Can Like Attract Like? A Study of Homonymous Gathering in Networks

Stéphane Devismes* Yoann Dieudonné† Arnaud Labourel‡

Abstract

A team of mobile agents, starting from distinct nodes of a network modeled as an undirected graph, have to meet at the same node and simultaneously declare that they all met. Agents execute the same algorithm, which they start when activated by an adversary or when an agent enter their initial node. While executing their algorithm, agents move from node to node by traversing edges of the network in synchronous rounds. Their perceptions and interactions are always strictly local: they have no visibility beyond their current node and can communicate only with agents occupying the same node. This task, known as *gathering*, is one of the most fundamental problems in distributed mobile systems. Over the past decades, numerous gathering algorithms have been designed, with a particular focus on minimizing their time complexity, i.e., the worst-case number of rounds between the start of the earliest agent and the completion of the task. To solve gathering deterministically, a common widespread assumption is that each agent initially has an integer ID, called *label*, only known to itself and that is distinct from those of all other agents. Labels play a crucial role in breaking possible symmetries, which, when left unresolved, may make gathering impossible. But must all labels be pairwise distinct to guarantee deterministic gathering?

In this paper, we conduct a deep investigation of this question by considering a context in which each agent applies a deterministic algorithm and has a label that may be shared with one or more other agents called *homonyms*. A team L of mobile agents, represented as the multiset of its labels, is said to be *gatherable* if, for every possible initial setting of L, there exists an algorithm, even dedicated to that setting, that solves gathering. Our contribution is threefold. First, we give a full characterization of the gatherable teams. Second, we design an algorithm that gathers all of them in $\text{poly}(n, \log \lambda)$ time, where n (resp. λ) is the order of the graph (resp. the smallest label in the team). This algorithm requires the agents to initially share only $O(\log \log \log \mu)$ bits of common knowledge, where μ is the multiplicity index of the team, i.e., the largest label multiplicity in L. Lastly, we show this dependency is almost optimal in the precise sense that no algorithm can gather every gatherable team in $\text{poly}(n, \log \lambda)$ time, with initially $o(\log \log \log \mu)$ bits of common knowledge.

As a by-product, we get the first deterministic $\text{poly}(n, \log \lambda)$-time algorithm that requires *no* common knowledge to gather *any* team in the classical case where all agent labels are pairwise distinct. While this was known to be achievable for teams of exactly two agents, extending it to teams of arbitrary size—under the same time and knowledge constraints—faced a major obstacle inherently absent in the two-agent scenario: that of termination detection. The synchronization techniques that enable us to overcome this obstacle may be of independent interest, as termination detection is a key issue in distributed systems.

Keywords: gathering, deterministic algorithm, mobile agent.

*MIS Lab., Université de Picardie Jules Verne, France. E-mail: stephane.devismes@u-picardie.fr

†MIS Lab., Université de Picardie Jules Verne, France. E-mail: yoann.dieudonne@u-picardie.fr

‡Aix Marseille Univ, CNRS, LIS, Marseille, France. Email: arnaud.labourel@lis-lab.fr

1 Introduction

1.1 Background

Gathering a group of mobile agents is a fundamental problem that has received significant attention in the field of distributed mobile systems. This interest largely stems from the fact that gathering often serves as a critical building block for more complex collaborative tasks. Hence, understanding the theoretical limits and algorithmic foundations of gathering has direct implications for a broad range of distributed problems.

Many studies assume that agents are equipped with unique identifiers—often referred to as labels in the literature—to help break symmetry and coordinate their actions. However, this assumption may not always be realistic or desirable. In practice, generating and maintaining globally unique labels/IDs can be costly, especially in large-scale or dynamically formed systems. Furthermore, in privacy-preserving scenarios, agents may deliberately blur their identity by revealing it only partially, thereby leading to the possible occurrence of homonyms. Although probabilistic approaches can, in principle, bypass the need for labels, they typically fail to provide absolute guarantees in terms of complexity and/or accuracy of the resulting solutions, which may be an unacceptable compromise in safety-critical or mission-critical applications.

This motivates a deep investigation of deterministic gathering in a more general context where some agents may share the same label, which we define as *homonymous gathering*. Naturally, the presence of homonyms introduces new challenges in symmetry breaking and coordination, making gathering significantly harder. Our goal is to precisely characterize when gathering remains feasible in such a context, and whether it can be done efficiently.

1.2 Model and Definitions

Given a team of $k \geq 2$ agents, an adversary selects a simple undirected connected graph $G = (V, E)$ with $n \geq k$ nodes and places the agents on k distinct nodes of G. As is standard in the literature on gathering (cf. the survey [31]), we make the following two assumptions regarding the labeling of the nodes and edges of G. The first assumption is that edges incident to a node v are locally ordered with a fixed port numbering (chosen by the adversary) ranging from 0 to $\texttt{deg}(v) - 1$ where $\texttt{deg}(v)$ is the degree of v (and thus, each edge has exactly two port numbers, one for each of its endpoints). The second 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. 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'.

Time is discretized into an infinite sequence of rounds $r = 1, 2, 3, \ldots$. At the beginning, every agent is said to be dormant and while an agent is dormant, it does nothing. The adversary wakes up some of the agents at possibly different rounds. A dormant agent, not woken up by the adversary, is woken up by the first agent that visits its starting node, if such an agent exists. Initially, each agent X is provided with three elements that are known to X: a deterministic algorithm $\mathcal{A}$, some common knowledge $\mathcal{K}$ (detailed thereafter), and a label ℓ_X corresponding to a positive integer. The algorithm (resp. the common knowledge) is the same for all agents. However, for any two agents X and X', labels ℓ_X and $\ell_{X'}$ may be identical or not. When $\ell_X = \ell_{X'}$, we say that agents X and X' are *homonyms*.

In the rest of this paper, the team of mobile agents will be designated by the multiset L of its agent's labels and $IS(L)$ will designate the initial setting that results from the choices of the adversary. Precisely, $IS(L)$ can be viewed as the underlying graph G in which each node v that is initially occupied by an agent X is colored by a couple (ℓ_X, t) where t is the round in which the adversary wants to wake up X or $\perp$ if such a round does not exist. As mentioned above, common knowledge $\mathcal{K}$ is a piece of information, called *advice*, that is initially given to all agents. Following the framework of algorithms with advice, see, e.g.,[18, 19, 20, 27, 29], this information will correspond to a single binary string (possibly empty) that is computed by an oracle $\mathcal{O}$ knowing $IS(L)$. More formally, $\mathcal{O}$ is defined as a function that associates a string from $\{0,1\}^*$ to every possible initial setting of any team L. The length of the binary string initially provided by the oracle will be called the size of $\mathcal{K}$.

An agent starts executing algorithm $\mathcal{A}$ in the round of its wake-up. When executing algorithm $\mathcal{A}$ at a node v in a round r, an agent X first performs some computations depending on $\mathcal{K}$ and its memory in round r that is denoted by $M_X(r)$ and that is formalized later (its memory includes its label ℓ_X). Then, after the computations, agent X chooses to do exactly one of the following things:

1. Declaring termination.
2. Waiting at its current node till the end of round r.
3. Moving to an adjacent node by a chosen port p.

In the first case, the agent definitively stops the execution of $\mathcal{A}$ at node v in round r. Otherwise, once the agent has processed the moving or waiting instruction, it is in round $r+1$ (at node v in the second case, or at node $\texttt{succ}(v,p)$ in the third case) and it continues the execution of algorithm $\mathcal{A}$. To ensure that the reference to the node occupied by an agent in any given round is unambiguous, we precisely assume that if the agent decides to take a port p when located at node v in round r, then it remains at node v till the end of round r and it is at node $\texttt{succ}(v,p)$ at the beginning of round $r+1$.

When entering a node, an agent learns the port of entry. When located at a node v, a non-dormant agent sees the degree of v and the other agents located at that node, and it can access all information they currently hold. (Agents' visions and interactions are thus always strictly local.) Also, if agents cross each other on an edge, traversing it simultaneously in different directions, they do not notice this fact.

An important notion used throughout this paper is the memory $M_X(r)$ of an agent X in a given round r. Intuitively, it corresponds to all information it has collected until round r, both by navigating in the graph and by exchanging information with other agents (we assume that the agents have no limitations on the amount of memory they can use). We formalize $M_X(r)$ as a triple. Precisely, if, in round r, agent X is dormant, or wakes up but is alone at its current node, then $M_X(r)$ is the triple $(\ell_X, \perp, \perp)$. Otherwise, we necessarily have $r > 1$ and $M_X(r)$ is the triple $(M_X(r-1), A_X(r), B_X(r))$, where the terms $A_X(r)$ and $B_X(r)$ are defined as follows. Suppose that agent X is at node v in round r and let $\{X_1, X_2, \ldots, X_h\}$ be the (possibly empty) set of the agents met by X at v in round r.

- If X enters node v in round r, then X was at some node u adjacent to v in round $r-1$ and $A_X(r) = (\texttt{deg}(v), \texttt{port}(u,v), \texttt{port}(v,u))$. Otherwise $A_X(r) = (\texttt{deg}(v), \perp, \perp)$.

- $B_X(r)=\emptyset$ if agent X is alone in round r. Otherwise, $B_X(r)=$ $\{(A_{X_1}(r), M_{X_1}(r-1)), (A_{X_2}(r), M_{X_2}(r-1)), \ldots, (A_{X_h}(r), M_{X_h}(r-1))\}$.

The rank of $M_X(r)$, denoted by $\texttt{rk}(M_X(r))$, is equal to 0 if $M_X(r) = (\ell_X, \perp, \perp)$, $1+\texttt{rk}(M_X(r-1))$ otherwise.

Under the above-described context, the agents are assigned the task of *gathering*: there must exist a round in which the agents are at the same node, simultaneously declare termination and stop.

A team L is said to be *gatherable* if for every initial setting $IS(L)$, there exists an algorithm $\mathcal{A}$, even specifically dedicated to $IS(L)$ (and thus regardless of $\mathcal{K}$) that allows the agents to solve gathering.[1] An algorithm $\mathcal{A}$ is said to be an *$\mathcal{O}$-universal gathering algorithm* if for every initial setting $IS(L)$ of any gatherable team L it solves the task of gathering with $\mathcal{K} = \mathcal{O}(IS(L))$. The *time complexity of $\mathcal{A}$ with $\mathcal{O}$* (i.e., when the agents are initially provided with advice from oracle $\mathcal{O}$) is the worst-case number of rounds since the wake-up of the earliest agent until all agents declare termination.

We now finish the presentation of this section by giving some additional conventions. The smallest label in L will be denoted by λ and, for any label ℓ, the length of its binary representation will be denoted by $|\ell|$. The number of occurrences of a given label in L will be referred to as the multiplicity of that label. Finally, the greatest common divisor of all the multiplicities of the labels in L will be called the *symmetry index* of L, while the largest multiplicity of any label in L will be called the *multiplicity index* of L.

1.3 Related Work

Historically, gathering was first investigated in the special case of exactly two agents, a variant known as *rendezvous*. This foundational variant was popularized by Schelling in *The Strategy of Conflict* [35], a seminal book in game theory and international relations, where he examined how two individuals might coordinate to meet at a common location without prior communication. This led to the introduction of focal points—salient solutions toward which individuals naturally gravitate in the absence of explicit coordination. Since then, rendezvous and its generalization—gathering—have been widely studied across very diverse fields such as economics, sociology, and, most relevant to this work, distributed mobile systems. In such systems, gathering has emerged as a fundamental coordination task, especially in environments where agents operate under strict limitations on communication and perception. This is due to the fact that, in such environments, the ability to regroup may become a necessary first step toward enabling more sophisticated forms of collaboration and collective decision-making that would otherwise be infeasible.

Even when focusing exclusively on distributed mobile systems, the gathering problem has given rise to a vast amount of research. One reason is that the problem offers many meaningful parameters to explore: the type of environment in which the agents evolve, whether algorithms are deterministic or randomized, whether the agents move in synchronous rounds or not, etc. Since our work is naturally aligned with research on deterministic gathering in graphs where agents operate in synchronous rounds, we will concentrate essentially on this setting in the remainder of the subsection. However, for the reader interested in gaining broader insight into the problem, we refer

[1] If $\mathcal{A}$ is specifically designed for $IS(L)$, then all necessary knowledge about $IS(L)$ can be "embedded directly" within the algorithm itself, making the use of $\mathcal{K}$ unnecessary in this case.

to [1, 10, 11, 23], which investigate asynchronous gathering in the plane or in graphs. Regarding randomized rendezvous, a good starting point is to go through [2, 3, 25].

The literature on deterministic gathering in graphs, as typically studied in the synchronous setting and thoroughly surveyed in [31], spans a wide range of variants shaped by the capabilities granted to the agents. Two particularly influential dimensions concern the ability to leave traces of their passage (e.g., by dropping markers or writing on local node memories) and the extent of their visibility, which may range from a global view of the network to a strictly local view limited to their current node. Perhaps the most compelling case arises when both dimensions are taken in their minimal forms—namely, no traces and node-local view only. Within this constrained and challenging context, a substantial body of work has focused on minimizing the time (i.e., the number of rounds) to achieve gathering. The core model adopted in most of these contributions is the one we formalize in Section 1.2, restricted to the special case where all agents have distinct labels (to break potential symmetries)—though sometimes with a variation in how agents are introduced into the network. Specifically, the literature considered two main scenarios in this regard: a *grounded scenario* and a *dropped-in scenario*. In the grounded scenario, all agents are initially present in the network, but start executing the algorithm when activated by an adversary or as soon as an agent enters their initial node (this potentially allows the algorithm to exert some control over the delays between agents' starting times). By contrast, in the dropped-in scenario, agents are not necessarily present at the outset: each of them appears in the network at a time and location entirely determined by an adversary and begins its execution in the round of its arrival. In this latter case, the delays between the starting times cannot be influenced by the algorithm executed by the agents.

Early works primarily focused on the dropped-in scenario, among which the key contributions are [13, 24, 36]. In [13], the authors provide a rendezvous algorithm (thus working for two agents only) whose time complexity is polynomial in the order n of the graph, in $|\lambda|$—where λ is the smaller of the two labels—and in the delay τ between the agents' starting times, even when time is counted from the start of the later agent. While a dependence on τ is naturally expected in the dropped-in scenario when measuring time from the start of the earlier agent (as the second agent's arrival is beyond the algorithm's control), it was unclear whether such a dependence remained unavoidable when counting time from the arrival of the later agent. This led the authors of [13] to ask whether a polynomial-time solution depending only on n and λ could be achieved in that case. In [24], a positive answer is brought to this question, with an algorithm ensuring rendezvous in time $\mathcal{O}(|\lambda|^3 + \ n^{15} \log^{12} n)$ from the start of the later agent. This was subsequently improved in [36], which reduced the complexity to $\tilde{\mathcal{O}}(n^5|\lambda|)$. On the negative side, [13] proves that any rendezvous algorithm necessarily has time complexity at least polynomial in n and $|\lambda|$ (specifically, at least $\Omega(n|\lambda|)$), even when $\tau = 0$.

To deal with more than two agents, the authors of [24] observe that it is sufficient for agents to execute a rendezvous algorithm with the following simple adaptation called the *sticking-together strategy*: whenever two or more agents meet, the agent with the smallest label continues its execution as if nothing had happened, while the others start following it, mimicking its actions from that point on. This way the agents will indeed eventually end up together. However, detecting all the agents are together in the dropped-in scenario requires the agents to know the exact number of agents that will be introduced by the adversary, regardless of the strategy being employed. This *de facto* rules out the possibility of designing gathering algorithms using no common knowledge, unless one entirely gives up on termination detection—a significant limitation in distributed

systems. This fundamental barrier disappears in the grounded scenario, which has emerged as the prevailing one for studying gathering, as reflected in the large body of work dedicated to it (see, e.g., [4, 5, 6, 15, 21, 22, 28, 30, 34]). Here, gathering *any* number of agents without relying on any common knowledge becomes possible (see, e.g., [6]). Moreover, since the delays between agents' starting times can now be algorithmically controlled to some extent, the reference time for rendezvous and gathering is taken from the start of the earliest agent. Interestingly, in this new reference time, the poly$(n, |\lambda|)$-time lower bound established by [13] in the dropped-in scenario remain valid in the grounded context, since it was proven even when $\tau = 0$. Yet, within this same context, and despite the numerous studies of the literature, no algorithm was known to guarantee gathering in polynomial time in n and $|\lambda|$, without assuming any common knowledge, except in the special case where teams are restricted to exactly two agents (cf. [7]).[2] In fact, the only existing gathering algorithms achieving such a complexity for teams of arbitrary size required the knowledge of a polynomial upper bound on the graph order (see [5, 6]).

Another intriguing aspect of the literature concerns the assumptions made about the agents' labeling. Specifically, gathering has only been studied at two opposite ends of the spectrum: either when all agents have pairwise distinct labels (as in the papers cited in the previous paragraph), or, much less frequently, when all are completely anonymous, equivalent to all agents sharing the same label (as in [14, 32]). Labels play a crucial role in breaking potential symmetries, which, if left unresolved, can often render gathering impossible, particularly in the anonymous case. However, requiring all labels to be distinct may be unnecessarily strong. Thus, the question of what can be achieved for gathering under such an intermediate assumption, where some degree of homonymy is allowed, remains entirely open.

To conclude, it is worth noting that the impact of homonymy has already been investigated, yet in the very different context of static message-passing networks, for classical problems such as leader election and sorting [37, 17, 9, 12].

1.4 Our Results

In this paper, we conduct a thorough study of the gathering problem in a standard model from the literature, which we extend to allow the presence of homonyms. Actually, in light of the discussions in the related work section, we are particularly motivated by the following open questions:

- Under what conditions is a team of agents—with possible homonyms—gatherable?
- Does there exist an algorithm allowing to gather any gatherable team in poly$(n, |\lambda|)$-time, without any initial common knowledge?
- And if not, what is the minimum amount of common knowledge required?

With these questions in mind, we make the following three contributions. First, we give a full characterization of the gatherable teams. Second, we design an algorithm that gathers all of them in poly$(n, |\lambda|)$ time. This algorithm requires the agents to initially share only $O(\log\log\log\mu)$ bits of common knowledge, where μ is the multiplicity index of the team. And finally, we show this

[2] More precisely, the authors of [7] analyze rendezvous under the assumption that agents may traverse edges at different speeds, and design an algorithm running in time polynomial in the order of the graph, the logarithm of the smaller label, and the maximum edge traversal time. In the particular case where every edge traversal time equals 1, this yields a rendezvous algorithm polynomial in the first two parameters.

dependency is almost optimal in the precise sense that no algorithm can gather every gatherable team in $\text{poly}(n, |\lambda|)$ time, with initially $o(\log\log\log\mu)$ bits of common knowledge.

As an important by-product, we also obtain the first $\text{poly}(n, |\lambda|)$-time algorithm that requires *no* common knowledge to gather *any* team in the classical scenario where all agent labels are pairwise distinct. While this result has been known to be achievable for the special case of exactly two agents, extending it to teams of arbitrary size has remained a fundamental open problem until now (cf. Section 1.3). The core difficulty of the extension lies in detecting the termination of the task. In a two-agent context, termination detection is trivial, as it inherently occurs at the time of the first meeting. However, when dealing with teams of arbitrary size, this is no longer the case: it becomes a fundamental challenge within the constraints of $\text{poly}(n, |\lambda|)$ time and no common knowledge (especially without any upper bound on the team size). Our ability to overcome this challenge is a direct consequence of the synchronization techniques we developed, in the general case with homonyms, to make the agents determine whether the gathering is done or not. These techniques are likely to be of independent interest, as termination detection is a key issue in distributed systems.

2 Preliminaries

In this section, we introduce some additional definitions and basic procedures that will be used throughout this paper.

Let us start by introducing the order $\prec$ on the set $\mathcal{M}$ of all possible memories. In particular, this order is made to have the heredity property stated in Lemma 2.1. Let f be any injective function from $\mathcal{M}$ to $\mathbb{N}$ (such a function necessarily exists since $\mathcal{M}$ is recursively enumerable). Given two memories $M_1 = (M_1', *, *)$ and $M_2 = (M_2', *, *)$, we have $M_1 \prec M_2$ if:

- $\texttt{rk}(M_1) < \texttt{rk}(M_2)$;
- Or $\texttt{rk}(M_1) = \texttt{rk}(M_2) = 0$ and $f(M_1) < f(M_2)$;
- Or $\texttt{rk}(M_1) = \texttt{rk}(M_2) > 0$, $M_1' = M_2'$ (resp. $M_1' \neq M_2'$) and $f(M_1) < f(M_2)$ (resp. $M_1' \prec M_2'$).

Note that $\prec$ induces a strict total order on $\mathcal{M}$.

When $M_1 \prec M_2$ (resp. $M_2 \prec M_1$), we will say that M_1 is *smaller* (resp. *larger*) than M_2. Below is a lemma related to memories.

Lemma 2.1 *Let A and B be two agents that are non dormant in a round r. If $M_A(r) \prec M_B(r)$, then $M_A(r') \prec M_B(r')$ in every round $r' \geq r$.*

Proof. Assume two agents A and B that are non dormant in a round r and $M_A(r) \prec M_B(r)$. We now show by induction on r' that $M_A(r') \prec M_B(r')$ in every round $r' \geq r$. The base case $r' = r$ is trivial.

Assume now that $r' > r$. Since A and B are non dormant in a round r and $r' > r$, we have both $\texttt{rk}(M_A(r')) > 0$ and $\texttt{rk}(M_B(r')) > 0$. So, we can let $M_A(r') = (M_A(r'-1), *, *)$ and $M_B(r') = (M_B(r'-1), *, *)$. Then, we have two cases:

- $\texttt{rk}(M_A(r')) \neq \texttt{rk}(M_B(r'))$.

So, $\texttt{rk}(M_A(r)) \neq \texttt{rk}(M_B(r))$ too. Now, in this case, $M_A(r) \prec M_B(r)$ implies that $\texttt{rk}(M_A(r)) < \texttt{rk}(M_B(r))$, which in turn implies $\texttt{rk}(M_A(r')) < \texttt{rk}(M_B(r'))$ and thus $M_A(r') \prec M_B(r')$.

- $\texttt{rk}(M_A(r')) = \texttt{rk}(M_B(r')) = k$. We already know that $k > 0$. Moreover, by induction hypothesis, $M_A(r'-1) \prec M_B(r'-1)$, which in particular means that $M_A(r'-1) \neq M_B(r'-1)$ (indeed, $\prec$ is a strict total order). We have then $M_A(r') \prec M_B(r')$ by definition of $\prec$.

Hence, the induction holds and the lemma follows. □

A sequence π of $2k$ non-negative integers $(y_1, y_2, \ldots, y_{2k})$ is said to be a *path* from a node u iff (1) $k = 0$ or (2) $k \geq 1$, $y_1 < \texttt{deg}(u)$, $\texttt{port}(\texttt{succ}(u, y_1), u) = y_2$ and $(y_3, \ldots, y_{2k})$ is a path from node $\texttt{succ}(u, y_1)$. The length of π, corresponding to its number of integers, will be denoted by $|\pi|$, and its i*th* integer will be referred to as $\pi[i]$. For every integer $0 \leq i \leq k$, $\pi[1, 2i]$ will denote the path corresponding to the prefix of π of length $2i$. Finally, the concatenation of the path π_1 from u to v and the path π_2 from v to w will be a path from u to w denoted by $\pi_1\pi_2$. This notion of path in particular used in the proof of the next lemma.

Lemma 2.2 *Let A and B be two agents sharing the same node v in a round r. We have $M_A(r) \neq M_B(r)$.*

Proof. Since A and B are initially placed at different nodes, $r > 1$ and both robots are awake at round r. Let P_A (resp. P_B) the path followed by A (resp. B) since its wake-up. Since A and B are located at the same node at round r but initially placed at different nodes, P_A necessarily differs from P_B, which implies that $M_A(r) \neq M_B(r)$. □

We will use log to denote the binary logarithm. We will say that an execution $\mathcal{E}$ of a sequence of instructions lasts T rounds if the number of edges it prescribes to traverse plus the number of rounds it prescribes to wait is equal to T. Moreover, if $\mathcal{E}$ starts in a round r, we will say that $\mathcal{E}$ *is completed in* (resp. *is completed by*) round $r + T$ if $\mathcal{E}$ lasts exactly T rounds (resp. at most T rounds).

We now present three procedures that will be used as building blocks to design our algorithm given in Section 4. They require no common knowledge $\mathcal{K}$.

The first procedure, due to Ta-Shma and Zwick [36], allows to gather a team made of exactly two agents X and X' having distinct labels. We will call it $\texttt{TZ}(\ell)$, where ℓ is the label of the executing agent. In [36], the authors show that if X and X' start executing procedure TZ in possibly different rounds, then they will meet after at most $(n \cdot \min(|\ell_X|, |\ell_{X'}|))^\alpha$ rounds since the start of the later agent, for some integer constant $\alpha \geq 2$.

The second procedure is based on universal exploration sequences (UXS) and is a corollary of the result of Reingold [33]. It is denoted by $\texttt{EXPLO}(N)$, where N is any integer such that $N \geq 2$. The execution of $\texttt{EXPLO}(N)$ by some agent X makes it traverse edges of the underlying graph G, without any waiting period. This execution terminates within at most N^β rounds, where $\beta \geq 2$ is some constant integer. Moreover, if N is an upper bound on n (i.e., the number of nodes in G), we have the guarantee that each node of G will be visited at least once by X during its execution of $\texttt{EXPLO}(N)$, regardless of its starting node.

The third procedure, explicitly described in [4] (and derived from a proof given in [8]), allows an agent to visit all nodes of G assuming there is a fixed token at its starting node and no token

anywhere else. As our model does not assume the existence of tokens, this routine will be later adapted to our needs by making certain agents play the role of token. We call this procedure `EST`, for *exploration with a stationary token*. The execution of `EST` in G lasts at most $8n^5$ rounds and at the end of the execution the exploring agent is back to its starting node. Since this will be important for our purposes, we give below an almost verbatim description of this procedure given in [4].

When executing procedure `EST`, the agent actually constructs and stores a *port-labeled BFS tree* T which is an appropriate representation of a BFS spanning tree of the underlying graph G. By appropriate, we mean a representation that enables the agent to traverse an actual BFS spanning tree of G. In more detail, there will be a correspondance between the nodes of T and the nodes of G: we will denote by $c(u)$ the node of T corresponding to the node u of G. T will be rooted at node $c(v)$ such that v is the node where the agent is located at the beginning of the procedure `EST`. Moreover, we will ensure that the incoming and outgoing ports along the simple path from the root $c(v)$ to any node $c(u)$ in T will describe a shortest path from v to u in G.

Nodes, edges and port numbers are added to the port-labeled BFS tree as follows. At the beginning, the agent is at node v: it initializes T to the root node $c(v)$ and then makes the *process* of $c(v)$. The process of any node $c(w)$ of T consists in checking each neighbor x of w in order to determine whether a corresponding node $c(x)$ has to be added to the tree or not. When starting the process of $c(w)$, the agent, which is located at w, takes port 0 to move to the neighbor $\texttt{succ}(w, 0)$ and then checks this neighbor.

When a neighbor x of w is being checked, the agent verifies whether a node corresponding to x has been previously added to T. To do this, for each node c of T, the agent considers the simple path P from node c to the root $c(v)$ in T and then, starting from node x, tries to successively take the following $\frac{|P|}{2}$ ports in the underlying graph G: $P[1], P[3], P[5] \ldots, P[|P| - 1]$. The agent gets the guarantee that c cannot correspond to x if after taking $0 \leq i \leq \frac{|P|}{2}$ ports, one of these three conditions is satisfied:

1. $0 < i$ and the port by which the agent enters its current node is not $P[2i]$.
2. $i < \frac{|P|}{2}$ and there is no port $P[2i + 1]$ at its current node.
3. $i = \frac{|P|}{2}$ and the current node does not contain the token.

If each of these conditions is never satisfied, then the agent leaves (resp. enters) successively by ports $P[1], P[3], P[5] \ldots, P[|P| - 1]$ (resp. $P[2], P[4], P[6] \ldots, P[|P|]$) and then meets the token: the agent thus gets the guarantee that c corresponds to x.

As soon as the agent gets the guarantee that c corresponds to x or not (after taking $0 \leq i \leq \frac{|P|}{2}$ ports), it goes back to node x. If $i > 0$, it does so by successively taking ports $P[2i], P[2i-2], P[2i-4] \ldots, P[2]$; otherwise it is already at x. After that, if c does not correspond to x, but it remains at least one node c' of T for which the agent does not know yet whether c' corresponds to x or not: it thus repeats with c' what it has done for c. Eventually, either the agent determines that some node c of T corresponds to x and x is *rejected*, or no node of T corresponds to x and x is *admitted*. If x ends up to be admitted, a corresponding node $c(x)$ is attached to $c(w)$ in T by an edge whose port number at $c(w)$ (resp. $c(x)$) is equal to $\texttt{port}(w, x)$ (resp. $\texttt{port}(x, w)$).

Once node x is admitted or rejected, the agent goes back to w by taking $\texttt{port}(x, w)$ and then moves to an unchecked neighbor of w, if any, following the increasing order of w's port numbers.

When all neighbors of w have been checked, the process of $c(w)$ is completed and the agent goes back to v using the simple path from $c(w)$ to $c(v)$ in T. Let $\mathcal{Y}$ be the set of all paths in the port-labeled BFS tree from its root to a node that has not yet been processed. If $\mathcal{Y} = \emptyset$ then procedure `EST` is completed (and the port-labeled BFS tree spans the underlying graph). Otherwise, the agent considers the lexicographically smallest path of $\mathcal{Y}$, which leads from the root $c(v)$ to some unprocessed node $c(s)$ in T, and follows this path in G to get at node s. When getting at this node, the agent starts the process of $c(s)$.

3 Characterization of the Gatherable Teams

The aim of this section is to prove Theorem 3.1, which provides a full characterization of the gatherable teams.

Theorem 3.1 *A team L is gatherable if and only if its symmetry index is 1.*

The proof of Theorem 3.1 follows directly from Lemmas 3.1 and 3.2 given below.

Lemma 3.1 *If the symmetry index of a team L is 1, then L is gatherable.*

Proof. Consider any team L of k agents for which the symmetry index is 1. By definition, L is gatherable if for every initial setting $IS(L)$, there exists an algorithm even specifically dedicated to $IS(L)$ that allows the agents to solve gathering. Hence, to show that the lemma holds, it is enough to show that Algorithm $\mathcal{A}$, whose pseudocode is given below in Algorithm 1, allows to solve the task of gathering, assuming that the multiset L and the graph order n are provided as input. In the rest of this proof, we will denote by r_0 the first round in which an agent wakes up and we will say that an agent is *advanced* if it has at least completed the execution of line 1 of Algorithm 1.

Besides Procedure `EXPLO` introduced in Section 2, the pseudocode of Algorithm 1 involves three particular functions. The first one is `CurMultLab`(): when an agent X calls it from a node u, the function simply returns the multiset of labels of all advanced agents that are currently at node u (including X's label if X is advanced). The second function is any fixed injective map $f : \mathscr{P}(L) \to \big[1..2^k\big]$, where $\mathscr{P}(L)$ is the set of multisubsets of L (such function exists as the cardinality of $\mathscr{P}(L)$ is at most 2^k). The third function is $\texttt{TZ}(\ell, i)$, where i and ℓ are positive integers. When calling $\texttt{TZ}(\ell, i)$ in a round r, the function returns the ith action (move through some port p or wait during the current round) that would be asked by Procedure $\texttt{TZ}(\ell)$ in an execution $\mathcal{E}$ of the procedure starting from the node occupied by the agent in round $r - i + 1$, assuming the agent is alone in the graph (in which case, $\mathcal{E}$ never terminates). Note that $\texttt{TZ}(\ell, i)$ is indeed computable by the agent in round r, because according to Algorithm 1, if $i \geq 2$, then, in round $r - 1$, the agent called $\texttt{TZ}(\ell, i - 1)$ and performed the $(i - 1)$th action of $\mathcal{E}$.

As stated in Section 2, Procedures `TZ` and `EXPLO` satisfy the following properties in a graph of order n. First, if two agents X and X' with distinct labels ℓ_X and $\ell_{X'}$ start executing, in possibly different rounds, $\texttt{TZ}(\ell_X)$ and $\texttt{TZ}(\ell_{X'})$, then they will meet after at most $(n \cdot \min(|\ell_X|, |\ell_{X'}|))^\alpha$ rounds since the start of the later agent, for some constant $\alpha \geq 2$. Second, an execution of $\texttt{EXPLO}(N)$ by an agent X lasts at most N^β rounds and allows X to visit every node of the graph if $n \leq N$. In particular, the latter property implies that every agent becomes advanced by round $r_0 + 2n^\beta$.

Algorithm 1: Algorithm $\mathcal{A}(L, n)$

```
perform EXPLO(n);
S := CurMultLab();
i := 1;
while S ≠ L do
    perform TZ(f(S), i);
    if S ≠ CurMultLab() then
        S := CurMultLab();
        i := 1;
    else
        i := i + 1;
declare termination;
```

The execution of the while loop of Algorithm $\mathcal{A}$ by an agent X can be viewed as a sequence of phases. In each phase, agent X executes step by step Procedure $\texttt{TZ}(f(S))$ where S is the multiset of labels of the advanced agents that are at X's node at the beginning of the phase. A phase ends as soon as the multiset of labels of the advanced agents that are currently at X's node changes: if the new multiset of advanced agents' labels S' is L, then X declares termination (and so does every other agent), otherwise it begins a new phase by restarting from scratch a step-by-step execution of Procedure TZ but with $f(S')$, instead of $f(S)$, as input.

Note that agent X declares termination in some round r, if, and only if, in this round the multiset of labels of all advanced agents that are at its current node corresponds to L (in which case, all agents also declare termination in round r). Also note that according to Algorithm $\mathcal{A}$, when two advanced agents meet, they stay together forever thereafter as they will necessarily perform the exact same actions from this event onward. Thus, since every agent becomes advanced by round $r_0 + 2n^\beta$, to prove the lemma, it is enough to prove the following: for any round $r' \geq r_0 + 2n^\beta$, if not all agents are gathered in round r', then there exist two agents that are not together in round r' that meet in some round $r'' > r'$. Suppose by contradiction it is not the case for some round r'.

Since the symmetry index of L is 1, there are two distinct nodes u and u' that are occupied in round r' by some agent X and some agent X', respectively, and such that the multiset of labels S of all agents that are at u in r' is different from the multiset of labels S' of all agents that are at v in r'. From the previous explanations, we know that X and X' cannot be together in any round of $[r_0 + 2n^\beta..r']$; moreover, the contradictive assumption implies that X and X' cannot meet after round r'. In other words, X and X' can never share the same node from round $r_0 + 2n^\beta$ on.

In view of Algorithm $\mathcal{A}$, the contradictive assumption and the fact that every agent becomes advanced by round r_0+2n^β, there is a round $r_0+2n^\beta \leq t \leq r'$ (resp. $r_0+2n^\beta \leq t' \leq r'$) such that in round $r'+(2^k n)^\alpha$, agent X (resp. X') has just finished the first $(2^k n)^\alpha+r'-t$ (resp. $(2^k n)^\alpha+r'-t'$) actions of some execution of $\texttt{TZ}(f(S))$ (resp. $\texttt{TZ}(f(S'))$). Assume without loss of generality that $t \leq t'$. It follows that agents X and X' necessarily meet in a round $t' \leq t'' \leq r' + (2^k n)^\alpha$ in view of the property of Procedure TZ recalled above and the fact that $1 \leq f(S) < f(S') \leq 2^k$ or $1 \leq f(S') < f(S) \leq 2^k$. However, since $r_0 + 2n^\beta \leq t'$, we obtain a contradiction with the fact that X and X' can never share the same node from round $r_0 + 2n^\beta$ on. Hence, the lemma follows. □

Lemma 3.2 *If the symmetry index of a team L is not 1, then L is not gatherable.*

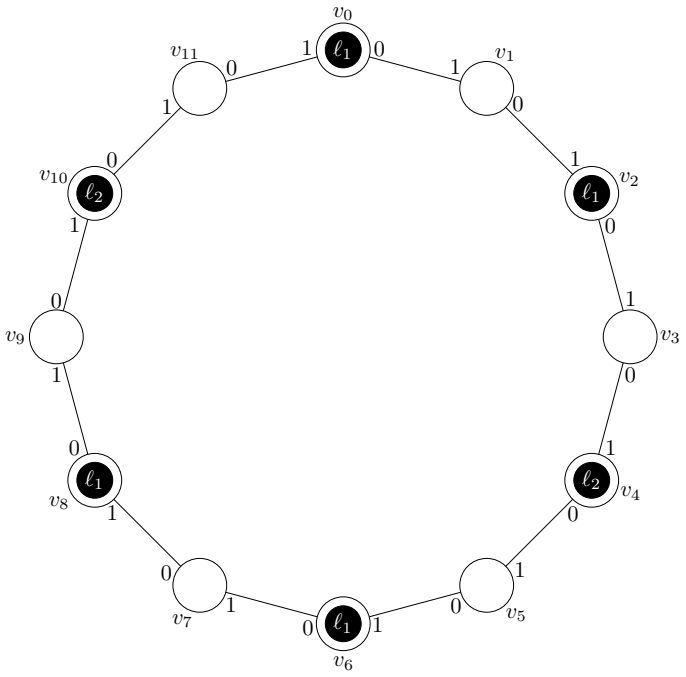

Figure 1: Initial configuration for a team of agents $\{\ell_1, \ell_1, \ell_1, \ell_1, \ell_2, \ell_2\}$ with $k = 6$, $\sigma = 2$ and with fixed map h such that $h(0) = \ell_1$, $h(1) = \ell_1$ and $h(2) = \ell_2$. Agents are represented as black disk with their label in white.

Proof. Let L be a team of k agents with symmetry index $\sigma \neq 1$, let $m_L(\ell)$ denote the multiplicity of label ℓ in L, and let Λ be the support of L. We construct an initial configuration $IS(L)$ as follows. Let the underlying graph be a ring G of size $k\sigma$, with nodes $v_0, v_1, \ldots, v_{k\sigma-1}$ arranged in clockwise order. Each node v_i has two ports: port 0 leads to v_{i+1}, and port 1 leads to v_{i-1} (indices are taken modulo $k\sigma$). Let $h : \{0, 1, \ldots, \frac{k}{\sigma} - 1\} \to \Lambda$ be any map such that $|h^{-1}(\ell)| = \frac{m_L(\ell)}{\sigma}$ for all $\ell \in \Lambda$ (such a map exists as $\frac{\sum_{\ell \in \Lambda} m_L(\ell)}{\sigma} = \frac{k}{\sigma}$). Now, place agents on the ring as follows: for each $i \in \{0, \ldots, \frac{k}{\sigma} - 1\}$ and each $j \in \{0, \ldots, \sigma - 1\}$, place an agent with label $h(i)$ at node $v_{i\sigma+jk}$. The other nodes, i.e., the nodes v_i such that $i \not\equiv 0 \mod \sigma$, do not initially contain any agent. Observe that for each $j \in \{0, \ldots, \sigma - 1\}$ the set of nodes $\{v_i \mid jk \leq i \leq (j+1)k - 1\}$ contains exactly $\frac{m_L(\ell)}{\sigma}$ agents with label ℓ. It follows that the agents placed in G correspond exactly to the team L. We assume that all agents are activated in round 1. See Figure 1 for an example of such a construction.

Let S_i^r denote the set of agents located at node v_i in round r. We will prove by induction that for any deterministic algorithm $\mathcal{A}$, the following property holds for all rounds $r \in \mathbb{N}^*$:

$H(r)$: For all $0 \leq i < j \leq k\sigma - 1$ such that $i \equiv j \mod k$, there exists a bijection $f_{i,j}^r : S_i^r \to S_j^r$ such that for every $X \in S_i^r$, the memory of X in round r equals that of $f_{i,j}^r(X)$.

First, we show the base case $H(1)$ of the induction. Let $i, j \in \{0, \ldots, k\sigma - 1\}$ with $i < j$ and $i \equiv j \equiv m \mod k$ for some m such that $0 \leq m < k$. If $m \not\equiv 0 \mod \sigma$, then v_i and v_j are both empty, and the bijection $f_{i,j}^1$ is trivial. Hence, we can assume that $m \equiv 0 \mod \sigma$. This means that v_i and v_j each host a single agent with label $h(m/\sigma)$. In round 1, each agent has initial memory $(\ell, \perp, \perp)$ where ℓ is its label. Therefore, for any such i, j, the singleton sets S_i^1 and S_j^1 contain agents

with identical memories. The bijection $f^1_{i,j}$ is also trivial in this case, and thus $H(1)$ necessarily holds.

Assume $H(r)$ holds for some round $r \geq 1$. We show that $H(r+1)$ also holds. Denote by $\pi(\cdot)$ the reduction modulo $k\sigma$. For any node v_i, let us partition S_i^{r+1} into three disjoint sets:

- $\dot{S}_i^{r+1}$: agents that decided in round r to stay idle at v_i,
- $\vec{S}_i^{r+1}$: agents that decided in round r to move to v_i from $v_{\pi(i-1)}$ via port 0,
- $\overleftarrow{S}_i^{r+1}$: agents that decided in round r to move to v_i from $v_{\pi(i+1)}$ via port 1.

By the inductive hypothesis $H(r)$, for all $i \equiv j \mod k$ such that $i < j$, there exist bijections $f^r_{i,j}$, $f^r_{\pi(i-1),\pi(j-1)}$ and $f^r_{\pi(i+1),\pi(j+1)}$ from S_i^r to S_j^r, $S^r_{\pi(i-1)}$ to $S^r_{\pi(j-1)}$ and $S^r_{\pi(i+1)}$ to $S^r_{\pi(j+1)}$, respectively, that all preserve the agents' memories in round r. Since the algorithm is deterministic, agents with the same memories take the same action. Hence, there exists a bijection from $\dot{S}_i^{r+1}$ to $\dot{S}_j^{r+1}$ that preserves the memories hold in round r. Similarly, such bijections exist between $\vec{S}_i^{r+1}$ and $\vec{S}_j^{r+1}$, and between $\overleftarrow{S}_i^{r+1}$ and $\overleftarrow{S}_j^{r+1}$. From these, we can construct a global bijection $f^{r+1}_{i,j} : S_i^{r+1} \to S_j^{r+1}$ that preserves both the memories hold in round r and the actions taken in round r. Let us now verify that this bijection necessarily preserves the agents' memories hold in round $r+1$. For any agent $X \in S_i^{r+1}$, define $X' = f^{r+1}_{i,j}(X)$. Their memories in round $r+1$ are respectively $M_X(r+1) = (M_X(r), A_X(r+1), B_X(r+1))$ and $M_{X'}(r+1) = (M_{X'}(r), A_{X'}(r+1), B_{X'}(r+1))$. Since $f^{r+1}_{i,j}$ preserves the memories hold in round r (resp. the actions taken in round r), we have $M_X(r) = M_{X'}(r)$ (resp. $A_X(r+1) = A_{X'}(r+1)$). The set $B_X(r+1)$ is also preserved because $B_X(r+1) = \{(A_Y(r+1), M_Y(r)) \mid Y \in S_i^{r+1} \setminus \{X\}\}$, $B_{X'}(r+1) = \{(A_{Y'}(r+1), M_{Y'}(r)) \mid Y' \in S_j^{r+1} \setminus \{X'\}\}$ and $f^{r+1}_{i,j}$ preserves the memories hold in round r as well as the actions taken in round r. Hence, $M_X(r+1) = M_{X'}(r+1)$, completing the inductive step. Therefore, $H(r)$ holds for all $r \geq 1$ by induction.

Consequently, in any round r, for any agent X at node v_i, there exists a corresponding agent $f^r_{i,j}(X)$ at node v_j for some $j \neq i$ such that $j \equiv i \mod k$. It follows that gathering is never achieved. □

4 A Polynomial-Time Algorithm Using Common Knowledge of Small Size

In this section, we present an algorithm, called $\mathcal{HG}$ (short for *Homonymous Gathering*). Essentially, this algorithm enables the gathering of any gatherable team L, with multiplicity index μ, in a time at most polynomial in n and $|\lambda|$, provided that $\mathcal{K}$ is the binary representation of $\lceil \log\log(\mu+1) \rceil$. Note that the size of $\mathcal{K}$ is thus in $O(\log\log\log\mu)$.

The remainder of this section is structured as follows. First, we provide the intuition underlying Algorithm $\mathcal{HG}$. Next, we present a detailed description of the algorithm. Finally, we establish its correctness and analyze its time complexity.

4.1 Intuition

At first glance, it could be quite tempting to simply reapply the strategy of Algorithm 1. However, for this algorithm to work correctly, it requires as input both the multiset of labels L and the graph order n. In particular, termination detection relies on the knowledge of L: an agent stops when, at its current node, the multiset of labels of the advanced agents (those having at least completed the execution of the first line of the algorithm) matches L. True, such a simple detection could also have been done knowing only the cardinality of L. But in our situation, we are nowhere near knowing L or its cardinality. When the oracle provides the binary representation of $\lceil \log \log(\mu+1) \rceil$, this does not even reveal the value of μ: it only yields an approximate upper bound U such that $\mu \leq U \leq (\mu+1)^2$.

Even setting aside the problem of initial knowledge, another difficulty remains: that of complexity. A quick analysis can show that the time required by Algorithm 1 to gather all agents is far from being polynomial in n and $|\lambda|$. Certainly, we could have improved the complexity to some extent, by designing the injective map f more carefully, but this would have not been enough without more radical change. Indeed, our method of progressively constructing larger and larger agent groups until only one remains, by applying function `TZ`, necessarily requires providing it with an input that is more than just the smallest label of the group, to avoid symmetries that could prevent gathering. Actually, the function must also take into account, in some way, the other labels in the group. This simple observation constitutes a fundamental barrier to reaching polynomial complexity in n and $|\lambda|$ using this strategy. We therefore need a significantly more sophisticated approach, whose main ideas are outlined below.

Our algorithm is structured around five states, between which an agent can switch and act accordingly. These states are: `cruiser`, `token`, `explorer`, `searcher`, and `shadow`. Among them, `shadow` and `searcher` can be introduced immediately as they are simple and play only a secondary role in the overall process. When an agent enters state `shadow`, it never leaves it: from then on, it only mirrors, in every round, the actions of a specific agent, called its *guide*—exiting through the same port, waiting when the guide waits, and declaring termination when the guide does. When an agent is in state `searcher`, it simply applies `EXPLO`(2^1), `EXPLO`(2^2), `EXPLO`(2^3), and so on, till finding an agent in state `token`, in which case it transits to state `shadow` by choosing this agent as its guide (this is unambiguous as we will prove that two agents can never be in state `token` at the same node in the same round). By contrast, the states other than `shadow` and `searcher` play a central role and require a detailed explanation, as they capture the essence of the algorithm.

A fundamental requirement is that, within a polynomial number of rounds in n and $|\lambda|$ from the first round r_0 in which any agent wakes up, every agent must have woken up. State `cruiser` is a first step toward fulfilling this condition. Upon waking up, an agent begins in this state, where the primary objective is to "quickly" meet another agent. For simplicity, and because it is sufficient for our purpose, the agent in state `cruiser` will precisely focus on finding a peer that is in state `cruiser` or `token`, and will deliberately ignore the meetings involving only agents in the other three states. To achieve this, the agent proceeds in phases $i = 1, 2, \ldots$ in which it executes procedure `EXPLO`(2^i) and then, for 2^i rounds, procedure `TZ` using its own label as input. This continues until it finally meets an agent in one of the two above-mentioned states. What happens then depends on the states of the agents involved in the meeting. If the meeting involves an agent X in state `token`, then our agent in state `cruiser` transits to state `shadow` by choosing X as its guide. Otherwise, all agents involved in the meeting are in state `cruiser`: depending on their current memories (which are all different in view of Lemma 2.2), one becomes `explorer`, one becomes `token`, while the

others, if any, transit to state `shadow` by choosing as their guide the agent becoming `token`.

At this point, using the properties of procedures `EXPLO` and `TZ`, together with the fact that an execution of the first $\lceil \log x \rceil$ phases lasts at most $H(x)$ rounds for some polynomial H, we can guarantee that the team's first meeting—necessarily involving only agents in state `cruiser`—occurs within $\text{poly}(n, |\lambda|)$ time from r_0. Precisely, if some agent is not woken up by the adversary by round $r_0 + H(\lceil \log n \rceil)$, then a meeting is guaranteed by this round because an execution of phase $\lceil \log n \rceil$ ensures that all nodes are visited via $\texttt{EXPLO}(2^{\lceil \log n \rceil})$. Otherwise, all executions of `TZ` initiated by the agents in a phase i start within a window of fewer than $H(\lceil \log n \rceil)$ rounds, and, in view of the meeting bound of procedure `TZ` (cf. Section 1.3), we can prove that a meeting occurs before any agent finishes the execution of `TZ` in phase $\lceil \log n \rceil + \lceil (\alpha \log(n|\lambda|)) \rceil$, for some constant α (i.e., by round $r_0 + H(2^{\lceil \log n \rceil + \lceil (\alpha \log(n|\lambda|)) \rceil})$. Hence, in either case, the team's first meeting indeed occurs in some round r_1, within $\text{poly}(n, |\lambda|)$ time from r_0. However, we cannot yet guarantee the fundamental requirement that, within such a time frame, every agent wakes up. To achieve this, additional mechanisms are needed. This is where the states `token` and `explorer` come into the picture.

When an agent A becomes an explorer in some round t at some node v, there is exactly one agent B that becomes a token in the same round at the same node. As the name suggests, the role of A in state `explorer` is to explore the graph. To do so, it emulates procedure `EST` (cf. Section 1.3), using B as its token. While in state `token`, agent B plays a passive role: it simply remains idle at node v. The only exception is a specific situation described later, in which it will interrupt this behavior and transition to state `searcher`. Meanwhile, during its emulation of `EST` started in round t, agent A follows the instructions of `EST`, but whenever it encounters an agent C in state `token` with $M_C(t) = M_B(t)$, it treats (rightly or wrongly) C as its own token B and proceeds accordingly. In the ideal situation where all agents have distinct labels, A can never confuse B with any other agent, and we can be certain that every node (resp. agent) is visited (resp. woken up) by the end of its exploration, which lasts at most $8n^5$ rounds—the worst-case complexity of `EST`. In particular, the port-labeled BFS tree constructed by A through this exploration indeed corresponds to a spanning tree of the underlying graph G. Unfortunately, we are not necessarily in such an ideal situation. While exploring the graph, agent A might sometimes mistakenly consider another agent as its token B, which could prevent it from visiting all nodes and, consequently, we cannot guarantee that all agents are woken up by the end of its exploration. Likewise, the tree constructed by A may actually correspond to a truncated BFS tree of the graph. However, what we can prove is that all explorers starting an exploration in round t with a token whose memory matches that of B in this round will, collectively, succeed in exploring the entire graph by round $t + 8n^5$. Roughly speaking, the union of their explorations, and, more specifically, of the trees they will construct, will span the whole graph, even though some trees may cover significantly larger portions than others. As a result, the algorithm satisfies the aforementioned fundamental requirement: every agent is necessarily woken up within $\text{poly}(n, |\lambda|)$ time from r_0.

While this is of course a significant step forward, we need to keep in mind that our ultimate goal is much stronger and requires all agents to gather at the same node within $\text{poly}(n, |\lambda|)$ time (from r_0). Perhaps surprisingly, we are quite close to achieving this if, instead of limiting each explorer A to a single exploration, they continue performing EST-based explorations until some conditions are met. Precisely, during each exploration that starts (resp. ends) in some round t (resp. t'), agent A considers (similarly as above) any agent C in state `token` as its own token B if $M_C(t) = M_B(t)$. Furthermore, A and its token B, which can be shown to be together in round t', both switch to state `searcher` in this round if, at some point in the interval $[t..t']$, A has encountered an agent C

in state `token` that entered this state before B, or in the same round as B but $M_B(t) \prec M_C(t)$. With these simple rules and the fact the team's symmetry index is 1, we can show the emergence of a kind of domino effect, by which, within $\text{poly}(n, |\lambda|)$ time, only a single explorer E and a single token T eventually remain, while all other agents become `shadow` of this token. In particular, E and T can be shown to enter states `explorer` and `token`, respectively, in round r_1 at the same node, and remain in them forever thereafter. Since each exploration lasts at most $8n^5$ rounds, it follows that, within $\text{poly}(n, |\lambda|)$ time, there exists a round r_2 in which all agents are together at the same node, with E in state `explorer`, T in state `token`, and all other agents in state `shadow`! However, a crucial problem persists: in the current design of our solution, the agents cannot detect this event, because they know a priori neither n nor λ. To overcome this—and, notably, to prevent the last surviving explorer from performing explorations ad indefinitely—we relied on two additional fundamental findings that we brought to light in our analysis. In particular, this is where we make use of the upper bound U on μ, derived by an agent from the oracle's advice, which, as explained earlier, satisfies $\mu \leq U \leq (\mu + 1)^2$.

The first finding is that when an explorer has completed U^c consecutive identical explorations for some prescribed constant c, then it can get a polynomial estimation of n. (Two explorations are considered identical if, in both, the explorer follows exactly the same path and encounters an agent it interprets as its token at the exact same times.) More precisely, we can prove that when such an event occurs for an explorer A, the order η of each of the identical (possibly truncated) BFS trees constructed during the U^c consecutive identical explorations of A satisfies $n \leq U\eta \leq (n+1)n^2$. Intuitively, this stems from the fact that, at some point, a group of at most μ explorers (including A) must have acted in a perfectly symmetric manner, notably by constructing identical trees (of order η) whose union spans the entire graph. Moreover, it is important to mention that starting from round r_2, the last surviving explorer can no longer confuse its token with another one and is therefore guaranteed to complete U^c consecutive identical explorations by round $r_2 + 8U^c n^5$.

The second finding is that while we can prove that the difference $r_2 - r_0$ is upper bounded by some polynomial $P(n, |\lambda|)$, we can also establish a more subtle and useful fact, which may seem counterintuitive at first: the difference $r_2 - r_0$ is upper bounded by some polynomial $P'(n, \tau)$, where τ is the minimum time spent by an agent in state `cruiser`. Why it is more subtle and useful? Consider any explorer X that completes U^c consecutive identical explorations. Let η_X be the order of the trees computed in these explorations and τ_X the time spent by X in state `cruiser`. Since $n \leq U\eta_X \leq (n+1)n^2$ and $\tau \leq \tau_X \leq P(n, |\lambda|)$, we necessarily have:

$$P'(n, \tau) \leq P'(U\eta_X, \tau_X) \leq P'((n+1)n^2, P(n, |\lambda|))$$

In other words, once an explorer X obtains its estimated upper bound $U\eta_X$ on the graph order, it can combine it with τ_X to determine an upper bound on the duration that must elapse before exceeding round r_2. And more importantly, all such upper bounds on this duration computed by the explorers are themselves upper bounded by the same polynomial in n and $|\lambda|$, without requiring prior knowledge of n or λ!

Consequently, we can ensure that gathering with detection is achieved within $\text{poly}(n, |\lambda|)$ time, by asking each explorer X to declare termination at the end of an exploration if:

(i) it has just completed at least U^c consecutive identical explorations, each producing a tree of order η_X, and

(ii) the total number of elapsed rounds since the beginning of its execution exceeds some polynomial bound in both the estimated graph size $U\eta_X$ and the time τ_X it spent in state `cruiser`.[3]

In practice, only the last surviving explorer eventually satisfies these conditions, at which point it is co-located with its token and all other agents (in state `shadow`). Of course, when the explorer declares termination, the algorithm will ask all the colocated agents to simultaneously do the same.

Overall, we end up with an algorithm—called $\mathcal{HG}$—whose design is remarkably simple. However, this simplicity comes at a price of a difficult and challenging analysis. One of the main difficulties was undoubtedly to show that our termination detection mechanism, relying on the two findings described above, indeed works within the expected time frame without producing any false positives.

4.2 Algorithm

In this section, we give a detailed description of procedure $\mathcal{HG}$ executed by an agent A of label ℓ_A. For ease of reading, we describe it as a list of states in which agent A can be and, for each of them, we specify the actions performed by A as well as the rules governing the transitions to other states. In the description of each state X, when we say "agent A transits to state Y" in a round r, we exactly mean that agent A remains idle in its current node in state X until the end of round r and enters state Y in round $r+1$ (precisely at the beginning of $r+1$). By doing so, each transition costs one extra round, but we have the guarantee that, in each round, agent A is in at most one state. The states in which agent A can be are: `cruiser`, `token`, `explorer`, `searcher` and `shadow`. Upon waking up, the agent starts in state `cruiser`. (If an agent is dormant or has completed the execution of $\mathcal{HG}$, it is considered to have no state.)

State `cruiser`.

In this state, agent A works in phases $i = 1, 2, \ldots$ In phase i, it executes procedure `EXPLO`(2^i) and then, for 2^i rounds, procedure `TZ`(ℓ_A). As soon as it meets another agent in state `cruiser` or `token` in a round r at a node v, it interrupts the execution of the current phase and transits either to state `shadow` or `explorer` or `token` in the same round, depending on the cases presented below. Let $\mathcal{S}$ be the set of all agents in state `cruiser` or `token` located at node v in round r, including agent A.

- *Case 1. All agents of $\mathcal{S}$ are in state* `cruiser` *in round r.* In this case, there are at least two agents of $\mathcal{S}$ that are in state `cruiser` in round r. If agent A has the largest (resp. the smallest) memory within $\mathcal{S}$ in round r, then it transits to state `token` (resp. `explorer`). Otherwise, it transits to state `shadow` and its guide is the agent having the largest memory within $\mathcal{S}$ in round r (the role of the guide is given in the description of state `shadow`).
- *Case 2. There is an agent of $\mathcal{S}$ that is in state* `token` *in round r.* Agent A transits to state `shadow` and its guide is the agent of S that is in state `token` in round r (we will show in Section 4.3 that, in every round, a node can contain at most one agent in state `token`).

State `searcher`.

In state `searcher`, agent A executes successively `EXPLO`(2), `EXPLO`(2^2), `EXPLO`(2^3), ..., `EXPLO`(2^i), and so on, until it meets an agent in state `token`. When such a meeting occurs at a

[3]For technical reasons, we substituted this second condition in our algorithm with carefully parametrized waiting periods inserted between explorations.

node v in a round r, agent A transits to state `shadow` in the same round and selects as its guide the agent that is in state `token` at node v in round r.

State `shadow`.

We will show that when agent A enters state `shadow` at a node v, it has exactly one guide that is also located at node v. At the end of each round, agent A simply takes the same decision as its guide regarding whether to remain idle, take a specific port, or declare termination (we will see that agent A always stays colocated with its guide and can always identify it unambiguously). If, in a round r, A's guide decides to transit itself to state `shadow` and selects an agent C as its guide, then, in round $r+1$, A's guide is C.

State `token`

While in this state, agent A remains idle and does nothing until it is, in a round r, in one of the following two situations:

1. agent A shares its current node with an agent in state `explorer` that transits to state `searcher`,
2. or all agents in state `explorer` at the node occupied by A declare termination.[4]

In the first situation (resp. second situation) agent A also transits to state `searcher` (resp. declares termination) in round r.

State `explorer`

We will show that when agent A enters state `explorer` at a node v, there is exactly one agent B that simultaneously enters state `token` at the same node. We will also show that B remains in state `token` (and thus idle at node v) as long as A remains in state `explorer`. In this state, agent A executes the protocol given in algorithm 2, which uses the notion of *seniority* that is defined as follows. The seniority of an agent B in state `token` (resp. `explorer`) in a round r' is the difference $r' - r$, where r is the round in which B entered state `token` (resp. state `explorer`). Since, in view of the description of the states, an agent can enter state `token` or `explorer` at most once, the difference is well defined. The protocol also uses function $\texttt{EST}^+$, which corresponds to procedure `EST` but with the following three changes. Consider an execution of $\texttt{EST}^+$ initiated by agent A in a round t from a node u. The first change is that to start this execution, we need to give function $\texttt{EST}^+$ any memory M as input. The second change is that each time agent A encounters an agent B that is in state `token` and such that $M_B(t) = M$,[5] agent A considers it is with its token. The third change is that $\texttt{EST}^+$ returns a triple $(b, \eta, trace)$. The first element, b, is a Boolean. Its value is true if, during the execution of $\texttt{EST}^+$, agent A encounters an agent C in state `token` of either higher seniority or equal seniority but such that $M \prec M_C(t)$. Otherwise, b is false. The second element, η, corresponds to the number of nodes in the BFS tree constructed during the execution of $\texttt{EST}^+$ (recall that, when executing `EST`, an agent constructs such a tree). Finally the

[4] We write "all agents in state `explorer`" instead of "an agent in state `explorer`" to formally avoid what might appear as a conflict in the description—namely, at the occupied by A, one `explorer` switching to state `searcher` while another declares termination. In practice, however, this can never happen: we prove that only one agent in state `explorer` will declare termination, and at that moment no other agents are in this state.

[5] Note that even if agent A is not with C in round t, it can determine $M_C(t)$ when meeting C in a round $t' > t$ using $M_C(t')$

third element, *trace*, is, as the name suggests, a trace of the execution. It is represented as the sequence $(p_1, q_1, m_1, p_2, q_2, m_2, \ldots, p_k, q_k, m_k)$ that satisfies the following two conditions. The first condition is that $(p_1, q_1, p_2, q_2, \ldots, p_k, q_k)$ is the path followed by agent A during the execution of $\texttt{EST}^+$. The second condition is that for every $1 \leq i \leq k$, $m_i = 1$ (resp. $m_i = 0$) if in round $t + i$ agent A is (resp. is not) with a token (in the sense of the second change given above).

Algorithm 2: Procedure executed by agent A while in state `explorer`

```
1  counter := 1;
2  x := the decimal number whose binary representation is K;
3  U := 2^(2^x); trace_old := the empty list;
4  τ := the number of rounds spent by agent A in state cruiser;
5  repeat
6      M := the memory of the agent in state token that is currently at the same node as A;
7      (b, η, trace_new) := EST+(M);
       /* Agent A is back at the node from which it has started the previous
          execution of EST+ */
8      if b = false then
9          wait Σ_{i=1}^{11⌈β log(ηUτ)⌉} 2^i rounds ;
10         if trace_old = trace_new then
11             counter := counter + 1;
12         else
13             counter := 1;
14         trace_old := trace_new;
15 until counter = U^{25β} or b = true;
16 if b = true then
17     transit to state searcher;
18 else
19     declare termination;
```

4.3 Correctness and Complexity Analysis

All the lemmas given in this subsection are stated under the following three assumptions. The first assumption is that the team of agents is obviously gatherable. The second assumption is that the algorithm executed by an agent when it wakes up is procedure $\mathcal{HG}$. The third assumption is that the advice $\mathcal{K}$, given by the oracle, corresponds to the binary representation of $\lceil \log \log(\mu + 1) \rceil$, where μ is the multiplicity index of the team. Thus, for ease of reading, these assumptions will not be explicitly repeated in the statements of the lemmas.

Furthermore, since $\mathcal{K}$ corresponds to the binary representation of $\lceil \log \log(\mu + 1) \rceil$, it follows that variable U used in Algorithm 2 is necessarily equal to $2^{(2^{\lceil \log \log(\mu+1) \rceil})}$. Thus, for simplicity, we will reuse U as the notation for this value throughout the proof below. Finally, we will denote by r_0 the first round in which at least one agent wakes up.

We start with the following lemma that is a direct consequence of procedure $\mathcal{HG}$.

Lemma 4.1 *Let S be any of the 5 states of procedure $\mathcal{HG}$. If an agent leaves state S in a round r, it cannot return to this state in any subsequent round.*

The next two lemmas establish key properties related to state `token`.

Lemma 4.2 *There exists an agent that enters state* `token` *within at most a polynomial number of rounds in n and $|\lambda|$ after round r_0.*

Proof. In Section 2, we mention that if two agents X and X' start executing procedure `TZ` in possibly different rounds, then they will meet after at most $(n \cdot \min(|\ell_X|, |\ell_{X'}|))^\alpha$ rounds since the start of the later agent, for some integer constant $\alpha \geq 2$. We also mention that for any positive integer $N \geq 2$, procedure `EXPLO`(N) lasts at most N^β rounds, for some integer constant $\beta \geq 2$.

In this proof, constants α and β play a central role. Indeed, we show below that some agent necessarily enters state `token` by round $r_0 + (16T)^\beta$ with $T = (n|\lambda|)^\alpha + (8n)^\beta$, which is enough to prove the lemma.

Assume by contradiction that no agent enters state `token` by round $r_0 + (16T)^\beta$. We have the following claim.

Claim 4.1 *Let X be any agent that wakes up in a round $r_0 \leq r \leq r_0 + (16T)^\beta$. Agent X never meets an agent in state* `token` *or* `cruiser` *before round $r_0 + (16T)^\beta$ and it remains in state* `cruiser` *from round r to round $r_0 + (16T)^\beta$ included.*

Proof of the claim. Suppose by contradiction that the claim does not hold. According to Algorithm $\mathcal{HG}$, an agent starts in state `cruiser` in the round of its wake-up and decides to transit from state `cruiser` to another state in some round if, and only if, it meets an agent in state `token` or `cruiser` in this round. Moreover, recall that when an agent decides to transit to a new state in some round, it enters this state in the next round. Hence, there is a round $r \leq r' \leq r_0 + (16T)^\beta - 1$ in which agent X decides to transit from state `cruiser` to another state because, in round r', it is at some node v with an agent in state `token` or `cruiser`. In the first case, it means that an agent enters state `token` by round $r' \leq r_0 + (16T)^\beta - 1$, which contradicts the assumption that no agent enters state `token` by round $r_0 + (16T)^\beta$. Hence, at node v in round r', there is no agent in state `token` and there are at least two agents in state `cruiser` (including agent X). Let $\mathcal{S}$ be the set of all agents in state `cruiser` at node v in round r. In view of Lemma 2.2, no two agents of $\mathcal{S}$ can have the same memory in round r. Hence, according to the rules applied by an agent in state `cruiser`, the agent having the largest memory in round r among the agents of $\mathcal{S}$ necessarily enters state `token` in round $r' + 1 \leq r_0 + (16T)^\beta$. This contradicts again the assumption that no agent enters state `token` by round $r_0 + (16T)^\beta$, which concludes the proof of this claim. ⋆

While in state `cruiser`, any agent X works in phases $i = 1, 2, \ldots$ where phase i consists of an execution of procedure `EXPLO`(2^i) followed by an execution, for 2^i rounds, of procedure `TZ`(ℓ_X). All of this is interrupted only when it meets another agent in state `cruiser` or `token`. For every positive integer k an entire execution of the first $\lceil \log k \rceil$ phases of state `cruiser` lasts at most $\sum_{i=1}^{\lceil \log k \rceil} (2^{i\beta} + 2^i)$, which is upper bounded by $(8k)^\beta$. Hence, an entire execution of the first $\lceil \log n \rceil$ phases lasts at most $(8n)^\beta$. Let A be any agent that wakes up in round r_0. Since $(8n)^\beta < (16T)^\beta$, we know by Claim 4.1 that agent A is in state `cruiser` from round r_0 to round $r_0 + (8n)^\beta$ included.

In view of the explanations given just above, it follows that an entire execution of phase $\lceil \log n \rceil$, and thus of procedure $\texttt{EXPLO}(2^{\lceil \log n \rceil})$, is completed by agent A by round $r_0 + (8n)^\beta$. Since an execution of procedure $\texttt{EXPLO}(2^{\lceil \log n \rceil})$ allows to visit every node of the underlying graph at least once, it follows that every agent enters state `cruiser` by round $r_0 + (8n)^\beta$. Indeed, otherwise it means that by some round $r \leq r_0 + (8n)^\beta$, agent A enters a node occupied by a dormant agent that immediately enters state `cruiser` in round r, which contradicts Claim 4.1. As a result, we have the guarantee that the delay between the wake-ups of any two agents is at most $(8n)^\beta$ rounds and, in view of Claim 4.1, every agent is in state `cruiser` from round $r_0 + (8n)^\beta$ to round $r_0 + (16T)^\beta$ included.

We stated above that, for every positive integer k, an entire execution of the first $\lceil \log k \rceil$ phases of state `cruiser` is upper bounded by $(8k)^\beta$. This implies that an entire execution of the first $\lceil \log T \rceil$ phases is upper bounded by $(8T)^\beta$. Moreover, $(8n)^\beta + (8T)^\beta < (16T)^\beta$, which means that every agent is in state `cruiser` from round $r_0 + (8n)^\beta$ to round $r_0 + (8n)^\beta + (8T)^\beta$ included. Hence, an entire execution of phase $\lceil \log T \rceil$ is completed by every agent by round $r_0 + (8n)^\beta + (8T)^\beta$. During its execution of phase $\lceil \log T \rceil$, any agent X starts an execution of procedure $\texttt{TZ}(\ell_X)$ for at least $T \geq (n|\lambda|)^\alpha + (8n)^\beta$ rounds. Since the delay between the wake-ups of any two agents is at most $(8n)^\beta$, the delay between the starting times of procedure `TZ` by any two agents in phase $\lceil \log T \rceil$ is also at most $(8n)^\beta$ rounds. Furthermore, by Theorem 3.1 and the assumption that the team is gatherable, it follows that the team must contain at least two agents with distinct labels. Hence, there is an agent B with label $\ell_B = \lambda$ (resp. an agent B' with a label $\ell_{B'} \neq \lambda$) that executes $\texttt{TZ}(\lambda)$ (resp. $\texttt{TZ}(\ell_{B'})$), from round r' to round $r' + (n|\lambda|)^\alpha$ included, where r' is the later of the starting rounds of the two agents of procedure `TZ` in phase $\lceil \log T \rceil$. According to the properties of procedure `TZ` recalled at the beginning of this proof, agent B meets agent B' in some round $r' \leq r'' \leq r' + (n|\lambda|)^\alpha$. Note that agent B and B' are necessarily woken up in round r''. Besides, $r' + (n|\lambda|)^\alpha < r_0 + (8n)^\beta + (8T)^\beta$ as an entire execution of phase $\lceil \log T \rceil$ is completed by every agent by round $r_0 + (8n)^\beta + (8T)^\beta$. In view of this and Claim 4.1, agent B and B' are both in state `cruiser` when they meet in round r'' and $r'' < r_0 + (16T)^\beta$. This is a contradiction with Claim 4.1. Hence, some agent enters state `token` by round $r_0 + (16T)^\beta$, which concludes the proof of the lemma. □

Lemma 4.3 *An agent enters state* `token` *(resp.* `explorer`*) at a node v in a round r if, and only if, exactly one agent enters state* `explorer` *(resp.* `token`*) at node v in round r. Moreover, in every round, there can never be two agents in state* `token` *that occupy the same node.*

Proof. According to Algorithm $\mathcal{HG}$, an agent A can enter state `token` (resp. `explorer`) at a node v in a round r if, and only if, the following three conditions are satisfied at node v in round $r - 1$:

1. The set $\mathcal{S}$ of the agents in state `cruiser` contains at least two agents, including agent A.
2. Agent A has the largest memory (resp. the smallest memory) among the agents of $\mathcal{S}$.
3. There is no agent in state `token`.

Moreover, Lemma 2.2 implies that the memories of any two agents of $\mathcal{S}$ in round $r - 1$ are different. Hence, an agent enters state `token` (resp. `explorer`) at a node v in a round r if, and only if, exactly one agent enters state `explorer` (resp. `token`) at node v in round r.

To prove the lemma, it now remains to show that in every round, there can never be two agents in state `token` that occupy the same node. Assume, by contradiction, there is a round r' in which two agents A and B are in state `token` at the same node u. Let r'_A (resp. r'_B) be the round in which agent A (resp. B) enters state `token` (r'_A, as well as r'_B, is unique and at most equal to r' in view of Lemma 4.1). According to what is stated above and the fact that an agent never moves while in state `token`, we know that agent A (resp. B) is in state `cruiser` at node u in round $r'_A - 1$ (resp. $r'_B - 1$) and has the largest memory among the agents in state `cruiser` at node u in round $r'_A - 1$ (resp. $r'_B - 1$). We also know that agent A (resp. B) is in state `token` at node u from round $r'_A - 1$ (resp. $r'_B - 1$) to round r' included.

If $r'_A = r'_B$ then agents A and B are both in state `cruiser` at node u in round $r'_A - 1$ and both have the largest memory among the agents in state `cruiser` (and thus the same memory) at node u in round $r'_A - 1$. However, this directly contradicts Lemma 2.2.

Consequently, we necessarily have $r'_A \neq r'_B$ and we can suppose, without loss of generality, that $r'_A < r'_B$. It follows that at node u in round $r'_B - 1$, agent A (resp. B) is in state `token` (resp. `cruiser`). However, according to the three conditions given at the beginning of this proof, agent B can enter state `token` in round r'_B at node u only if there is no agent in state `token` in round $r'_B - 1$ at node u. This means that agent B cannot enter state `token` in round r'_B: this is a contradiction with the definition of round r'_B.

As a result, in every round, there can never be two agents in state `token` that occupy the same node, which ends the proof of the lemma. □

Consider any agent A (resp. B) that ends up entering state `explorer` (resp. `token`). By Lemma 4.1, we know that agent A (resp. B) enters state `explorer` (resp. `token`) exactly once. By Algorithm 2, we also know that once an agent becomes `explorer` (resp. `token`), it will never enter state `token` (resp. `explorer`) thereafter. Hence, we can unambiguously define `home`(A) (resp. `home`(B)) as the node at which agent A (resp. B) enters state `explorer` (resp. `token`). Moreover, when it does so, there is exactly one agent that simultaneously enters state `token` (resp. `explorer`) at `home`(A) (resp. `home`(B)) in view of Lemma 4.3. In the remainder, this other agent will be denoted by `tok`(A) (resp. `exp`(B)) (since agent A (resp. B) enters state `explorer` (resp. `token`) exactly once, the definition of `tok`(A) (resp. `exp`(B)) is unambiguous). By a slight misuse of language, we will occasionally refer to `tok`(A) as the token of A and `exp`(B) as the explorer of B. Note that we have `home`(A) = `home`(B) if B = `tok`(A) (or equivalently if A = `exp`(B)). Having established these definitions, we can now proceed to the next lemma.

Lemma 4.4 *Let $\mathcal{E}$ be an execution of* `EST`$^+$ *started in a round r by an agent A. We have the following four properties.*

1. *In $\mathcal{E}$, the input given to function* `EST`$^+$ *is $M_{\texttt{tok}(A)}(r)$.*
2. *$\mathcal{E}$ lasts at most $8n^5$ rounds.*
3. *In round r and in the round when $\mathcal{E}$ is completed, agent A (resp.* `tok`(A)*) is in state* `explorer` *(resp.* `token`*) at* `home`(A)*.*
4. *Let $(*, \eta, *)$ be the triple returned by function* `EST`$^+$ *when $\mathcal{E}$ is completed. If there exists no agent $X \neq$* `tok`(A) *such that $M_X(r) = M_{\texttt{tok}(A)}(r)$, then all nodes of the underlying graph G are visited by A during $\mathcal{E}$ and $\eta = n$, otherwise $1 \leq \eta \leq n$.*

Proof. Before starting, just observe that an agent can start an execution of $\texttt{EST}^+$ only when in state `explorer` and it cannot decide to leave this state during an execution of $\texttt{EST}^+$. This implies that agent A is in state `explorer` in round r and in the round when $\mathcal{E}$ is completed if such a round exists (of course, we show below that such a round exists). This observation must be kept in mind when reading the proof as it will not always be repeated in order to lighten the text.

We now start this proof by assuming that property $\Psi(\mathcal{E}, A)$, made of the following two conditions, is satisfied:

- In round r, agent A is at $\texttt{home}(A)$ with $\texttt{tok}(A)$ that is in state `token` and
- during $\mathcal{E}$, agent $\texttt{tok}(A)$ never decides to switch from state `token` to state `searcher` or to declare termination.

Our goal is to first establish that, under this assumption, the lemma holds, and then to demonstrate that property $\Psi(\mathcal{E}, A)$ is necessarily satisfied. Note that property $\Psi(\mathcal{E}, A)$ immediately implies that $\texttt{tok}(A)$ always remains in state `token` at $\texttt{home}(A)$ during $\mathcal{E}$ and, in particular, in the round when $\mathcal{E}$ is completed, if any. Also note that property $\Psi(\mathcal{E}, A)$ and Lemma 4.3 ensure that $\texttt{tok}(A)$ is the only agent in state `token` at $\texttt{home}(A)$ in round r. Thus, by lines 6 and 7 of Algorithm 2, the input given to function $\texttt{EST}^+$ in $\mathcal{E}$ is necessarily well defined and indeed corresponds to $M_{\texttt{tok}(A)}(r)$, which proves the first property of the lemma.

As mentioned in Section 2, an execution of procedure `EST` by an agent Y in a setting where a single token is present in the underlying graph G, at the node from which Y starts this procedure, allows this agent to visit all nodes of G in at most $8n^5$ rounds. Additionally, at the end of the execution, agent Y is back with its token and has constructed a BFS spanning tree of G. Hence, if there is no agent $X \neq \texttt{tok}(A)$ such that $M_X(r) = M_{\texttt{tok}(A)}(r)$, agent A can never confuse its token with another one during $\mathcal{E}$ and thus, in view of property $\Psi(\mathcal{E}, A)$, the second, third and fourth properties of the lemma also hold.

Consequently, suppose that there is at least one agent $X \neq \texttt{tok}(A)$ such that $M_X(r) = M_{\texttt{tok}(A)}(r)$. In this case, agent A may sometimes mistakenly confuse its token $\texttt{tok}(A)$ with another agent. By taking a close look at the description of procedure `EST` (cf. Section 2), on which $\texttt{EST}^+$ is based, we can observe that, during $\mathcal{E}$, agent A never takes into account the presence or absence of $\texttt{tok}(A)$ except in one specific situation. This occurs only when it decides, during the process of a node $c(x)$ of the BFS tree T under construction, corresponding to a node x of G, whether it adds to T a node $c(x')$ corresponding to a neighbor x' of x (i.e, whether a node corresponding to x' has already been added before or not). This means that confusing its token $\texttt{tok}(A)$ with another agent may only occur at times of these decisions. However, while such a confusion can lead agent A, during the process of $c(x)$, to wrongly reject x' if no corresponding node has not yet been added to the tree, it cannot lead the agent to wrongly admit x' i.e., adding to T a node corresponding to x' while such a node has been previously added. Nor can such a confusion prevent the process of $c(x)$ from being completed at the node from which it is started, i.e., x. Also observe that execution $\mathcal{E}$ consists of alternating periods of two different types. The first type corresponds to periods when agent A processes a node of the BFS tree. The second type corresponds to periods when:

- Agent A, having just finished processing a node of T corresponding to some node w in G, moves from w to a node corresponding to an unprocessed node of T, following a path from this tree.

- Or there is no remaining node to process and agent A, having just finished the last process of a node of T corresponding to some node w in G, moves from w to the node corresponding to the root of T.

From these explanations, property $\Psi(\mathcal{E}, A)$ and the fact that the root of the BFS tree T constructed by A corresponds to the node from which $\mathcal{E}$ is started by A (actually it is the first node to be processed), we get two consequences. The first consequence is that the tree constructed by agent A during $\mathcal{E}$ is either a spanning tree of G or a non-empty truncated spanning tree of G (i.e., a non-empty tree that can be obtained from a spanning tree of G by removing one or more of its subtrees). The second consequence is that execution $\mathcal{E}$ is eventually completed at the node from which $\mathcal{E}$ is started by A. In view of these two consequences and property $\Psi(\mathcal{E}, A)$, the third and fourth properties of the lemma hold. Concerning the second property, note that, whether or not there is at least one agent $X \neq \texttt{tok}(A)$ such that $M_X(r) = M_{\texttt{tok}(A)}(r)$, a period of the first (resp. second) type, during $\mathcal{E}$, takes at most $4n^3$ (resp. $2n$) rounds. Moreover, in $\mathcal{E}$, the number of periods of the first type, as well as the number of periods of the second type, is equal to the number of nodes that are added to BFS tree, i.e., at most n (we explained above that the tree spans all or part of G). Thus, $\mathcal{E}$ indeed lasts at most $8n^5$ rounds, which proves the second property. This concludes the proof that the lemma is true if property $\Psi(\mathcal{E}, A)$ is satisfied.

It now remains to show that property $\Psi(\mathcal{E}, A)$ is necessarily satisfied. We will say that a round r^* invalidates $\Psi(\mathcal{E}, A)$ if at least one of the following conditions is met:

- $\mathcal{E}$ starts in round r^* and either agent A is not at $\texttt{home}(A)$ in round r^* or $\texttt{tok}(A)$ is not in state `token` at $\texttt{home}(A)$ in round r^*.
- $\mathcal{E}$ starts no later than round r^* but is not completed by round r^*, and, in this round, $\texttt{tok}(A)$ either decides to switch from state `token` to state `searcher` or to declare termination.

Assume by contradiction that $\Psi(\mathcal{E}, A)$ is not satisfied. This implies that there exists a round r^* that invalidates $\Psi(\mathcal{E}, A)$. Also assume, without loss of generality, that r^* is minimal in the following precise sense: for every execution $\mathcal{E}'$ of function $\texttt{EST}^+$ by any agent A' in state `explorer`, if a round r' invalidates $\Psi(\mathcal{E}', A')$ then $r^* \leq r'$. (Note that $\mathcal{E}'$ (resp. A') is not necessarily different from $\mathcal{E}$ (resp. A).)

First consider the case in which $\mathcal{E}$ starts in round r^* and either agent A is not at $\texttt{home}(A)$ in round r^* or $\texttt{tok}(A)$ is not in state `token` at $\texttt{home}(A)$ in round r^*. Note that the first execution of function $\texttt{EST}^+$ by agent A starts in the round when agent A enters state `explorer` at $\texttt{home}(A)$. Moreover, by the definition of $\texttt{tok}(A)$, $\texttt{tok}(A)$ enters state `token` at $\texttt{home}(A)$ in the round when A enters state `explorer`. Hence, $\mathcal{E}$ cannot correspond to the first execution of function $\texttt{EST}^+$ by agent A. In other words, $\mathcal{E}$ corresponds to the ith execution of function $\texttt{EST}^+$ by agent A, for some $i \geq 2$.

Denote by $\mathcal{E}'$ the $(i-1)$th execution of function $\texttt{EST}^+$ by agent A and denote by r'_1 (resp. r'_2) the round in which $\mathcal{E}'$ starts (resp. is completed). In view of Algorithm $\mathcal{HG}$, agent A can execute function $\texttt{EST}^+$ only when in state `explorer`, which, by Lemma 4.1, it enters at most once. In particular, this means that agent A is in state `explorer` from round r'_1 to (at least) round r^* included. By the definition of $\mathcal{E}'$, we have $r'_1 \leq r'_2 \leq r^*$, and thus, by the minimality of r^*, we get the guarantee that no round invalidates property $\Psi(\mathcal{E}', A)$. From this and the fact, proven above, that the four properties of the lemma hold if $\Psi(\mathcal{E}', A)$ is satisfied, we know that the triple

$(b, \eta, *)$ returned by function $\texttt{EST}^+$, when $\mathcal{E}'$ is completed in round r'_2, is such that $1 \leq \eta \leq n$. By line 15 of Algorithm 2 and the fact that agent A is in state `explorer` from round r'_1 to (at least) round r^* included, we also know that the first element b of the triple is the Boolean value false. Hence, by lines 8 and 9 of Algorithm 2, we know that agent A starts in round r'_2 a waiting period of some positive duration (the duration is well defined and positive, since $1 \leq \eta \leq n$ and $U = 2^{(2^{\lceil \log \log(\mu+1) \rceil})} \geq 2$). This waiting period is necessarily completed in round r^* because agent A is in state `explorer` from round r'_2 to (at least) round r^* included and the next execution of function $\texttt{EST}^+$ by agent A, after $\mathcal{E}'$, is $\mathcal{E}$ that starts in round r^*. This implies that agent A is at $\texttt{home}(A)$ in round r^*, as agent A is already at $\texttt{home}(A)$ in round r'_2 by the fact that no round invalidates property $\Psi(\mathcal{E}', A)$.

Consequently, for the case under analysis, we know that $\texttt{tok}(A)$ is not in state `token` at $\texttt{home}(A)$ in round r^*. This means that, in some round $r'' < r^*$ at node $\texttt{home}(A)$, $\texttt{tok}(A)$ decides to switch from state `token` to state `searcher` or to declare termination. According to the description of states `token` and `explorer`, this is due to the fact that an agent A'' in state explorer has completed an execution $\mathcal{E}''$ of function $\texttt{EST}^+$ at $\texttt{home}(A)$ in some round $r''' \leq r''$ and then has waited $r'' - r'''$ rounds (while staying in state `explorer`) before deciding in round r'' to switch from state `explorer` to state `searcher` or to declare termination. Note that, in view of the minimality of r^*, $\Psi(\mathcal{E}'', A'')$ necessarily holds. Therefore, according to what has been proved in the first part of the proof, we can state that, in round r''', agent A'' is with $\texttt{tok}(A'')$ at $\texttt{home}(A'')$ and $\texttt{tok}(A'')$ is in state `token`. If $r'' = r'''$, then $\texttt{tok}(A)$ and $\texttt{tok}(A'')$ are both in state `token` at $\texttt{home}(A) = \texttt{home}(A'')$, and thus $\texttt{tok}(A) = \texttt{tok}(A'')$, or otherwise we get a contradiction with Lemma 4.3. If $r''' < r''$, we can also get the guarantee that $\texttt{tok}(A) = \texttt{tok}(A'')$ but it requires a bit more explanation. Precisely, if $r''' < r''$, agent A'' is in state `explorer` at $\texttt{home}(A) = \texttt{home}(A'')$ in each round of $[r'''..r'']$. Hence, if no agent transits from state `explorer` to state `searcher` at $\texttt{home}(A'')$ in some round of $[r'''..r'' - 1]$, we know, by the description of state `token`, that $\texttt{tok}(A'')$ is in state `token` at $\texttt{home}(A'')$ in each round of $[r'''..r'']$. As above, this necessarily means that $\texttt{tok}(A)$ and $\texttt{tok}(A'')$ are both in state `token` at $\texttt{home}(A) = \texttt{home}(A'')$, and thus $\texttt{tok}(A) = \texttt{tok}(A'')$ by Lemma 4.3. But what if some agent X transits from state `explorer` to state `searcher` at $\texttt{home}(A'')$ in some round t of $[r'''..r'' - 1]$? In view of the minimality of r^* and lines 7-17 of Algorithm 2, we know that, in round t, agent X is with $\texttt{tok}(X)$ at $\texttt{home}(X) = \texttt{home}(A'')$ and $\texttt{tok}(X)$ is in state `token`. Moreover, assuming, without loss of generality, that t is the first round of $[r'''..r'' - 1]$ in which an agent transits from state `explorer` to state `searcher` at $\texttt{home}(A'')$, $\texttt{tok}(A'')$ is also in state `token` at $\texttt{home}(X) = \texttt{home}(A'')$ in round t, in view of the description of state `token` and the fact that in each round of $[r'''..t]$ $\texttt{tok}(A'')$ is with an agent in state `explorer` that does not declare termination (namely A''). Thus, $\texttt{tok}(X) = \texttt{tok}(A'')$ by Lemma 4.3, which implies that $X = A''$ because, by definition, $\texttt{tok}(X)$ is the token of exactly one agent, namely X. However, by Lemma 4.1, an agent can enter state `explorer` at most once, which means that agent $A'' = X$ cannot be in state `explorer` in round $r'' > t$. This is a contradiction confirming that, for the first case of our analysis, we must have $\texttt{tok}(A) = \texttt{tok}(A'')$.

By definition, the fact that $\texttt{tok}(A) = \texttt{tok}(A'')$ implies that $A = A''$. However, agent A'' decides to leave state `explorer` in round $r'' < r^*$ and agent A is still in this state in round r^*, which means that $A'' \neq A$ in view of Lemma 4.1. This is a contradiction that shows that the first case of our analysis cannot occur.

The second case to consider is when $\mathcal{E}$ starts by round r^* but is not completed by round r^*, and $\texttt{tok}(A)$ decides in this round to switch from state `token` to state `searcher` or to declare termination. However, using similar arguments as those in the analysis of the first case, we again

reach a contradiction.

As a result, round r^* cannot invalidate property $\Psi(\mathcal{E}, A)$. This shows that $\Psi(\mathcal{E}, A)$ is necessarily satisfied and concludes the proof of the lemma. □

The next lemma highlights some relationship between a token and its explorer.

Lemma 4.5 *Agent A transits from state* `explorer` *to state* `searcher` *in a round r if, and only if,* $\texttt{tok}(A)$ *transits from state* `token` *to state* `searcher` *in round r.*

Proof. From the round when an agent has completed its last execution of line 7 of Algorithm 2 to the round when it executes line 17 or 19 of Algorithm 2 (this round included), the agent does not move and is always in state `explorer`. Hence, by Lemma 4.4, we have the following claim.

Claim 4.2 *When an agent B in state* `explorer` *declares termination or switches to state* `searcher`*, it is necessarily at* $\texttt{home}(B)$.

Below, we prove both directions of the equivalence separately. We start with the forward direction of the equivalence. Thus, assume that an agent A transits from state `explorer` to state `searcher` in a round r. We need to show that $\texttt{tok}(A)$ transits from state `token` to state `searcher` in round r. Note that, according to Claim 4.2, agent A is in state `explorer` at $\texttt{home}(A)$ in round r.

There is a case that can be handled easily. It is the one where, in each round, if an agent B is in state `explorer` at $\texttt{home}(B)$, then $\texttt{tok}(B)$ is in state `token` at $\texttt{home}(B)$ in the same round. Indeed, this case immediately implies that $\texttt{tok}(A)$ is in state `token` at $\texttt{home}(A)$ with agent A in round r. Consequently, by the transition rule given in the description of state `token`, we know that agent $\texttt{tok}(A)$ transits from state `token` to state `searcher` in round r.

So, consider the complementary case in which there exists a round where an agent B is in state `explorer` at $\texttt{home}(B)$, but $\texttt{tok}(B)$ is not in state `token` at $\texttt{home}(B)$ in this round. Let r' be the first round in which this occurs. In view of the description of state `token`, it turns out that there is a round $r'' < r'$ in which the following two conditions are satisfied: (1) $\texttt{tok}(B)$ (resp. some agent D) is in state `token` (resp. `explorer`) at $\texttt{home}(B)$ and (2) agent D declares termination or switches to state `searcher`. According to Claim 4.2, we know that agent D is at $\texttt{home}(D)$ in round r''. Therefore $\texttt{home}(B) = \texttt{home}(D)$. Moreover, by the definition of round r' and the fact that $r'' < r'$, we have the guarantee that $\texttt{tok}(D)$ is in state `token` at $\texttt{home}(B) = \texttt{home}(D)$ in round r''. Given that $\texttt{tok}(B)$ and $\texttt{tok}(D)$ are both in state `token` at $\texttt{home}(B) = \texttt{home}(D)$ in round r'', we necessarily have $\texttt{tok}(B) = \texttt{tok}(D)$ by Lemma 4.3. This implies that $B = D$. However, since, in round r'', agent D is in state `explorer` and declares termination or switches to state `searcher`, we know that $B \neq D$ in view of Lemma 4.1 and the fact that an agent declaring termination in round r'' is considered to have no state in any subsequent round. This is a contradiction, which concludes the proof of the forward direction of the equivalence.

Now, consider the backward direction. Assume for the sake of contradiction that $\texttt{tok}(A)$ transits from state `token` to state `searcher` in round r, but agent A does not transit from state `explorer` to state `searcher` in a round r. In view of the description of state `token` and the fact that while in state `token` an agent does not move, we know that, in round r, agent $\texttt{tok}(A)$ is at $\texttt{home}(A)$ with an agent $B \neq A$ that transits from state `explorer` to state `searcher`. According to Claim 4.2, we have $\texttt{home}(A) = \texttt{home}(B)$. Moreover, since B transits from state `explorer` to state `searcher` in round r, we know that, during its last iteration of the repeat loop of Algorithm 2, the execution of

$\texttt{EST}^+$ returns $(\mathit{true}, *, *)$ and line 9 of Algorithm 2 is not executed. Thus, an execution of $\texttt{EST}^+$ is completed by agent B in round r, which means, by Lemma 4.4, that $\texttt{tok}(B)$ is in state `token` at $\texttt{home}(A) = \texttt{home}(B)$ in round r. From this point, we can again reach a contradiction by showing both that $A = B$ and $A \neq B$, following similar arguments to those used in the analysis of the forward direction. This concludes the proof of the backward direction of the equivalence, and thus the proof of the lemma. □

In the sequel, we denote by r_1 the first round in which at least one agent enters state `explorer`. This round, which is involved in the next lemma, exists by Lemmas 4.2 and 4.3.

Lemma 4.6 *Let EXP be the (non-empty) set of agents that enters state* `explorer` *in round r_1. There exists an agent of EXP that never switches from state* `explorer` *to state* `searcher`*.*

Proof. Assume, for the sake of contradiction, that the claim does not hold. For every agent A of EXP, denote by r_A the round in which it decides to switch from state `explorer` to state `searcher` (this round is unique by Lemma 4.1). Let r^* be the latest of these rounds and let A' be an agent of EXP such that for every $A \in EXP$, $M_{r^*}(\texttt{tok}(A))$ is smaller than or equal to $M_{r^*}(\texttt{tok}(A'))$.

By the first property of Lemma 4.4, we know that at the beginning of every execution of function $\texttt{EST}^+$ by agent A' the input of the function is well defined and corresponds to the current memory of $\texttt{tok}(A')$. Consequently, the description of state `explorer` and the fact that A' decides to switch from state `explorer` to state `searcher` in round $r_{A'}$ implies that, in some round $r_1 \leq r \leq r_{A'}$, there is an agent B in state `token` that has either (1) a higher seniority than A' or (2) the same seniority as A' but $M_r(\texttt{tok}(A')) \prec M_r(B)$.

By the definition of EXP, agent A' is among the earliest to enter state `explorer`, which means that the first case cannot occur. Therefore, the second case must occur. As a result, agent B enters state `token` in round r_1 and thus, there is an agent B' such that $B = \texttt{tok}(B')$ (or equivalently such that $B' = \texttt{exp}(B)$) that enters state `explorer` in round r_1. This implies that agent B' belongs to EXP. Moreover, in view of Lemma 4.5 and the fact that B enters state `token` at most once (cf. Lemma 4.1), we know that $r_{B'} \geq r$. Thus, in view of Lemma 2.1, the inequality $M_r(\texttt{tok}(A')) \prec M_r(\texttt{tok}(B'))$ implies that $M_{r^*}(\texttt{tok}(A')) \prec M_{r^*}(\texttt{tok}(B'))$. From this and the fact that $B' \in EXP$, we get a contradiction with the definition of A'. Hence, the claim is necessarily true. □

Until now, we have essentially established properties concerning states `token` and `explorer`. The following lemma introduces some properties related to state `shadow`.

Lemma 4.7 *Let A be an agent in state* `shadow` *in a round r at a node v. Agent A has exactly one guide B in round r. Moreover, in this round, agent B is at node v, can be unambiguously identified by A and is either in state* `searcher` *or* `token`*.*

Proof. Let r' be the round in which agent A enters state `shadow` and let v' be the node occupied by A in round r'. Note that, by Lemma 4.1, r' is unique and agent A is in state `shadow` from round r' to round r included.

First assume that $r = r'$. Under this assumption, we have $v = v'$. According to Algorithm $\mathcal{HG}$, agent A decides to transit to state `shadow`, at node v' in round $r' - 1$, from either (1) state `cruiser` or (2) state `searcher`. In the first case, agent A selects as its guide an agent that enters state `token` at node v' in round r'. In view of Lemma 4.3, there is at most one agent that can be in state

`token` at node v' in round r', which implies that the lemma holds in the first case. In the second case, agent A selects as its guide an agent B that is in state `token` in round $r'-1$ at node v' (cf. the description of state `searcher`) and that is in state `token` or `searcher` at node v' in round r' (cf. the description of state `token`). By Lemma 4.3, there is at most one agent that is in state `token` at node v' in round $r'-1$. Moreover, this means that, among the agents occupying v' in round r', B is the only agent that was in state `token` at this node in the previous round. Hence, the lemma also holds in the second case, and, by extension, when $r = r'$.

In light of the first part of the proof, we know there exists a non negative integer i such that the lemma holds if $r = r' + i$. As a result, to show that the lemma holds when $r > r'$, it is enough to show that the lemma holds if $r = r' + i + 1$. So, we now assume that $r = r' + i + 1$ and we denote by v'' the node occupied by A in round $r' + i$.

At the beginning of the proof, we observed that agent A is in state `shadow` from round r' to round r included. Since $r = r' + i + 1 > r'$, this means that agent A is necessarily in state `shadow` in round $r' + i$. Hence, by the definition of i, we know that, in round $r' + i$, the agent B playing the role of A's guide is either in state `searcher` or `token`, occupies node v'' and can be unambiguously identified by A. Note that if agent B is in state `searcher` in round $r' + i$, then it is in state `searcher` or `shadow` in round $r' + i + 1$. Below, we consider two complementary cases.

First, consider the case where agent B is in state `searcher` in rounds $r' + i$ and $r' + i + 1$. According to the description of state `shadow`, in round $r' + i + 1$, agent B is at node v and remains the unique guide of A. Since, for every integer $N \geq 2$, an execution of procedure `EXPLO`(N) contains no waiting periods, the nodes occupied by B in rounds $r' + i$ and $r' + i + 1$ are different, i.e., $v \neq v''$. Note that, by Lemma 2.2, we have $M_B(r' + i) \neq M_X(r' + i)$ for every agent $X \neq B$ at node v'' in round $r' + i$. Moreover, among the agents at node v in round $r' + i + 1$, only those that were at node v'' in round $r' + i$ enter v by port `port`(v'', v) in round $r' + i + 1$. As a result, agent A can unambiguously identify B in round $r' + i + 1$, which proves the lemma in this case.

Now, consider the case where agent B is either (1) in state `searcher` in round $r' + i$ and in state `shadow` in round $r' + i + 1$ or (2) in state `token` in round $r' + i$. In view of Algorithm $\mathcal{HG}$, in round $r' + i + 1$, A's guide is necessarily an agent that was in state `token` at node v'' in round $r' + i$ (it corresponds to the guide of B if B is in state `shadow` in round $r' + i + 1$, or to B itself otherwise). Hence, using similar arguments to those above (when assuming $r = r'$ and A's guide is in state `token` in round $r - 1$), we can prove that the lemma holds in the second case as well. This completes the proof of the lemma. □

The next lemma pinpoints a specific situation in which all nodes of the underlying graph are visited at least once. In particular, it involves function `EST`$^+$ and the notion of "admitted node" (cf. the description of procedure `EST` in Section 2).

Lemma 4.8 *Let A be an agent in state* `explorer` *that starts an execution of* `EST`$^+$ *in a round r. Let $\mathcal{X}$ be the set of all agents B in state* `explorer` *in round r such that $M_{\texttt{tok}(A)}(r) = M_{\texttt{tok}(B)}(r)$ ($\mathcal{X}$ includes agent A). Every agent of $\mathcal{X}$ starts an execution of* `EST`$^+$ *in round r. Moreover, each node of the underlying graph G is visited and admitted by at least one agent of $\mathcal{X}$ during its execution of* `EST`$^+$ *started in round r.*

Proof. We first want to prove that every agent of $\mathcal{X}$ starts an execution of `EST`$^+$ in round r. By assumption, it is the case for agent A. So, consider any agent $A' \neq A$ from $\mathcal{X}$. By Lemma 4.4, in round r, `tok`(A) is in state `token` and occupies `home`(A) with agent A. Moreover, we know that

while in state `token`, an agent does not move. Hence, since $M_{\texttt{tok}(A)}(r) = M_{\texttt{tok}(A')}(r)$, we know that, in round r, $\texttt{tok}(A')$ is in state `token` at $\texttt{home}(A')$ with an agent Y in state `explorer` that has the same seniority as $\texttt{tok}(A')$ (and thus as A') and that starts an execution of $\texttt{EST}^+$. Note that we necessarily have $\texttt{home}(A') = \texttt{home}(Y)$ in view of Lemma 4.4. Consequently, agents Y and A' enter state `explorer` at node $\texttt{home}(A')$ in the round when agent $\texttt{tok}(A')$ enters state `token` at $\texttt{home}(A')$. By Lemma 4.3, this necessarily means that $Y = A'$, which proves that every agent of $\mathcal{X}$ indeed starts an execution of $\texttt{EST}^+$ in round r.

Now, it remains to show that each node of the underlying graph G is visited and admitted by at least one agent of $\mathcal{X}$ during its execution of $\texttt{EST}^+$ started in round r. As mentioned earlier, every agent of $\mathcal{X}$ is at its home in round r. This implies that during its execution of $\texttt{EST}^+$ started in this round, each agent B of $\mathcal{X}$ constructs a tree T_B rooted at a node corresponding to $\texttt{home}(B)$. Although, in this execution, agent B can sometimes confuse its token with the token of another agent of $\mathcal{X}$, this can never lead agent B to add a "wrong" path in T_B. Precisely, at each stage of the construction of T_B, each path from the root of T_B to any of its node always corresponds to a path from $\texttt{home}(B)$ to some node in G. This particularly implies that when admitting a node x of G (and thus adding a corresponding node to T_B), agent B is necessarily located at node x. Hence, for the purpose of the rest of this proof, it is enough to show that each node of the underlying graph G is admitted by at least one agent of $\mathcal{X}$ during its execution of $\texttt{EST}^+$ started in round r. Assume, by contradiction, it is not the case for some node v of G.

Let $\mathcal{H}$ be the set of homes of all the agents from $\mathcal{X}$ and let $\pi = (p_1, q_1, p_2, q_2, \ldots, p_k, q_k)$ be the lexicographically smallest shortest path from a node of $\mathcal{H}$ to node v. Let C be an agent of $\mathcal{X}$ such that π is a path from $\texttt{home}(C)$ to v (it is not needed here, but one can show that C is unique, using Lemma 4.3). Finally, let $\mathcal{E}_C$ be the execution of $\texttt{EST}^+$ started by agent C in round r. As mentioned at the beginning of this proof, in round r, agent $\texttt{tok}(C)$ is in state `token` and occupies node $\texttt{home}(C)$ with agent C.

During $\mathcal{E}_C$, agent C constructs a tree T_C rooted at a node corresponding to $\texttt{home}(C)$. According to procedure `EST`, when admitting a node x of G, agent C only attaches a corresponding node $c(x)$ to a node of T_C by an edge with two port numbers, so that the the simple path from the root of T_C to $c(x)$ is a path from $\texttt{home}(C)$ to x. Moreover, adding a node to T_C occurs only when admitting a node of G. Hence, since v is not admitted by C in $\mathcal{E}_C$, we know that, at the end of $\mathcal{E}_C$, there exists an integer $0 \leq i < k$ such that $\pi[1, 2i]$ is a simple path in T_C from its root to some of its node, while $\pi[1, 2(i+1)]$ is not (as defined in Section 2, $\pi[1, 0]$ is considered to be the empty path). According to Lemma 4.4, $\mathcal{E}_C$ eventually terminates. Hence, in view of the description of procedure `EST`, agent C eventually moves from $\texttt{home}(C)$ to some node u by following path $\pi[1, 2i]$, which leads to some node $c(u)$ from the root of T_C, and then starts the process of $c(u)$. Denote by Pr this process. During Pr, agent C checks each neighbor of u in order to determine whether admitting the neighbor or not. In particular, it checks $\texttt{succ}(u, p_{i+1})$ which, by the definition of i, is necessarily not admitted during Pr. Let $u' = \texttt{succ}(u, p_{i+1})$. Since, by the first property of Lemma 4.4, the input given to function $\texttt{EST}^+$ in $\mathcal{E}_C$ is $M_{\texttt{tok}(C)}(r)$, it follows from the descriptions of `EST` and $\texttt{EST}^+$ that there exists a simple path φ from node u' satisfying the following two conditions:

- Condition 1. When agent B starts checking u' in Pr, φ is a simple path in T_C from some node w to the root of T_C.
- Condition 2. For some agent Z, agent C encounters an agent $\texttt{tok}(Z)$ in state `token` such that $M_{\texttt{tok}(C)}(r) = M_{\texttt{tok}(Z)}(r)$, right after following path φ from node u' during Pr.

Recall that, in round r, agent $\texttt{tok}(C)$ is in state `token` and occupies node $\texttt{home}(C)$ with agent C. Also recall that, while in state `token`, an agent does not move. Hence, the fact that $M_{\texttt{tok}(C)}(r) = M_{\texttt{tok}(Z)}(r)$ implies that, in round r, $\texttt{tok}(Z)$ is in state `token` at $\texttt{home}(Z)$ with an agent in state `explorer` that has the same seniority as $\texttt{tok}(Z)$ and that starts an execution of $\texttt{EST}^+$. Using the same reasonning as at the beginning of this proof, we can prove that this agent is necessarily Z. As a result, since $M_{\texttt{tok}(A)}(r) = M_{\texttt{tok}(C)}(r) = M_{\texttt{tok}(Z)}(r)$, we have $Z \in \mathcal{X}$ and $\texttt{home}(Z) \in \mathcal{H}$.

According to procedure `EST`, when agent C finishes the process of a node of T_C in $\mathcal{E}_C$, the next node to process, if any, is the one reached by following the lexicographically smallest shortest path from the root of T_C to an unprocessed node of T_C. Moreover, during the process of a node $c(w)$ of T_C, corresponding to a node w of G, the neighbors of w are checked, and thus possibly admitted, following the increasing order of the port numbers at this node. In particular, when admitting a neighbor w' of w during the process of $c(w)$, the agent precisely attaches a node $c(w')$ to $c(w)$ by an edge whose port number at node $c(w)$ is $\texttt{port}(w, w')$. This implies that if a subsequent neighbor w'' of w is admitted during the same process, a corresponding node $c(w'')$ will be attached to $c(w)$ with an edge whose port number at node $c(w)$ is greater than $\texttt{port}(w, w')$. Therefore, when agent B starts checking u' in Pr, we have the guarantee that each simple path from the root of T_C to any of its node is either shorter than path $\pi[1, 2i](\texttt{port}(u, u'), \texttt{port}(u', u))$ or of equal length but lexicographically smaller. Note that $\pi[1, 2i](\texttt{port}(u, u'), \texttt{port}(u', u)) = \pi[1, 2(i + 1)]$ and is a path from $\texttt{home}(\texttt{C})$ to u', by the definition of u and u'. Hence, in view of Condition 1, the reverse of path φ, call it $\bar{\varphi}$, is shorter than $\pi[1, 2(i + 1)]$ or of equal length but lexicographically smaller. By Condition 2 and the fact that $\texttt{tok}(Z)$ stays at $\texttt{home}(Z)$ while in state `token`, it follows that $\bar{\varphi}(p_{i+2}, q_{i+2}, p_{i+3}, q_{i+3}, \ldots, p_k, q_k)$ is a path from a node of $\mathcal{H}$, namely $\texttt{home}(Z)$, to node v that is shorter than π or of the same length but lexicographically smaller. This contradicts the definition of π and thus the existence of v, which terminates the proof. □

Every execution by an agent in state `explorer` of the repeat loop of Algorithm 2 will be viewed as a sequence of consecutive steps $j = 1, 2, 3, 4, \ldots$ where step j is the part of the execution corresponding to the jth iteration of this loop. We will say that two steps, executed by the same agent or not, are identical if they have the same duration and the triple returned by function $\texttt{EST}^+$ in the first step is equal to the triple returned by $\texttt{EST}^+$ in the second step. The notion of step is used in the following four lemmas.

Note that, in view of Algorithm 2, an agent can transit to state `explorer` only from state `cruiser` and, by Lemma 4.1, once an agent leaves state `cruiser`, it cannot return to it. Hence, in the statement of the next lemma, the value τ is well defined.

Lemma 4.9 *Let s be a step completed in some round by an agent A in state* `explorer`*, in which function* $\texttt{EST}^+$ *returns a triple $(b, \eta, trace)$ such that $b =$ false. Let τ be the number of rounds that agent A spends in state* `cruiser`*. The duration T of step s satisfies the following bounds: $(\eta U\tau)^{11\beta} \leq T \leq 2^{13}(nU\tau)^{11\beta}$.*

Proof. Since, during step s, function $\texttt{EST}^+$ returns a triple $(b, \eta, trace)$ such that $b =$ *false*, we know, in view of line 8 of Algorithm 2, that the waiting period at line 9 of Algorithm 2 is executed in step s. Note that $\beta \geq 2, U \geq 2$, $\tau \geq 1$ (the agent spends at least one round in state `cruiser`) and, by Lemma 4.4, $1 \leq \eta \leq n$. This means that the execution time of line 9 of Algorithm 2 in step s can be can be lower bounded (resp. upper bounded) by $(\eta U\tau)^{11\beta}$ (resp. $2^{12}(nU\tau)^{11\beta}$). Moreover, according to Lemma 4.4, an execution of line 7 of Algorithm 2 lasts at most $8n^5$ rounds. Hence,

step s lasts at least (resp. at most) $(\eta U\tau)^{11\beta}$ (resp. $8n^5 + 2^{12}(nU\tau)^{11\beta} \leq 2^{13}(nU\tau)^{11\beta}$) rounds, which proves the lemma. □

Below is arguably one of the most important lemma of this section. It will serve as the main argument to show that gathering is indeed done when a sequence of $U^{25\beta}$ consecutive identical steps is completed by some agent (cf. the proof of Lemma 4.11).

Lemma 4.10 *Let r be the first round, if any, in which a sequence of $U^{25\beta}$ consecutive identical steps is completed by some agent A in state* `explorer`*. No agent has declared termination before round r and, in this round, agent* `tok`(A) *(resp. every agent different from A and* `tok`(A)*) is in state* `token` *(resp.* `shadow`*).*

Proof. Note that an agent in state `explorer` (resp. `token`) declares termination only when it has completed a sequence of $U^{25\beta}$ consecutive identical steps (resp. only when it is with an agent in state `explorer` that declares termination). By the definition of round r, this means there is no round $r' < r$ in which an agent in state `explorer` or `token` declares termination. Moreover, while in state `searcher` or `cruiser`, an agent cannot declare termination and an agent in state `shadow` declares termination only if its guide declares termination. However, by Lemma 4.7, in each round, the guide of an agent in state `shadow` is either in state `searcher` or `token`. Hence, no agent declares termination before round r. From this, Lemma 4.5 and the description of state `token`, it particularly follows that if agent `tok`(A) is not in state `token` in round r, then agent A transits from state `explorer` to state `searcher` before round r, which contradicts Lemma 4.1 and the fact that agent A is in state `explorer` in round r.

Consequently, to prove the lemma, it remains to show that, in round r, every agent, different from A and `tok`(A), is in state `shadow`. In the rest of this proof, given a step s executed by an agent X in state `explorer`, we will denote by r_s (resp. r'_s) the round in which s is started (resp. completed) and by $\mathcal{X}_s$ the set of agents X' in state `explorer` in round r_s such that $M_{\texttt{tok}(X)}(r_s) = M_{\texttt{tok}(X')}(r_s)$ ($\mathcal{X}_s$ includes agent X). We will also say that a sequence of l consecutive (not necessarily identical) steps $S = s_1, s_2, \ldots, s_l$ completed in round r'_{s_l} by an agent in state `explorer` is l-perfect if, the next two properties hold:

1. In step s_l, function $\texttt{EST}^+$ returns a triple whose the first element is the Boolean value false.
2. For every $1 \leq i \leq l$, $\mathcal{X}_{s_i} = \mathcal{X}_1$ and a step identical to s_i is completed by each agent of $\mathcal{X}_{s_i}$ in round r'_{s_i}.

Note that the first property and lines 15 to 17 of Algorithm 2 immediately imply that function $\texttt{EST}^+$ returns a triple whose the first element is the Boolean value false in each step of S. Also note that in every round, there cannot be more than μ non-dormant agents that have the same memory. Hence, since for each agent X there exists at most one agent `exp`(X), it follows that the cardinality of $\mathcal{X}_{s_i}$ is at most μ for every $1 \leq i \leq l$.

We now proceed with the following two claims.

Claim 4.3 *Consider any sequence of $k(\mu+1)$ consecutive steps $s_1, s_2, \ldots, s_{k(\mu+1)}$ completed in round $r'_{s_{k(\mu+1)}}$ by an agent X in state* `explorer`*, where k is a positive integer. There exists a positive integer $j \leq k\mu$ such that the subsequence $s_j, s_{j+1}, \ldots, s_{j+k-1}$ is k-perfect.*

Proof of the claim. Let us first prove that, for every $1 \leq i < k(\mu+1)$, if an agent of $\mathcal{X}_{s_i}$ starts in round r_{s_i} a step that will not be identical to s_i, then this agent cannot belong to $\mathcal{X}_{s_{i+1}}$. Suppose by contradiction this does not hold. Hence, there exist an integer $1 \leq i < k(\mu+1)$ and an agent X' such that X' is in $\mathcal{X}_{s_i} \cap \mathcal{X}_{s_{i+1}}$ while the step started in round r_{s_i} by agent X' is not identical to s_i. By Lemma 2.1, we necessarily have $M_X(r_{s_{i+1}}) \neq M_{X'}(r_{s_{i+1}})$. Moreover, by the definition of $\mathcal{X}_{s_{i+1}}$ and Lemma 4.4, we know that $M_{\texttt{tok}(X)}(r_{s_{i+1}}) = M_{\texttt{tok}(X')}(r_{s_{i+1}})$ and, in round $r_{s_{i+1}}$, agent X is in state `explorer` at $\texttt{home}(X)$ with $\texttt{tok}(X)$ that is in state `token`. Thus, in round $r_{s_{i+1}}$, $\texttt{tok}(X')$ is in state `token` at $\texttt{home}(X')$ with an agent Y in state `explorer` that starts a step and such that $M_X(r_{s_{i+1}}) = M_Y(r_{s_{i+1}})$. This particularly implies that $Y \neq X'$. However from Lemmas 4.3 and 4.4, it follows that $\texttt{tok}(X') = \texttt{tok}(Y)$, which implies that $X' = Y$ as, by definition, $\texttt{tok}(X')$ is the token of exactly one agent, namely X'. This is a contradiction that proves the implication given at the beginning of the proof of the claim.

In light of this and Lemma 2.1, we can state that, for every integer $1 \leq i < k(\mu+1)$, $\mathcal{X}_{s_{i+1}} \subset \mathcal{X}_{s_i}$ if at least one agent of $\mathcal{X}_{s_i}$ starts in round r_{s_i} a step that will not be identical to s_i, $\mathcal{X}_{s_{i+1}} \subseteq \mathcal{X}_{s_i}$ otherwise. Moreover, lines 15-17 of Algorithm 2 and the fact that the steps $s_1, s_2, \ldots, s_{k(\mu+1)}$ are consecutive imply that function $\texttt{EST}^+$ necessarily returns a triple whose the first element is the Boolean value false in each of these steps, except possibly in step $s_{k(\mu+1)}$. Hence, if the claim does not hold, it means that for every integer $0 \leq m \leq \mu - 1$, the cardinality of $\mathcal{X}_{s_{1+km}}$ is smaller than that of $\mathcal{X}_{s_{1+k(m+1)}}$. However, since the cardinality of $\mathcal{X}_{s_1}$ is at most μ (cf. the explanations given just above the statement of the claim), $\mathcal{X}_{s_{1+k\mu}}$ must be empty, which is impossible as $X \in \mathcal{X}_{s_{1+k\mu}}$. Consequently, the claim necessarily holds. ⋆

Claim 4.4 *Let k be a positive integer and let $S = s_1, s_2, \ldots, s_k$ be a k-perfect sequence of steps completed in round r'_{s_k} by an agent X in state* `explorer`. *For every integer $1 \leq i \leq k$, each node is visited by at least one agent of $\mathcal{X}_{s_i}$ in some round $r_{s_i} \leq t \leq r'_{s_i}$. Moreover, if all the steps of S are identical, then the second elements of the triples returned by function $\texttt{EST}^+$ in the steps of S all have the same value η, with $\eta \geq \frac{n}{\mu}$.*

Proof of the claim. From Lemma 4.8 and the fact that S is a k-perfect sequence of steps, we immediately get the guarantee that for every integer $1 \leq i \leq k$, each node is visited by at least one agent of $\mathcal{X}_{s_i}$ during its execution of $\texttt{EST}^+$ within the time interval $[r_{s_i} .. r'_{s_i}]$. Moreover, if all the steps of S are identical, it is obvious that the second elements of the triples returned by function $\texttt{EST}^+$ in the steps of S all have the same value η. Therefore, to prove the claim, it is enough to show that the second element of the triple returned by function EST^+ in s_1, call it η_1, is at least $\frac{n}{\mu}$.

By Lemma 4.8, we know that each node is admitted by at least one agent of $\mathcal{X}_{s_1}$ during its execution of $\texttt{EST}^+$ within the time interval $[r_{s_1} .. r'_{s_1}]$. Each time a node is admitted during an execution of $\texttt{EST}^+$, a new node is added to the BFS tree under construction in this execution. Hence, the sum of the orders of the trees built by the agents of $\mathcal{X}_{s_1}$ during their execution of $\texttt{EST}^+$ in the time interval $[r_{s_1} .. r'_{s_1}]$ is at least n. Note that these trees all have the same order η_1 as a step identical to s_1 is completed by each agent of $\mathcal{X}_{s_i}$ in round r'_{s_1}. Also recall that the cardinality of $\mathcal{X}_{s_1}$ is at most μ (cf. the explanations given just before Claim 4.3). As a result, $\eta_1 \geq \frac{n}{\mu}$, which concludes the proof of this claim. ⋆

In the sequel, we denote by r^* the round in which agent A (resp. $\texttt{tok}(A)$) enters state `explorer` (resp. `token`) at $\texttt{home}(A)$ and we denote by τ the number of rounds spent by agent A in state `cruiser` (we necessarily have $\tau \geq 1$). Since we established above that $\texttt{tok}(A)$ is in state `token` in

round r, we know, in view of the description of state `token` and Lemma 4.1, that agent `tok`(A) is in state `token` at `home`(A) from round r^* to round r included.

Moreover, observe that $U^{25\beta} \geq (U+1)U^{23\beta} \geq (\mu+1)U^{23\beta}$ because $U \geq 2$ and $U \geq \mu$. Also recall that a sequence of $U^{25\beta}$ consecutive identical steps is completed in round r by agent A. Hence, in view of Claim 4.3, we get the guarantee that a $U^{23\beta}$-perfect sequence of identical steps $P = \rho_1, \rho_2, \ldots, \rho_{U^{23\beta}}$ is completed in round $r'_{\rho_{U^{23\beta}}} \leq r$ by agent A (while in state `explorer`).

By Claim 4.4, we know that when ρ_1 is completed, all nodes of the graph have been visited at least once. Therefore, we have the following claim.

Claim 4.5 *Every agent enters state* `cruiser` *by round* r'_{ρ_1}.

With the claim below, we give a round from which we will no longer see an agent in state `cruiser`.

Claim 4.6 *From round* r'_{ρ_2} *onwards, no agent will be in state* `cruiser`.

Proof of the claim. Assume by contradiction that the claim does not hold. By Lemma 4.1 and Claim 4.5, this means there exists an agent X that wakes up in some round $t \leq r'_{\rho_1}$ and that is in state `cruiser` from round t to at least round r'_{ρ_2} included. Note that by Lemma 4.9 and Claim 4.4, $r'_{\rho_2} - r'_{\rho_1}$ is at least equal to $(\frac{n}{\mu}U\tau)^{11\beta} \geq (n\tau)^{11\beta} \geq 1$.

In view of Lemma 2.2 and the transition rules from state `cruiser`, we know that when an agent in state `cruiser` meets an agent in state `cruiser` or `token` in some round, it enters state `token`, `explorer` or `shadow` in the next round. Thus, from round t to round $r'_{\rho_2} - 1$ included, agent X never encounters an agent in state `cruiser` or `token`, otherwise we get a contradiction with the definition of X.

While in state `cruiser`, agent X works in phases $i = 1, 2, \ldots$ where phase i consists of an execution of procedure `EXPLO`(2^i) followed by an execution, for 2^i rounds, of procedure `TZ`(ℓ_X). An entire execution of the ith phase lasts $2^{i\beta} + 2^i$ rounds and, by the properties of procedure `EXPLO`, allows to visit each node of the underlying graph at least once if $2^i \geq n$.

Clearly, when agent A wakes up, agent X cannot have entirely completed the first $\lceil \log n \rceil$ phases of state `cruiser`, because otherwise, by round $r'_{\rho 1}$, X wakes up an agent that immediately enters state `cruiser` (namely A), which is a contradiction. Hence, in view of the duration of a phase in state `cruiser`, we know that agent X cannot have started by round r^* the $(\lceil \log n \rceil + \lceil \log \tau \rceil + 1)$th phase of state `cruiser`. However, $\lceil \log n \rceil + \lceil \log \tau \rceil + 1 \leq 4\lceil \log(n\tau) \rceil + 1$, and an entire execution of the first $4\lceil \log(n\tau) \rceil + 1$ phases of state `cruiser` lasts at most $\sum_{i=1}^{4\lceil \log(n\tau) \rceil + 1}(2^{i\beta} + 2^i) < 2^{5\beta+2}(n\tau)^{4\beta}$ rounds, which is at most $\frac{2^{5\beta+2}}{(n\tau)^{7\beta}}(n\tau)^{11\beta} \leq (n\tau)^{11\beta}$ rounds because $n\tau \geq 2$ and $\beta \geq 2$. Since $(n\tau)^{11\beta} \leq r'_{\rho_2} - r'_{\rho_1}$, it follows that an entire execution of the $(4\lceil \log(n\tau) \rceil + 1)$th phase of state `cruiser` is started (resp. completed) by agent A after round r^* (resp. before round r'_{ρ_2}). This execution allows agent X to visit each node of the graph and to meet an agent in state `token` before round r'_{ρ_2} (as `tok`(A) is in state `token` at `home`(A) from round r^* to round r included). This is a contradiction, which concludes the proof of the claim. ⋆

The next claim gives a restriction on the set of agents that can be in state `explorer` in round $r'_{\rho_{U^{22\beta}}}$ and after.

Claim 4.7 *From round* $r'_{\rho_{U^{22\beta}}}$ *onwards, no agent outside of* $\mathcal{X}_{\rho_{U^{22\beta}}}$ *will be in state* `explorer`.

Proof of the claim. Assume by contradiction that the claim does not hold. This means there exists an agent Y outside of $\mathcal{X}_{\rho_{U^{22\beta}}}$ that is in state `explorer` in round $r'_{\rho_{U^{22\beta}}}$ or after. Since an agent can transit to state `explorer` only from state `cruiser`, Claim 4.6 implies that agent Y is in state `explorer` from round r'_{ρ_2} to round $r'_{\rho_{U^{22\beta}}}$ included, and thus from round r'_{ρ_2} to round $r_{\rho_{U^{22\beta}}}$.

We know, by Lemma 4.9 and Claim 4.4, that $r_{\rho_{U^{22\beta}}} - r'_{\rho_2} \geq (U^{22\beta} - 3)(\frac{n}{\mu}U\tau)^{11\beta} \geq (U^{22\beta} - 3)(n\tau)^{11\beta}$. Moreover, we have $(U^{22\beta} - 3)(n\tau)^{11\beta} \geq U^{11\beta}(nU\tau)^{11\beta} - 3(n\tau)^{11\beta}$ which is at least $U^9 2^{13}(nU\tau)^{11\beta} - 3(n\tau)^{11\beta} \geq (\mu+1)2^{13}(nU\tau)^{11\beta} + 2^{13}(nU\tau)^{11\beta}$. Hence, Claim 4.3 and Lemma 4.9 guarantee that agent Y executes a k-perfect sequence of steps $c_1, c', \ldots, c_k$ for some $k \geq 1$, with $r_{c_1} \geq r'_{\rho_2}$ and $r'_{c_k} \leq r_{\rho_{U^{22\beta}}}$. In view of the definition of such a sequence and Claim 4.4, a step identical to c_1 is completed in round r'_{c_1} by every agent of $\mathcal{X}_{c_1}$, $\texttt{tok}(A)$ is visited by some agent of $\mathcal{X}_{c_1}$ in some round of $[r_{c_1}..r'_{c_1}]$ and function $\texttt{EST}^+$ returns a triple whose the first element is the Boolean value false in each step completed by an agent of $\mathcal{X}_{c_1}$ in round r'_{c_1}. Therefore, by the description of state `explorer` and Lemma 4.4, we have one of the following properties in round r_{c_1}: either (1) the seniority of $\texttt{tok}(Y)$ is greater than that of $\texttt{tok}(A)$ or (2) their seniorities are equal but the memory of $\texttt{tok}(Y)$ is larger than or equal to the memory of $\texttt{tok}(A)$.

Since agent Y is in state `explorer` from round r'_{ρ_2} to round $r'_{\rho_{U^{22\beta}}}$ included, we know that agent $\texttt{tok}(Y)$ is in state `token` at $\texttt{home}(Y)$ from round r_{c_1} to round $r'_{\rho_{U^{22\beta}}}$ included, in view of Lemma 4.5 and the fact that no agent has declared termination before round r. Moreover, $r_{c_1} \leq r_{\rho_{U^{22\beta}}}$ because step c_k is completed in round r'_{c_k} and $r'_{c_k} \leq r_{\rho_{U^{22\beta}}}$. Thus, Claim 4.4 and the fact that $P = \rho_1, \rho_2, \ldots, \rho_{U^{23\beta}}$ is a $U^{23\beta}$-perfect sequence imply that $\texttt{tok}(Y)$ is visited by some agent of $\mathcal{X}_{\rho_{U^{22\beta}}}$ in some round of $[r_{\rho_{U^{22\beta}}}..r'_{\rho_{U^{22\beta}}}]$ and function $\texttt{EST}^+$ returns a triple whose the first element is the Boolean value false in each step completed by an agent of $\mathcal{X}_{\rho_{U^{22\beta}}}$ in round $r'_{\rho_{U^{22\beta}}}$. Therefore, by the description of state `explorer` and Lemma 4.4, we have one of the following properties in round $r_{\rho_{U^{22\beta}}}$: either (1) the seniority of $\texttt{tok}(A)$ is greater than that of $\texttt{tok}(Y)$ or (2) their seniorities are equal but the memory of $\texttt{tok}(A)$ is larger than or equal to the memory of $\texttt{tok}(Y)$. By Lemma 2.1 and the fact that $r_{\rho_{U^{22\beta}}} \geq r_{c_1}$, it means that, in round r_{c_1}, either (1) the seniority of $\texttt{tok}(A)$ is greater than that of $\texttt{tok}(Y)$ or (2) their seniorities are equal but the memory of $\texttt{tok}(A)$ is larger than or equal to the memory of $\texttt{tok}(Y)$. According to what we established at the end of the previous paragraph, it necessarily follows that $M_{\texttt{tok}(A)}(r_{c_1}) = M_{\texttt{tok}(Y)}(r_{c_1})$, and thus $M_{\texttt{tok}(A)}(r_{\rho_3}) = M_{\texttt{tok}(Y)}(r_{\rho_3})$ as $r_{\rho_3} = r'_{\rho_2} \leq r_{c_1}$.

Recall that agent Y is in state `explorer` from round r'_{ρ_2} to round $r'_{\rho_{U^{22\beta}}}$ included, which means it is in this state in round r_{ρ_3}. Consequently, Lemma 4.8 and the fact that $M_{\texttt{tok}(A)}(r_{\rho_3}) = M_{\texttt{tok}(Y)}(r_{\rho_3})$ imply that Y starts an execution of $\texttt{EST}^+$, and thus a step, in round r_{ρ_3}. Hence, $Y \in \mathcal{X}_{\rho_3}$. However, $\mathcal{X}_{\rho_3} = \mathcal{X}_{\rho_{U^{22\beta}}}$ as P is a $U^{23\beta}$-perfect sequence. This contradicts the fact that $Y \notin \mathcal{X}_{\rho_{U^{22\beta}}}$, which proves the claim.

$\star$

Before concluding the proof of the lemma, we still need two more claims that are proven below.

Claim 4.8 *In round $r_{\rho_{U^{23\beta}}}$, no agent is in state* `searcher`.

Proof of the claim. Assume by contradiction that there is an agent X in state `searcher` in round $r_{\rho_{U^{23\beta}}}$. Note that, in view of Algorithm 2, an agent can transit to state `searcher` only from state `explorer` or `token`. Moreover, by Lemma 4.3, if an agent Y enters state `token` in some round then, in the same round, there exists an agent Y' that enters state `explorer` for which agent Y becomes the token. Thus, by Lemma 4.5, we know that an agent transits to state `searcher` in a round t iff an agent in state `explorer` transits to state `searcher` in round t.

By Lemma 4.1 and the fact that P is a $U^{23\beta}$-perfect sequence of steps, we know that the agents of $\mathcal{X}_{\rho_{U^{22\beta}}} = \mathcal{X}_{\rho_{U^{23\beta}}}$ are in state `explorer` from round $r_{\rho_{U^{22\beta}}}$ to round $r_{\rho_{U^{23\beta}}}$ included. Moreover, Claim 4.7 implies that every agent outside of $r_{\rho_{U^{22\beta}}}$ cannot be in state `explorer` in any round of $[r_{\rho_{U^{22\beta}}} .. r_{\rho_{U^{23\beta}}}]$. Hence, in each round of $[r_{\rho_{U^{22\beta}}} .. r_{\rho_{U^{23\beta}}} - 1]$, no agent in state `explorer` transits to state `searcher`. Since we stated above that an agent transits to state `searcher` in a round t iff an agent in state `explorer` transits to state `searcher` in round t, it follows that agent X is in state `searcher` from some round $t' \leq r_{\rho_{U^{22\beta}}}$ to round $r_{\rho_{U^{23\beta}}}$ included.

While in state `searcher`, agent X works in phases $i = 1, 2, \ldots$ where phase i consists of an execution of procedure `EXPLO`(2^i). This is interrupted only when agent X meets an agent in state `token`, in which case it transits to state `shadow` in the round of this meeting. For every positive integer k, an entire execution of the first $\lceil \log k \rceil$ phases lasts at most $\sum_{i=1}^{\lceil \log k \rceil} 2^{i\beta}$, which is upper bounded by $(4k)^\beta$. Note that by Lemma 4.9 and Claim 4.4, $r_{\rho_{U^{23\beta}}} - r_{\rho_{U^{22\beta}}}$ is at least equal to $(\frac{n}{\mu} U\tau)^{11\beta} \geq (n\tau)^{11\beta}$. Hence, since $t' \leq r_{\rho_{U^{22\beta}}}$ and $(n\tau)^{11\beta} > (4n)^\beta$, the first $\lceil \log n \rceil$ phases of state `searcher` are started and completed by agent X during a time interval included in $[t' .. r_{\rho_{U^{23\beta}}} - 1]$. Given that an execution of procedure `EXPLO`(2^i) allows the executing agent to visit at least once every node of the graph if $2^i \geq n$, it follows that agent X visits every node of the graph in the time interval $[t' .. r_{\rho_{U^{23\beta}}} - 1]$.

Recall that r_1 is the first round in which there is at least one agent that enters state `explorer`. By Lemma 4.3, we know that r_1 is also the first round in which there is at least one agent that enters state `token`. As mentioned above, agent X can transit to state `searcher` only from state `explorer` or `token`. Thus, by Lemmas 4.5 and 4.6, we know that $t' \geq r_1$ and there is an agent that is in state `token` from round r_1 to round $r_{\rho_{U^{23\beta}}}$ included. As a result, while in state `searcher`, agent X meets an agent in state `token` in some round t'' of $[t' .. r_{\rho_{U^{23\beta}}} - 1]$ and thus enters state `shadow` in round $t'' + 1$. This contradicts the fact that agent X is in state `searcher` from some round t' to round $r_{\rho_{U^{23\beta}}}$ included. Therefore, the claim holds.

⋆

Claim 4.9 *In round $r_{\rho_{U^{23\beta}}}$, every agent in state* `token` *is the token of an agent of $\mathcal{X}_{\rho_{U^{22\beta}}}$.*

Proof of the claim. Consider any agent X in state `token` in round $r_{\rho_{U^{23\beta}}}$. By definition, we know that when X enters state `token` in some round then, in the same round, there exists an agent called `exp`(X) that enters state `explorer` for which X becomes the token. We also know that states `token` and `explorer` can be left only by declaring termination or by transiting to state `searcher`. Moreover, we proved at the beginning of the proof that no agent declares termination before round r and thus by round $r_{\rho_{U^{23\beta}}}$ as $r_{\rho_{U^{23\beta}}} < r$. Hence, by Lemma 4.5, agent `exp`(X) is in state `explorer` in round $r_{\rho_{U^{23\beta}}}$, which means it belongs to $\mathcal{X}_{\rho_{U^{22\beta}}}$ by Claim 4.7. This concludes the proof of this claim. ⋆

We are now ready to conclude the proof of the lemma. Lemma 4.7 and Claims 4.5, 4.6 and 4.8 imply that, in round $r_{\rho_{U^{23\beta}}}$, every agent that is neither in state `explorer` nor `token` is in state `shadow` and shares its current node with an agent in state `token` that is its guide. Claims 4.7 and 4.9, together with the fact that P is a $U^{23\beta}$-perfect sequence of steps, imply that, in round $r_{\rho_{U^{23\beta}}}$, every agent in state `explorer` (resp. `token`) is an agent of $\mathcal{X}_{\rho_{U^{22\beta}}} = \mathcal{X}_{\rho_{U^{23\beta}}}$ (resp. is the token of an agent of $\mathcal{X}_{\rho_{U^{22\beta}}} = \mathcal{X}_{\rho_{U^{23\beta}}}$). Hence, in view of Lemma 4.4, we know that in round $r_{\rho_{U^{23\beta}}}$, every agent is at a node occupied by the token of an agent of $\mathcal{X}_{\rho_{U^{23\beta}}}$. By the definition of $\mathcal{X}_{\rho_{U^{23\beta}}}$, the tokens of the agents of $\mathcal{X}_{\rho_{U^{23\beta}}}$ all have the same memory in round $r_{\rho_{U^{23\beta}}}$, which means that the

symmetry index of the team is at least equal to the number of nodes occupied by these tokens in round $r_{\rho_{U^{23\beta}}}$. However, since the team is gatherable, Theorem 3.1 implies that the symmetry index of the team is 1. Consequently all the agents occupies the same node in round $r_{\rho_{U^{23\beta}}}$ and, in view of Lemma 4.3, only agent $\texttt{tok}(A)$ is in state `token` in this round. By Lemma 4.5, the description of state `token` and the fact that no agent declares termination before round r, we know that, in round $r_{\rho_{U^{23\beta}}}$, an agent X cannot be in state `explorer` if $\texttt{tok}(X)$ is not in state `token` in this round. This means that every agent different from A is not in state `explorer` in round $r_{\rho_{U^{23\beta}}}$.

From the above explanations, it follows that, in round $r_{\rho_{U^{23\beta}}}$, every agent different from A and $\texttt{tok}(A)$ is in state `shadow` and has $\texttt{tok}(A)$ as its guide. Since $\texttt{tok}(A)$ is in state `token` from round $r_{\rho_{U^{23\beta}}}$ to round r included, we know by the description of state `shadow` and Lemma 4.7 that every agent different from A and $\texttt{tok}(A)$ is still in state `shadow` in round r. This completes the proof of the lemma. □

With the last two lemmas below, we are ready to wrap up the proof. Specifically, the former states that the problem is solved when a sequence of $U^{25\beta}$ consecutive identical steps is completed by an agent in state `explorer`, while the latter guarantees that such an event occurs after at most a polynomial time in n and $|\lambda|$ from r_0.

Lemma 4.11 *When a sequence of $U^{25\beta}$ consecutive identical steps is completed in a round r by an agent in state* `explorer`*, then, in this round, all agents are together and declare termination.*

Proof. Without loss of generality, we assume that r is the first round when a sequence of $U^{25\beta}$ consecutive identical steps is completed by an agent A in state `explorer`. We know by lines 15-17 of Algorithm 2, that in each of these steps, function $\texttt{EST}^+$ returns a triple whose the first element is the Boolean value false. Given that a step is made of an execution of function $\texttt{EST}^+$ followed by a possible waiting period, we also know by the third property of Lemma 4.4 that agent A is at $\texttt{home}(A)$ in round r. Thus, by lines 15-19 of Algorithm 2, agent A declares termination at $\texttt{home}(A)$ in round r.

In view of Lemma 4.10 and the fact that while in state `token` an agent remains idle, $\texttt{tok}(A)$ is in state `token` at $\texttt{home}(A)$ in round r. Furthermore, for the agents, other than A and $\texttt{tok}(A)$, we know by Lemma 4.10 that they are in state `shadow` in round r, which means that agent A is the only agent in state `explorer` in round r. Thus, by the description of state `token` and the fact that A declares termination at $\texttt{home}(A)$ in round r, $\texttt{tok}(A)$ also declares termination in this round.

We mentioned just above that all the agents, except A and $\texttt{tok}(A)$, are in state `shadow` in round r. By Lemma 4.7 and the fact that agent $\texttt{tok}(A)$ is the only agent in state `token` in round r, we also know that these agents occupy in round r the same node as their guide i.e., $\texttt{tok}(A)$. In view of the description of state `shadow`, it follows that, in round r, these agents in state `shadow` are all at $\texttt{home}(A)$ and declare termination, as their guide does so in the same round.

Consequently, we proved that all agents, including A and $\texttt{tok}(A)$, are at $\texttt{home}(A)$ in round r and declare termination in this round, which ends the proof of the lemma. □

Lemma 4.12 *A sequence of $U^{25\beta}$ consecutive identical steps is completed by an agent in state* `explorer` *within at most a polynomial number of rounds in n and $|\lambda|$ after round r_0.*

Proof. Recall that r_1 denotes the first round in which at least one agent enters state `explorer`. By Lemmas 4.2 and 4.3, the difference $r_1 - r_0$ is at most some polynomial $P(n, |\lambda|)$.

Let EXP be the non-empty set of agents that enters state `explorer` in round r_1. By Lemma 4.6, we know there exists an agent A^* of EXP that never switches from state `explorer` to state `searcher`.

In view of lines 6-17 of Algorithm 2, this implies that each execution of $\texttt{EST}^+$ by agent A^* returns a triple whose the first element is the Boolean value false, and is immediately followed by an execution of line 9 of Algorithm 2. Furthermore, given two triples returned by any two executions of $\texttt{EST}^+$ by agent A^*, if their third elements (which correspond to the traces of the execution of $\texttt{EST}^+$) are identical, then their second elements (which correspond to the orders of the BFS trees constructed during the executions of $\texttt{EST}^+$) are also identical and, in both cases, the waiting period at line 9 of Algorithm 2 that immediately follows the execution of $\texttt{EST}^+$ lasts the same amount of time. From this, it follows that a sequence of $U^{25\beta}$ consecutive steps executed by agent A^* are identical if, and only if, the third elements of the triples returned by the executions of $\texttt{EST}^+$ within these steps are all identical. In view of lines 8-15 of Algorithm 2, it also follows that agent A^* stops executing the repeat loop of this algorithm when, and only when, it has completed a sequence of $U^{25\beta}$ consecutive identical steps.

Consider any sequence of $U^{25\beta}$ consecutive steps that is started (resp. completed) by agent A^* in some round r (resp. r'). If the $U^{25\beta}$ steps of this sequence are not all identical then, according to the explanations given above, there must exist two triples, among those returned during the execution of the sequence, whose the third elements differ. In view of function $\texttt{EST}^+$, this can happen only if there is a round $r \leq r'' \leq r'$ such that at least one of the following conditions is satisfied:

1. $M_{r''}(\texttt{tok}(A)) = M_{r''}(B)$ and $M_{r''-1}(\texttt{tok}(A)) \neq M_{r''-1}(B)$ for some agent B
2. $M_{r''}(\texttt{tok}(A)) \neq M_{r''}(B)$ and $M_{r''-1}(\texttt{tok}(A)) = M_{r''-1}(B)$ for some agent B

However, by Lemma 2.1, we know that the first condition cannot actually be satisfied. By the same lemma and the fact that there are at most n agents, we also know that the number of rounds that can fulfilled the second condition is at most n. Moreover, $U = 2^{(2^{\lceil \log\log(\mu+1)\rceil})} \leq (n+1)^2$ and, by Lemma 4.9, each execution of a step by agent A^* lasts at most $2^{13}(nUP(n,|\lambda|))^{11\beta}$ rounds (we can indeed apply Lemma 4.9 because, as mentioned above, each execution of $\texttt{EST}^+$ by A^* returns $(\mathit{false}, *, *)$). This implies that a sequence of $U^{25\beta}$ consecutive steps executed by agent A^* lasts a number of rounds that is at most some polynomial $Q(n,|\lambda|)$ and agent A^* can execute at most n pairwise disjoint such sequences, in which all the $U^{25\beta}$ steps are not identical. Hence, since agent A^* stops executing the repeat loop of Algorithm 2 when, and only when, it has completed a sequence of $U^{25\beta}$ consecutive identical steps, it must have completed such a sequence by round $r_1 + (n+1)Q(n,|\lambda|) \leq P(n,|\lambda|) + (n+1)Q(n,|\lambda|)$, which proves the lemma.

□

Lemmas 4.11 and 4.12 directly imply the next theorem.

Theorem 4.1 *Let $\mathcal{O}$ be the oracle that, for each team L with multiplicity index μ, associates the binary representation of $\lceil \log\log(\mu+1)\rceil$ to each initial setting $IS(L)$ (the size of $\mathcal{K}$ is therefore in $O(\log\log\log\mu)$). Algorithm $\mathcal{HG}$ is an $\mathcal{O}$-universal gathering algorithm and its time complexity with $\mathcal{O}$ is polynomial in n and $|\lambda|$.*

Note that any team with multiplicity index 1 also has symmetry index 1, and is thus gatherable by Theorem 3.1. As a result, from Algorithm $\mathcal{HG}$, we can derive another algorithm, call it $\mathcal{HG}^+$, that

allows to gather any team of multiplicity index 1 without requiring any initial common knowledge. Specifically, this can be achieved by making the following two modifications to Algorithm 2:

1. Assign value 2 instead of $2^{(2^x)}$ to variable U at line 3 ($2^{(2^{\lceil \log\log(\mu+1)\rceil})} = 2$ when $\mu = 1$).
2. Remove line 2.

By definition, a team for which the multiplicity index is 1 is a team in which all agents' labels are pairwise distinct. Hence, in view of Theorem 4.1, we obtain the corollary below.

Corollary 4.1 *Without using any common knowledge $\mathcal{K}$, Algorithm $\mathcal{HG}^+$ allows to gather, in polynomial time in n and $|\lambda|$, any team in which all agents' labels are pairwise distinct.*

5 Time Complexity vs Size of $\mathcal{K}$: A Negative Result

In this section, we establish a negative result on the size of $\mathcal{K} = \mathcal{O}(IS(L))$ required for an algorithm to gather any gatherable team L in a time at most polynomial in n and $|\lambda|$, regardless of the adversary's choices. Essentially, it states that to gather any gatherable team in polynomial time in n and $|\lambda|$, the size of advice used by Algorithm $\mathcal{HG}$ presented in the previous section is almost optimal. Its proof follows an approach similar to [9].

Theorem 5.1 *Let $\mathcal{O}$ be an oracle such that the size of $\mathcal{K} = \mathcal{O}(IS(L))$, for any gatherable team L and any initial setting $IS(L)$, is in $o(\log\log\log\mu)$, where μ is the multiplicity index of L. There exists no $\mathcal{O}$-universal gathering algorithm whose time complexity with $\mathcal{O}$ is polynomial in n and $|\lambda|$.*

Proof. Let $\mathcal{O}$ be an oracle that gives an advice of size $f(\mu)$ in $o(\log\log\log\mu)$ for any initial setting $IS(L)$. Since $f(\mu) = o(\log\log\log\mu)$, for every $\varepsilon > 0$, there exists a constant μ_0 such that for every $\mu \geq \mu_0, f(\mu) \leq \varepsilon.\log\log\log\mu$. Hence, we may assume that for any initial setting $IS(L)$ with a sufficientally large μ, we can fix a constant $c > 0$ to be defined later such that the size of the advice is less than $\lceil c.\log\log\log(\mu)\rceil$. Let assume for the sake of contradiction that there exists an $\mathcal{O}$-universal gathering algorithm $\mathcal{A}$ whose time complexity with oracle $\mathcal{O}$ is less than $b.n^d|\lambda|^d$ for some integer constants $b, d \geq 1$.

We define a set of initial settings $\mathcal{I} = \{IS(L_i) \mid i \in \mathbb{N}^*\}$. Each initial setting $IS(L_i) \in \mathcal{I}$ corresponds to some ring R_i of size $n_i \geq 3$, with nodes $x_0, x_1, \ldots, x_{n_i-1}$ arranged in clockwise order with exactly one agent on each node. Each node x_j has two ports: port 0 leads to x_{j+1}, and port 1 leads to x_{j-1} (indices are taken modulo n_i). Each initial setting $IS(L_i)$ is represented by a word W_i of length n_i on the alphabet of labels such that the j-th letter of W_i corresponds to the label of the agent placed in node x_{j-1} of R_i. We define words W_i recursively. We set $W_1 = 112$ and so $IS(L_1)$ corresponds to the ring of size 3 with the first two nodes each containing an agent with label 1 and the third node containing an agent with label 2. Let $P(n) = 4.b.n^{d-1}$. For $i \in \mathbb{N}^*$, we set $W_{i+1} = (W_i)^{P(|W_i|)}.\ell$ with ℓ the label $i+2$. That is, W_{i+1} is constructed by taking $P(|W_i|)$ copies of W_i and adding the label $i+2$. First, we show some properties on each L_i and W_i.

Claim 5.1 *The following properties are true for all $i \in \mathbb{N}^*$:*

1. *L_i is gatherable,*
2. *$|W_{i+1}| = 4b|W_i|^d + 1$ and*

3. the length of the binary representation of the smallest label in L_i is 1.

Proof of the claim.

1. For all $i \in \mathbb{N}^*$, L_i has a symmetry index equal to 1 since it contains exactly one instance of label $i+1$. It follows by Theorem 3.1 that L_i is gatherable.
2. Observe that $|W_{i+1}| = P(|W_i|)|W_i| + 1 = 4b.|W_i|^{d-1}|W_i| + 1 = 4b.|W_i|^d + 1$.
3. For all $i \in \mathbb{N}^*$, the smallest label of L_i is 1 and so is the length of its binary representation.

$\star$

Let $i \in \mathbb{N}^*$. By Claim 5.1, we have $|W_{i+1}| = 4b|W_i|^d + 1$. Hence, we have $|W_{i+1}| \leq |W_i|^{d+b+3}$. Now, since $|W_1| = 3$, we have $|W_i| \leq 3^{(d+b+3)^i}$ and so $\mu_i \leq 3^{(d+b+3)^i}$, where μ_i is the multiplicity index of L_i. Hence, it follows that there exists a positive constant c' such that we have $\log\log \mu_i \leq c'.i$ for all $i \geq 1$. We fix the constant c such that the number of binary words of length at most $\lceil c \log(m) \rceil$ is less than $\lfloor \frac{m}{c'} \rfloor$ for a sufficiently large positive integer m. Let $\mathcal{I}_m = \{IS(L_i) \mid 1 \leq i \leq \lfloor \frac{m}{c'} \rfloor\}$. Observe that for all $IS(L_i) \in \mathcal{I}_m$, $\log\log \mu_i \leq c'.i \leq c' \lfloor \frac{m}{c'} \rfloor \leq m$. Hence, for $IS(L_i) \in \mathcal{I}_m$, the size of the advice $\mathcal{K}$ given by $\mathcal{O}$ is at most $\lceil c. \log\log\log(\mu_i) \rceil \leq \lceil c. \log m \rceil$. By the choice of c, the number of different advices given by $\mathcal{O}$ to all $IS(L_i) \in \mathcal{I}_m$ is less than $\lfloor \frac{m}{c'} \rfloor$. Since $|\mathcal{I}_m| = \lfloor \frac{m}{c'} \rfloor$, it follows that there are two initial settings $IS(L_s)$ and $IS(L_q)$ in $\mathcal{I}_m$ that both receive the same advice. The remainder of the proof then consists in showing that the fact that the two initial settings $IS(L_s)$ and $IS(L_q)$ receive the same advice makes Algorithm $\mathcal{A}$ fails. To that goal, we can assume w.l.o.g. that $s < q$. Observe that, by construction, W_q contains occurrences of word $(W_s)^{P(|W_s|)}$; let W be the first of these occurrences. Let z_i be the node of $IS(L_s)$ corresponding to the $(i+1)$-th letter of the word W_s and let y_j be the node of $IS(L_q)$ corresponding to the $(j+1)$-th letter of the word W. Let Z_i^r (resp. Y_j^r) denote the set of agents located at node z_i (resp. y_j) in round r. In order to show a contradiction, we show the following claim.

Claim 5.2 *The following property is true for all round r such that $1 \leq r \leq |W|/4$.*

$H(r)$: For all $0 \leq i \leq |W_s| - 1$ and all $r - 1 \leq j \leq |W| - r$ such that $i \equiv j \mod |W_s|$, there exists a bijection $f_{i,j}^r : Z_i^r \to Y_j^r$ such that for every agent $Z \in Z_i^r$, the memory state of Z in round r equals that of $f_{i,j}^r(Z)$.

Proof of the claim. We will show $H(r)$ by induction on r. First, we show the base case $H(1)$ of the induction. Let $i \in \{0, \ldots, |W_s| - 1\}$ and $j \in \{0, \ldots, |W| - 1\}$ such that $i \equiv j \mod |W_s|$. The nodes z_i and y_j both starts with exactly one agent having the $(i+1)$-th letter of W_s as label. Since at round 1 the memory of an agent only contains its label, the singleton sets Z_i^1 and Y_j^1 contain agents with identical memory. The bijection $f_{i,j}^1$ is trivial in this case, so $H(1)$ holds.

Assume $H(r)$ holds for some round r such that $1 \leq r \leq |W|/4 - 1$. We show that $H(r+1)$ also holds. By the inductive hypothesis $H(r)$, for all $i \in \{0, \ldots, |W_s| - 1\}$ and all $j \in \{r-1, \ldots, |W| - r\}$ such that $i \equiv j \mod |W_s|$, there exists a bijection $f_{i,j}^r : Z_i^r \to Y_j^r$ such that for every agent $Z \in Z_i^r$, the memory state of Z in round r equals that of $f_{i,j}^r(Z)$. Now, $\{r-1, \ldots, |W| - r\} = \{r-1\} \cup \{(r+1) - 1, \ldots, |W| - (r+1)\} \cup \{|W| - r\}$. So, for all $k \in \{(r+1) - 1, \ldots, |W| - (r+1)\}$, $\forall Y \in Y_k^{r+1}$, $Y \in \cup_{j \in \{r-1, \ldots, |W|-r\}} Y_j^r$. Since the algorithm is deterministic, agents with the same advice and the same memory take the same action. Thus, from the bijections $f_{i,j}^r$ (with $i \in \{0, \ldots, |W_s| - 1\}$, $j \in \{r-1, \ldots, |W| - r\}$, and $i \equiv j \mod |W_s|$), we can deduce that for all $i \in \{0, \ldots, |W_s| - 1\}$

and all $k \in \{(r+1)-1, \ldots, |W|-(r+1)\}$ such that $i \equiv k \mod |W_s|$, there exists a bijection $f_{i,k}^{r+1} : Z_i^{r+1} \to Y_j^{r+1}$ such that for every agent $Z \in Z_i^{r+1}$, the memory state of Z in round $r+1$ equals that of $f_{i,k}^{r+1}(Z)$, and we are done.

$\star$

Let λ_s be the smallest label of L_s. There exists $i \in \{0, \ldots, |W_s|-1\}$ such that agents in $IS(L_s)$ must declare termination at node z_i at some round $r \leq |W|/4$. Indeed, algorithm $\mathcal{A}$ applied on $IS(L_s)$ has time complexity less than $b.(n_s)^d.(|\lambda_s|)^d = b.(n_s)^d = n_s.P(n_s)/4 = |W|/4$ as $|\lambda_s| = 1$ by Claim 5.1. Since $|W| - 2r + 1 \geq \frac{|W|}{2} \geq \frac{P(|W_s|)|W_s|}{2} \geq 2|W_s|$, there are two distinct integers j and j' such that $r-1 \leq j, j' \leq |W|-r$, $i \equiv j \mod |W_s|$ and $i \equiv j' \mod |W_s|$. By Claim 5.2, there exist two bijections $f_{i,j}^r$ and $f_{i,j'}^r$ such that for every $Z \in Z_i^r$, the memory state of Z at round r equals that of $f_{i,j}^r(Z)$ and $f_{i,j'}^r(Z)$. It follows that $f_{i,j}^r(Z)$ and $f_{i,j'}^r(Z)$ must also declare termination at round r when algorithm $\mathcal{A}$ applied on $IS(L_q)$. Since $f_{i,j}^r(Z)$ and $f_{i,j'}^r(Z)$ are on distinct nodes y_j and $y_{j'}$, we obtain a contradiction with the fact that $\mathcal{A}$ is an $\mathcal{O}$-universal gathering algorithm. $\square$

6 Conclusion

Until now, the problem of gathering had been studied at two extremes: either when agents are equipped with pairwise distinct labels, or when they are entirely anonymous. Our paper bridged this gap by considering the more general context of labeled agents with possible homonyms. In this context, we fully characterized the teams that are gatherable and we analyzed the question of whether a single algorithm can gather all of them in time polynomial in n and $|\lambda|$. This question was natural given the well-known poly$(n, |\lambda|)$-time lower bound of [13] for gathering just two agents with distinct labels without initial common knowledge. On the negative side, we showed that no algorithm can gather every gatherable team in poly$(n, |\lambda|)$ time, even if they initially share $o(\log\log\log\mu)$ bits of common knowledge, where μ is the multiplicity index of the team. On the positive side, we designed an algorithm that gathers all of them in poly$(n, |\lambda|)$ time using only $O(\log\log\log\mu)$ bits of common knowledge.

As a general open question, it is legitimate to ask what happens to complexity, or even simply feasibility, when we vary the amount of shared knowledge across the full spectrum that ranges from no knowledge to complete knowledge of the initial setting. Actually, using the same arguments as in the proof of Theorem 5.1, we can show that no algorithm can gather all gatherable teams if the agents initially share no knowledge about their initial setting.[6] But what about the rest of the spectrum? Are there sharp threshold effects or only smooth trade-offs? This is an interesting but challenging avenue for future work.

In the classical scenario where all agent labels are pairwise distinct, we obtained the first poly$(n, |\lambda|)$-time algorithm that requires no common knowledge to gather teams of arbitrary size. This result closed a fundamental open problem when agents traverse edges in synchronous rounds. Intriguingly, a similar problem remains open when the assumption of synchrony is removed, and agents instead traverse edges at speeds that an adversary may arbitrarily vary over time (as in

[6] For any algorithm $\mathcal{A}$ supposedly achieving this, one can construct two initial settings $IS(L)$ and $IS(L')$, around two gatherable teams L and L', that would necessarily require two initial different pieces of advice from an oracle to be solved by $\mathcal{A}$. Precisely, this can be done using a design principle similar to the one employed to construct the set of initial settings $\mathcal{I}$ in the proof of Theorem 5.1. This yields a direct contradiction with the fact that $\mathcal{A}$ can work without any common knowledge.

[11, 16, 26]). It is then no longer a matter of determining complexity in terms of time (since time is entirely controlled by an adversary), but rather in terms of cost, that is, the total number of edge traversals performed by all agents in order to gather. Specifically, with no initial common knowledge, the cost of asynchronous gathering is lower-bounded by a polynomial in n and $|\lambda|$ (cf. [26]), but no algorithm guaranteeing such a polynomial cost complexity under the same knowledge constraint is known, except in the special case of two agents [16]. As in the synchronous model, the challenge of generalization is closely tied to termination detection. However, the tools we used to tackle this rely on synchronicity, especially waiting periods, which lose their meaning when agents move asynchronously. Thus, developing new techniques to address this problem in this hasher environment constitutes another interesting direction for future research.

References

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

[2] Steve Alpern. Rendezvous search: A personal perspective. *Operations Research*, 50(5):772–795, 2002.

[3] Steve Alpern. *The theory of search games and rendezvous*. International Series in Operations Research and Management Science, Kluwer Academic Publishers, 2003.

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

[5] Sébastien Bouchard, Yoann Dieudonné, and Anissa Lamani. Byzantine gathering in polynomial time. *Distributed Comput.*, 35(3):235–263, 2022.

[6] Sébastien Bouchard, Yoann Dieudonné, and Andrzej Pelc. Want to gather? no need to chatter! *SIAM J. Comput.*, 52(2):358–411, 2023.

[7] Sébastien Bouchard, Yoann Dieudonné, Andrzej Pelc, and Franck Petit. On deterministic rendezvous at a node of agents with arbitrary velocities. *Inf. Process. Lett.*, 133:39–43, 2018.

[8] 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.

[9] Jérémie Chalopin, Emmanuel Godard, and Yves Métivier. Election in partially anonymous networks with arbitrary knowledge in message passing systems. *Distributed Comput.*, 25(4):297–311, 2012.

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

[11] Jurek Czyzowicz, Andrzej Pelc, and Arnaud Labourel. How to meet asynchronously (almost) everywhere. *ACM Trans. Algorithms*, 8(4):37:1–37:14, 2012.

[12] Dariusz Dereniowski and Andrzej Pelc. Topology recognition and leader election in colored networks. *Theoretical Computer Science*, 621:92–102, 2016.

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

[14] Yoann Dieudonné and Andrzej Pelc. Anonymous meeting in networks. *Algorithmica*, 74(2):908–946, 2016.

[15] 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.

[16] Yoann Dieudonné, Andrzej Pelc, and Vincent Villain. How to meet asynchronously at polynomial cost. *SIAM J. Comput.*, 44(3):844–867, 2015.

[17] Paola Flocchini, Evangelos Kranakis, Danny Krizanc, Flaminia L. Luccio, and Nicola Santoro. Sorting and election in anonymous asynchronous rings. *Journal of Parallel and Distributed Computing*, 64(2):254–265, 2004.

[18] Pierre Fraigniaud, David Ilcinkas, and Andrzej Pelc. Oracle size: a new measure of difficulty for communication tasks. In Eric Ruppert and Dahlia Malkhi, editors, *Proceedings of the Twenty-Fifth Annual ACM Symposium on Principles of Distributed Computing, PODC 2006, Denver, CO, USA, July 23-26, 2006*, pages 179–187. ACM, 2006.

[19] Christian Glacet, Avery Miller, and Andrzej Pelc. Time vs. information tradeoffs for leader election in anonymous trees. *ACM Trans. Algorithms*, 13(3):31:1–31:41, 2017.

[20] Barun Gorain, Avery Miller, and Andrzej Pelc. Four shades of deterministic leader election in anonymous networks. *Distributed Comput.*, 36(4):419–449, 2023.

[21] Jion Hirose, Junya Nakamura, Fukuhito Ooshita, and Michiko Inoue. Weakly byzantine gathering with a strong team. *IEICE Trans. Inf. Syst.*, 105-D(3):541–555, 2022.

[22] Jion Hirose, Junya Nakamura, Fukuhito Ooshita, and Michiko Inoue. Fast gathering despite a linear number of weakly byzantine agents†. *Concurr. Comput. Pract. Exp.*, 36(14), 2024.

[23] 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.*, 41(1):26–46, 2012.

[24] Dariusz R. Kowalski and Adam Malinowski. How to meet in anonymous network. *Theor. Comput. Sci.*, 399(1-2):141–156, 2008.

[25] Evangelos Kranakis, Danny Krizanc, and Sergio Rajsbaum. Mobile agent rendezvous: A survey. In *Structural Information and Communication Complexity, 13th International Colloquium, SIROCCO 2006, Chester, UK, July 2-5, 2006, Proceedings*, pages 1–9, 2006.

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

[27] Avery Miller and Andrzej Pelc. Fast rendezvous with advice. *Theor. Comput. Sci.*, 608:190–198, 2015.

[28] Anisur Rahaman Molla, Kaushik Mondal, and William K. Moses Jr. Fast deterministic gathering with detection on arbitrary graphs: The power of many robots. In *IEEE International Parallel and Distributed Processing Symposium, IPDPS 2023, St. Petersburg, FL, USA, May 15-19, 2023*, pages 47–57. IEEE, 2023.

[29] Nicolas Nisse and David Soguet. Graph searching with advice. *Theor. Comput. Sci.*, 410(14):1307–1318, 2009.

[30] Andrzej Pelc. Deterministic gathering with crash faults. *Networks*, 72(2):182–199, 2018.

[31] 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.

[32] Andrzej Pelc and Ram Narayan Yadav. Using time to break symmetry: Universal deterministic anonymous rendezvous. In Christian Scheideler and Petra Berenbrink, editors, *The 31st ACM on Symposium on Parallelism in Algorithms and Architectures, SPAA 2019, Phoenix, AZ, USA, June 22-24, 2019*, pages 85–92. ACM, 2019.

[33] Omer Reingold. Undirected connectivity in log-space. *J. ACM*, 55(4):17:1–17:24, 2008.

[34] Ashish Saxena and Kaushik Mondal. A further study on weak byzantine gathering of mobile agents. *Theor. Comput. Sci.*, 1022:114892, 2024.

[35] Thomas Schelling. *The Strategy of Conflict*. Oxford University Press, Oxford, 1960.

[36] Amnon Ta-Shma and Uri Zwick. Deterministic rendezvous, treasure hunts, and strongly universal exploration sequences. *ACM Trans. Algorithms*, 10(3):12:1–12:15, 2014.

[37] Masafumi Yamashita and Tiko Kameda. Electing a leader when processor identity numbers are not distinct (extended abstract). In *Proceedings of the 3rd International Workshop on Distributed Algorithms*, page 303–314, Berlin, Heidelberg, 1989. Springer-Verlag.