced view of an agent") to designate the view (resp. enhanced view) of its initial starting node.

How are these notions involved in our characterization? In fact, in [Dieudonné & Pelc 2016] we show that any initial configuration is gatherable if, and only if, it respects the following condition Φ: there exist agents with different views, and each agent has a unique enhanced view.

Let us start explaining why condition Φ is necessary. First, suppose that an initial configuration C does not satisfy the first part of condition Φ, i.e., that the views of all agents are identical. If the adversary wakes up every agent in the same round in configuration C, then no pair of agents can meet. Indeed, assume, for the of contradiction, that some agents A_1 and A_2 are the first to meet at some round t (counted from the common start). Since the initial views of the agents are identical and they execute the same deterministic algorithm, we can easily show that they take the exact same decisions and see the exact same things in each round till round t, and hence they both enter the node at which they first meet by the same port. This implies that A_1 and A_2 are already together in round $t-1$, which is a contradiction showing that the first part of condition Φ must be verified.

Now, suppose that a configuration C does not satisfy the second part of condition Φ, i.e., that there exists two distinct agents A and A' that have identical enhanced views. Let $B \neq A$ be any agent, and suppose that q is a sequence of port numbers that leads from the initial position of agent A to that of agent B in the enhanced view of agent A. (Notice that there may be many such sequences, corresponding to different paths leading from A to B.) Then there exists an agent $B' \neq A'$ such that q is a sequence of port numbers that leads from the initial position of agent A' to that of agent B' in the enhanced view of agent A'. It can be easily shown that agent B' has the same enhanced view as agent B and, since agents A and A' are different, agents B and B' are different as well. Thus, for every agent, there exists another agent whose enhanced view is identical. Call such agents *homologs*.

If the adversary wakes up all agents in the same round in configuration C, then, although some meetings of agents are possible, homologs will never meet. Indeed, suppose by contradiction that two homologs A and A' meet for the first time in some round t. Since A and A' have the same enhanced view at the beginning, it can be shown by induction on the round number that their memory will be identical in every round (in particular, it can be shown that whenever A meets an agent B in some round, its homolog A' meets a homolog B' of B in the same round). This means that they both enter the node at which they first meet in round t by the same port. Hence, A and A' are already together in round $t-1$, which is a contradiction proving that the second part of condition Φ, and by extension condition Φ itself, must be verified by any gatherable configuration.

Showing that respecting condition Φ is sufficient for a configuration to be gatherable is much more difficult. In [Dieudonné & Pelc 2016], it is shown by constructing an algorithm that takes as input an upper bound on the size of the graph and that

accomplishes gathering with detection of all agents, when executed by agents starting from any initial configuration satisfying condition Φ. (Note that, this is enough to prove that condition Φ is sufficient, as for any specific configuration satisfying this condition even a gathering algorithm dedicated to this specific configuration does the trick to show that it is gatherable, and such a dedicated algorithm "knows" the exact size of the graph. Of course, our algorithm accomplishes much more: knowing just an upper bound on the size of the graph, it gathers *all* configurations satisfying condition Φ.) Our algorithm works in time polynomial in the upper bound on the graph size given as input. Therefore, if this upper bound is polynomial in the graph size, the algorithm is polynomial in the graph size as well.

We close this subsection by outlining below some of the ideas that are behind this algorithm.

At a high level, the algorithm works in two stages. The aim of the first stage for any agent is to meet another agent, in order to perform later an exploration in which one of the agents will play the role of a token and the other the role of an explorer (roles will be decided comparing memories of the agents: any two agents that first meet in a round t at a node v, necessarily enter this node in round t by distinct ports or exactly one of both agents already occupies node v in round $t-1$). To this end, once it is woken up, an agent starts an exploration of the graph using procedure `Explo`(n) (introduced in Section 2.2.2), where n is the known upper bound on the graph size. The aim of this exploration is to wake up all, possibly still dormant agents. Afterward, in order to create meetings, agents use a procedure from [Ta-Shma & Zwick 2014], that aims at gathering two agents with distinct labels in `GBM`, assuming they appear at their starting positions in possibly different rounds (originally, it is assumed in `GBM` that all agents are initially present in the network and thus the scenario of [Ta-Shma & Zwick 2014] is harder to handle). We call this procedure `TZ`(l), where l is a positive integer given as input parameter. If two agents with distinct labels execute this procedure by assigning their label to the input parameter (they do not need to know an upper bound on the number of nodes), then the meeting of the two agents is guaranteed in a polynomial time $\mathcal{P}_{TZ}$ in the network size and in the logarithm of the smaller of both labels after the appearance of the later agent (the polynomial increases in each of the variables). This remains true even if one of the agents never move after its appearance. However, our agents have no label... One way to differentiate some agents and give them labels is to find their views: the views (truncated to n) from the initial positions of the agents could serve as labels because the first part of condition Φ ensures that there are at least two distinct views, and thus this would lead to get at least two distinct labels, which is enough for our purpose. Nevertheless, it is *a priori* not at all clear how to find the view of an agent (truncated to n) in time polynomial in n (the size of the view truncated to n is exponential in n). To circumvent this issue, we use a procedure from [Czyzowicz *et al.* 2012a] that we call `Sign`(n), where n is the known upper bound on the graph size given as input parameter. The time complexity of this procedure is polynomial in n and it allows an agent to construct a label of polynomial size in n such that the obtained labels of two agents are different if their

views are initially different. Then, performing procedure `TZ` for a sufficiently long time with such constructed labels, can enable us to guarantee a meeting for every agent.

In the second stage, each explorer explores the graph using procedure `Explo`(n) and then backtracks to its token left at the starting node of the exploration. After the backtrack, memories of the token and of the explorer are updated to check for anomalies caused by other agents meeting in the meantime either the token or the explorer. Note that, due to the fact that some agents may have identical memories at this stage of the algorithm, an explorer may sometimes falsely consider another token as its own, due to their identical memories. By contrast, an end of each backtrack is a time when an explorer can be sure that it is on its token and the token can be sure that its explorer is with it.

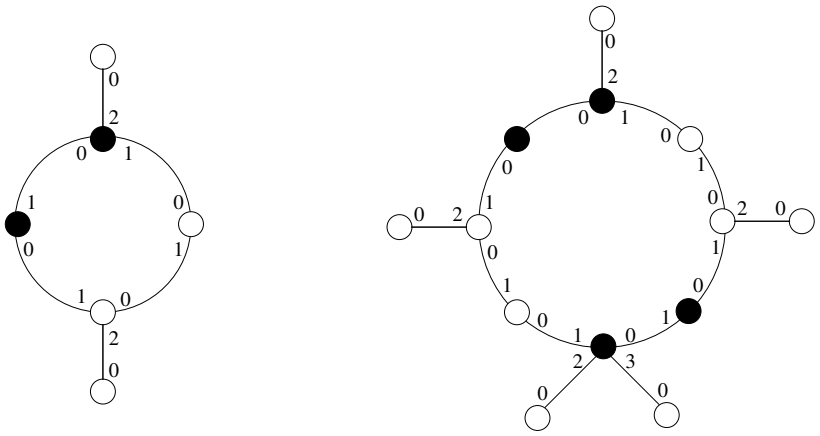
Explorers repeat these explorations with backtrack again and again, with the aim of creating meetings with other agents and detecting anomalies. As a consequence of these anomalies, some agents merge with others, mergers being decided on the basis of the memories of the agents. Each explorer eventually either merges with some token or performs an exploration without anomalies. In the latter case it waits a prescribed amount of time with its token: if no new agent comes during the waiting time, the end of the gathering is declared, otherwise another exploration is launched. It can be proved that eventually, due to the second part of condition Φ, all agents merge with the same token A and then, after the last exploration made by the explorer of A and after undisturbed waiting time, the end of the gathering is correctly declared.

3.2.2 Unknown upper bound on the size of the graph

If no upper bound on the size of the graph is known, then we show there is no algorithm for gathering *with detection* all gatherable configurations. Nevertheless, we still show in this case an algorithm that gathers all gatherable configurations: all agents from any gatherable configuration eventually stop forever at the same node (although no agent is ever sure that gathering is over). Our algorithm is polynomial in the (unknown) size of the graph. Below, we offer a closer look at these results, and we start with the negative one.

Consider the following initial configurations. In configuration C, the graph is a 4-cycle with clockwise oriented ports 0,1 at each node, and with additional nodes of degree 1 attached to two non-consecutive nodes. There are two agents starting at a node of degree 2 and at its clockwise neighbor, (cf. Fig. 3.1 (a)). In configuration D_k, where k is a positive integer, the graph is constructed as follows. Take a cycle of size $4k$ with clockwise oriented ports 0,1 at each node. Call clockwise consecutive nodes of the cycle $v_0, \ldots, v_{4k-1}$ (names are used only to explain the construction) and attach two nodes of degree 1 to v_0 and one node of degree 1 to every other node with even index. Initial positions of agents are at nodes v_i, where $i = 4j$ or $i = 4j-1$, for some j (cf. Fig. 3.1 (b)).

Each of the configurations C and D_k, for $k \geq 2$, is gatherable. Indeed, in each



(a) CONFIGURATION C (b) CONFIGURATION D_8

Figure 3.1: An illustration of configurations C and D_8. Black nodes correspond to nodes occupied by agents

of these configurations there exist agents with different views (agents starting at nodes of degree 2 and of degree 3) and each agent has a unique enhanced view (this is obvious for configuration C and follows from the existence of a unique node of degree 4 for configurations D_n). Hence, each of these configurations satisfies condition Φ and consequently, there is an algorithm for gathering with detection each of them (cf. Section 3.2.1). However, there is no algorithm that gathers with detection all configurations C and D_k for every $k \geq 2$, in the absence of any additional knowledge. Indeed, suppose, for contradiction, that $\mathcal{A}$ is such an algorithm. Also suppose that the adversary always wakes up all agents simultaneously, and let $t \geq 1$ be the time after which agents in configuration C stop at the same node and declare that gathering is over. Consider the configuration D_{2t} and two consecutive agents antipodal to the unique node of degree 4, i.e., starting from nodes v_{4t} and v_{4t-1}. Call X the agent starting at a node of degree 2 in configuration C and call Y the agent starting at its clockwise neighbor (of degree 3) in this configuration. Call X' the agent starting at node v_{4t-1} and call Y' the agent starting at node v_{4t} in configuration D_{2t}.

In the first t rounds of the executions of algorithm $\mathcal{A}$ starting from configurations C and D_{2t} the memories of the agents X and X' and of the agents Y and Y' are the same. This easily follows by induction on the round number. Thus, after t rounds, agents X' and Y' starting from configuration D_{2t} stop and declare that gathering is over. However, this is impossible, as the distance between the agents that are initially at nodes v_0 and v_{4t} is at least $2t$ in round t, which contradicts the existence of Algorithm $\mathcal{A}$.

We now turn our attention to our positive result, by describing the high-level idea of our algorithm that accomplishes gathering without detection from any gatherable

configuration, in the absence of any additional knowledge.

Since in our present scenario, no upper bound on the size of the graph is known, already guaranteeing any meeting between agents must be done differently than when such an upper bound is initially known. Hence, after waking up, each agent proceeds in phases $i = 1, 2, \ldots$, where in phase i it "supposes" that the graph has size at most 2^i. In each phase, an appropriate label based on procedure `Sign`(2^i) (cf. Section 3.2.1) is computed and, using this label, procedure `TZ` (cf. Section 3.2.1) is performed sufficiently long to guarantee a meeting at most at the end of phase $\lceil \log_2 m \rceil$, where m is the real size of the graph. If no meeting occurs in some phase for a sufficiently long time, the agent starts the next phase.

Another important difference occurs after the meeting, when one of the agents becomes an explorer and the other its token. Unlike in the case of a known upper bound on the size of the graph, there is no way for any explorer to be sure at any point of the execution that it has already visited the entire graph. Clearly procedure `Explo` involved in the previous scenario (cf. Section 3.2.1), would be of no use to guarantee that, as it requires as input an upper bound on m that is alas unknown. True, the explorer can somehow simulate BFS using as a marker the agent playing the role of its token, which does not require the knowledge of an upper bound. In fact, there exists a procedure based on BFS [Chalopin *et al.* 2010], which we will call `EST` (standing for Exploration With a Stationary Token), that allows an exploring agent to learn the graph size m and visit all the nodes in polynomial time in m, if it can always distinguish unambiguously its starting node. Thus, simulating `EST`, by considering to be at the starting node when encountering the agent playing the role of its token, may sound like the perfect idea. Unfortunately, in our context, an explorer cannot be always sure that it visits its own token, because memories of several agents playing the role of tokens can be identical at various stages of the execution, and hence these "tokens" may be indistinguishable for the explorer. Nevertheless, our algorithm succeeds in accomplishing the task by using a mechanism which is analogous to the "butterfly effect". By repeating simulations of `EST`, even a slight asymmetry in a remote part of the graph is eventually communicated to all agents and ensures that at some point some explorer will visit the entire graph (although in some graphs no explorer can ever be sure of it at any point of an execution) and then all agents will eventually be able to gather at the token of one of these explorers. Making all agents eventually decide on the same token is made possible by condition Φ, and demonstrating it is one of the main difficulties to overcome in the proof of correctness and complexity of our algorithm.

3.3 Byzantine gathering

In this section, we present our contributions [Dieudonné *et al.* 2014, Bouchard *et al.* 2016, Bouchard *et al.* 2022] to the problem of gathering that have been made in model `GBM`, but by assuming:

- The team is made of two or more agents that all have distinct pairwise labels

and distinct pairwise starting nodes.

- Among the agents, at most f are Byzantine.
- The adversary wakes up some of the agents, at least one of which is good (*i.e.,* not Byzantine), in possibly different rounds.
- When several agents are at the same node v in the same round t, they see, for each agent x at node v, the label of agent x and all information it wants to share with the others in round t. This transmission of information is done in a "shouting" mode at the beginning of the round: all the transmitted information by all agents at node v at the beginning of round t becomes common knowledge for agents that are currently at node v in round t.

What do we mean precisely by "Byzantine agent"? In fact, throughout our studies, we have been led to distinguish two kinds of Byzantine agents: the strongly Byzantine and the weakly Byzantine ones. A strongly Byzantine agent can choose an arbitrary port when it moves, can change its label in every round (in particular by forging the label of another agent or by creating a completely new one) and can convey arbitrary information to other agents, while a weakly Byzantine agent can do the same, except changing its label. Needless to say, that the strongly Byzantine agents are much more difficult to handle than the weakly Byzantine ones.

No matter the type, we cannot count on the Byzantine agents to cooperate. Hence, in such a context, we can just hope to gather only the good agents. (Thus, when referring to gathering in the remainder of this section, it is always understood that it concerns the gathering of the good agents only.) This naturally raises the following question: what is the minimum number of good agents that guarantees gathering of all of them? In [Dieudonné *et al.* 2014], we study this question, assuming that f is initially known to all agents: we bring exact answers when coping with weakly Byzantine agents, and give approximate ones when coping with strongly Byzantine agents, both when an upper bound on the size of the network is known and when it is not. For weakly Byzantine agents, we show that any number of good agents permit to solve the problem when an upper bound on the graph size is known. If no such upper bound is known, then this minimum number is $f+2$. More precisely, we show a polynomial algorithm that gathers all good agents in an arbitrary network, provided that there are at least $f+2$ of them. We also provide a matching lower bound: we prove that if the number of good agents is at most $f+1$, then they are not able to solve the gathering problem in some networks.

For strongly Byzantine agents, we come up with negative and positive results too. On the negative side, we give a lower bound of $f+1$, when an upper bound on the graph size is known: we show that f good agents cannot gather in the presence of f Byzantine agents even in a ring of known size. When no upper bound on the graph size is known, the lower bound of $f+2$ that holds for weakly Byzantine agents, is obviously also true for strongly Byzantine agents. On the positive side, we give gathering algorithms for at least $2f+1$ good agents when an upper bound

on the graph size is known, and for at least $4f + 2$ good agents when no such upper bound is known. In [Bouchard *et al.* 2016], we significantly improve on these positive results: when an upper bound on the size of the network is known (resp. no upper bound on the size of the network is known), we provide an algorithm working with at least a number of good agents that perfectly matches the lower bound of $f + 1$ (resp. $f + 2$) identified in [Dieudonné *et al.* 2014]. Consequently, in each of the considered scenarios, we get the exact minimum number of good agents that guarantees gathering of all of them: these results are summarized in Table 3.1. It is worth noticing that when an upper bound on the graph size is known, the gap between the number of good agents permitting gathering with weakly and with strongly Byzantine agents is very significant (1 vs. $f+1$), while this gap completely disappears when such an upper bound is unknown: the minimum number of good agents is then $f + 2$, regardless of whether the bad agents are weakly or strongly Byzantine.

	Weakly Byzantine	**Strongly Byzantine**
Known upper bound on the graph size	1	$f+1$
Unknown upper bound on the graph size	$f+2$	$f+2$

Table 3.1: Exact minimum numbers of good agents guaranteeing gathering of all of them, in various scenarios, in presence of at most f Byzantine agents.

Unfortunately, the algorithms dealing with strongly Byzantine agents in our two papers [Dieudonné *et al.* 2014, Bouchard *et al.* 2016], all have the major disadvantage of having a time complexity that is exponential in the size of the network and the largest label of a good agent. To circumvent this problem, we propose in [Bouchard *et al.* 2022] to make a concession on the proportion of strongly Byzantine agents within the team, in order to get an algorithm offering a significantly lower complexity. Precisely, in this paper, we design a gathering algorithm working in all graphs of size at most n in time polynomial in n and the logarithm of the smallest label ℓ of a good agent, provided the agents are in a *strong team*, *i.e.,* a team where the good agents are at least some quadratic polynomial in f. Our algorithm requires that the agents are initially provided with a common input that can be coded in $O(\log\log\log n)$ bits. Nevertheless, we prove this size of input is asymptotically optimal to obtain a polynomial time complexity in n and ℓ.

The rest of this section is organized as follows. First, in Section 3.3.1, we give the high-level ideas of our algorithms allowing to solve gathering, in the presence of at most f weakly Byzantine agents, with at least one good agent (resp. at least $f+2$ good agents) when an (resp. no) upper bound on the graph size is initially known. Next, in Section 3.3.2, we do a similar thing but with strongly Byzantine agents. Precisely, we give the high-level ideas of our algorithms allowing to solve gathering, in the presence of at most f strongly Byzantine agents, with at least $f + 1$ agents (resp. at least $f + 2$ good agents) when an (resp. no) upper bound on the graph size is initially known. Then, in Section 3.3.3, we explain why our algorithms are optimal in terms of required number of good agents. Finally, in Section 3.3.4, we

provide the intuitions that are behind our algorithm of [Bouchard *et al.* 2022] that solves gathering in polynomial time provided the agents are in a strong team.

3.3.1 Algorithms dealing with weakly Byzantine agents.

In this subsection, we describe the core ideas that are behind our two algorithms from [Dieudonné *et al.* 2014] dealing with weakly Byzantine agents. The first algorithm, which will be called here `WeakByzKnownUpperBound`, permits to solve gathering with any (positive) number of good agents,[2] when an upper bound n on the graph size is *a priori* known. It works in polynomial time in n and in the logarithm of the largest label within the team.

In Algorithm `WeakByzKnownUpperBound`, when an agent is woken up by the adversary, it starts executing the procedure `TZ` (cf. Section 3.2.1), by using its label l as input parameter. At the first meeting with one or more agents, the current procedure `TZ`(l) is interrupted, and the agent starts executing procedure `TZ`(l'), where l' is the minimum of labels among the colocated agents. This way, if all agents are good, they will start traveling together, with the objective of continuing this process in order to get groups of increasing size, till achieving finally the gathering. However, this idea cannot be implemented directly, due to the presence of Byzantine agents. Indeed, the agent of the group with the smallest label l' can be Byzantine and leave the group. Continuing `TZ`(l') is then dangerous because that agent could join another group, thus causing two groups to execute the same `TZ`(l'), and possibly preventing them from meeting each other due to symmetry considerations. Thus, we want to interrupt the current `TZ` and switch to `TZ`(l''), where l'' is the smallest remaining label. However, this solution is vulnerable to another danger. A Byzantine agent with label 1 could meet another agent with label x, leave it, then meet it again, and so on. According to the idea described above, the agent with label x would alternate between executing `TZ`(x) and `TZ`(1), interrupting the current procedure whenever the Byzantine agent labeled 1 joins or leaves. This could go forever, creating a vicious circle, and, by extending this behavior to other agents, the gathering of all good agents might never be accomplished. Hence, we need a mechanism that permits a group of good agents marching together to disregard "rogue" agents that already compromised themselves as Byzantine. To do so, we make use of *blacklists* to indicate such rogue agents. Informally, the blacklist of an agent at a given round t is the largest subset B of agents currently at the same node, such that other agents at this node have *collectively* seen each element of B leave the group in the past, possibly at different times. This definition guarantees that the blacklists of all good agents at a node in a given round are identical. In Algorithm `WeakByzKnownUpperBound`, we adopt the rule that whenever a group of agents changes, they interrupt the current `TZ` and restart it with the smallest label

[2] Note that if there is only one good agent, this means that at the time of stopping for good it may be alone, in which case it learns that there are no other good agents, or it may meet some agents, in which case it only knows that if there are other good agents, they are at the same node. In the latter case, the only good agent cannot deduce that there are no other good agents.

belonging to the *kernel* of the group, which corresponds to the set of all agents' labels of the group that are not blacklisted. Again, good agents will travel together.

Using such an approach, we can prove that the number of times when an agent is interrupted in the execution of procedure `TZ` is less than $2n^2$. Therefore, in view of the definition of the polynomial $\mathcal{P}_{TZ}$ (cf. Section 3.2.1), which is related to procedure `TZ`, it can be proven that if a group of agents execute together `TZ`(μ), for some label μ belonging to their kernel, without interruption for $2n^2 \cdot \mathcal{P}_{TZ}(n, \lceil \log \mu \rceil)$ rounds, then gathering of all good agents is achieved. (In fact, gathering has been already achieved at the beginning of the execution of this `TZ`(μ), but we are sure the agents in the group become aware of it after time $2n^2 \cdot \mathcal{P}_{TZ}(n, \lceil \log \mu \rceil)$.) Since μ is at most equal to the largest label L, we can show that such an event necessarily occurs by round $4n^4 \cdot \mathcal{P}_{TZ}(n, \lceil \log L \rceil)$. Besides, when occurring, all good agents of the group (which must now contain all good agents) can detect it because they all know n, and thus can declare the completion of gathering and stop.

The second algorithm, which will be called here `WeakByzUnknownUpperBound`, permits to solve gathering, when no upper bound on the graph size is known, provided there are at least $f + 2$ good agents. It works in polynomial time in the graph size and in the logarithm of the largest label within the team.

In Algorithm `WeakByzUnknownUpperBound`, each agent of a group also maintains and updates, similarly as in the previous algorithm, a blacklist and a kernel. The algorithm essentially consists of three parts. The aim of part 1 is to gather a group of agents with a kernel of size at least $f + 2$. Once such a group is formed,[3] the second part of the algorithm starts. It aims at exploring the graph and constructing its map, and in particular, finding an estimate on the graph size. Once such an estimate is known to the good agents, they start the third part of the algorithm, which is similar to Algorithm `WeakByzUnknownUpperBound`.

Let us go a little bit deeper into the description of these parts. The first part is a variation of `WeakByzUnknownUpperBound`, except that its completion for a good agent A is not determined by meeting any particular deadline (such a deadline is yet impossible to establish, as no upper bound on the size of the graph is known). Rather, it occurs when the kernel size of its group becomes at least $f + 2$. More precisely, whenever its kernel changes, an agent starts executing $TZ(\mu)$, where μ is the smallest label in the new kernel. This is iterated until the kernel gets to size at least $f + 2$, and another technical condition, given later in this description (when presenting some complications), is satisfied.

The second part is where the current algorithm differs most significantly from Algorithm `WeakByzKnownUpperBound`, and where the main technical difficulties are. The goal of this part is to discover the size of the underlying graph. At a high level, it is performed by implementing a fault-tolerant version of the exploration procedure `EST` (cf. Section 3.2.2), in which a group of at least $f + 1$ agents plays the role of the token and one agent plays the role of the explorer. Multiple such explorations are performed: agents from the kernel of the group of size at least $f + 2$ assume in turn

[3] Recall that the good agents initially know f.

the role of the explorer, all others playing the role of the token for this exploration. Once all explorations are performed, all good agents in the group know the graph size. It is in part 2 where the assumption that there are at least $f+2$ good agents is crucial. This assumption permits to guarantee the creation of a group with a kernel of size at least $f+2$. In contrast, using a kernel of size $f+1$ would necessitate using a token of size f, and such a token cannot be used reliably, as it may be composed of Byzantine agents that have not yet compromised themselves, and might suddenly move to another node during exploration, thus enticing a good explorer to learn a wrong graph size.

The third part mainly differs from Algorithm `WeakByzUnknownUpperBound` in the following two points. First, whenever the kernel changes, a good agent restarts the algorithm from the very beginning, rather than restarting only the third part. More precisely, it starts executing $TZ(\mu)$, where μ is the smallest label in the new kernel. Second, the time bound Δ after which good agents know that the gathering is over, is now different and particularly has to take into account also an upper bound on the exploration part of the algorithm.

As the acute reader will have quickly guessed, the implementation of the above algorithm has to take care of a lot of complications that have not been addressed here. For instance, what happens when an agent executing the first part encounters an agent playing the role of an explorer or a token? Or what happens when agents playing the role of a token see bad behaviors when the explorer is absent? In fact, most of the complications can be easily swept away with technical ingredients which it is not relevant to dwell on here to understand the main intuitive ideas. Two difficulties seem nevertheless worth explaining. They are both related to the implementation of the second part of Algorithm `WeakByzUnknownUpperBound`.

The first difficulty is that, when an explorer is a Byzantine agent, it may not come back to its group. Hence, all agents in the group must have a time bound after which they "give up" on this explorer, treat it as Byzantine and restart the algorithm. If they knew the graph size, they could obviously easily compute such a time bound, but it is not the case at this stage. To tackle this, the agents of the group will act in steps $i = 1, 2, 3, \ldots$. In each step i, the agents compute an upper bound on the time required for an explorer to visit all nodes of the graph via procedure `EST`, assuming that the graph size is at most 2^i. Of course, this time must be enough to let an explorer come back to its token if it realizes that 2^i is not a good upper bound. Therefore, in each step i, agents from the kernel of the group will perform time-limited explorations. If at some point an explorer does not come back from its exploration after the time limit, then it is Byzantine. In this case, the agents in the token blacklist the Byzantine explorer and restart the algorithm. Otherwise, if at the end of step i everything has gone "smoothly", then the agents start the third part of Algorithm `WeakByzUnknownUpperBound` (resp. step $i+1$) if they realize that 2^i is (resp. is not) a good upper bound on the graph size.

The second difficulty is even more serious. It may occur when there are some Byzantine agents among the agents playing the role of a token. Consider the situation when one agent of a group, the explorer, left a node v in round t, where a group

$\mathcal{T}$ of at least $f+1$ agents belonging to the kernel of the group remains playing the role of the token for this explorer. Obviously, if during its exploration the explorer meets the entire group $\mathcal{T}$, it can deduce that it has arrived at node v. Indeed, $f+1$ agents could not move when the algorithm instructs them to stay (recall that according to the algorithm, the size of $\mathcal{T}$ is at least equal to $f+1$, when the explorer starts its exploration). However, if the explorer meets only a subset $\mathcal{X} \subset \mathcal{T}$ at node w, two cases are possible.

The first case occurs when the size of $\mathcal{T}$, in round t, is at least $2f+1$. In this case, since there are at most f Byzantine agents in the token, we know that, after the explorer has left, at most f agents can leave the token and at least $f+1$ agents remain idle at v. Consequently, if the size of $\mathcal{X}$ is strictly smaller than $f+1$ then the explorer does not consider $\mathcal{X}$ and continues as if nothing happened: it is sure that $w \neq v$. Otherwise, the explorer is sure that it reached node v, i.e., that $v = w$. Both the explorer and the agents in $\mathcal{X}$ change state to regular, blacklist all the agents belonging to $\mathcal{T} \setminus \mathcal{X}$ and restart the algorithm from scratch.

The second case occurs when the size of $\mathcal{T}$ is strictly smaller than $2f+1$. If the size of $\mathcal{X}$ is at least $f+1$, the agents apply the same mechanism as above: the missing agents are blacklisted and the agents restart the algorithm from the beginning. If $|\mathcal{X}| < |\mathcal{T}| - f$ (where $|\mathcal{X}|$ and $|\mathcal{T}|$ are the sizes of $\mathcal{X}$ and $\mathcal{T}$ respectively), then the explorer does not consider $\mathcal{X}$ and continues as if nothing happened: it is sure that $v \neq w$. Otherwise, we have $|\mathcal{T}| - f \leq |\mathcal{X}| \leq f$ and it might be impossible for the explorer to decide whether it arrived at v or not, i.e., whether $v = w$ or $v \neq w$. Indeed, there are at least two possible situations:

- $\mathcal{X}$ is the set of good agents that remained at v and the agents belonging to $\mathcal{T} \setminus \mathcal{X}$ are Byzantine. In this case, $v = w$.
- $\mathcal{X}$ is a set of Byzantine agents and the other agents belonging to $\mathcal{T} \setminus \mathcal{X}$ remained at v. In this case, $v \neq w$.

This kind of ambiguity could possibly arise when the explorer reaches a node occupied by this group $\mathcal{X}$ via a new port. It cannot decide whether this node is v or another node w and it cannot pinpoint any Byzantine agent. The error resulting from a possibly wrong identification of the node occupied by the token could potentially lead a good agent to a wrong estimate of the graph size. We now describe how to prevent this.

Whenever the above ambiguity arises, the explorer backtracks to the original node v by the previously used path, informs the remaining agents of this ambiguity and marks the kernel of the original group from which it started the exploration, as well as all its subsets, as *grey*. Notice that the explorer may be unable to blacklist any particular agent as Byzantine if all agents that left v in the absence of the explorer returned to v before the explorer. Nevertheless, the explorer knows that there are Byzantine agents in the kernel of the original group from which it started the exploration.

On the side of the token whose members learned from an explorer that an ambiguity occurred, the reaction is the following. If the ambiguity is plausible, i.e., if $|\mathcal{T}| < 2f + 1$ and during the absence of the explorer there was a round in which a subset $Y \subset \mathcal{T}$ of size between $|\mathcal{T}| - f$ and f was separated from the rest of the token, then agents in the token mark the kernel of the original group, as well as all its subsets, as grey and restart the algorithm from scratch. Otherwise, the agents in the token know that the explorer is Byzantine. They blacklist the explorer and restart the algorithm.

Notice that a kernel K is marked grey when it becomes known that it contains Byzantine agents, but at no time all agents in K have necessarily seen together a faulty behavior of a particular agent from K, and hence nobody in K can be blacklisted.[4] In addition to marking the kernel grey, if after the return of the explorer to the token, there are agents missing from the original kernel, both the explorer and the agents remaining in the token blacklist the missing agents and restart the algorithm from scratch.

Marking a kernel as grey has the following effect. When a new group of at least $f + 2$ agents will be formed in the future during part 1 of the algorithm, but the kernel of the good agents in the group is either smaller than $f + 2$ or is grey then, instead of switching to the second part of Algorithm `WeakByzUnknownUpperBound`, the first part is continued because certainly at least one other good agent is outside of the group. This part will continue until a group is formed in some round t, such that, for each good agent a in this group, the kernel is of size at least $f+2$ and is not grey. This prevents a vicious circle which would result if kernels were not marked grey.

It can be shown that at some point the kernel will be such as to prevent the above ambiguity. This will permit all good agents to find the correct graph size needed to start the third part of the algorithm.

3.3.2 Algorithms dealing with strongly Byzantine agents.

The mechanism of blacklist, that is used with weakly Byzantine agents and that permits to maintain lists of labels corresponding to agents having exhibited an "inconsistent" behavior, becomes obsolete in a context in which faulty agents can always change their label. Hence, when facing strongly Byzantine agents, we need to devise completely new strategies. It is what we have done in [Bouchard *et al.* 2016], by designing an algorithm, which will be called here `StrongByzKnownUpperBound` (resp. `StrongByzUnknownUpperBound`), which allows the good agents to solve the problem of gathering, provided that they are at least $f + 1$ (resp. $f + 2$), when an (resp. no) upper bound n on the size of the underlying graph G is known. Below,

[4] When a kernel is marked as grey, all its subsets are also marked as grey. However, when a kernel S is grey, this does not imply that a superset S' of S is grey. In fact, unless S' or one of its supersets have been marked as grey, S' is not grey even if there exists a subset $S \subsetneq S'$ such that S is marked as grey. In particular, if S' contains some other agents not seen before, it is not grey, even if there is a subset of S' which is grey.

we give an informal description of these algorithms and we start with Algorithm `StrongByzKnownUpperBound`.

Let us first assume an ideal situation in which each agent would have as input its label, value f, and, instead of n, a parameter $\rho = (G^*, L^*)$ corresponding to the initial configuration of the agents in the graph such that:

- G^* represents graph G with all port numbers, in which each node is assigned an identifier belonging to $\{1, \cdots, |G|\}$, so that the node identifiers are pairwise distinct. (Note that the representation G^* contains more information than there is in the actual graph G as it also includes node identifiers that do not exist in G.)
- $L^* = \{(v_1, l_1), (v_2, l_2), \cdots, (v_k, l_k)\}$ where $(v_i, l_i) \in L^*$ iff there is a good agent having label l_i which is initially placed in G at the node having identifier v_i in G^*. Remark that $k \geq f + 1$.

Let us also assume that all the agents in the graph are woken up at the same time by the adversary. In such an ideal situation, gathering all good agents can be easily achieved by ensuring that each agent moves towards the node v where the agent having the smallest label is located. Each agent can indeed do that by using the knowledge of $\rho = (G^*, L^*)$ and its own label. Of course, all the good agents do not necessarily reach node v at the same time. However, each agent can compute the remaining time which is required to wait at node v in order to be sure that all good agents are at node v: again, this time can be computed using $\rho = (G^*, L^*)$ and the fact that all agents are woken up in the same round. Unfortunately, the agents are not in such an ideal situation. First, every agent is not necessarily woken up by the adversary, and for those that are woken by the adversary, this is not necessarily in the same round. Second, the agents do not have configuration ρ as input of the algorithm. In Algorithm `StrongByzKnownUpperBound`, we cope with the first constraint by requiring the first action to be a traversal of the entire graph using procedure `Explo` (cf. Section 2.2.2) with the known upper bound n as input, which permits to wake up all encountered agents that are still dormant. In this way, the agents are "almost synchronized", as the delay between the starting times of any two agents is at most the time complexity of `Explo`(n): the waiting time periods can be adjusted regarding this maximum delay. The second constraint, *i.e.,* the non-knowledge of ρ, is more complicated to deal with. To handle the lack of information about ρ, we use a technique that is an extension of ideas introduced in [Czyzowicz *et al.* 2012b] for the case of two agents evolving in a fault-free environment. Through this technique, agents make successive assumptions about it that are "tested" one by one. More precisely, let $\mathcal{P}$ be the recursively enumerable set of all the configurations $\rho_i = (G_i^*, L_i^*)$ such that G_i^* is a connected graph of at most n nodes and $|L_i^*| \geq f + 1$. Let $\Theta = (\rho_1, \rho_2, \rho_3, \cdots)$ be a fixed enumeration of $\mathcal{P}$ (all good agents agree on this enumeration). Each agent proceeds in phases numbered $1, 2, 3, \cdots$. In each phase i, an agent supposes that $\rho = \rho_i$ and, similarly as in the ideal situation, tries to go to the node which is supposed to correspond to node v, where v is the node where

the agent having the smallest label is initially located (according to ρ_i). For some reasons, when $\rho_i \neq \rho$ some agents may be unable to make such a motion or some others may realize that the current hypothesis ρ_i is incorrect (it is for instance the case, if an agent is supposed to exit a node by a port that does not exist, or if it sees a node having a degree too large according to ρ_i). As a consequence, these agents will consider that, rightly, $\rho_i \neq \rho$. On the other hand, whether $\rho_i \neq \rho$ or not, some other good agents may reach a node for which they had no reason to think it is not v (and thus $\rho_i \neq \rho$). The danger here is that when reaching the supposed node v these successful agents could see all the $|L_i^*|$ labels of ρ_i (with the possible "help" of some Byzantine agents). At this point, it may be tempting to consider that gathering is over, but this could be wrong, especially in the case where $\rho_i \neq \rho$ and some good agents did not reach a supposed node v in phase i. To circumvent this problem, the idea is to get the good agents that think $\rho_i = \rho$ to fetch the (possible) others for which $\rho_i \neq \rho$ via a traversal of the entire graph using procedure `Explo`(n). To allow this, an agent for which $\rho_i \neq \rho$ will wait a prescribed amount of rounds in order to leave enough time for possible good agents to fetch it. For our purposes, it is important to prevent the agents from being haphazardly fetched by any group, especially those containing only Byzantine agents. Hence, our algorithm is designed in such a way that within each phase at most one group, called a *tower* and made up of at least $f+1$ agents (supposedly thinking that ρ_i is the correct hypothesis), will be allowed to fetch the other agents via an entire traversal of the graph. Why $f+1$? Because a set of $f+1$ agents has some conviction power: if at least $f+1$ agents are together and claim that they form a tower, then we can believe them, because there is at least one good agent among them. When a tower has finished the execution of procedure `Explo`(n) in some phase i, our algorithm guarantees that all good agents are together and declare gathering is over at the same time (whether the assumed configuration ρ_i corresponds to the real initial configuration or not). On the other hand, in every phase i, if a tower is not created or "vanishes" (because there are not at least $f+1$ agents inside of it anymore) before the completion of its traversal, no good agent will declare that gathering is over in phase i. In the worst case, the good agents will have to wait until assuming a good hypothesis about the real initial configuration, in order to witness the creation of a tower that will proceed to an entire traversal of the network (and thus declare gathering is over).

Let us now provide some of the main intuitive ingredients on which `StrongByzUnknownUpperBound` is based. This algorithm inevitably shares some similarities with Algorithm `StrongByzknownUpperBound`, but it also presents several major differences to tackle the lack of knowledge about the network size. Among the most notable differences, there is the way of enumerating the configurations. Previously, the agents were considering the enumeration $\Theta = (\rho_1, \rho_2, \rho_3, \cdots)$ of $\mathcal{P}$ where $\mathcal{P}$ is the set of every configuration corresponding to a graph in which there are at least $f+1$ robots with pairwise distinct labels and whose size is at most the given upper bound n. Now, instead of considering Θ, the agents will consider the enumeration $\Omega = (\phi_1, \phi_2, \phi_3, \cdots)$ of $\mathcal{Q}$ where $\mathcal{Q}$ is the set of all configurations corresponding to graphs of any size (instead of size at most n only) in which there are at

least $f+2$ agents (instead of at least $f+1$) with pairwise distinct labels. Note that, as for set $\mathcal{P}$, set $\mathcal{Q}$ is also recursively enumerable.

Another difference stems from the function performed by a tower, which we also find here. In Algorithm `StrongByzknownUpperBound`, the role of a tower is to fetch all awaiting good agents (which know that the tested configuration is not good) via procedure `Explo`(n): in the new algorithm, we keep the exact same strategy. However, to be able to use procedure `Explo` with a parameter corresponding to an upper bound on the size of the network, it is necessary, for the good agents that are members of a tower, to get an estimate of such an upper bound. Hence, in our solution, before being considered as a tower and then authorized to make a traversal of the graph, a group of agents will have to learn the size of the graph. To do this, at least each good agent of the group will be required to make a simulation of procedure `EST` (cf. Section 3.2.2) by playing the role of an explorer and using the others as its token, as in the second part of Algorithm `WeakByzUnknownUpperBound` (cf. Section 3.3.1). To carry out these simulations, it is also required for the group of agents to contain initially at least $f+2$ members (explorer + token), even though it is subsequently required for a group of agents forming a tower to contain at least $f+1$ members. Why $f+2$? Because, as with Algorithm `WeakByzUnknownUpperBound`, Algorithm `StrongByzknownUpperBound` is designed in such a way that if during the simulation of procedure `EST` by an agent playing the role of an explorer, we have the guarantee there are always at least $f+1$ agents playing the role of its token, then the explorer will be able to recognize its own token without any ambiguity (and thus will act as if it performed procedure `EST` with a "genuine" token). Of course, the agents will not always have such a guarantee (especially due to the possible bad behavior of Byzantine agents when testing a wrong configuration) and will not be able to detect in advance whether they will have it or not. Besides, some other problems may arise including, for example, some Byzantine explorers which take too much time to explore the graph (or even worse, "never finish" the exploration). However, in all cases, we can make sure that the good agents never learn an erroneous size of the graph (even with the duplicity of Byzantine agents when testing a wrong configuration). We can also make sure that the good agents will learn the size of the network when testing a good configuration at the latest (as the creation of a group of at least $f+2$ agents and the aforementioned guarantee are ensured when testing a good configuration). As for Algorithm `StrongByzknownUpperBound`, in the worst case the good agents will have to wait until assuming a good hypothesis about the real initial configuration, in order to declare gathering is over.

3.3.3 Lower bounds on the minimum number of good agents.

In order to show that our algorithms presented in Sections 3.3.1 and 3.3.2 are optimal in terms of required number of good agents, it is enough to demonstrate two properties. The first (resp. second) property is that, in the presence of at most f strongly (resp. weakly) Byzantine agents, there is no deterministic algorithm permitting to solve the gathering problem with a team containing at most f (resp. $f+1$)

good agents, when the graph size is initially known (resp. when no upper bound on the graph size is initially known). Below, we sketch the arguments underlying the proofs of these two properties, which are given in [Dieudonné *et al.* 2014].

We start with the first one. For the sake of simplicity, we voluntarily restrict our explanations to the case where there are exactly f good agents and f strongly Byzantine agents, as the ideas involved in this particular case can be easily extended to fully demonstrate the first property. Suppose, towards contradiction, that there exists a gathering algorithm $\mathcal{A}$ for f good agents in the presence of f strongly Byzantine ones. Take an oriented ring of $n = 2f$ nodes $v_0, \ldots, v_{2f-1}$, with port 0 (resp. 1) leading clockwise (resp. counterclockwise) and place f good agents, labeled $1, \ldots, f$, in nodes $v_0, \ldots, v_{f-1}$ respectively. Also place in this ring, f faulty agents, labeled $1, \ldots, f$, in nodes $v_f, \ldots, v_{2f-1}$ respectively. We obtain an initial configuration $\mathcal{I}$. For notational convenience, we will refer to the kth faulty agent as $\hat{k}$. Now, consider an execution EX_1 from $\mathcal{I}$ in which all the agents are woken up in the same round and in which the faulty agents behave as good agents (except for the fact that they use a false label), namely, agent $\hat{k}$ (whose label is k) acts according to the instructions of Algorithm $\mathcal{A}$ for label k.

Observe that due to symmetry, the agents k and $\hat{k}$ will act symmetrically for every k, and subsequently will remain at diametrically opposing nodes throughout the execution. Since A is a gathering algorithm, it is guaranteed to gather the good agents $1, \ldots, f$ at some node v_i in some round t. By symmetry, in the same round, the faulty agents $\hat{1}, \ldots, \hat{f}$ are gathered at the diametrically opposing node $v_{i+f \bmod 2f}$.

Now, take an initial configuration $\mathcal{I}'$ that is similar to $\mathcal{I}$, except that the starting node of agent 1 (resp. $\hat{k}$) is v_f (resp. v_0), and consider an execution EX_2 in which, as in EX_1, the agents are woken up in the same round and the faulty agents, apart from lying about their label, behave as good agents. Necessarily, this execution will also end in round t with the agents $\hat{1}, 2, \ldots, f$ gathered at node v_i, and the agents $1, \hat{2}, \ldots, \hat{f}$ gathered at node $v_{i+f \bmod 2f}$. Hence, in execution EX_2 the algorithm fails to gather the good agents. This is a contradiction proving the first property.

The explanations about the first property being done, we can turn our attention to the second property stipulating that in the presence of at most f weakly Byzantine agents, there is no deterministic algorithm permitting to solve the gathering problem with a team containing at most $f + 1$ good agents, when no upper bound on the graph size is initially known. Suppose, by contradiction, that there exists such an algorithm and call it $\mathcal{B}$. For the sake of simplicity, we narrow down our arguments to the case in which there are exactly $f = 1$ faulty agents and $f + 1 = 2$ good agents, as the ideas involved in this particular case, once understood, can be easily generalized.

Take a clique of 6 nodes in which the port numbering is symmetric, i.e., for any nodes u and v, $\texttt{port}(u, v) = \texttt{port}(v, u)$, and place at distinct nodes two good agents labeled 1 and 2 and a weakly Byzantine agent labeled 3. From such an initial configuration, it can be proved that there exists an execution EX_3 of $\mathcal{B}$ in which the good agents end up solving the gathering problem in some round t without ever having seen the bad agent.

Now, take a ladder graph made of $4t+3$ rungs, labeled $rg_1, rg_2, \ldots, rg_{4t+3}$ from bottom to top, and first place a good agent with label 3 at one of the nodes of rg_1. Second, place a good agent with label 1 at one of the nodes of rung rg_{2t+2} and a weakly Byzantine agent with label 2 at the other node of rg_{2t+2}. Third, for every integer $1 \leq i \leq 4t+3$ and for each node u_i of rg_i, add an edge from u_i to the node of rg_{i+1} to which it is not adjacent. Finally, assign numbers to edges' ports so that the port numbering is symmetric. From the resulting configuration C, consider an execution EX_4 of $\mathcal{B}$ where the adversary wakes up all agents in the same round and the Byzantine agent labeled 2 acts as follows. For each round $s = 1, 2, \ldots, t$, if the agents labeled 1 and 2 are (resp. are not) together in round k of execution EX_3 (execution in which both these agents are good), then the agents labeled 1 and 2 are together (resp. occupy different nodes of the same rung) in round s of execution EX_4 (execution in which the agent having label 2 is Byzantine). This is always possible, in view of the structure of the graph in configuration C and the fact that agents labeled 1 and 2 start from the same rung in execution EX_4. Moreover, in each round $s = 1, 2, \ldots, t$, the Byzantine agent in round s of execution EX_4 gives exactly the same information to agent 1, as the agent with label 2 in round s of execution EX_3. Hence, using the fact that, by round t in EX_4, agent 1 can neither meet agent 3 nor being in a node with a degree different from 5, it can be proved, from the point of view of agent 1, that the first t rounds of execution EX_3 look exactly identical to the first t rounds of execution EX_4. Therefore, in round t of execution EX_4, agent 1 consider the task of gathering is terminated and stops the execution of $\mathcal{B}$. This is a contradiction with the definition of $\mathcal{B}$, which closes our argumentation about the second property.

3.3.4 An efficient algorithm dealing with strongly Byzantine agents

The algorithms presented in Sections 3.3.2, dealing with strongly Byzantine agents, have very huge complexities. As explained, these solutions are all based on a common strategy that consists in enumerating the possible initial configurations, and successively testing them one by one. Once the testing reaches the correct initial configuration, the gathering can be achieved. However, in order to get a significantly more efficient algorithm, such a costly strategy must be abandoned in favor of a completely new one. It is what we have done in [Bouchard *et al.* 2022] with a gathering algorithm that will be called here `GatherStrongTeam`.

In this subsection, we give the high-level idea of this algorithm. It works in all graphs of size at most n, in time polynomial in n and in the logarithm of the smallest label ℓ of a good agent, provided the agents are in a *strong team*, *i.e.,* the number of good agents is at least $5f^2 + 6f + 2$. Unlike the algorithms presented in Sections 3.3.2, Algorithm `GatherStrongTeam` does not require the agents to get initially, as global knowledge, the value of f, but just the value $\lceil \log \log n \rceil$.[5]

[5]We will not detail it in the present manuscript, but in [Bouchard *et al.* 2022], we show that the size of this global knowledge, which belongs to $\mathcal{O}(\log \log \log n)$, is asymptotically optimal to get a gathering algorithm polynomial in n and ℓ with strong teams.

Algorithm `GatherStrongTeam` relies on procedure `Explo`(n), presented in Section 2.2.2, allowing an agent to visit every node of any graph of size at most n in a number of rounds polynomial in n, which will be denoted here by $X(n)$. Algorithm `GatherStrongTeam` also relies on two other technical procedures. The first procedure, called `Group`$(\mathcal{T}, n, \beta)$, guarantees that a group of at least $4f + 2$ good agents terminate the execution of this procedure in the same round and at the same node if (1) the graph size is at most n, (2) all good agents start executing `Group`$(\mathcal{T}, n, \beta)$ within an interval of at most $\mathcal{T}$ consecutive rounds, and (3) the good agents are partitioned into two sets: those for which $\beta = 1$ and those for which $\beta = 0$. The execution by each good agent of `Group`$(\mathcal{T}, n, \beta)$ lasts at most a number of rounds that is polynomial in n and $\mathcal{T}$, which will be denoted here by $Q(n, \mathcal{T})$.

The second technical procedure, called `Merge`$(\mathcal{T}, n)$, guarantees that all good agents finish the execution of this procedure at the same node and in the same round if (1) the graph size is at most n, (2) all good agents start executing `Merge`$(\mathcal{T}, n)$ within an interval of at most $\mathcal{T}$ consecutive rounds, and (3) at least $4f + 2$ of them even do so in the same round and from the same node. The execution by each good agent of `Merge`$(\mathcal{T}, n)$ lasts at most a number of rounds that is polynomial in n and $\mathcal{T}$.

We now see how these procedures are involved in the design of Algorithm `GatherStrongTeam` and, in order to better describe the high-level idea, we first consider a situation that would be ideal to solve gathering with a strong team and that would be as follows. Instead of assigning distinct labels to all agents, the adversary assigns to each of them just one bit $\beta \in \{0, 1\}$, so that there is at least one good agent for which $\beta = 0$ and at least one good agent for which $\beta = 1$. Such a situation would clearly constitute an infringement of our model, but would allow the simple protocol described in Algorithm 1 to solve the problem in a time that is polynomial in n when global knowledge $\mathcal{GK} = \lceil \log \log n \rceil$. Let us briefly explain why.

Algorithm 1: Algorithm executed by every good agent in the ideal situation.

1 $\beta :=$ the bit assigned to me by the adversary;
2 Execute $\mathcal{A}(\beta)$;
3 Declare that gathering is achieved;

Algorithm 2: $\mathcal{A}(\beta)$ executed by a good agent.

1 $N := 2^{(2^{\mathcal{GK}})}$;
2 Execute `Explo`(N);
3 Execute `Group`$(X(N), N, \beta)$;
4 Execute `Merge`$(X(N) + Q(X(N), N), N)$

Algorithm 1 consists mainly of a call to $\mathcal{A}(\beta)$ that is given by Algorithm 2. Since $\mathcal{GK} = \lceil \log \log n \rceil$, we know that at the first line of Algorithm 2, N is a polynomial upper bound on n, and the execution of `Explo`(N) in a call to $\mathcal{A}(\beta)$ by the first woken-up good agent permits to visit every node of the graph and to wake up all dormant agents. As a result, the delay between the starting times of `Group`$(X(N), N, \beta)$ by any two good agents of the strong team is at most $X(N)$.

According to the properties of procedure `Group`, this guarantees in turn that the delay between the starting times of $\texttt{Merge}(X(N)+Q(X(N),N),N)$ by any two good agents is at most $X(N)+Q(X(N),N)$, and at least $4f+2$ good agents start this procedure at the same time from the same node. Hence, in view of the properties of procedure `Merge`, all good agents declare gathering is achieved after a polynomial number of rounds (w.r.t. n).

Unfortunately, we are not in such an ideal situation. At first glance, one might argue that it is not really a problem because all agents are assigned distinct labels that are, after all, distinct binary strings. Thus, by ensuring that each good agent applies on its label a transformation that is similar to that mentioned in Section 2.2.1, and then processes one by one each bit b_i of its transformed label by executing $\mathcal{A}(b_i)$, we can guarantee (with some minor technical adjustments) that the conditions of the ideal situation are recreated when the agents process the jth bit of their transformed label, for some $j \in \mathcal{O}(\ell)$, and by extension, the fact that all good agents end up being together after at most a time polynomial in n and ℓ.

Unfortunately, it is not enough for our purpose. Indeed, in the ideal situation, there is just one bit to process: thus, *de facto* every good agent knows that every good agent knows that gathering will be done at the end of this single process. However, it is no longer the case when the agents have to deal with sequences of bit processes: the good agents have a priori no mean to detect collectively when they are gathered.

To circumvent this problem, we want to reach a round in which every good agent knows that every good agent knows that gathering is done. To do so, we put in place a strategy in which we need the agents to consider a transformed label slightly different to that used in Section 2.2.1. Precisely, each agent applies a transformation permitting to obtain a new label for which the binary representation is even, has asymptotically the same size as the original label and that has the following property. Given any two agents, for which the sizes of their transformed labels are s and s', there exists a positive integer $i \leq \frac{min(s,s')}{2}$ (resp. $\frac{min(s,s')}{2} < j \leq min(s,s')$) such that the ith bits (resp. jth bits) of their transformed labels are different. How does it help solve the gathering? To understand this, let us return to our sequence of bit processes, but now applied to labels transformed in this way. When a good agent has finished reading the first half of its transformed label – call such an agent *experienced* – it has the guarantee that the gathering of all good agents has been done at least once. Hence, when an experienced agent starts to process the second half of its transformed label, it actually knows an approximation of the number of good agents with a margin of error of f at the most. For the sake of convenience, let us consider that an experienced agent knows the exact number μ of good agents: the general case adds a slight level of complexity that is unnecessary to understand the intuition. So, each time an experienced agent completes the process of a bit in the second half of its transformed label, it is at a node containing either less than μ agents or at least μ agents. In the first case, the experienced agent is sure that the gathering is not achieved. In the second case, the experienced agent is in doubt. In our solution, we build on this doubt. How do we do that? So far, each bit process

was just made of one call to procedure $\mathcal{A}$: now at the end of each bit process, we add a waiting period of some prescribed length, followed by an extra step that consists in applying $\mathcal{A}$ again, but this time according to the following rule. If during the waiting period it has just done, an agent Y was at a node containing, for a sufficiently long period, an agent pretending to be experienced and in doubt (this agent may be Y itself), then agent Y is said to be *optimistic* and the second step corresponds to the execution of $\mathcal{A}(0)$. Otherwise, agent Y is said to be *pessimistic* and the second step corresponds to the execution of $\mathcal{A}(1)$.

If at least one good agent is optimistic within a given second step, then the gathering of all good agents is done at the end of this step. Indeed, through similar arguments of partition to those used for the ideal situation, we can show it is the case when at least another agent is pessimistic. However, it is also, more curiously, the case when there are no pessimistic agents at all. This is due in part to the fact that two good experienced agents cannot have been in doubt at two distinct nodes during the previous waiting period (otherwise, we would get a contradiction with the definition of μ). Thus, all good agents start $\mathcal{A}(0)$ from at most $f + 1$ distinct nodes (as the Byzantine agents can mislead the good agents in at most f distinct nodes during the waiting period), which implies by the pigeonhole principle that at least $4f + 2$ good agents start it from the same node. Combined with some other technical arguments, we are able to show that the conditions, given earlier, for a proper execution of procedure `Merge`, are fulfilled when the agents execute it at the end of $\mathcal{A}(0)$, thereby guaranteeing again gathering of all good agents.

As a result, the addition of an extra step to each bit process gives us the following interesting property: when a good agent is optimistic at the beginning of a second step, at its end the gathering is done and, more importantly, the optimistic agent knows it because its existence ensures it. Note that, it is a great progress, but unfortunately it is not yet sufficient, particularly because the pessimistic agents do not have the same kind of guarantee. The way of remedying this is to repeat once more the same kind of algorithmic ingredient as above. More precisely, at the end of each second step, we add again a waiting period of some prescribed length, followed by a third step that consists in applying $\mathcal{A}$ in the following manner. If during the waiting period it has just done, an agent Y was at a node containing, for a sufficiently long period, an agent pretending to be optimistic, then the third step of agent Y corresponds to the execution of $\mathcal{A}(0)$ and it becomes optimistic if it was not. Otherwise, the third step of agent Y corresponds to the execution of $\mathcal{A}(1)$ and the agent stays pessimistic.

By doing so, we made a significant move forward. To understand why, we want to invite the reader to reconsider the case when there is at least one good agent that is optimistic at the beginning of a second step. As we have seen earlier, at the end of this second step, all good agents are necessarily gathered and every optimistic agent knows it. In view of the last changes made to our solution, when starting the third step, every good agent is then optimistic. As explained above the absence of pessimistic good agent is very helpful, and using here the same arguments, we are sure that when finishing the third step, all good agents are gathered and every

good agent knows it because all of them are optimistic. Actually, it is even a little more subtle: the optimistic agents of the first generation (i.e., those that were already optimistic when starting the second step) know that the gathering is done and know that every good agent knows it. Concerning the optimistic agents of the second generation (i.e., those that became optimistic only when starting the third step), they know that the gathering is done, but do not know whether the other agents know it or not. (Recall that we want to reach a round in which every good agent knows that every good agent knows that gathering is done.) However, we are very close to such a consensus. To reach it, at the end of a third step, the optimistic agents of the first generation make themselves known to all agents. Note that if there were at least $f+1$ agents declaring to be optimistic agents of the first generation and if f were part of $\mathcal{GK}$, the consensus would be reached. Indeed, among the agents declaring to be optimistic of the first generation, at least one is necessarily good and every agent can notice it: at this point we can show that every good agent knows that every good agent knows that gathering is done.

However, the agents do not know f. That being said, at the end of a third step, note that an optimistic agent knowing that the gathering is done can compute an approximation $\tilde{f}$ of the number of Byzantine agents. More precisely, if the number of agents gathered at its node is p, the optimistic agent knows that the number of Byzantine agents cannot exceed $\tilde{f} = \max\{y|(5y+1)(y+1)+1 \leq p\}$ according to the definition of a strong team. Based on this fact, we are saved. Indeed, our algorithm is designed in such a way that all good agents correctly declare that the gathering is achieved in the same round after having computed the same approximation $\tilde{f}$ and noticed at least $\tilde{f}+1$ agents that claim being optimistic of the first generation during a third step. We show that such an event necessarily occurs before any agent finishes the kth bit process of its transformed label, for some $k \in \mathcal{O}(l)$, which permits to obtain the promised polynomial complexity. This is where our feat of strength is: obtaining such a complexity with a small amount of global knowledge, while ensuring that the Byzantine agents cannot confuse the good agents in any way. Actually, our algorithm is judiciously orchestrated so that the only thing Byzantine agents can really do is just to accelerate the resolution of the problem.

A natural open question that may come to mind is whether we could do the same thing by reducing the ratio between the good agents and the Byzantine agents. For example, could it be still possible to solve the problem in polynomial time with a global knowledge of size $\mathcal{O}(\log\log\log n)$ if the number of good agents is at most $o(f^2)$? Note that the answer to this question may be negative, but then may become positive with a little bit more global knowledge. Actually, we can even easily show that the answer is true if the agents are initially given a complete map of the graph with all port numbers, and in which each node v is associated to the list of all labels of the good agents initially occupying node v. However, the size of $\mathcal{GK}$ is then huge as it belongs to $\Omega(n^2)$. In fact, in this case what is really interesting is to find the optimal size for $\mathcal{GK}$. This observation allows us to conclude with the following open problem that is more general and appealing.

What are the trade-offs among the time complexity, the ratio good/Byzantine

agents and the amount of global knowledge to solve gathering?

Bringing an exhaustive and complete answer to this question appears to be really challenging, but would turn out to be a major step forward in our understanding of the problem.

3.4 Gathering without any direct means of communication

In all the contributions that have been exposed so far in the present chapter, agents sharing the same node can exchange information. In [Bouchard *et al.* 2023b], we ask if this ability of talking is needed to solve the gathering problem, and the answer turns out to be no. In support of this, we design two deterministic algorithms that accomplish gathering in a model, deriving from `GBM`, in which:

- The team is made of two or more agents that all have distinct pairwise labels and that all start from distinct nodes.
- The adversary wakes up some of the agents (at least one) in possibly different rounds.
- In each round, the only information an agent can get about those sharing the same node as it, is their number (in particular, it cannot talk or see the labels of the other co-located agents).

Our first algorithm, called `GatherKnownUpperBound`, assumes that agents know some upper bound n on the size of the network, and works in time polynomial in n and in the length ℓ of the smallest label. Our second algorithm, called `GatherUnknownUpperBound`, does not assume any knowledge about the network, but its complexity is at least exponential in the size of the network and in the labels of agents. Its purpose is to show feasibility of gathering under this harsher scenario.

As a by-product of our techniques we obtain, in the same weak model, solutions to two fundamental problems, whether the agents initially know some upper bound n on the size of the network or not. The first problem is that of leader election: one agent is elected as a leader and all agents learn its identity. The second problem is that of gossiping: each agent has a message at the beginning, and all agents must end up learning all messages. This result about gossiping is perhaps our most surprising finding: agents devoid of any transmitting devices can solve the most general information exchange problem, as long as they can count the number of agents present at visited nodes.

In the absence of direct communication, a natural idea that comes to mind to solve the gathering problem, is to emulate the unavailable mechanism of communication using moves of agents. In fact, this is the basic approach that we adopt. However, this idea, while natural, turns out to be very delicate to put in use. Indeed, without special care, one gets soon to a dangerous situation where communication

movements of one group of agents can interfere with communication movements of another closely located group. On top of this difficulty we have another one: movements of agents must serve to accomplish two different goals, one is to communicate with other agents, and the other is to travel in order to meet. Hence, we face the danger of "travelling" movements interfering with "communication" movements. Moreover, it should be stressed that an agent does not, in fact, "see" another agent entering or leaving its node: it can only see the cardinality of the set of agents occupying its current node, and, e.g., notice changes in it while waiting at a node. Hence, for example, an agent will not notice any change, if one other agent leaves its node trying to communicate something, and another agent enters its node simply navigating in the graph. Of course, all the above challenges are entirely absent when co-located agents can directly exchange information. Overcoming these difficulties in the design of our algorithms is the main technical contribution of our paper [Bouchard *et al.* 2023b].

The intuitions that are behind Algorithms `GatherKnownUpperBound` and `GatherUnknownUpperBound` are presented blow, in Section 3.4.1 and 3.4.2 respectively.

3.4.1 Known upper bound on the size of the graph

In order to better describe the high-level idea of Algorithm `GatherKnownUpperBound`, working when an upper bound n on the network is known, let us assume an ideal situation in which all the agents have labels of the same length μ. Let us also assume that all agents are initially woken up at the same time by the adversary.

In such an ideal situation, we can solve the gathering problem via a strategy made up of consecutive steps $1, 2, 3$, etc. The agents start each step i simultaneously. At the beginning of step i they are distributed over k_i distinct nodes. The intended goal is either to get the agents to declare gathering at the same time once step i is completed if $k_i = 1$, or otherwise to get all agents to start simultaneously step $i+1$ from at most $k_{i+1} \leq \lfloor \frac{k_i}{2} \rfloor$ distinct nodes. Since the beginning of step 1 coincides with the round when the adversary wakes up all agents, we can consider a step $i \geq 1$ initiated at the same time by all agents from k_i distinct nodes, and explain how to reach the intended goal mentioned above. This is the purpose of the following paragraphs in which we will use the notion of *invisibility* that can be intuitively defined as follows: two agents (or two groups of agents), executing the same sequence of instructions X starting at the same round but from two distinct nodes, are said to be invisible to each other if they do not meet when executing X.

At the beginning of step i, an agent first applies the simple procedure described in Algorithm 3 that is based on the graph traversal routine `Explo`(n) (cf. Section 2.2.2), whose time complexity is a polynomial in n that will be denoted by $X(n)$. This procedure consists of two successive parts of equal durations: the effective part, in which each node is visited at least once, and the backtrack part, in which the agent

executes in reverse order all edge traversals made during the effective part.

Algorithm 3:

1 c := the number of agents at my current node;
2 execute `Explo`(n) and interrupt it as soon as there are more than c agents in my node;
3 wait until the time spent executing Algorithm 3 is precisely $X(n)$ rounds;

It should be noted that the agents that are initially together remain so during the execution of Algorithm 3. Hence, after having applied this algorithm, an agent is either (1) with more agents than at the beginning of the step or (2) with exactly the same number of agents.

Let us first focus on the former situation. This situation necessarily implies that two groups of agents starting step i from two distinct nodes are not invisible to each other during the execution of the procedure `Explo`(n), and thus even not invisible to each other during the effective part of `Explo`(n), due to symmetry arguments. Therefore, as soon as the first $\frac{X(n)}{2}$ rounds of the execution of Algorithm 1 are elapsed, we get at most two kinds of groups: the old ones (if any) that have not met any group yet and the new ones (at least one exists) that result from the merge of at least two old groups. In view of the fact that the old groups, which have not merged yet, were invisible to each other when executing the effective part of `Explo`(n) and the fact that every new group remains idle during the last $\frac{X(n)}{2}$ rounds, we have the guarantee that each remaining old group meets a new one when executing the backtrack part of `Explo`(n). Thus, the execution by every agent of Algorithm 3 lasts exactly $X(n)$ rounds: when it is completed, the agents are all situated in at most $\lfloor \frac{k_i}{2} \rfloor$ distinct nodes. Note that all agents know this, and know that every agent knows this because if an agent ends up sharing its node with more agents than at the beginning of the step, it follows from the above explanations that this is the case for the other agents as well. Hence, we can fulfill our intended goal by just requiring an agent to start step $i+1$ if, after having applied Algorithm 3, it is with more agents than at the beginning of the step.

Now, let us focus on the latter situation in which, after applying Algorithm 3, an agent is exactly with the same number of agents as at the beginning of step i. For an agent experiencing this situation, it could be tempting to think that everyone is together as, after all, it has not met any new agent when executing Algorithm 3. However, at this stage, this would be premature and thus dangerous. In fact, it can be shown that the current situation implies that: either $k_i = 1$ (i.e. all agents are indeed together), or $k_i \geq 2$ but the k_i groups are pairwise invisible to each other when executing Algorithm 3 (the procedure `Explo`(n) does not guarantee rendezvous of two agents starting at different nodes). However, these possible invisibilities that could appear detrimental at first glance, we turn them to our advantage. Indeed, we are actually in a convenient situation to allow each agent, using movements, to communicate with the agents of its group, without being disturbed by the agents of the other groups. Those communications aim at ensuring that in each group the agents end up knowing the label of one of them.

To achieve this, still in step i, the agents will act in phases $1, 2, 3 \ldots, \mu$, each lasting $2 \cdot X(n)$ rounds. At the beginning of phase k, we have the following property $\mathcal{Q}(k)$: in every group Gr, there is an agent with label L such that all agents of Gr know the prefix p_{k-1} of length $k-1$ of the binary representation of L (note that $\mathcal{Q}(1)$ is trivially satisfied at the beginning of the first phase). Let us see how these agents proceed to have the property $\mathcal{Q}(k+1)$ satisfied when phase k is completed.

During the first (resp. last) $X(n)$ rounds of phase k, the agents having a label whose prefix is $p_{k-1}0$ execute `Explo`(n) (resp. remain idle) while the others remain idle (resp. execute `Explo`(n)). The respective invisibilities of the groups come into the picture as they imply the following crucial property: two agents belonging to two distinct groups and executing `Explo`(n) in the first (resp. last) $X(n)$ rounds cannot meet each other within phase k. This is crucial because it means that when a set of agents belonging to the same group move together by executing `Explo`(n), they visit a node that contains only agents belonging to this set. Hence, by comparing the number of agents sharing its node at the beginning of phase k to the minimum number of agents with which it was at some node when executing `Explo`(n), every agent can determine the number of agents of its group for which $p_{k-1}0$ is a prefix. Once phase k is completed, if an agent concludes that there is at least one agent in its group that has a label prefixed by $p_{k-1}0$, then all agents of the group conclude the same, and then p_k is set to $p_{k-1}0$, otherwise p_k is set to $p_{k-1}1$.

After phase μ, in each group Gr all agents know the same label p_μ of an agent belonging to Gr: this label can now be used to break the invisibility of Gr via procedure `TZ` and its associated polynomial $\mathcal{P}_{TZ}$, both introduced in Section 3.2.1. Indeed, by requiring each agent, once its execution of phase μ is completed, to execute Algorithm 4 that relies on procedure `TZ` and the polynomial $\mathcal{P}_{TZ}$, we can show, using similar arguments as before, that we reach a configuration where: either (1) the number of groups is at most $\lfloor \frac{k_i}{2} \rfloor$ and the cardinality of each of them has increased, or (2) there is only one group and its cardinality has remained unchanged since the beginning of the phase. Note that every agent can detect in which of these two situations it is, just by looking at the cardinality of its group. If the agents are in the first situation, then they start step $i+1$, otherwise they declare that gathering is over: whichever is the case, they can do it at the same time in view of the last line of Algorithm 4. Hence, in our ideal scenario, we can prove that gathering is declared after at most $\lceil \log n \rceil$ steps, leading to a time complexity polynomial in n and μ.

Of course, things get more complicated when we are in a scenario that is not necessarily ideal. However, it is through those conceptual principles, together with extra algorithmic ingredients (to circumvent the possible desynchronizations between the wake-ups of the agents as well as the possible different label lengths) that we finally obtain a gathering algorithm working in the general case with a time complexity

polynomial in n and in the length of the smallest label among the agents.

Algorithm 4:

```
1 c := the number of agents at my current node;
2 p_μ := the label learned when phase μ is completed;
3 execute TZ(p_μ) and interrupt it as soon as there are more than c agents in
    my node or the execution has lasted P_TZ(n, μ) rounds;
4 if there are c agents in my node then
5 |   execute Explo(n) and interrupt it as soon as there are more than c
  |     agents in my node;
6 wait until the time spent executing Algorithm 4 is precisely
    P_TZ(n, μ) + X(n) rounds;
```

3.4.2 Unknown upper bound on the size of the graph

We now turn our attention to the high-level idea of Algorithm `GatherUnknownUpperBound`, which permits to solve gathering without direct means of communication in the case where they are not initially given any upper bound on the graph size.

A preliminary question that may come to mind when considering this harsher scenario, is how to guarantee gathering even if in each round an agent had the capacity to exchange all information available to it with the other agents sharing its node. Actually, an answer to this question have already been provided in Section 3.3.4, when describing a general mechanism, to solve gathering in presence of strongly Byzantine agents when no upper bound on the network size is initially known, which is based on the following simple observation. If the agents were woken up at the same time by the adversary and were initially given the description of the initial configuration ϕ (i.e., the complete map of the underlying graph, with all port numbers, in which a node v is labeled L iff v is the starting node of the agent labeled L), then the problem could be solved by applying a simple rule: upon its wake-up, each agent moves, using the map, towards the node containing the smallest label, and then declares that gathering is over when all agents are in that node.

One of the main goal of the general mechanism outlined in Section 3.3.4 is to solve Byzantine gathering (with communication) by trying to recreate similar favorable conditions to those given in the above observation. As it will be really useful for our purpose, let us briefly reformulate here how it works at a high level, by removing the Byzantine agents from the picture. Let $\Omega = (\phi_1, \phi_2, \phi_3, \cdots)$ be a fixed enumeration of the (recursively enumerable) set of all initial configurations. An agent proceeds in consecutive phases $h = 1, 2, 3, \cdots$, where in each phase h, an agent tries to achieve gathering by making a hypothesis that the initial configuration ϕ is ϕ_h. This assumption will be called hypothesis h. When making hypothesis h, the agent first executes `Explo`(n_h) (cf. Section 2.2.2) where n_h is the supposed graph size (in the hope of waking up the remaining dormant agents, if any), and then, if its label belongs to ϕ_h, tries to go to the node that supposedly corresponds to the starting node v_h of the agent having the smallest label in ϕ_h. Once an agent reaches

a node that supposedly corresponds to v_h, it waits some amount of time sufficient to allow the other agents to join it if the hypothesis h is good (i.e., correct). Note that in the case where the hypothesis h is not good, some agents may notice it more or less rapidly, for instance if ϕ_h does not contain their label or if the path they have to follow to reach v_h simply does not exist in the real network. Hence, the process of a phase h we have described so far can lead to one of the following three situations: (1) the hypothesis h is good and all agents think rightly that gathering is over, (2) the hypothesis h is not good and all agents know it, or (3) the hypothesis h is not good, but some agents do not know it because for some of them everything appears to have gone very well (in particular they are at the node supposedly corresponding to v_h with agents having the labels they should have w.r.t ϕ_h). The third situation may especially occur if the number of supposed agents or the supposed size of the graph in ϕ_h is lower than it actually is. At this point, an "optimistic" agent may think that it is in the first situation, while really it is in the third. This is why each time a phase is completed, the optimistic agents, if any, execute a checking protocol based on a simulation of the exploration protocol `EST` (cf. Section 3.2.2), in which the role of the token is played by some of them. If after having executed the checking protocol, the optimistic agents notice they have constructed a map of the graph corresponding to that of ϕ_h and they have not encountered agents thinking that hypothesis h is wrong (before switching to the next phase, these agents stay idle enough time in order to be detected by possible optimistic agents), they can be sure to have accomplished gathering. Otherwise, the agents that were optimistic join the camp of the agents knowing that hypothesis h is wrong, and after a certain time, each agent goes back to its initial node in order to start phase $h+1$.

The high-level idea of our algorithm consists in emulating this general mechanism in a context in which the agents are devoid of direct means of communication. However, this turns out to be much easier said than done. Indeed, to settle properly such an emulation, we have to face numerous challenges. For instance, how can an agent become optimistic about a supposed configuration ϕ_h, if it cannot see the labels of the agents sharing its node? Or, how can an agent recognize its token played by some agents during an execution of the checking protocol? In the next subsection, we bring algorithmic solutions to solve these problems. However, one particular problem was really more important than the others, in the sense that its resolution appeared as a *sine qua non* condition to solve most of the other problems raised by our emulation. The problem is this: in order to emulate the mechanism outlined above in a correct way, an agent has to avoid confusing another agent acting under the same hypothesis as itself, with an agent acting under a different hypothesis. While this is not at all an issue when an agent can exchange arbitrary messages with the other agents sharing its current node (indeed, it is then enough to ignore the agents that indicate processing a different hypothesis), this becomes a real difficulty in our context. To tackle it, we put in place two fundamental schemes.

The first scheme consists in slowing down every agent more and more as it progresses through successive hypotheses, so that an agent processing a hypothesis h avoids being misled by an agent processing a hypothesis $h' > h$. Actually, using

various strategies of moves such as, for example, some "dancing" protocols (that consist in leaving and entering the same node in carefully chosen rounds), and by interleaving, between each step of every hypothesis $h > 1$, waiting periods lasting the time of processing the hypotheses 1 to $h-1$ in the worst case, every agent can detect an agent that processes a hypothesis larger than its own current hypothesis. However, this turns out to be ineffective to protect an agent from being confused by agents processing smaller hypotheses. This is why we introduce a second scheme that divides the processing of each hypothesis into two parts: the main part corresponding, strictly speaking, to the emulation of the general mechanism with, in particular, the previously mentioned slowdowns and dances, that is preceded by a preprocessing part, the heart of the second scheme, and which relies on the notions of *kernel* and *ball* defined below.

A kernel $\mathcal{K}(v,h)$ is the set of all nodes that may be visited by an agent processing the main part of hypothesis h, starting from a node v. In our solution, the set $\mathcal{K}(v,h)$ does not depend on the label of the executing agent: in fact, the label can influence the way the nodes of $\mathcal{K}(v,h)$ are visited, but not the set of visited nodes. Given a kernel $\mathcal{K}$, its ball is the set of all its critical nodes. A node u is said to be critical for a kernel $\mathcal{K}(v,h)$ if u belongs to $\mathcal{K}(v,h)$, or if an agent, when initially located at node u, may visit a node of $\mathcal{K}(v,h)$ before processing hypothesis h, either during a preprocessing part or during a main part of some hypothesis $h' < h$. These notions are used within the second scheme, in the following way. An agent executing the preprocessing part of a hypothesis h performs the following actions, without paying attention to other agents: first, it visits all nodes of the ball of $\mathcal{K}(v,h)$, where v is the node it occupies at the beginning of the preprocessing and of the main part of hypothesis h, then it goes back to v, and finally it waits the maximum time required for an agent, which would have just been woken up, to begin the execution of hypothesis h (in our solution the time to reach hypothesis h is bounded by some function depending only on the supposed graph sizes of the previous hypotheses). By doing so, we have the guarantee that an agent A executing the main part of a hypothesis h, the kernel of which is $\mathcal{K}$, cannot encounter any agent executing the preprocessing part or the main part of a previous hypothesis. Indeed, otherwise this would imply that an agent, initially located at a node belonging to the ball of $\mathcal{K}$, has been woken up after the traversal by agent A of the ball of $\mathcal{K}$, which would be a contradiction. Note that, visiting all nodes of the ball of $\mathcal{K}(v,h)$ can be made by following all the (not necessary simple) paths of length $d_1 + d_2$ from node v, where d_1 (resp. d_2) is the maximum distance between two nodes visited during the main part of hypothesis h (resp. during the processing of the hypotheses 1 to $h-1$). Also note that the executing agent will abort the traversal of the ball of $\mathcal{K}(v,h)$ as soon as it occupies a node having a degree at least equal to the graph size of ϕ_h. This precaution does not cause any problem because if such an event occurs the agent has the guarantee that $\phi_h \neq \phi$, and before starting phase $h+1$, it no longer has to worry about agents processing hypotheses different from its own. This is nonetheless crucial, as we need the agents to know in advance the worst-case time to process any given hypothesis (to settle consistently some of the aforementioned

waiting periods).

To get all of this to work, we must not forget to link the schemes to each other: in particular, the slowdowns of the first scheme have to be also added between the steps of the preprocessing parts, and every ball traversal of the second scheme has to take into account the dancing protocols of each main part resulting from the first scheme.

In order to illustrate the importance of the joint application of the two presented schemes, let us close our intuitive explanations by considering one of the problems, mentioned earlier, encountered in putting in place our emulation: that of an exploration, during which an agent plays the role of an explorer while the others play the role of a token, performed to check whether a supposed hypothesis is good or not. Actually, our solution is designed so that for each hypothesis h, such explorations can be triggered by at most one group G_h of agents located at the same node u_h, those thinking that hypothesis h may be good: the role of explorer is assigned in turn to all agents of G_h using the order over the labels that are in ϕ_h. Note that these explorations must be seen as the last part of the process that consists in testing hypothesis h. They are launched by agents that think hypothesis h may be good but that have not acquired yet any guarantee about the validity of the hypothesis (the other agents also test hypothesis h but do not go as far as the agents of G_h in the process of this test because they "soon realize" that this hypothesis is wrong).

When an agent becomes explorer, it executes protocol `EST` using as a token the other agents of G_h, which then stay idle at node u_h. As soon as the execution of `EST` is completed or as soon as the explorer notices that the map under construction of the network does not match that of ϕ_h, it goes back to its token to let the other agents, which have not yet done so, execute `EST` and reach the same conclusion about the validity of hypothesis h. The key thing for an explorer is to perform a "clean" exploration, *i.e.,* not to confuse the group of agents representing its token with another group, as otherwise it might reach a wrong conclusion. Such a confusion can be avoided by requiring the agents of G_h to perform twice together a traversal of all the nodes located at distance at most $d+1$ from node u_h before starting the simulations of `EST`, where d is the maximal distance that may separate an explorer from its token during a simulation of `EST` from u_h. If some agents are encountered during one of these two collective traversals, which can be easily detected through a rise of cardinality, then all the agents of G_h have the guarantee that ϕ_h is not good and they do not even need to proceed further with the simulations of `EST`. On the other hand, if no other agent is encountered during any of these traversals, then in view of the two schemes, we have the guarantee that each simulation of `EST` by an agent of G_h will be clean. Indeed, the second scheme ensures that an explorer cannot meet an agent processing a hypothesis $h' < h$. Moreover, an explorer cannot meet an agent processing a hypothesis $h' > h$ because the first scheme ensures that such an agent is too slow to reach or to be already in the zone of the nodes at distance at most d from u_h (in which the successive simulations of `EST` are done) without having been detected during one of the previous two traversals. Of course, one might still argue that an explorer could be bothered by agents that also test hypothesis h but

that do not belong to G_h. However, this kind of situation will never occur. In fact, the agents that do not belong to G_h, will quickly notice that the hypothesis h is not good, when processing the main part of hypothesis h, and thus by adding judicious waiting periods as we did in Algorithm `GatherUnknownUpperBound`, we can prove they will be detected by all agents of G_h during one of their two collective traversals.

Chapter 4

Treasure Hunt

Contents

4.1 Introduction

The treasure hunt problem can be viewed as a variant of the rendezvous problem in which one of the agents, called the treasure, is always stationary. To the best of our knowledge, this problem was first introduced by Richard E. Bellman in his seminal paper [Bellman 1963]. Since then, the problem has been investigated under various scenarios, in which the search can be deterministic or randomized, and the environment may be a graph or a plane (or a portion of it). The best surveys on treasure hunt are certainly [Alpern & Gal 2003] and [Ghosh & Klein 2010], although the former is mainly concerned with randomized search strategies.

When devising a treasure hunt algorithm, the aim is often to provide a solution that optimizes the asymptotic cost complexity, or even the competitive ratio, which is, in our current context, the worst-case cost incurred by the algorithm, for which the treasure's location is unknown, divided by the worst-case cost incurred by the optimal algorithm working with knowledge of the treasure's location. Roughly speaking, it can be interpreted as a measure of the additional distance that is traveled due to the lack of knowledge on the position of the treasure.

One of the most famous result of the field is given by the following paper [Beck & Newman 1970], in which the authors show that the best competitive ratio for deterministic treasure hunt, with no range of vision, in a one-dimensional space (i.e., an infinite line) is 9. This optimal competitive ratio is obtained by a doubling zigzag strategy, known as *cow path*: go left at distance 1, then right at distance 2, left at distance 4, and so on. If randomization is allowed, then there is a strategy with a smaller competitive ratio [Kao *et al.* 1996]. Instead of doubling the search distance, the strategy uses a ratio r that is approximately equal to 3.59

and a random number ε belonging to $[0,1)$. More precisely, till finding the treasure, the agent acts in steps $1,2,3,\ldots$, where in each step i, the probability of going left (resp. right) at distance $r^{i\varepsilon}$ is 0.5. If the treasure is not found by the end of the step, the agent goes back to its initial position and starts the next step. Such a strategy is proven to be optimal in [Kao *et al.* 1998] and the expected competitive ratio is around 4.5911, which is almost twice as good as the best that can be done deterministically.

Still in the infinite line, if the agent initially knows the exact distance D that separates it from the treasure, we can get a significantly lower competitive ratio by asking the agent to walk a distance D to the right and, if the treasure is not found, then to walk a distance $2D$ to the left. Indeed, it can be easily observed that the competitive ratio of this straightforward algorithm is 3, which is optimal when the agent has prior knowledge of D. As a result, in a scenario where the agent initially knows an upper bound Δ on D, the competitive ratio of an optimal treasure hunt algorithm must lie somewhere between 3 and 9. A more accurate answer is brought by [López-Ortiz & Schuierer 2001], in which the authors not only show that the competitive ratio of any search strategy is at least $9-\mathcal{O}(\frac{1}{\log^2\Delta})$ when an upper bound Δ is known by the agent, but also construct an algorithm achieving this competitive ratio (albeit with a different constant factor in the "big-Oh" term).

The problem can be generalized to find a target in the plane. In this case, the agent needs to have a radius of vision of any positive value ε, allowing it to see all its surroundings at distance at most ε from its current location, or otherwise the treasure hunt problem is impossible to solve due to enumerability reasons (cf. Section 1.2.2). It is well known that the optimal cost to find a treasure located at distance at most $D > \varepsilon$ in the plane belongs to $\Theta(D^2)$, if the agent does not have any prior information on the location of its target. Indeed, a cost of $\mathcal{O}(D^2)$ can be reached by tracing a square spiral starting at the initial position p of the agent in which the length of each edge increases by ε for every half-turn, while the lower bound $\Omega(D^2)$ can be proven using the fact that after having followed any trajectory of length $\frac{D^2}{4\varepsilon}$, there exists necessarily a point in the disc of area $\pi\dot{D}^2$ centered at p that has not been seen by the agent. In order to circumvent this bound of $\Omega(D^2)$, some studies have investigated the problem by assuming that additional information is initially provided to the agent and by analyzing the effect of decreasing its amount [Baeza-Yates *et al.* 1993, Pelc & Yadav 2019, Pelc & Yadav 2021], while others studies have supposed the search is restricted to some polygons [Klein 1991, Kleinberg 1994, Lee *et al.* 1997, Kranakis & Spatharis 1997, Icking *et al.* 2004]. In [Lu *et al.* 2016] (resp. [Chrobak *et al.* 2015]) the problem is extended to a scenario in which there are several searchers that cooperate to find a treasure in the plane (resp. in an infinite line), and then further studied assuming the agents may be prone to faults in [Czyzowicz *et al.* 2019a] (resp. in [Czyzowicz *et al.* 2019b, Czyzowicz *et al.* 2021]).

Apart from the two surveys [Alpern & Gal 2003] and [Ghosh & Klein 2010], all the aforementioned studies are exclusively related to continuous environments. However, a lot of effort has also been dedicated to studying the problem in discrete

environments modeled as graphs, for which treasure hunt and exploration are closely interrelated. The primary focus of past research on graph exploration has predominantly been on labeled graphs (*i.e.,* the same as in GBM, but by assuming that all nodes have distinct pairwise labels that permit to distinguish them unambiguously). The fastest exploration algorithm to date for arbitrary labeled graphs is the one given in [Panaite & Pelc 1999], which allows an agent to traverse all edges of a graph $G = (V, E)$ using at most $|E| + \mathcal{O}(|V|)$ moves. Indeed, the "penalty" of the algorithm of [Panaite & Pelc 1999], *i.e.,* the worst-case number of traversals made in excess of the trivial lower bound $|E|$, is linear in $|V|$, which is significantly better than the other classic strategies of exploration that all have a penalty that is at least super-linear with respect to this parameter (for instance, the standard Depth-First Search algorithm, which takes $2|E|$ moves, has a penalty that is quadratic in $|V|$). In [Betke *et al.* 1995], the authors introduce the problem of fuel-constrained exploration of a labeled graph in which the mobile agent has a fuel tank that can be replenished only at its starting node s. The size of this tank is $B = 2(1 + \alpha)r$, for some positive real constant α, where r, called the radius of the graph, is the maximum distance from s to any other node. The tank imposes an important constraint, as it forces the agent to make at most $\lfloor B \rfloor$ edge traversals before having to refuel at node s, otherwise the agent will be left with an empty tank and unable to move, preventing further exploration of the graph. The authors of [Betke *et al.* 1995] give fuel-constrained algorithms running in $\mathcal{O}(|E| + |V|)$ but for some classes of graphs only. The study is continued in [Awerbuch & Kobourov 1998] and in [Awerbuch *et al.* 1999], in which are provided an $\mathcal{O}(|E| + |V|^{1+o(1)})$ algorithm and an $\mathcal{O}(|E| + |V| \log^2 |V|)$ that both work in arbitrary graphs. A fuel-constrained exploration algorithm requiring at most $\mathcal{O}(|E| + |V|)$ edge traversals is finally obtained in [Duncan *et al.* 2006]. Although the authors of [Awerbuch *et al.* 1999] and [Duncan *et al.* 2006] are mainly interested in exploration of finite unknown graphs, they also get interesting corollaries for the treasure hunt problem, still under the same fuel constraint, assuming that a treasure is located at distance at most $D \geq 1$ from node s. Precisely, in [Awerbuch *et al.* 1999] (resp. [Duncan *et al.* 2006]), a treasure hunt algorithm working at cost $O(e(D + o(D)) + (s(D + o(D))^{1+o(1)})$ (resp. $\mathcal{O}(e((1 + \alpha)D))$) is provided, where, for any positive integer x, $s(x)$ is the number of nodes at distance at most x from node s and $e(x)$ is the number of edges whose at least one endpoint is at distance less than x from s. However, the cost of both these algorithms cannot be bounded by any function of $e(d)$ (as $e(D + o(D))$ and $e((1 + \alpha)D)$ may be arbitrarily larger than $e(d)$), seeming to confirm the following conjecture of Awerbuch, Betke, Rivest and Singh, formulated for the fuel-restricted model:

Conjecture ([Awerbuch *et al.* 1999]): *Is it possible (we conjecture not) to find a treasure in time nearly linear in the number of those vertices and edges whose distance to the source is less than or equal to that of the treasure?*[1]

[1] Time in this conjecture is what we call cost, i.e., the worst-case number of edge traversals until

When the nodes of the graphs have no label (*i.e.,* as in GBM), the task of treasure hunt becomes undeniably harder to solve. Nonetheless, it remains feasible, whether probabilistically or deterministically. If randomization is allowed, we know from [Aleliunas *et al.* 1979] that a simple random walk can cover, with high probability, all vertices of an unlabeled graph (and thus can find the treasure) at a cost polynomial in its size. If randomization is not allowed, an entire exploration can still be conducted, using the universal exploration sequences (known as UXS) introduced in [Koucký 2002], to which we briefly alluded in Section 2.2.2. Formally, a sequence of integers $x_1, x_2, \ldots, x_k$ is said to be a UXS for a class $\mathcal{G}$ of graphs, if it allows an agent, starting at any node of any graph $G \in \mathcal{G}$, to visit at least once every node of G in $k+1$ steps as follows. In step 1, the agent leaves its starting node v_1 by port 0 and enters node $v_2 = \texttt{succ}(v, 0)$. In step $2 \leq i \leq k+1$, the agent leaves its current node v_i by port $q = (p + x_{i-1}) \mod \texttt{deg}(\texttt{v}_\texttt{i})$, where p is the port by which it entered v_i in step $i-1$, and enters node $v_{i+1} = \texttt{succ}(v_i, q)$. In [Reingold 2008], it is proven that for any positive integer n, a UXS of polynomial length in n, for the class of all graphs of size at most n, can be computed deterministically in logarithmic space and in polynomial time in n, thus implying a deterministic solution for finding a treasure in any unlabeled graph at a cost polynomial in its size (whether this size is initially known or not).

What did we do about the problem of treasure hunt? We contributed to the treasure hunt problem both in the plane and in arbitrary graphs. For the plane, we considered a scenario in which the searching agent gets hints to find the treasure [Bouchard *et al.* 2020b]. However, contrary to other studies related to treasure hunt with advice, the agent does not get all the hints at the start: instead, they are provided at the beginning and at the end of each of its moves under the form of a positive angle that is supposed to contain the treasure. We investigated the problem of how these hints permit the agent to lower the cost of finding the treasure and obtained interesting results depending on how large are the hints.

For arbitrary graphs, we refuted the conjecture, given above, of Awerbuch, Betke, Rivest and Singh by showing an algorithm [Bouchard *et al.* 2023a] for the fuel-constraint model that permits to find a treasure at distance at most D from a source node using at most $\mathcal{O}(e(D) \log D)$ edge traversals (which is indeed nearly linear in $e(D)$).

All of this is further described in the following two sections.

4.2 Treasure hunt in the plane with angular hints

In [Bouchard *et al.* 2020b], we address the problem of treasure hunt in the plane with angular hints. More precisely, we consider model PBM, but by assuming:

- The adversary wakes up exactly one agent, called the searching agent, while

finding the treasure.

the other, called the treasure and initially located at distance at most D from the searching agent, always remains dormant/idle.

- In the beginning and after each move, the searching agent gets a hint consisting of a positive angle smaller than 2π whose vertex is at its current position and within which the treasure is contained.

Without any hints, the cost of finding the treasure would be $\Theta(D^2)$. But with them, we can beat this cost. Indeed, we show that if all angles given as hints are at most π, then the cost can be lowered to $O(D)$, which is the optimal complexity. We also show that if all angles are at most β, where $\beta < 2\pi$ is a constant unknown to the searching agent, then the cost is at most $O(D^{2-\varepsilon})$, for some $\varepsilon > 0$. Finally, we observe that arbitrary angles smaller than 2π given as hints cannot be of significant help: using such hints the cost complexity $\Theta(D^2)$ cannot be beaten.

For both our positive results, we give deterministic algorithms achieving the above costs. Both algorithms work in phases "assuming" that the treasure is contained in increasing squares centered at the initial position of the agent. The common principle behind both algorithms is to move the agent to strategically chosen points in the current square, depending on previously obtained hints, and sometimes perform exhaustive search of small rectangles from these points, in order to guarantee that the treasure is not there. This is done in such a way that, in a given phase, obtained hints together with small rectangles exhaustively searched, eliminate a sufficient area of the square assumed in the phase to eventually permit finding the treasure.

In both algorithms, the points to which the agent travels and where it gets hints are chosen in a natural way, although very differently in each of the algorithms. The main difficulty is to prove that the distance travelled by the agent is within the promised cost. In the case of the first algorithm, it is possible to cheaply exclude large areas not containing the treasure, and thus find the treasure asymptotically optimally. For the second algorithm, the agent eliminates smaller areas at each time, due to less precise hints, and thus finding the treasure costs more.

The next two subsections are dedicated to the presentation of the intuitions that are behind the two algorithms.

4.2.1 Angles at most π

In this section, we consider the case when all angles given as hints are at most π. Without loss of generality, we can assume that they are all equal to π, completing any smaller angle to π in an arbitrary way: this makes the situation even harder for the agent, as hints become less precise. For such hints we show an algorithm, called `TreasureHunt1`, that finds the treasure at cost $O(D)$. This is of course an optimal complexity, as the treasure can be at any point at distance at most D from the starting point of the agent.

For angles of size π, every hint is in fact a half-plane whose boundary line L contains the current location of the agent. For simplicity, we can code such a hint

as a couple $(L, right)$ or $(L, left)$, whenever the line L is not horizontal, depending on whether the indicated half-plane is to the right (i.e., East) or to the left (i.e., West) of L. For any non-horizontal line L this is non-ambiguous. Likewise, when L is horizontal, we can code a hint as a couple (L, up) or $(L, down)$, depending on whether the indicated half-plane is up (i.e., North) from L or down (i.e., South) from L.

In view of the work on ϕ-self-approaching curves (cf. [Aichholzer *et al.* 2001]) we first note that there is a big difference of difficulty between obtaining our result in the case when angles given as hints are bounded by some angle ϕ_0 strictly smaller than π and when they are *at most* π, as we assume. A ϕ-self-approaching curve is a planar oriented curve such that, for each point B on the curve, the rest of the curve lies inside a wedge of angle ϕ with apex in B. In [Aichholzer *et al.* 2001], the authors prove the following property of these curves: for every $\phi < \pi$ there exists a constant $c(\phi)$ such that the length of any ϕ-self-approaching curve is at most $c(\phi)$ times the distance D between its endpoints. Hence, for hints bounded by some angle ϕ_0 strictly smaller than π, our result could possibly be derived from the existing literature: roughly speaking, the agent should follow a trajectory corresponding to any ϕ_0-self-approaching curve to find the treasure at a cost linear in D. Even then, transforming the continuous scenario of self-approaching curves to our discrete scenario presents some difficulties. However, the crucial problem is this: the constant $c(\phi)$ from [Aichholzer *et al.* 2001] diverges to infinity as ϕ approaches π, hence the result from [Aichholzer *et al.* 2001] cannot be used when hints are arbitrary angles smaller than π. Moreover, the result of [Aichholzer *et al.* 2001] holds only when $\phi < \pi$ (the authors also emphasize that for each $\phi \geq \pi$, the property is false), and thus the above derivation is no longer possible for our purpose when $\phi = \pi$. Actually, this is the real difficulty of our problem: handling angles equal to π, i.e., half-planes.

We further observe that a rather straightforward treasure hunt algorithm of cost $O(D \log D)$, for hints being angles of size π, can be obtained using an immediate corollary of a theorem proven in [Grünbaum 1960]: each line passing through the centroid of a convex polygon cuts the polygon into two convex polygons with areas differing by a factor of at most $\frac{5}{4}$. Suppose for simplicity that D is known. Starting from the square of side length $2D$, centered at the initial position of the agent, this permits to reduce the search area from P to at most $\frac{5P}{9}$ in a single move. Hence, after $O(\log D)$ moves, the search area is small enough to be exhaustively searched by procedure `RectangleScan` at cost $O(D)$. However, the cost of each move during the reduction is not under control and can be only bounded by a constant multiple of D, thus giving the total cost bound $O(D \log D)$. By contrast, our algorithm controls both the remaining search area and the cost incurred in each move, yielding the optimal complexity $O(D)$ of the cost.

Now, let us turn our attention to the high-level idea of Algorithm `TreasureHunt1`. In this algorithm, the agent acts in phases $j = 1, 2, 3, \ldots$ where in each phase j the agent "supposes" that the treasure is in a straight square R_j centered at the initial position of the agent, and of side length 2^j. When executing a phase j, the agent successively moves to distinct points with the aim of using the

hints at these points to narrow the search area that initially corresponds to R_j. In our algorithm, this narrowing is made in such a way that the remaining search area is always a straight rectangle. Often this straight rectangle is a strict superset of the intersection of all hints that the agent was given previously. This would seem to be a waste, as we are searching some areas that have been previously excluded. However, this loss is compensated by the ease of searching description and subsequent analysis of the algorithm, due to the fact that, at each stage, the search area is very regular.

During a phase, the agent proceeds to successive reductions of the search area by moving to distinct locations, until it obtains a rectangular search area that is small enough to be searched directly at low cost using a brute-force procedure that we called `RectangleScan`, which permits to see all the points within a given rectangle at a cost linear in its area. In our algorithm, such a final execution of `RectangleScan` in a phase is triggered as soon as the rectangle has a side smaller than 4. If the treasure is not found by the end of this execution of procedure `RectangleScan`, the agent learns that the treasure cannot be in the supposed straight square R_j and starts the next phase from scratch by forgetting all previously received hints. This forgetting again simplifies subsequent analysis. The algorithm terminates at the latest by the end of phase $j_0 = \lceil \log_2 D \rceil + 1$, in which the supposed straight square R_{j_0} is large enough to contain the treasure. Hence, if the cost of a phase j is linear in 2^j, then the cost of the overall solution is linear in the distance D.

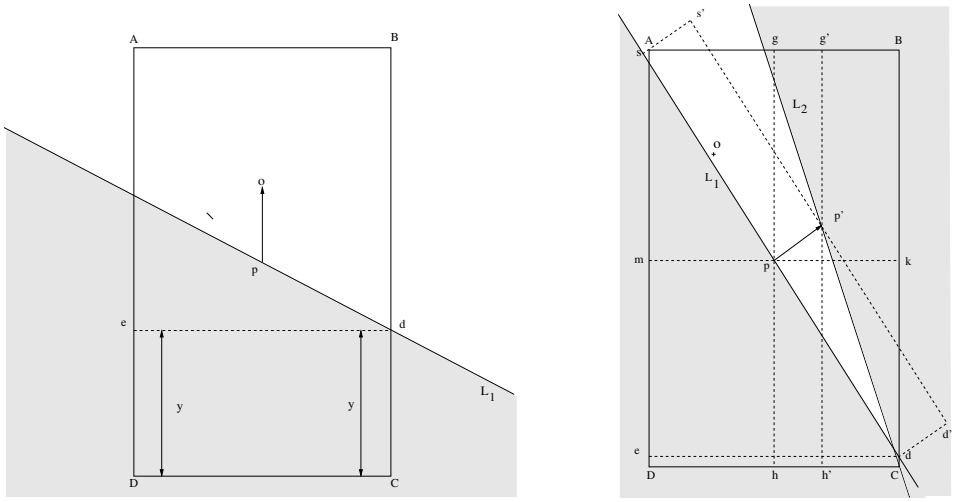
In order to give the reader deeper insights in the reasons why our solution is valid and has linear cost, we need to give more precise explanations on how the search area is reduced during a given phase $j \geq 2$ (when $j = 1$, the agent makes no reduction and directly scans the small search area using procedure `RectangleScan`). Suppose that in phase $j \geq 2$ the agent is at the center p of a search area corresponding to a straight rectangle R, every side of which has length between 4 and 2^j (note that this is the case at the beginning of the phase), and denote by A, B, C and D the vertices of R starting from the top left corner and going clockwise. In order to reduce rectangle R, the agent uses the hint at point p. The obtained hint denoted by (L_1, x_1) can be of two types: either a *good* hint or a *bad* hint. A good hint is a hint whose line L_1 divides one of the sides of R into two segments such that the length y of the smaller one is at least 1. A bad hint is a hint that is not good.

If the received hint (L_1, x_1) is good, then the agent narrows the search area to a rectangle $R' \subset R$ having the following three properties:

1. $R \setminus R'$ does not contain the treasure.
2. The difference between the perimeters of R and R' is $2y \geq 2$.
3. The distance from p to the center o of R' is exactly $\frac{y}{2}$.

and then moves to the center o of R'.

An illustration of such a reduction is depicted in Figure 4.1(a), in which the reduced search area R' is the rectangle $ABde$.



(*a*) A good hint $(L_1, right)$ (*b*) A bad hint $(L_1, right)$

Figure 4.1: In Figure (a) the agent received a good hint $(L_1, right)$ at the center p of a rectangular search area $R = ABCD$ and hence it moved to the center o of the rectangle $R' = ABde$. In Figure (b) it received a bad hint $(L_1, right)$ at the point p and hence it moved to point p' from which it got a hint $(L_2, left)$. From there, it will scan the two rectangles $ss'd'd$ and $gg'h'h$ using procedure `RectangleScan`, and then it will move to the center o of the rectangle $R' = Agpm$. In both figures, the half-planes excluded by the hints are shaded.

If the agent receives a bad hint, say $(L_1, right)$, at the center of a rectangular search area R, we cannot apply the same method as the one used for a good hint: this is the reason for the distinction between good and bad hints. If we applied the same method as before, we could obtain a rectangular search area R' such that the difference between the perimeters of R and R' is at least $2y$. However, in the context of a bad hint, the difference $2y$ may be very small (even null), and hence there is no significant reduction of the search area. In order to tackle this problem, when getting a bad hint at the center p of R, the agent moves to another point p' which is situated in the half-plane $(L_1, right)$ at distance 2 from p, perpendicularly to L_1. This point p' is chosen in such a way that, regardless of what is the second hint, we can ensure that two important properties described below are satisfied.

The first property is that by combining the two hints, the agent can decrease the search area to a rectangle $R' \subset R$ whose perimeter is smaller by 2 compared to the perimeter of R, as it is the case for a good hint, and such that $R \setminus R'$ does not contain the treasure. This decrease follows either directly from the pair of hints, or indirectly after having scanned some relatively small rectangles using procedure `RectangleScan`. In the example depicted in Fig. 4.1 (b), after getting the second hint $(L_2, left)$, the agent executes procedure `RectangleScan` in rectangle $ss'd'd$ followed by `RectangleScan` in rectangle $gg'h'h$, and moves to the center o of the new search area R' that is the rectangle $Agpm$. Note that the part of R' not excluded by the two hints and by the procedure `RectangleScan` executed in rectangles $ss'd'd$ and $gg'h'h$ is only the small quadrilateral bounded by line L_2 and the segments $[AB]$, $[s'd']$ and $[gh]$. However, in order to preserve the homogeneity of the process, we consider the entire new search area R' which is a straight rectangle whose perimeter is smaller by at least 2, compared to that from R. This follows from the fact that no side of R has length smaller than 4. The agent finally moves to the center of R'.

The second property is that all of this (i.e., the move from p to p', the possible scans of small rectangles and finally the move to the center of R') is done at a cost linear in the difference of perimeters of R and R'. The two properties together ensure that, even with bad hints, the agent manages to reduce the search area in a significant way and at a small cost. So, regardless of whether hints are good or not, we can show that the cost of phase j is in $\mathcal{O}(2^j)$ and the treasure is found during this phase if the initial square is large enough. The difficulty of the solution is in showing that the moves prescribed by our algorithm in the case of bad hints guarantee the two above properties, and thus ensure the correctness of the algorithm and the cost linear in D.

4.2.2 Angles bounded by $\beta < 2\pi$

In this section, we give the high-level idea of our algorithm called `TreasureHunt2` that permits to find the treasure at a cost subquadratic when all hints are angles upper bounded by some constant $\beta < 2\pi$, unknown to the agent. Note that the way of defining a hint in the previous subsection is no longer appropriate here: we need

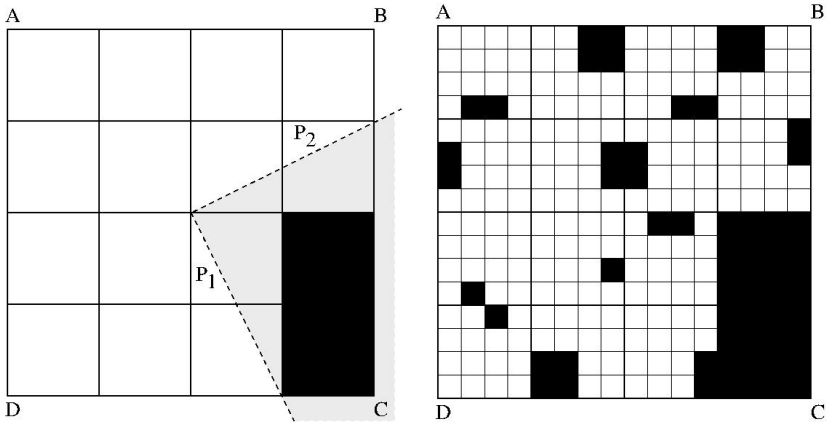
a more general definition. Hence, in this context, a hint given to the agent currently located at some point a is formally described as an ordered pair (P_1, P_2) of half-lines originating at a such that the angle clockwise from P_1 to P_2 (including P_1 and P_2) contains the treasure. For a hint (P_1, P_2) we denote by $\overline{(P_1, P_2)}$ the complement of (P_1, P_2).

In Algorithm `TreasureHunt2`, similarly as in the previous algorithm, the agent acts in phases $j = 1, 2, 3, \ldots$, where in each phase j the agent "supposes" that the treasure is in the straight square centered at its initial position and of side length 2^j. The intended goal is to search each supposed square at relatively low cost, and to ensure the discovery of the treasure by the time the agent finishes the first phase for which the initial supposed square contains the treasure. However, the similarity with the previous solution ends there: indeed, the hints that may now be less precise do not allow us to use the same strategy within a given phase. Hence, we adopt a different approach that we outline below and that uses the following notion of tiling. Given a square S with side of length $x > 0$, $Tiling(i)$ of S, for any non-negative integer i, is the partition of square S into 4^i squares with side of length $\frac{x}{2^i}$. Each of these squares, called *tiles*, is closed, i.e., contains its border, and hence neighboring tiles overlap in the common border.

Let us consider a simpler situation in which the angle of every hint $[P_1, P_2]$ is always equal to the bound β: the general case, when the angles may vary while being at most β, adds a level of technical complexity that is unnecessary to understand the intuition. In the considered situation, the angle of each excluded zone $\overline{[P_1, P_2]}$ is always the same as well. The following property holds in this case: there exists an integer i_β such that for every square S and every hint $[P_1, P_2]$ given at the center of S, at least one tile of $Tiling(i_\beta)$ of S belongs to the excluded zone $\overline{[P_1, P_2]}$.

In phase j, the agent performs k steps: we will indicate later how the value of k should be chosen. At the beginning of the phase, the entire square S is white. In the first step, the agent gets a hint $[P_1, P_2]$ at the center of S. By the above property, we know that $\overline{[P_1, P_2]}$ contains at least one tile of $Tiling(i_\beta)$ of S, and we have the guarantee that such a tile cannot contain the treasure. All points of all tiles included in $\overline{[P_1, P_2]}$ are painted black in the first step. This operation does not require any move, as painting is performed in the memory of the agent. As a result, at the end of the first step, each tile of $Tiling(i_\beta)$ of S is either black or white, in the following precise sense: a black tile is a tile all of whose points are black, and a white tile is a tile all of whose *interior* points are white.

In the second step, the agent repeats the painting procedure at a finer level. More precisely, the agent moves to the center of each white tile t of $Tiling(i_\beta)$ of S. When it gets a hint at the center of a white tile t, there is at least one tile of $Tiling(i_\beta)$ of t that can be excluded. As in the first step, all points of these excluded tiles are painted black. Note that a tile of $Tiling(i_\beta)$ of t is actually a tile of $Tiling(2i_\beta)$ of S. Moreover, each tile of $Tiling(i_\beta)$ of S is made of exactly 4^{i_β} tiles of $Tiling(2i_\beta)$ of S. Hence, as depicted in Figure 4.2, the property we obtain at the end of the second step is as follows: each tile of $Tiling(2i_\beta)$ of S is either black or white.



(*a*) At the end of a first step for a hint $[P_1, P_2]$

(*b*) At the end of a second step

Figure 4.2: White and black tiles at the end of the first and the second step of a phase, for square $S = ABCD$ and $i_\beta = 2$.

In the next steps, the agent applies a similar process at increasingly finer levels of tiling. More precisely, in step $2 < s \leq k$, the agent moves to the center of each white tile of $Tiling((s-1)i_\beta)$ of S and gets a hint that allows it to paint black at least one tile of $Tiling(s \cdot i_\beta)$ of S. At the end of step s, each tile of $Tiling(s \cdot i_\beta)$ of S is either black or white. We can show that at each step s the agent paints black at least $\frac{1}{4^{i_\beta}}$th of the area of S that is white at the beginning of step s.

After step k, each tile of $Tiling(k \cdot i_\beta)$ of S is either black or white. These steps permit the agent to exclude some area without having to search it directly, while keeping some regularity of the shape of the black area. The agent paints black a smaller area than excluded by the hints but a more regular one. This regularity enables in turn the next process in the area remaining white. Indeed, the agent subsequently executes a brute-force searching that consists in moving to each white tile of $Tiling(k \cdot i_\beta)$ of S in order to scan it using the procedure `RectangleScan`. If, after having scanned all the remaining white tiles, it has not found the treasure, the agent repaints white all the square S and enters the next phase, *i.e.,* phase $j+1$. Note that if the treasure is initially in S, the agent necessarily sees it by the end of phase j because, during the brute-force searching mentioned above, the agent gets at distance at most 1 of all points belonging to a white tile of $Tiling(k \cdot i_\beta)$ of S. Thus, we have the guarantee that the agent finds the treasure by the end of phase $\lceil \log_2 D \rceil + 1$, as in this phase the initial supposed square is large enough to contain the treasure. The question is: how much do we have to pay for all of this? In fact, the cost depends on the value that is assigned to k in each phase j. The value of k must be large enough so that the distance travelled by the agent during the brute-force searching is relatively small. At the same time, this value must be small enough so that the distance travelled during the k steps is not too large. A good trade-off can be reached when $k = \lceil \log_{4^{i_\beta}} \sqrt{2^j} \rceil$. Indeed, as proven in

[Bouchard *et al.* 2020b], it is due to this carefully chosen value of k that we can beat the cost $\Theta(D^2)$ necessary without hints, and get a complexity of $\mathcal{O}(D^{2-\varepsilon})$, where ε is a positive real depending on i_β, and hence depending on the angle β.

4.3 Refutation of Awerbuch, Betke, Rivest and Singh's conjecture

The main result of our paper [Bouchard *et al.* 2023a] is the refutation of Awerbuch, Betke, Rivest and Singh's conjecture (given in the introduction of this chapter). This conjecture has been formulated for a variant of model `GBM`. The differences between this variant and `GBM` are precisely as follows:

- The adversary wakes up exactly one agent, called the searching agent, while the other, called the treasure, always remains dormant/idle.
- The nodes of the underlying graphs have distinct pairwise labels.
- The searching agent has a fuel tank that can be replenished only at its starting node s. The size of this tank is $B = 2(1+\alpha)r$, for some positive real constant α, where r, called the radius of the graph, is the maximum distance from s to any other node. Each edge traversal consumes one unit of fuel. When the fuel tank is full, the agent can make at most $\lfloor B \rfloor$ edge traversals before having to refuel at node s, as with less than one unit of fuel, the agent cannot move anymore.

Our refutation is conducted in two steps. The first step consists in designing a deterministic treasure hunt algorithm working in the above model variant, but without the fuel constraint, at cost $\mathcal{O}(e(D)\log D)$, where D is an upper bound on the distance between s and the treasure. The second step consists in showing how to modify this algorithm so that it works at an asymptotically equal cost even with the additional fuel constraint, *i.e.,* in the model in which the conjecture has been established. Since we observe that no treasure hunt algorithm can beat cost $\Theta(e(d))$ for all graphs in the models of the first and the second step, it turns out that our algorithms are also almost optimal, *i.e.,* optimal up to a logarithmic factor (indeed, $D < e(D)$ and thus $\log D < \log e(D)$), which leaves open the question of whether it is possible to get rid of the factor $\mathcal{O}(\log D)$.

The major difficulty of our work residing in the first step, the purpose of the rest of this section will essentially be to sketch an intuitive overview of our algorithm that permits to find the treasure at an almost-optimal cost in the above model variant but deprived of the fuel constraint. To this end, and to simplify the discussion, we will assume that the underlying graph G is countably infinite with nodes of finite degrees. We will rely on the notion of *largest explored ball*. Given a non-negative integer k, the ball $B_k(G,s)$ is the subgraph of G induced by nodes at distance at most k from the source node s without edges for which the two endpoints are at

distance exactly k from s in G (thus the number of edges in $B_k(G,s)$ is $e(k)$). Now, by "largest explored ball", at a given phase of treasure hunt, we mean the ball $B_f(G,s)$ where f is the largest integer such that each edge of $B_f(G,s)$ has been traversed at least once. This largest integer f is the radius of the largest explored ball.

At a high level, our algorithm works in phases $i = 1, 2, 3, \ldots$ and immediately stops as soon as the treasure is found. At the beginning of phase i, the agent is located at node s and the radius of the largest explored ball is equal to f_i. The goal for the agent is to terminate the phase at node s while satisfying at least one of the following three conditions unless, of course, the treasure has been found before.

- *Condition 1.* The agent has entirely explored ball $B_{f_i+1}(G,s)$, $e(f_i+1) \geq 2e(f_i)$ and the cost of the phase is $\mathcal{O}(e(f_i+1))$.
- *Condition 2.* The agent has entirely explored ball $B_{2f_i}(G,s)$, $f_i \geq 1$ and the cost of the phase is $\mathcal{O}(e(f_i))$.
- *Condition 3.* The agent has entirely explored ball $B_{f_i+k}(G,s)$ for some positive integer k, $e(f_i+k+1) \geq 2e(f_i)$, $f_i \geq 2$, and the cost of the phase is $\mathcal{O}(e(f_i)\log f_i)$.

Actually, the conditions we really seek to meet in our algorithm are a little more intricate than those presented above, because we needed stronger technical requirements in order to be subsequently able to show the transformation of the second step mentioned earlier. However, this would add an unnecessary level of complexity to understand the intuition, hence we omit these technical details here.

The radius of the largest explored ball at the beginning of phase $i+1$, i.e., f_{i+1} is guaranteed to have a minimal value that depends on the condition that is fulfilled at the end of phase i. Precisely, if it is Condition 1 or Condition 3 that is fulfilled at the end of phase i, then there is a positive integer k such that $B_{f_i+k}(G,s)$ is entirely explored at the end of phase i, which means that $f_{i+1} \geq f_i + k$. And, if it is Condition 2 that is fulfilled at the end of phase i, then $B_{2f_i}(G,s)$ is entirely explored at the end of phase i, which means that $f_{i+1} \geq 2f_i$. Note that, the fact that $f_i \geq 1$ in Condition 2 guarantees that the radius of the largest explored ball increases by at least one at the end of *any* phase in which the treasure is not found. Also note that, in Condition 3, the requirement $f_i \geq 2$ is due to the presence of $\log f_i$ within the big O notation.

Before seeing how we implement our strategy, let us briefly examine why it permits us to get a cost quasi-linear in $e(D)$. Since $f_1 = 0$ and the radius of the largest explored ball increases by at least one during each phase in which the treasure is not found, the agent necessarily finds the treasure by the end of some phase $\lambda \leq D$, and $f_i < f_\lambda < D$ for every $1 \leq i < \lambda$. During each phase satisfying Condition 1, the size of the largest explored ball at least doubles, which means that the total cost of these phases is upper bounded by twice the worst-case cost of the last phase satisfying Condition 1 i.e., $\mathcal{O}(e(f_\lambda+1))$. Concerning the phases fulfilling Condition 2, their

number is at most $\mathcal{O}(\log(f_\lambda+1))$ and the cost of each of them cannot be more than $\mathcal{O}(e(f_\lambda))$, which implies that their total cost is $\mathcal{O}(e(f_\lambda)\log(f_\lambda+1))$. It remains to consider the case of the phases satisfying Condition 3. Given such a phase i, we have the guarantee that the size of the largest explored ball at least doubles between the beginning of phase i and the end of phase $i+1$, provided phase $i+1$ exists and is not prematurely interrupted by the discovery of the treasure. Indeed, at the end of phase i, the agent has at least entirely explored ball $B_{f_i+k}(G,s)$ for some positive integer k and $e(f_i+k+1) \geq 2e(f_i)$, while at the end of the (not prematurely interrupted) phase $i+1$ the agent has at least entirely explored ball $B_{f_{i+1}+1}(G,s)$ with $f_{i+1} \geq f_i+k$. Using this, it can be shown that the total cost of the phases satisfying Condition 3 is at most four times the worst-case cost of the last phase satisfying this condition, *i.e.,*$\mathcal{O}(e(f_\lambda)\log(f_\lambda+1))$. Given that the last phase λ can be viewed as a truncated phase that should have normally satisfied one of the three conditions, our sketch of analysis leads to the conclusion that the cost incurred by the agent till the discovery of the treasure is in $\mathcal{O}(e(f_\lambda+1)\log(f_\lambda+1))$, which is $\mathcal{O}(e(D)\log D)$ and is in line with our expectations.

Having justified the pertinence of such a strategy, we can turn our attention to its implementation. To do so, we need to introduce a technical building block, called `GlobalExpansion`(l,m) to which we will go back at the end of this section to give additional details. Always executed from the source node s, it is a function that returns a boolean and whose two input parameters are positive integers except m that may be sometimes equal to the special symbol $\perp$. Assuming that $B_f(G,s)$ is the largest explored ball, the execution of `GlobalExpansion`$(l,\perp)$ permits the agent to traverse all the edges of $B_{f+l}(G,s)$ that are outside of $B_f(G,s)$ before coming back to node s. Under the same assumption, the execution of `GlobalExpansion`(l,m), when m is a positive integer, consists for the agent in acting as if m was $\perp$ but with the following extra requirement: as soon as more than m distinct edges outside of $B_f(G,s)$ have been traversed during the execution of the function, the agent backtracks to node s and aborts this execution. If m is $\perp$ or at least large enough to avoid an aborted execution, the agent ends up exploring $B_{f+l}(G,s)$ and the function returns true. Otherwise, the function returns false. It should be stressed that all of this is made while guaranteeing two properties. The first one is that the agent is always in $B_{f+2l-1}(G,s)$ during the execution of `GlobalExpansion`(l,m). The second is that the cost of the execution of `GlobalExpansion`(l,m) is $\mathcal{O}(e(f+2l-1))$ (resp. $\mathcal{O}(\min\{e(f)+m, e(f+2l-1)\})$) when $m = \perp$ (resp. $m \neq \perp$). Both these properties will turn out to be crucial to ensure a proper design of the phases. Finally, even if by chance the agent could explore a larger ball, we will assume for the ease of our intuitive explanations that $B_{f+l}(G,s)$ (resp. $B_f(G,s)$) is the largest ball explored by the agent at the end of `GlobalExpansion`(l,m) in the case where the returned value is true (resp. false).

Let us consider a phase i of our algorithm and, in order not to burden the text with a lot of "unless the treasure is found", let us assume that the treasure will not be found by the end of it. Phase i is made of at most three successive attempts, each of them aiming at fulfilling at least one of the three conditions

described earlier, with the help of our building block. In the first attempt, the agent executes `GlobalExpansion`$(1, \perp)$ from node s, the cost of which is $\mathcal{O}(e(f_i + 1))$. At the end of this execution, the agent is at node s and $B_{f_i+1}(G, s)$ has been entirely explored by the agent. If $e(f_i + 1) \geq 2e(f_i)$ or $f_i \leq 1$, the first attempt is a success as Condition 1 or Condition 2 is fulfilled, and the agent directly switches to phase $i + 1$. Otherwise, the attempt is a failure, but we can nonetheless observe that the cost incurred because of the attempt is just $\mathcal{O}(e(f_i))$ because $e(f_i + 1) < 2e(f_i)$.

If the first attempt has failed, the agent starts the second attempt of phase i that consists of an execution of function `GlobalExpansion`$(f_i - 1, e(f_i))$. The hope here is to expand by a distance of $f_i - 1$ the radius of the largest explored ball, which is $B_{f_i+1}(G, s)$. According to the properties of `GlobalExpansion` and the fact that $e(f_i + 1) < 2e(f_i)$, the cost of this execution, and thus of the second attempt, is $\mathcal{O}(e(f_i))$. If `GlobalExpansion`$(f_i - 1, e(f_i))$ returns true, then at the end of the second attempt, the radius of the largest explored ball is $2f_i$. Hence, the cost of the first two attempts being equal to $\mathcal{O}(e(f_i))$ and f_i being at least 2, Condition 2 is satisfied and the agent starts phase $i + 1$ without making the third attempt.

On the other hand, if `GlobalExpansion`$(f_i - 1, e(f_i))$ returns false, it is a different story. Indeed, the largest explored ball is still only $B_{f_i+1}(G, s)$ and we cannot ensure the fulfillment of Condition 1 or Condition 2. This is exactly where Condition 3 comes into the picture. In order to remedy the failures of the two previous attempts, the agent will start a third and last attempt which consists of a binary search that is described in Algorithm 5. At the end of this process, Condition 3 is guaranteed to be satisfied.

Algorithm 5: Third attempt

```
1 floor := f_i + 1; ceil := 3f_i − 2; l := ⌊(ceil − floor)/2⌋;
2 while l ≥ 1 and |B_floor(G, s)| < 2e(f_i) do
3     success := GlobalExpansion(l, e(f_i));
4     if success = true then
5         floor := floor + l; l := ⌊(ceil − floor)/2⌋;
6     else
7         ceil := floor + 2l − 1; l := ⌊l/2⌋;
```

In order to better understand why we can get such a guarantee, let us take a look at the properties that are satisfied during the third attempt and at its end.

Since the execution of `GlobalExpansion`$(f_i - 1, e(f_i))$ returned false, the agent has explored at least $e(f_i)$ distinct edges outside of ball $B_{f_i+1}(G, s)$ during the second attempt. Moreover, during this execution, the agent was always in $B_{3f_i-2}(G, s)$ according to the properties of `GlobalExpansion`. As a result, in view of the first line of Algorithm 5, we necessarily have the following feature before the execution of the while loop of Algorithm 5: $B_{floor}(G, s)$ is the largest explored ball and $e(ceil) \geq 2e(f_i)$. Actually, by carefully examining the pseudocode of the while loop and using again the properties of `GlobalExpansion`, it can be inductively proven that this feature is a loop invariant. Alone, this loop invariant is not enough to bring the

sought guarantee, but as highlighted below, it is of precious help to do the job.

The number of iterations of the while loop can be shown to be $\mathcal{O}(\log f_i)$. Furthermore, at the beginning of each iteration, $B_{floor}(G,s)$ has size smaller than $2e(f_i)$ in view of the condition of the while loop, and is the largest explored ball in view of the loop invariant. Hence, according to the cost property of `GlobalExpansion`, each execution of `GlobalExpansion`$(l, e(f_i))$ costs at most $\mathcal{O}(e(f_i))$ like the previous two attempts, which gives a total cost of $\mathcal{O}(e(f_i)\log f_i)$ of the whole phase. This corresponds exactly to the target value of Condition 3. Along with this, at the end of the while loop, the size of $B_{floor}(G,s)$ is at least $2e(f_i)$, or $l<1$. In the first case, we immediately have $e(floor+1) \geq 2e(f_i)$, while in the second case it can be shown that $ceil \leq floor+1$. This, combined with the fact that $e(ceil)$ is always at least $2e(f_i)$ (by the loop invariant) and the fact that $floor$ is always at least f_i+1, allows us to show the last missing piece of the puzzle, which is precisely this: when Algorithm 5 terminates, ball $B_{f_i+k}(G,s)$ is entirely explored and $e(f_i+k+1) \geq 2e(f_i)$ for some integer $k \geq 1$.

To conclude with the intuitive explanations, let us give, as promised, some more insight concerning the building block `GlobalExpansion`(l,m). At first glance, one might think that `GlobalExpansion` could be directly derived from the exploration algorithm `CFX`(v,r,α) of [Duncan *et al.* 2006], which permits to explore a ball $B_r(G,v)$ at a cost of $\mathcal{O}\left(\frac{|B_{(1+\alpha)r}(G,v)|}{\alpha}\right)$ for any given real $\alpha > 0$ (this corresponds to a cost of $\mathcal{O}\left(\frac{e((1+\alpha)r)}{\alpha}\right)$ when $v = s$) provided $\alpha r \geq 1$. Indeed, the task of `GlobalExpansion`(l,m) that consists in expanding the radius f of the largest explored ball by a distance l in the case where m is appropriately set, can be done with `CFX`$(s, f+l, \alpha)$. However, in this case we want the cost of this expansion to be $\mathcal{O}(e(f+2l-1))$, which is an important property of our strategy. This cannot be guaranteed using `CFX`$(s, f+l, \alpha)$ because, in order to get a cost depending on $e(f+2l-1)$, we would have to set α to a value lower than $\frac{l-1}{f+l}$, which cannot lead to a cost that is linear in $e(f+2l-1)$, as $\frac{l-1}{f+l}$ can be arbitrarily small. True, during the design we could have been "less demanding" about some of the properties of `GlobalExpansion`(l,m), but not significantly enough to permit the use of `CFX`$(s, f+l, \alpha)$ without spoiling the validity or the cost complexity of our strategy. Another solution that may come to mind would be to apply `CFX`(v, l, α) from each node v located on the boundary of the largest explored ball $B_f(G,s)$. Visiting each node of the boundary can be done in $\mathcal{O}(e(f))$. Hence, this solution looks attractive because by setting α to $\frac{1}{2}$ or less (which overcomes the above problem of the arbitrarily small value) and provided the zones explored by the different executions of `CFX` do not overlap, we would get a cost that is linear in $e(f+2l-1)$. The bad news is that there may be overlaps. Of course, some overlaps can be easily avoided, especially those appearing within $B_f(G,v)$, but some others cannot without running the risk of missing some nodes of $B_{f+l}(G,s)$ that are outside of $B_f(G,s)$. These "necessary overlaps" may be pernicious and may occur in a way that prevents us from guaranteeing a cost of $\mathcal{O}(e(f+2l-1))$.

So, what did we do? Although it was not possible to use `CFX` as a black box, we managed to tailor `GlobalExpansion` by adapting to our needs an elegant algorithmic technique used in `CFX`. Through a set of judiciously pruned trees spanning some already explored area, it allowed us to satisfy the desired cost property of `GlobalExpansion` by controlling and amortizing efficiently the number of times the same edges are traversed.

Chapter 5

Conclusion and perspectives

Throughout this manuscript, we have presented most of our work that has been made over the past ten years, by adopting, to the extent possible, an intuitive point of view. The main guiding thread of all this work can actually be summarized by the following central thematic question:

"What can and cannot be achieved by a group of mobile agents when they operate under difficult conditions?"

We have seen that behind the term "difficult", we understand, for example, a completely erratic asynchrony, the eventuality of having to deal with Byzantine faults, the possible lack of direct means of communication, the fact of being strongly limited in terms of energy, etc. We have addressed this question by focusing essentially on the problem of rendezvous and some of its variants, which are the problems of gathering and treasure hunting. This focus does not constitute a restriction as drastic and fundamental as one might think at first sight on the scope of our results, since meeting is a prerequisite building block for solving more sophisticated tasks in mobile distributed systems. The existing results in the literature related to this problem, including in particular those we have obtained and presented in the previous sections, allow us to outline some answers to the question raised above. Nevertheless, there are still aspects that have not been sufficiently explored. This is particularly the case for the optimal complexity of rendezvous and its variants under the various constraints that have been previously mentioned, and, more generally, for the precise interrelation between the magnitude of these constraints and the complexity that can then be expected. This observation forms the main basis of our medium- and long-term research project, which is articulated around three distinct axes.

The first of these axes consists in finding new original strategies to significantly beat the complexities, under the same restrictive assumptions, of the existing results concerning the rendezvous, gathering, and treasure hunt problems. Ideally, the goal is to obtain algorithms of optimal complexity that will improve those already established. The best candidates for such improvements are certainly the algorithms of [Bouchard *et al.* 2019, Bouchard *et al.* 2023b], which are described in Chapters 2 and 3, because they are, for most of them, either exponential or polynomial but with exponents having... three digits. And there is a good reason explaining this situation: in both cases, most of our work merely consisted in demonstrating feasi-

bility, or even polynomiality, but without seeking to optimize complexity. And what is true for these two results, is also true in a general way, although sometimes to a lesser extent, for lots of the algorithms from the literature concerning rendezvous and its variants under the difficult constraints listed above. In other words, although the latest rendezvous algorithms in highly constrained environments have allowed substantial progress to be made, there is still a long way to go in order to optimize them, and thus, this is naturally a very promising avenue of research.

Asking the question of improving the existing complexities, in order to reach optimality, also implies asking the question of the fundamental limits of the possible improvements. For this first axis, it is therefore also important to shed light on the lower bounds of the complexities that can be achieved. In fact, most of the lower bound results that exist in the literature concerning rendezvous in harsh conditions, either are quite elementary or do not take advantage of harshness. Thinking about it, this can seem quite surprising. Indeed, we know that demonstrating a lower complexity bound implies, in a manner of speaking, demonstrating the impossibility for any algorithm to "do less". However, when we are faced with difficult conditions such as asynchrony or the occurrence of Byzantine faults, to name just a few, we have a much larger number of possible combinations to cause any algorithm to fail than if we were in conditions that can be qualified as standard.

In this regard, a result of great beauty would be, for example, to show a lower bound that perfectly matches the quasi-linear complexity of our treasure hunt algorithm [Bouchard *et al.* 2023a], which contradicts the conjecture of B. Awerbuch, M. Betke, R.L. Rivest and M. Singh [Awerbuch *et al.* 1999] (cf. Chapter 4). The only lower bound that has been demonstrated stipulates that the problem cannot be solved at a sublinear cost, and it is quite easy to prove. Of course, one could argue that perhaps it is the complexity of the algorithm that should be further improved to achieve linearity. However, up to now, almost all efforts have been focused on the development of an efficient algorithm which is very complex, and almost none on the lower bound, for which the only existing proof is almost trivial. Thus, even if no eventuality can be ruled out, it seems that it is most likely on the side of the lower bound that improvements should be made.

The second axis of research is to rigorously determine the influence of difficult conditions on the efficiency that can be expected from a meeting algorithm. For example, we do not currently know what the additional cost on efficiency is of asynchrony or lack of communication. Does this translate into a constant, polylogarithmic or polynomial factor on complexity, or even more?

In one of our papers [Dieudonné & Pelc 2014b], which has not been covered in the previous chapters, we lay a first stone for such analyses in the context of mobile agents. In this paper, we show that for the problem of rendezvous in `PBM`, the optimal cost of its deterministic solution in the asynchronous scenario has higher order of magnitude than that in the synchronous scenario, when the agents know the initial distance D separating the agents. More precisely, we prove that in the synchronous scenario rendezvous can be performed at cost $O(D\ell)$, where ℓ is the length of the binary representation of the smaller label, while cost $\Omega(D^2 + D\ell)$

is needed for asynchronous completion of rendezvous. Hence, for instances with $\ell = o(D)$, the optimal cost of asynchronous rendezvous is asymptotically larger than that of synchronous rendezvous. However, while the upper bound $O(D\ell)$ for the cost of synchronous rendezvous is tight, the tightness of the lower bound $\Omega(D^2+D\ell)$ for the cost of asynchronous rendezvous remains fully open. Indeed, the best known algorithm solving asynchronous rendezvous in PBM with cost polynomial in D and ℓ is our algorithm of [Bouchard *et al.* 2019] presented in Chapter 2, and the polynomial describing the cost of this algorithm is very large. True, this algorithm works in a case where the agents do not know D initially, and adding the knowledge of D would likely decrease its cost significantly. But bringing the exponents even to single digits seems to be a challenging problem.

It is important to emphasize that the knowledge by the agents of the initial distance D separating them plays a crucial role in our arguments used in [Dieudonné & Pelc 2014b]. Without this knowledge the cost of even synchronous rendezvous in PBM is at least D^2 and the gap between it and our lower bound for the asynchronous rendezvous disappears. It is still open if any gap in cost between synchrony and asynchrony remains in this case, as no result concerning such a gap has been shown for the task of rendezvous before or after this work.

It should be also underlined that the result from [Bampas *et al.* 2010] (already discussed in the introduction of Chapter 2) permits to show that if knowledge of D is replaced by each agent knowing its own position in a global system of coordinates, then the gap between synchrony and asynchrony must be smaller than in our case. Indeed, under this scenario (often referred to as rendezvous with GPS), an asynchronous rendezvous algorithm with cost $O(D^2 polylog(D))$ is given by the authors of [Bampas *et al.* 2010]. (In this paper, the initial positions in a global system of coordinates also play the role of distinct labels of agents.) On the other hand, without knowledge of D, the cost of even synchronous rendezvous in the plane is $\Omega(D^2)$, even with GPS. So, under this scenario, we do not know if a gap in cost between synchrony and asynchrony exists, but if it does, it is at most polylogarithmic, while under our scenario without GPS but with the knowledge of D it is polynomial for instances in which $\ell = O(D^\alpha)$, for $\alpha < 1$.

Apart from these snippets of observations and results, the mystery surrounding the real impact of asynchrony on the cost of rendezvous (but also of other tasks executed by mobile agents) remains complete, and the same is true for the other constraints we discussed earlier. Thus, what would be involved here, is examining the ratio between the optimal cost of an algorithm under a given constraint and that of an optimal algorithm without this constraint. At first, it might be too ambitious to show exact ratios, but defining non-trivial bounds on them would already represent a significant step forward.

Another relevant line of research, which can also be included in this second axis, consists in determining the trade-offs that can be made between the efficiency of the gathering and the degree of intensity of the constraints. To illustrate our point, let us consider the scenario we discussed in Chapter 3 in which the agents do not have direct means of communication. As explained, we have shown that, despite the

difficulties inherent to this scenario, the gathering remains always possible. Nevertheless, it must be recognized that the assumed conditions are extreme. A more realistic scenario would certainly be one in which the agents could exchange information directly using dedicated devices, but in a relatively limited way. In this case, though, a question naturally comes to mind: it is the one concerning the link that may exist between the amount of information that agents are able to exchange and the cost that can be expected from a gathering algorithm. In other words, how can the complexity of gathering vary according to the maximum volume of data that is allowed to be shared? An ideal result would be the characterization of the whole spectrum and, in particular, the identification of possible threshold effects. However, it should be acknowledged that establishing such a result is a daunting task, as it requires bringing to light some of the parametric functions underlying the complexity, which are, in essence, very difficult to handle. Be that as it may, such a result would be fundamental, since one could then not only rule out inefficient or inappropriate algorithms, but also precisely adjust, according to the desired final complexity, the communication capacities of the agents, and by extension, minimize the manufacturing costs.

We are certainly not limited to the context of communication, and if we let our imagination run wild, we are quickly led to similar questions in lots of distinct contexts, including for example when agents are limited in terms of energy or susceptible to crashes. How does the resolution time vary depending on the amount of energy that can be spent by the agents? And how does it vary depending on the number of failures? These are also questions that deserve to be thoroughly studied.

The third and last axis of our research project aims at identifying possible relationships among the models used for mobile agents. In fact, some studies have recently investigated this aspect [Flocchini *et al.* 2019, Buchin *et al.* 2021, Buchin *et al.* 2022], but limited themselves to models in which the agents have unrestricted vision and little or no memory of their past actions. Conversely, our research has been made assuming the agents have little or no vision but unrestricted memory. These couples "strong memory/weak vision" and "weak memory/strong vision" appear to be at the root of an important dividing line among the studies in the field and thus, it would be particularly interesting to widen the scope and find some comparisons between models from both sides. Is there some kind of equivalence between these couples? To what extent could they be interchanged without affecting the expressive power of the models? What about the relationships with models that would lie between the two extremes, *i.e.,* where each agent has some reasonable memory of past events and can have a partial perception of the network and/or the positions of the team? And, more fundamentally, between the two capabilities, memory and vision, is one more important than the other? These are examples of interesting avenues worth exploring.

We can go beyond just highlighting mere comparisons, by giving even possible reductions between models. This would be especially useful, from models with difficult constraints to models with more lenient constraints (*e.g.,* from asynchronous to synchronous ones), even if the reductions are only valid for some classes of problems.

In the field of classical distributed systems (in which the entities composing the networks are fixed), a common way to compare the expressive power of two models M and M' is to try to design a meta-algorithm, called a *transformer*, which takes as input an algorithm A written for model M and produces an algorithm $T(A)$ that achieves the same specification as A but in M'. If such a transformer exists, then M is at least as powerful as M': any problem that can be solved in M can also be solved in M', and any impossibility result in M' is also an impossibility result in M. When transformers exist in both directions and without any additional assumptions, the two models M and M' are considered equivalent in terms of expressive power, which means that we can use either one without any loss of generality regarding feasibility issues. What is worth noting is that even if models M and M' have the same expressive power, one may be easier to manipulate than the other for designing algorithms, establishing proofs of validity, constructing counterexamples, etc. Depending on the use case, one will therefore choose the model that appears to be the most appropriate: this is where the real value of knowing clear reductions between models lies. In particular, reductions from models with harsh constraints to models with softer constraints, through transformers, would be considerable assets that would greatly facilitate future research in the field of mobile agents, in which they are currently lacking. Of course, the use of transformers can result in additional costs on the final obtained complexities, which cannot be ignored. Nonetheless, this does not alter the crucialness of this research path, which should be regarded as orthogonal to our first two axes focusing on complexity issues. This is our strong conviction, and we will work towards this goal in the coming years.

Bibliography

[Agmon & Peleg 2006] Noa Agmon and David Peleg. *Fault-Tolerant Gathering Algorithms for Autonomous Mobile Robots.* SIAM J. Comput., vol. 36, no. 1, pages 56–82, 2006. (Cited on page 32.)

[Aichholzer *et al.* 2001] Oswin Aichholzer, Franz Aurenhammer, Christian Icking, Rolf Klein, Elmar Langetepe and Günter Rote. *Generalized self-approaching curves.* Discret. Appl. Math., vol. 109, no. 1-2, pages 3–24, 2001. (Cited on page 70.)

[Aldous 1991] David J. Aldous. *Meeting times for independent Markov chains.* Stochastic Processes and their Applications, vol. 38, no. 2, pages 185–193, 1991. (Cited on page 25.)

[Aleliunas *et al.* 1979] Romas Aleliunas, Richard M. Karp, Richard J. Lipton, László Lovász and Charles Rackoff. *Random Walks, Universal Traversal Sequences, and the Complexity of Maze Problems.* In 20th Annual Symposium on Foundations of Computer Science, San Juan, Puerto Rico, 29-31 October 1979, pages 218–223. IEEE Computer Society, 1979. (Cited on page 68.)

[Alpern & Gal 2003] Steve Alpern and Shmuel Gal. The theory of search games and rendezvous, volume 55 of *International series in operations research and management science.* Kluwer, 2003. (Cited on pages 14, 65 and 66.)

[Alpern 2002] Steve Alpern. *Rendezvous Search: A Personal Perspective.* Oper. Res., vol. 50, no. 5, pages 772–795, 2002. (Cited on page 14.)

[Awerbuch & Kobourov 1998] Baruch Awerbuch and Stephen G. Kobourov. *Polylogarithmic-Overhead Piecemeal Graph Exploration.* In Peter L. Bartlett and Yishay Mansour, editors, Proceedings of the Eleventh Annual Conference on Computational Learning Theory, COLT 1998, Madison, Wisconsin, USA, July 24-26, 1998, pages 280–286. ACM, 1998. (Cited on page 67.)

[Awerbuch *et al.* 1999] Baruch Awerbuch, Margrit Betke, Ronald L. Rivest and Mona Singh. *Piecemeal Graph Exploration by a Mobile Robot.* Inf. Comput., vol. 152, no. 2, pages 155–172, 1999. (Cited on pages 10, 67 and 84.)

[Baeza-Yates *et al.* 1993] Ricardo A. Baeza-Yates, Joseph C. Culberson and Gregory J. E. Rawlins. *Searching in the Plane.* Inf. Comput., vol. 106, no. 2, pages 234–252, 1993. (Cited on page 66.)

[Bampas *et al.* 2010] Evangelos Bampas, Jurek Czyzowicz, Leszek Gasieniec, David Ilcinkas and Arnaud Labourel. *Almost Optimal Asynchronous Rendezvous in Infinite Multidimensional Grids.* In Distributed Computing, 24th International Symposium, DISC 2010, Cambridge, MA, USA, September 13-15, 2010. Proceedings, pages 297–311, 2010. (Cited on pages 15 and 85.)

[Barrière *et al.* 2007] Lali Barrière, Paola Flocchini, Pierre Fraigniaud and Nicola Santoro. *Rendezvous and Election of Mobile Agents: Impact of Sense of Direction.* Theory Comput. Syst., vol. 40, no. 2, pages 143–162, 2007. (Cited on page 32.)

[Beck & Newman 1970] Anatole Beck and D. J. Newman. *Yet more on the linear search problem.* Israel Journal of Mathematics, vol. 8, no. 4, pages 419–429, 1970. (Cited on page 65.)

[Bellman 1963] Richard Bellman. *An Optimal Search.* SIAM Review, vol. 5, no. 3, pages 274–274, 1963. (Cited on page 65.)

[Betke *et al.* 1995] Margrit Betke, Ronald L. Rivest and Mona Singh. *Piecemeal Learning of an Unknown Environment.* Mach. Learn., vol. 18, no. 2-3, pages 231–254, 1995. (Cited on page 67.)

[Bouchard *et al.* 2015] Sébastien Bouchard, Yoann Dieudonné and Bertrand Ducourthial. *Byzantine Gathering in Networks.* In Christian Scheideler, editor, Structural Information and Communication Complexity - 22nd International Colloquium, SIROCCO 2015, Montserrat, Spain, July 14-16, 2015, Post-Proceedings, volume 9439 of *Lecture Notes in Computer Science*, pages 179–193. Springer, 2015. (Cited on page 8.)

[Bouchard *et al.* 2016] Sébastien Bouchard, Yoann Dieudonné and Bertrand Ducourthial. *Byzantine gathering in networks.* Distributed Comput., vol. 29, no. 6, pages 435–457, 2016. (Cited on pages 8, 9, 33, 39, 41 and 46.)

[Bouchard *et al.* 2017] Sébastien Bouchard, Marjorie Bournat, Yoann Dieudonné, Swan Dubois and Franck Petit. *Asynchronous Approach in the Plane: A Deterministic Polynomial Algorithm.* In Andréa W. Richa, editor, 31st International Symposium on Distributed Computing, DISC 2017, October 16-20, 2017, Vienna, Austria, volume 91 of *LIPIcs*, pages 8:1–8:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. (Cited on page 7.)

[Bouchard *et al.* 2018a] Sébastien Bouchard, Yoann Dieudonné and Anissa Lamani. *Byzantine Gathering in Polynomial Time.* In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx and Donald Sannella, editors, 45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic, volume 107 of *LIPIcs*, pages 147:1–147:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. (Cited on page 8.)

[Bouchard *et al.* 2018b] Sébastien Bouchard, Yoann Dieudonné, Andrzej Pelc and Franck Petit. *Deterministic Treasure Hunt in the Plane with Angular Hints.* In Wen-Lian Hsu, Der-Tsai Lee and Chung-Shou Liao, editors, 29th International Symposium on Algorithms and Computation, ISAAC 2018, December

16-19, 2018, Jiaoxi, Yilan, Taiwan, volume 123 of *LIPIcs*, pages 48:1–48:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. (Cited on page 10.)

[Bouchard *et al.* 2018c] Sébastien Bouchard, Yoann Dieudonné, Andrzej Pelc and Franck Petit. *On deterministic rendezvous at a node of agents with arbitrary velocities*. Inf. Process. Lett., vol. 133, pages 39–43, 2018. (Cited on pages 7, 16, 23 and 29.)

[Bouchard *et al.* 2019] Sébastien Bouchard, Marjorie Bournat, Yoann Dieudonné, Swan Dubois and Franck Petit. *Asynchronous approach in the plane: a deterministic polynomial algorithm*. Distributed Comput., vol. 32, no. 4, pages 317–337, 2019. (Cited on pages 7, 16, 17, 23, 83 and 85.)

[Bouchard *et al.* 2020a] Sébastien Bouchard, Yoann Dieudonné and Andrzej Pelc. *Want to Gather? No Need to Chatter!* In Yuval Emek and Christian Cachin, editors, PODC '20: ACM Symposium on Principles of Distributed Computing, Virtual Event, Italy, August 3-7, 2020, pages 253–262. ACM, 2020. (Cited on page 8.)

[Bouchard *et al.* 2020b] Sébastien Bouchard, Yoann Dieudonné, Andrzej Pelc and Franck Petit. *Deterministic Treasure Hunt in the Plane with Angular Hints*. Algorithmica, vol. 82, no. 11, pages 3250–3281, 2020. (Cited on pages 10, 68 and 76.)

[Bouchard *et al.* 2021] Sébastien Bouchard, Yoann Dieudonné, Arnaud Labourel and Andrzej Pelc. *Almost-Optimal Deterministic Treasure Hunt in Arbitrary Graphs*. In Nikhil Bansal, Emanuela Merelli and James Worrell, editors, 48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference), volume 198 of *LIPIcs*, pages 36:1–36:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. (Cited on page 10.)

[Bouchard *et al.* 2022] Sébastien Bouchard, Yoann Dieudonné and Anissa Lamani. *Byzantine gathering in polynomial time*. Distributed Comput., vol. 35, no. 3, pages 235–263, 2022. (Cited on pages 8, 9, 33, 39, 41, 42 and 51.)

[Bouchard *et al.* 2023a] Sébastien Bouchard, Yoann Dieudonné, Arnaud Labourel and Andrzej Pelc. *Almost-Optimal Deterministic Treasure Hunt in Unweighted Graphs*. ACM Trans. Algorithms, 2023. to appear. (Cited on pages 10, 11, 68, 76 and 84.)

[Bouchard *et al.* 2023b] Sébastien Bouchard, Yoann Dieudonné and Andrzej Pelc. *Want to Gather? No Need to Chatter!* SIAM Journal on Computing, vol. 52, no. 2, pages 358–411, 2023. (Cited on pages 8, 9, 33, 56, 57 and 83.)

[Bouzid *et al.* 2013] Zohir Bouzid, Shantanu Das and Sébastien Tixeuil. *Gathering of Mobile Robots Tolerating Multiple Crash Faults*. In 2013 IEEE 33rd In-

ternational Conference on Distributed Computing Systems, pages 337–346, 2013. (Cited on page 32.)

[Bramas *et al.* 2023] Quentin Bramas, Anissa Lamani and Sébastien Tixeuil. *Stand up indulgent gathering.* Theor. Comput. Sci., vol. 939, pages 63–77, 2023. (Cited on page 32.)

[Buchin *et al.* 2021] Kevin Buchin, Paola Flocchini, Irina Kostitsyna, Tom Peters, Nicola Santoro and Koichi Wada. *Autonomous Mobile Robots: Refining the Computational Landscape.* In IEEE International Parallel and Distributed Processing Symposium Workshops, IPDPS Workshops 2021, Portland, OR, USA, June 17-21, 2021, pages 576–585. IEEE, 2021. (Cited on page 86.)

[Buchin *et al.* 2022] Kevin Buchin, Paola Flocchini, Irina Kostitsyna, Tom Peters, Nicola Santoro and Koichi Wada. *On the Computational Power of Energy-Constrained Mobile Robots: Algorithms and Cross-Model Analysis.* In Merav Parter, editor, Structural Information and Communication Complexity - 29th International Colloquium, SIROCCO 2022, Paderborn, Germany, June 27-29, 2022, Proceedings, volume 13298 of *Lecture Notes in Computer Science*, pages 42–61. Springer, 2022. (Cited on page 86.)

[Chalopin *et al.* 2007] Jérémie Chalopin, Shantanu Das and Nicola Santoro. *Rendezvous of Mobile Agents in Unknown Graphs with Faulty Links.* In Andrzej Pelc, editor, Distributed Computing, 21st International Symposium, DISC 2007, Lemesos, Cyprus, September 24-26, 2007, Proceedings, volume 4731 of *Lecture Notes in Computer Science*, pages 108–122. Springer, 2007. (Cited on page 32.)

[Chalopin *et al.* 2010] Jérémie Chalopin, Shantanu Das and Adrian Kosowski. *Constructing a Map of an Anonymous Graph: Applications of Universal Sequences.* In Chenyang Lu, Toshimitsu Masuzawa and Mohamed Mosbah, editors, Principles of Distributed Systems - 14th International Conference, OPODIS 2010, Tozeur, Tunisia, December 14-17, 2010. Proceedings, volume 6490 of *Lecture Notes in Computer Science*, pages 119–134. Springer, 2010. (Cited on page 39.)

[Chalopin *et al.* 2014] Jérémie Chalopin, Yoann Dieudonné, Arnaud Labourel and Andrzej Pelc. *Fault-Tolerant Rendezvous in Networks.* In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt and Elias Koutsoupias, editors, Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part II, volume 8573 of *Lecture Notes in Computer Science*, pages 411–422. Springer, 2014. (Cited on page 7.)

[Chalopin *et al.* 2016] Jérémie Chalopin, Yoann Dieudonné, Arnaud Labourel and Andrzej Pelc. *Rendezvous in networks in spite of delay faults.* Distributed

Comput., vol. 29, no. 3, pages 187–205, 2016. (Cited on pages 7, 16, 24, 26, 28 and 29.)

[Chrobak *et al.* 2015] Marek Chrobak, Leszek Gasieniec, Thomas Gorry and Russell Martin. *Group Search on the Line.* In Giuseppe F. Italiano, Tiziana Margaria-Steffen, Jaroslav Pokorný, Jean-Jacques Quisquater and Roger Wattenhofer, editors, SOFSEM 2015: Theory and Practice of Computer Science - 41st International Conference on Current Trends in Theory and Practice of Computer Science, Pec pod Sněžkou, Czech Republic, January 24-29, 2015. Proceedings, volume 8939 of *Lecture Notes in Computer Science*, pages 164–176. Springer, 2015. (Cited on page 66.)

[Cicerone *et al.* 2019] Serafino Cicerone, Gabriele Di Stefano and Alfredo Navarra. *Asynchronous Robots on Graphs: Gathering.* In Paola Flocchini, Giuseppe Prencipe and Nicola Santoro, editors, Distributed Computing by Mobile Entities, Current Research in Moving and Computing, volume 11340 of *Lecture Notes in Computer Science*, pages 184–217. Springer, 2019. (Cited on page 15.)

[Cieliebak *et al.* 2012] Mark Cieliebak, Paola Flocchini, Giuseppe Prencipe and Nicola Santoro. *Distributed Computing by Mobile Robots: Gathering.* SIAM J. Comput., vol. 41, no. 4, pages 829–879, 2012. (Cited on pages 15 and 32.)

[Cohen & Peleg 2005] Reuven Cohen and David Peleg. *Convergence Properties of the Gravitational Algorithm in Asynchronous Robot Systems.* SIAM J. Comput., vol. 34, no. 6, pages 1516–1528, 2005. (Cited on page 15.)

[Cohen & Peleg 2008] Reuven Cohen and David Peleg. *Convergence of Autonomous Mobile Robots with Inaccurate Sensors and Movements.* SIAM J. Comput., vol. 38, no. 1, pages 276–302, 2008. (Cited on page 15.)

[Collins *et al.* 2010] Andrew Collins, Jurek Czyzowicz, Leszek Gasieniec and Arnaud Labourel. *Tell Me Where I Am So I Can Meet You Sooner.* In Automata, Languages and Programming, 37th International Colloquium, ICALP 2010, Bordeaux, France, July 6-10, 2010, Proceedings, Part II, pages 502–514, 2010. (Cited on pages 15 and 21.)

[Czyzowicz *et al.* 2012a] Jurek Czyzowicz, Adrian Kosowski and Andrzej Pelc. *How to meet when you forget: log-space rendezvous in arbitrary graphs.* Distributed Comput., vol. 25, no. 2, pages 165–178, 2012. (Cited on pages 14, 34 and 36.)

[Czyzowicz *et al.* 2012b] Jurek Czyzowicz, Andrzej Pelc and Arnaud Labourel. *How to meet asynchronously (almost) everywhere.* ACM Trans. Algorithms, vol. 8, no. 4, pages 37:1–37:14, 2012. (Cited on pages 15, 17, 21 and 47.)

[Czyzowicz *et al.* 2019a] Jurek Czyzowicz, Maxime Godon, Evangelos Kranakis and Arnaud Labourel. *Group search of the plane with faulty robots.* Theor. Comput. Sci., vol. 792, pages 69–84, 2019. (Cited on page 66.)

[Czyzowicz *et al.* 2019b] Jurek Czyzowicz, Evangelos Kranakis, Danny Krizanc, Lata Narayanan and Jaroslav Opatrny. *Search on a line with faulty robots.* Distributed Comput., vol. 32, no. 6, pages 493–504, 2019. (Cited on page 66.)

[Czyzowicz *et al.* 2021] Jurek Czyzowicz, Konstantinos Georgiou, Evangelos Kranakis, Danny Krizanc, Lata Narayanan, Jaroslav Opatrny and Sunil M. Shende. *Search on a Line by Byzantine Robots.* Int. J. Found. Comput. Sci., vol. 32, no. 4, pages 369–387, 2021. (Cited on page 66.)

[D'Angelo *et al.* 2012] Gianlorenzo D'Angelo, Gabriele Di Stefano and Alfredo Navarra. *How to Gather Asynchronous Oblivious Robots on Anonymous Rings.* In Marcos K. Aguilera, editor, Distributed Computing - 26th International Symposium, DISC 2012, Salvador, Brazil, October 16-18, 2012. Proceedings, volume 7611 of *Lecture Notes in Computer Science*, pages 326–340. Springer, 2012. (Cited on page 32.)

[Das *et al.* 2014] Shantanu Das, Dariusz Dereniowski, Adrian Kosowski and Przemyslaw Uznanski. *Rendezvous of Distance-Aware Mobile Agents in Unknown Graphs.* In Magnús M. Halldórsson, editor, Structural Information and Communication Complexity - 21st International Colloquium, SIROCCO 2014, Takayama, Japan, July 23-25, 2014. Proceedings, volume 8576 of *Lecture Notes in Computer Science*, pages 295–310. Springer, 2014. (Cited on page 14.)

[Das *et al.* 2019] Shantanu Das, Riccardo Focardi, Flaminia L. Luccio, Euripides Markou and Marco Squarcina. *Gathering of robots in a ring with mobile faults.* Theor. Comput. Sci., vol. 764, pages 42–60, 2019. (Cited on page 32.)

[Défago *et al.* 2006] Xavier Défago, Maria Gradinariu, Stéphane Messika and Philippe Raipin Parvédy. *Fault-Tolerant and Self-stabilizing Mobile Robots Gathering.* In Distributed Computing, 20th International Symposium, DISC 2006, Stockholm, Sweden, September 18-20, 2006, Proceedings, pages 46–60, 2006. (Cited on page 32.)

[Défago *et al.* 2019] Xavier Défago, Maria Potop-Butucaru and Sébastien Tixeuil. *Fault-Tolerant Mobile Robots.* In Paola Flocchini, Giuseppe Prencipe and Nicola Santoro, editors, Distributed Computing by Mobile Entities, Current Research in Moving and Computing, volume 11340 of *Lecture Notes in Computer Science*, pages 234–251. Springer, 2019. (Cited on page 32.)

[Défago *et al.* 2020] Xavier Défago, Maria Potop-Butucaru and Philippe Raipin Parvédy. *Self-stabilizing gathering of mobile robots under crash or Byzantine faults.* Distributed Comput., vol. 33, no. 5, pages 393–421, 2020. (Cited on page 32.)

[Dessmark *et al.* 2006] Anders Dessmark, Pierre Fraigniaud, Dariusz R. Kowalski and Andrzej Pelc. *Deterministic Rendezvous in Graphs.* Algorithmica, vol. 46, no. 1, pages 69–96, 2006. (Cited on pages 4, 14 and 19.)

[Dieudonné & Pelc 2012a] Yoann Dieudonné and Andrzej Pelc. *Deterministic network exploration by a single agent with Byzantine tokens.* Inf. Process. Lett., vol. 112, no. 12, pages 467–470, 2012. (Cited on page 6.)

[Dieudonné & Pelc 2012b] Yoann Dieudonné and Andrzej Pelc. *Deterministic Network Exploration by Anonymous Silent Agents with Local Traffic Reports.* In Artur Czumaj, Kurt Mehlhorn, Andrew M. Pitts and Roger Wattenhofer, editors, Automata, Languages, and Programming - 39th International Colloquium, ICALP 2012, Warwick, UK, July 9-13, 2012, Proceedings, Part II, volume 7392 of *Lecture Notes in Computer Science*, pages 500–512. Springer, 2012. (Cited on page 6.)

[Dieudonné & Pelc 2013a] Yoann Dieudonné and Andrzej Pelc. *Anonymous Meeting in Networks.* In Sanjeev Khanna, editor, Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013, pages 737–747. SIAM, 2013. (Cited on page 8.)

[Dieudonné & Pelc 2013b] Yoann Dieudonné and Andrzej Pelc. *Deterministic Polynomial Approach in the Plane.* In Fedor V. Fomin, Rusins Freivalds, Marta Z. Kwiatkowska and David Peleg, editors, Automata, Languages, and Programming - 40th International Colloquium, ICALP 2013, Riga, Latvia, July 8-12, 2013, Proceedings, Part II, volume 7966 of *Lecture Notes in Computer Science*, pages 533–544. Springer, 2013. (Cited on page 7.)

[Dieudonné & Pelc 2014a] Yoann Dieudonné and Andrzej Pelc. *Deterministic Network Exploration by Anonymous Silent Agents with Local Traffic Reports.* ACM Trans. Algorithms, vol. 11, no. 2, pages 10:1–10:29, 2014. (Cited on page 6.)

[Dieudonné & Pelc 2014b] Yoann Dieudonné and Andrzej Pelc. *Price of asynchrony in mobile agents computing.* Theor. Comput. Sci., vol. 524, pages 59–67, 2014. (Cited on pages 5, 84 and 85.)

[Dieudonné & Pelc 2015] Yoann Dieudonné and Andrzej Pelc. *Deterministic polynomial approach in the plane.* Distributed Comput., vol. 28, no. 2, pages 111–129, 2015. (Cited on pages 7, 16 and 17.)

[Dieudonné & Pelc 2016] Yoann Dieudonné and Andrzej Pelc. *Anonymous Meeting in Networks.* Algorithmica, vol. 74, no. 2, pages 908–946, 2016. (Cited on pages 8, 33 and 35.)

[Dieudonné & Pelc 2017] Yoann Dieudonné and Andrzej Pelc. *Impact of Knowledge on Election Time in Anonymous Networks.* In Christian Scheideler

and Mohammad Taghi Hajiaghayi, editors, Proceedings of the 29th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA 2017, Washington DC, USA, July 24-26, 2017, pages 207–215. ACM, 2017. (Cited on page 6.)

[Dieudonné & Pelc 2019] Yoann Dieudonné and Andrzej Pelc. *Impact of Knowledge on Election Time in Anonymous Networks.* Algorithmica, vol. 81, no. 1, pages 238–288, 2019. (Cited on page 6.)

[Dieudonné & Petit 2007] Yoann Dieudonné and Franck Petit. *Circle formation of weak robots and Lyndon words.* Inf. Process. Lett., vol. 101, no. 4, pages 156–162, 2007. (Cited on page 6.)

[Dieudonné & Petit 2008] Yoann Dieudonné and Franck Petit. *Squaring the Circle with Weak Mobile Robots.* In Seok-Hee Hong, Hiroshi Nagamochi and Takuro Fukunaga, editors, Algorithms and Computation, 19th International Symposium, ISAAC 2008, Gold Coast, Australia, December 15-17, 2008. Proceedings, volume 5369 of *Lecture Notes in Computer Science*, pages 354–365. Springer, 2008. (Cited on page 6.)

[Dieudonné & Petit 2009] Yoann Dieudonné and Franck Petit. *Scatter of Robots.* Parallel Process. Lett., vol. 19, no. 1, pages 175–184, 2009. (Cited on page 6.)

[Dieudonné & Petit 2012] Yoann Dieudonné and Franck Petit. *Self-stabilizing gathering with strong multiplicity detection.* Theor. Comput. Sci., vol. 428, pages 47–57, 2012. (Cited on page 32.)

[Dieudonné *et al.* 2008a] Yoann Dieudonné, Ouiddad Labbani-Igbida and Franck Petit. *Circle formation of weak mobile robots.* ACM Trans. Auton. Adapt. Syst., vol. 3, no. 4, pages 16:1–16:20, 2008. (Cited on page 6.)

[Dieudonné *et al.* 2008b] Yoann Dieudonné, Ouiddad Labbani-Igbida and Franck Petit. *On the solvability of the localization problem in robot networks.* In 2008 IEEE International Conference on Robotics and Automation, ICRA 2008, May 19-23, 2008, Pasadena, California, USA, pages 480–485. IEEE, 2008. (Cited on page 6.)

[Dieudonné *et al.* 2009a] Yoann Dieudonné, Shlomi Dolev, Franck Petit and Michael Segal. *Brief announcement: deaf, dumb, and chatting robots.* In Srikanta Tirthapura and Lorenzo Alvisi, editors, Proceedings of the 28th Annual ACM Symposium on Principles of Distributed Computing, PODC 2009, Calgary, Alberta, Canada, August 10-12, 2009, pages 308–309. ACM, 2009. (Cited on page 6.)

[Dieudonné *et al.* 2009b] Yoann Dieudonné, Shlomi Dolev, Franck Petit and Michael Segal. *Deaf, Dumb, and Chatting Asynchronous Robots.* In Tarek F. Abdelzaher, Michel Raynal and Nicola Santoro, editors, Principles of Distributed Systems, 13th International Conference, OPODIS 2009, Nîmes,

France, December 15-18, 2009. Proceedings, volume 5923 of *Lecture Notes in Computer Science*, pages 71–85. Springer, 2009. (Cited on page 6.)

[Dieudonné *et al.* 2010a] Yoann Dieudonné, Ouiddad Labbani-Igbida and Franck Petit. *Deterministic Robot-Network Localization is Hard.* IEEE Trans. Robotics, vol. 26, no. 2, pages 331–339, 2010. (Cited on page 6.)

[Dieudonné *et al.* 2010b] Yoann Dieudonné, Franck Petit and Vincent Villain. *Brief announcement: leader election vs pattern formation.* In Andréa W. Richa and Rachid Guerraoui, editors, Proceedings of the 29th Annual ACM Symposium on Principles of Distributed Computing, PODC 2010, Zurich, Switzerland, July 25-28, 2010, pages 404–405. ACM, 2010. (Cited on page 6.)

[Dieudonné *et al.* 2010c] Yoann Dieudonné, Franck Petit and Vincent Villain. *Leader Election Problem versus Pattern Formation Problem.* In Nancy A. Lynch and Alexander A. Shvartsman, editors, Distributed Computing, 24th International Symposium, DISC 2010, Cambridge, MA, USA, September 13-15, 2010. Proceedings, volume 6343 of *Lecture Notes in Computer Science*, pages 267–281. Springer, 2010. (Cited on page 6.)

[Dieudonné *et al.* 2012a] Yoann Dieudonné, Bertrand Ducourthial and Sidi-Mohammed Senouci. *COL: A data collection protocol for VANET.* In 2012 IEEE Intelligent Vehicles Symposium, IV 2012, Alcal de Henares, Madrid, Spain, June 3-7, 2012, pages 711–716. IEEE, 2012. (Cited on page 6.)

[Dieudonné *et al.* 2012b] Yoann Dieudonné, Andrzej Pelc and David Peleg. *Gathering despite mischief.* In Yuval Rabani, editor, Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012, pages 527–540. SIAM, 2012. (Cited on page 8.)

[Dieudonné *et al.* 2013a] Yoann Dieudonné, Florence Levé, Franck Petit and Vincent Villain. *Deterministic geoleader election in disoriented anonymous systems.* Theor. Comput. Sci., vol. 506, pages 43–54, 2013. (Cited on page 6.)

[Dieudonné *et al.* 2013b] Yoann Dieudonné, Andrzej Pelc and Vincent Villain. *How to meet asynchronously at polynomial cost.* In Panagiota Fatourou and Gadi Taubenfeld, editors, ACM Symposium on Principles of Distributed Computing, PODC '13, Montreal, QC, Canada, July 22-24, 2013, pages 92–99. ACM, 2013. (Cited on page 7.)

[Dieudonné *et al.* 2014] Yoann Dieudonné, Andrzej Pelc and David Peleg. *Gathering Despite Mischief.* ACM Trans. Algorithms, vol. 11, no. 1, pages 1:1–1:28, 2014. (Cited on pages 8, 9, 33, 39, 40, 41, 42 and 50.)

[Dieudonné *et al.* 2015] Yoann Dieudonné, Andrzej Pelc and Vincent Villain. *How to Meet Asynchronously at Polynomial Cost.* SIAM J. Comput., vol. 44, no. 3, pages 844–867, 2015. (Cited on pages 7, 16, 21 and 23.)

[Dieudonné *et al.* 2019] Yoann Dieudonné, Shlomi Dolev, Franck Petit and Michael Segal. *Explicit Communication Among Stigmergic Robots.* Int. J. Found. Comput. Sci., vol. 30, no. 2, pages 315–332, 2019. (Cited on page 6.)

[Dijkstra 1974] Edsger W. Dijkstra. *Self-stabilizing Systems in Spite of Distributed Control.* Commun. ACM, vol. 17, no. 11, pages 643–644, 1974. (Cited on page 15.)

[Duncan *et al.* 2006] Christian A. Duncan, Stephen G. Kobourov and V. S. Anil Kumar. *Optimal constrained graph exploration.* ACM Trans. Algorithms, vol. 2, no. 3, pages 380–402, 2006. (Cited on pages 67 and 80.)

[Flocchini *et al.* 2004] Paola Flocchini, Evangelos Kranakis, Danny Krizanc, Nicola Santoro and Cindy Sawchuk. *Multiple Mobile Agent Rendezvous in a Ring.* In Martin Farach-Colton, editor, LATIN 2004: Theoretical Informatics, 6th Latin American Symposium, Buenos Aires, Argentina, April 5-8, 2004, Proceedings, volume 2976 of *Lecture Notes in Computer Science*, pages 599–608. Springer, 2004. (Cited on page 32.)

[Flocchini *et al.* 2005] Paola Flocchini, Giuseppe Prencipe, Nicola Santoro and Peter Widmayer. *Gathering of asynchronous robots with limited visibility.* Theor. Comput. Sci., vol. 337, no. 1-3, pages 147–168, 2005. (Cited on page 15.)

[Flocchini *et al.* 2019] Paola Flocchini, Nicola Santoro and Koichi Wada. *On Memory, Communication, and Synchronous Schedulers When Moving and Computing.* In Pascal Felber, Roy Friedman, Seth Gilbert and Avery Miller, editors, 23rd International Conference on Principles of Distributed Systems, OPODIS 2019, December 17-19, 2019, Neuchâtel, Switzerland, volume 153 of *LIPIcs*, pages 25:1–25:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. (Cited on page 86.)

[Fraigniaud & Pelc 2008] Pierre Fraigniaud and Andrzej Pelc. *Deterministic Rendezvous in Trees with Little Memory.* In Gadi Taubenfeld, editor, Distributed Computing, 22nd International Symposium, DISC 2008, Arcachon, France, September 22-24, 2008. Proceedings, volume 5218 of *Lecture Notes in Computer Science*, pages 242–256. Springer, 2008. (Cited on page 14.)

[Fraigniaud & Pelc 2013] Pierre Fraigniaud and Andrzej Pelc. *Delays Induce an Exponential Memory Gap for Rendezvous in Trees.* ACM Trans. Algorithms, vol. 9, no. 2, pages 17:1–17:24, 2013. (Cited on page 14.)

[Ghosh & Klein 2010] Subir Kumar Ghosh and Rolf Klein. *Online algorithms for searching and exploration in the plane.* Computer Science Review, vol. 4, no. 4, pages 189–201, 2010. (Cited on pages 65 and 66.)

[Grünbaum 1960] Branko Grünbaum. *Partitions of mass-distributions and convex bodies by hyperplanes.* Pacific J. Math., vol. 10, no. 4, pages 1257–1261, 1960. (Cited on page 70.)

[Icking *et al.* 2004] Christian Icking, Rolf Klein, Elmar Langetepe, Sven Schuierer and Ines Semrau. *An Optimal Competitive Strategy for Walking in Streets.* SIAM J. Comput., vol. 33, no. 2, pages 462–486, 2004. (Cited on page 66.)

[Izumi *et al.* 2012] Taisuke Izumi, Samia Souissi, Yoshiaki Katayama, Nobuhiro Inuzuka, Xavier Défago, Koichi Wada and Masafumi Yamashita. *The Gathering Problem for Two Oblivious Robots with Unreliable Compasses.* SIAM J. Comput., vol. 41, no. 1, pages 26–46, 2012. (Cited on page 15.)

[Kao *et al.* 1996] Ming-Yang Kao, John H. Reif and Stephen R. Tate. *Searching in an Unknown Environment: An Optimal Randomized Algorithm for the Cow-Path Problem.* Inf. Comput., vol. 131, no. 1, pages 63–79, 1996. (Cited on page 65.)

[Kao *et al.* 1998] Ming-Yang Kao, Yuan Ma, Michael Sipser and Yiqun Lisa Yin. *Optimal Constructions of Hybrid Algorithms.* J. Algorithms, vol. 29, no. 1, pages 142–164, 1998. (Cited on page 66.)

[Klasing *et al.* 2008] Ralf Klasing, Euripides Markou and Andrzej Pelc. *Gathering asynchronous oblivious mobile robots in a ring.* Theor. Comput. Sci., vol. 390, no. 1, pages 27–39, 2008. (Cited on page 32.)

[Klasing *et al.* 2010] Ralf Klasing, Adrian Kosowski and Alfredo Navarra. *Taking advantage of symmetries: Gathering of many asynchronous oblivious robots on a ring.* Theor. Comput. Sci., vol. 411, no. 34-36, pages 3235–3246, 2010. (Cited on page 32.)

[Klein 1991] Rolf Klein. *Walking an Unknown Street with Bounded Detour.* Comput. Geom., vol. 1, pages 325–351, 1991. (Cited on page 66.)

[Kleinberg 1994] Jon M. Kleinberg. *On-line Search in a Simple Polygon.* In Daniel Dominic Sleator, editor, Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms. 23-25 January 1994, Arlington, Virginia, USA, pages 8–15. ACM/SIAM, 1994. (Cited on page 66.)

[Koucký 2002] Michal Koucký. *Universal traversal sequences with backtracking.* Journal of Computer and System Sciences, vol. 65, no. 4, pages 717–726, 2002. Special Issue on Complexity 2001. (Cited on page 68.)

[Kowalski & Malinowski 2008] Dariusz R. Kowalski and Adam Malinowski. *How to meet in anonymous network.* Theor. Comput. Sci., vol. 399, no. 1-2, pages 141–156, 2008. (Cited on page 14.)

[Kranakis & Spatharis 1997] Evangelos Kranakis and Anthony Spatharis. *Almost optimal on-line search in unknown streets.* In Proceedings of the 9th Canadian Conference on Computational Geometry, Kingston, Ontario, Canada, August 11-14, 1997, 1997. (Cited on page 66.)

[Kranakis *et al.* 2006] Evangelos Kranakis, Danny Krizanc and Sergio Rajsbaum. *Mobile Agent Rendezvous: A Survey.* In Paola Flocchini and Leszek Gasieniec, editors, Structural Information and Communication Complexity, 13th International Colloquium, SIROCCO 2006, Chester, UK, July 2-5, 2006, Proceedings, volume 4056 of *Lecture Notes in Computer Science*, pages 1–9. Springer, 2006. (Cited on page 14.)

[Lee *et al.* 1997] Jae-Ha Lee, Chan-Su Shin, Jae-Hoon Kim, Sung Yong Shin and Kyung-Yong Chwa. *New Competitive Strategies for Searching in Unknown Star-Shaped Polygons.* In Jean-Daniel Boissonnat, editor, Proceedings of the Thirteenth Annual Symposium on Computational Geometry, Nice, France, June 4-6, 1997, pages 427–429. ACM, 1997. (Cited on page 66.)

[López-Ortiz & Schuierer 2001] Alejandro López-Ortiz and Sven Schuierer. *The ultimate strategy to search on m rays?* Theor. Comput. Sci., vol. 261, no. 2, pages 267–295, 2001. (Cited on page 66.)

[Lu *et al.* 2016] Qi Lu, Joshua P. Hecker and Melanie E. Moses. *The MPFA: A multiple-place foraging algorithm for biologically-inspired robot swarms.* In 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2016, Daejeon, South Korea, October 9-14, 2016, pages 3815–3821. IEEE, 2016. (Cited on page 66.)

[Marco *et al.* 2006] Gianluca De Marco, Luisa Gargano, Evangelos Kranakis, Danny Krizanc, Andrzej Pelc and Ugo Vaccaro. *Asynchronous deterministic rendezvous in graphs.* Theor. Comput. Sci., vol. 355, no. 3, pages 315–326, 2006. (Cited on page 15.)

[Miller & Pelc 2015] Avery Miller and Andrzej Pelc. *Fast rendezvous with advice.* Theor. Comput. Sci., vol. 608, pages 190–198, 2015. (Cited on page 14.)

[Miller & Pelc 2016] Avery Miller and Andrzej Pelc. *Time versus cost tradeoffs for deterministic rendezvous in networks.* Distributed Comput., vol. 29, no. 1, pages 51–64, 2016. (Cited on page 14.)

[Miller & Saha 2020] Avery Miller and Ullash Saha. *Fast Byzantine Gathering with Visibility in Graphs.* In Cristina Maria Pinotti, Alfredo Navarra and Amitabha Bagchi, editors, Algorithms for Sensor Systems - 16th International Symposium on Algorithms and Experiments for Wireless Sensor Networks, ALGOSENSORS 2020, Pisa, Italy, September 9-10, 2020, Revised Selected Papers, volume 12503 of *Lecture Notes in Computer Science*, pages 140–153. Springer, 2020. (Cited on page 32.)

[Ooshita *et al.* 2017] Fukuhito Ooshita, Ajoy K. Datta and Toshimitsu Masuzawa. *Self-stabilizing Rendezvous of Synchronous Mobile Agents in Graphs.* In Paul G. Spirakis and Philippas Tsigas, editors, Stabilization, Safety, and Security of Distributed Systems - 19th International Symposium, SSS 2017, Boston, MA, USA, November 5-8, 2017, Proceedings, volume 10616 of *Lecture Notes in Computer Science*, pages 18–32. Springer, 2017. (Cited on page 15.)

[Pagli *et al.* 2015] Linda Pagli, Giuseppe Prencipe and Giovanni Viglietta. *Getting close without touching: near-gathering for autonomous mobile robots.* Distributed Computing, vol. 28, no. 5, pages 333–349, 2015. (Cited on page 15.)

[Panaite & Pelc 1999] Petrisor Panaite and Andrzej Pelc. *Exploring Unknown Undirected Graphs.* J. Algorithms, vol. 33, no. 2, pages 281–295, 1999. (Cited on page 67.)

[Park & Hutchinson 2017] Hyongju Park and Seth A. Hutchinson. *Fault-Tolerant Rendezvous of Multirobot Systems.* IEEE Transactions on Robotics, vol. 33, no. 3, pages 565–582, 2017. (Cited on page 32.)

[Pattanayak *et al.* 2019] Debasish Pattanayak, Kaushik Mondal, H. Ramesh and Partha Sarathi Mandal. *Gathering of mobile robots with weak multiplicity detection in presence of crash-faults.* J. Parallel Distributed Comput., vol. 123, pages 145–155, 2019. (Cited on page 32.)

[Pelc & Yadav 2019] Andrzej Pelc and Ram Narayan Yadav. *Cost vs. Information Tradeoffs for Treasure Hunt in the Plane.* CoRR, vol. abs/1902.06090, 2019. (Cited on page 66.)

[Pelc & Yadav 2021] Andrzej Pelc and Ram Narayan Yadav. *Advice complexity of treasure hunt in geometric terrains.* Information and Computation, vol. 281, page 104705, 2021. (Cited on page 66.)

[Pelc 2018] Andrzej Pelc. *Deterministic gathering with crash faults.* Networks, vol. 72, no. 2, pages 182–199, 2018. (Cited on page 32.)

[Pelc 2019] Andrzej Pelc. *Deterministic Rendezvous Algorithms.* In Paola Flocchini, Giuseppe Prencipe and Nicola Santoro, editors, Distributed Computing by Mobile Entities, Current Research in Moving and Computing, volume 11340 of *Lecture Notes in Computer Science*, pages 423–454. Springer, 2019. (Cited on page 14.)

[Reingold 2008] Omer Reingold. *Undirected connectivity in log-space.* J. ACM, vol. 55, no. 4, pages 17:1–17:24, 2008. (Cited on pages 22 and 68.)

[Schelling 1960] Thomas Schelling. The strategy of conflict. Oxford University Press, Oxford, 1960. (Cited on page 13.)

[Suzuki & Yamashita 1999] Ichiro Suzuki and Masafumi Yamashita. *Distributed Anonymous Mobile Robots: Formation of Geometric Patterns.* SIAM J. Comput., vol. 28, no. 4, pages 1347–1363, 1999. (Cited on pages 15 and 32.)

[Ta-Shma & Zwick 2014] Amnon Ta-Shma and Uri Zwick. *Deterministic Rendezvous, Treasure Hunts, and Strongly Universal Exploration Sequences.* ACM Trans. Algorithms, vol. 10, no. 3, pages 12:1–12:15, 2014. (Cited on pages 14 and 36.)

[Yamashita & Kameda 1996] Masafumi Yamashita and Tsunehiko Kameda. *Computing on Anonymous Networks: Part I-Characterizing the Solvable Cases.* IEEE Trans. Parallel Distributed Syst., vol. 7, no. 1, pages 69–89, 1996. (Cited on page 34.)

[Yu & Yung 1996] Xiangdong Yu and Moti Yung. *Agent Rendezvous: A Dynamic Symmetry-Breaking Problem.* In Friedhelm Meyer auf der Heide and Burkhard Monien, editors, Automata, Languages and Programming, 23rd International Colloquium, ICALP96, Paderborn, Germany, 8-12 July 1996, Proceedings, volume 1099 of *Lecture Notes in Computer Science*, pages 610–621. Springer, 1996. (Cited on page 32.)