Gathering Teams of Deterministic Finite Automata on a Line

Younan Gao
Department of Computer Science, University of Milano-Bicocca, Italy

Andrzej Pelc
Université du Québec en Outaouais, Gatineau, Canada

Abstract

Several mobile agents, modelled as deterministic finite automata, navigate in an infinite line in synchronous rounds. All agents start in the same round. In each round, an agent can move to one of the two neighboring nodes, or stay idle. Agents have distinct labels which are integers from the set $\{1, \ldots, L\}$. They start in teams, each of which consists of x agents, for some fixed integer x. Agents in a team have the same starting node. The adversary decides the compositions of teams, and their starting nodes. Whenever an agent enters a node, it sees the entry port number and the states of all collocated agents; this information forms the input of the agent on the basis of which it transits to the next state and decides the current action. The aim is for all agents to gather at the same node and stop. Gathering is feasible, if this task can be accomplished for any decisions of the adversary, and its time is the worst-case number of rounds from the start till gathering.

We consider the feasibility and time complexity of gathering teams of agents, and give a complete solution of this problem. It turns out that both feasibility and complexity of gathering depend on the crucial parameter x which is the size of teams. For the oriented line, gathering is impossible if $x = 1$, and it can be accomplished in time $O(D)$, for $x > 1$, where D is the distance between the starting nodes of the most distant teams. This complexity is of course optimal. For the unoriented line, the situation is different. For $x = 1$, gathering is also impossible, but for $x = 2$, the optimal time of gathering is $\Theta(D \log L)$, and for $x \geq 3$ the optimal time of gathering is $\Theta(D)$.

Solving the gathering problem for agents that are finite automata navigating in an infinite environment requires new methodological tools. Traditional gathering techniques in graphs are *count driven*: agents make decisions based on counting steps. Since distances between agents may be unbounded, agents have to count unbounded numbers of steps. When agents are finite automata, counting unbounded numbers of steps is impossible, hence we must use different methods. In all our gathering algorithms, changes of the agents' behavior are triggered not by counting steps but by events which are meetings between agents during which they interact. Hence our new technique is *event driven*. Designing the behavior of the agents based on meeting events, so as to guarantee gathering regardless of the adversary's decisions is our main methodological contribution.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases Gathering, deterministic finite automaton, mobile agent, team of agents, line, time

Digital Object Identifier 10.4230/LIPIcs.OPODIS.2024.11

Funding *Younan Gao*: Partially supported by the Research Chair in Distributed Computing at the Université du Québec en Outaouais and MUR 2022YRB97K, PINC, Pangenome Informatics: from Theory to Applications.
Andrzej Pelc: Partially supported by NSERC discovery grant RGPIN-2024-03767 and by the Research Chair in Distributed Computing at the Université du Québec en Outaouais.

1 Introduction

The background and the problem. Several mobile agents, navigating in a network, have to meet at the same node. This basic task, known as gathering, has been thoroughly studied in the literature. It has many applications, both in everyday life and in computer science.

Mobile agents can be people who have to meet in a city whose streets form a network. In computer science applications, agents can represent either software agents in computer networks, or mobile robots in networks formed by, e.g., corridors of a contaminated mine, where the access is dangerous for humans. The purpose of gathering may be to exchange data previously collected by software agents that accessed a distributed database, or to coordinate some future task of mobile robots, such as network decontamination, based on previously collected samples of the ground or of air. Since, due to cost reasons, mobile agents are often simple devices with limited memory, sensor capacity and computation power, it is natural to model them as deterministic finite automata that can interact only when they meet.

We consider the feasibility and time complexity of gathering mobile agents, modelled as automata, in one of the simplest networks that is the infinite line, i.e. the infinite graph all of whose nodes have degree 2. While the gathering problem in networks has a long research history, mobile agents in this context were usually modelled either as machines with unbounded memory or it was assumed that agents can "see" other agents at a distance. To the best of our knowledge, this classic problem has never been investigated before for agents modelled as deterministic finite automata navigating in an infinite environment and interacting with other agents only at meetings.[1]

The model. The environment in which agents navigate is modelled as an infinite line, i.e., the infinite graph all of whose nodes have degree 2. Nodes of the line do not have labels, and ports at each node are labeled -1 and 1. We consider two variations of this port-labeled graph: in the *oriented line*, ports corresponding to any edge at both extremities of it are different; in the *unoriented line*, port labeling at any node is arbitrary.

There are several agents navigating in a line. They are modelled as deterministic finite Mealy automata. Agents move in synchronous rounds. All agents start in the same round 0. In each round, an agent can choose one of the following actions: take the port -1, take the port 1, or stay idle. Agents have distinct labels which are integers from the set $\{1, \ldots, L\}$. Agents start in R teams, each of which consists of x agents, for some fixed integer x, where $xR \leq L$. Agents in a team have the same starting node, called the *base* of the team. The integer x, the size L of the space of labels and the total number R of teams are known in advance and can be used in the design of the agents. [2] Throughout the paper, we assume that $R > 1$. If there is only one team, gathering is trivially achieved in the wake-up round. The adversary knows all the agents and decides the composition of teams and their bases. In the case of the unoriented line, the adversary also decides the port labeling at each node.

When entering a node, an agent sees the entry port number, and sees the set of states of all currently collocated agents. If in the current round the agent stays idle, it also sees this set of states. (This is how agents interact). This information, together with the size L of the label space and the number R of teams, forms the input of the agent in a given round. (L and R form the fixed part of the input that never changes). Based on its state and this input, the agent transits to some state, and produces an output action belonging to the set $\{-1, 1, 0\}$, to be executed in the next round. Output action -1 means taking port -1, output action 1 means taking port 1, and output action 0 means staying idle in the next round. The sequence of the above actions forms the *trajectory* of an agent. Each agent has a special state STOP, upon transition to which, it stays idle forever.

---

[1] It could seem that the recent paper [17] is a counterexample to this rule. However, the focus of [17] is on computing functions by teams of automata, and the environment in which agents navigate is the rooted oriented half-line. In this graph, gathering is trivial: all agents go toward the root and stop.

[2] The discussion of the above assumptions is deferred to the full version of this paper.

We now formalize the above intuitions. All agents are represented by the same deterministic finite Mealy automaton $\mathcal{A} = (\mathcal{I}, O, Q, \delta, \lambda)$. The agent with label $\ell \in \{1, \ldots, L\}$ has the set of states Q^ℓ, where all sets Q^ℓ are pairwise disjoint. Thus the label of an agent can be recognized from each of its states. $Q = Q^1 \cup \cdots \cup Q^L$ denotes the union of sets of states of all the agents. The size of Q, i.e., the number of states of the automaton, is denoted by H. Let $\mathcal{Q}$ be the set of all subsets of Q. The common input alphabet for all agents is the set $\mathcal{I} = \{(L, R)\} \times \{-1, 1, 0\} \times \mathcal{Q}$. This corresponds to the intuition that an input $I \in \mathcal{I}$ is a triple whose first term is the *fixed part of the input* consisting of the pair of integers (L, R), whose second term is the *move part of the input* which is either the entry port number in the current round or 0 if the agent is idle in the current round (it is also 0 in round 0), and whose third term is the *states part of the input* which is the set of states of other currently collocated agents. This set may be empty, if the agent is currently alone. The common output alphabet $O = \{-1, 1, 0\}$ corresponds to the output actions intuitively described above.

It remains to define the transition function δ and the output function λ. The transition function $\delta : Q \times \mathcal{I} \to Q$ takes the current state of an agent and its current input, and produces the state to which this agent transits in the next round. The fact that Q^ℓ is the set of states of agent ℓ is reflected by the following restriction: if $q \in Q^\ell$, then $\delta(q, I) \in Q^\ell$, for any input I, i.e., under any input, the transition function maps a state of a given agent to a state of the same agent. The output function $\lambda : Q \times \mathcal{I} \to O$ takes the current state of an agent and its current input, and produces the output action to be executed in the next round.

According to the established custom in the literature on automata navigating in graphs, we present the behavior of our automata by designing procedures that need only remember a constant number of bits, and thus can be executed by deterministic finite automata, rather than formally describing the construction of a Mealy automaton by defining its output and state transition functions.

The aim is for all agents to gather at the same node and transit to state STOP. The first round in which all the agents are at the same node in state STOP is called the *gathering round*. Gathering is feasible, if this task can be accomplished for any decisions of the adversary, and its time is the worst-case (largest) gathering round, over all decisions of the adversary.

Our results. We give a complete solution of the problem of feasibility and time complexity of gathering teams of agents on the infinite line. It turns out that both feasibility and complexity of gathering depend on the crucial parameter x which is the size of teams. For the oriented line, gathering is impossible if $x = 1$, and it can be accomplished in time $O(D)$, for any $x > 1$, where D is the distance between the bases of the most distant teams. The first fact means that, for $x = 1$ and for arbitrary agents, the adversary can place them in the oriented line (at distinct nodes) in such a way that they will never gather. The second fact means that, for any $x > 1$ and any $R > 1$, there exist $x \cdot R$ agents, each assigned a distinct label drawn from $\{1, \ldots, L\}$, where $L \geq xR$, such that if the adversary composes them in arbitrary R teams of size x and selects the R bases of the teams as arbitrary nodes of the oriented line with the most distant nodes at distance D, then the agents will gather in time $O(D)$. This complexity is of course optimal.

For the unoriented line, the situation is different. For $x = 1$, gathering is also impossible which means that, for $x = 1$ and for arbitrary agents, the adversary can choose a port labeling of the line, and can place the agents at distinct nodes in such a way that they will never gather. This directly follows from the above impossibility result on the oriented line, as the adversary can choose the port labeling as in the oriented line.

However, for $x = 2$, the optimal time of gathering in the unoriented line turns out to be $\Theta(D \log L)$. To show this, we prove two facts. First, we show that there exist $2R$ agents, each assigned a distinct label drawn from $\{1, \ldots, L\}$, where $L \geq 2R$, such that if the adversary chooses an arbitrary port labeling of the line, composes the agents in arbitrary R teams of size 2 and selects the R bases of the teams as arbitrary nodes of the line with the most distant nodes at distance D, then the agents will gather in time $O(D \log L)$. Second, we prove that this complexity is optimal, even for two teams of agents, each of size 2. In fact, we show that the "difficult" port labeling of the line can be chosen the same for any agents: it is the *homogeneous* port labeling in which, for every edge h, port numbers at both extremities of h are equal. More precisely, we show that for any agents, the adversary can select two teams of agents of size 2 and choose bases of these teams as nodes at an arbitrarily large distance D on the line with homogeneous port labeling, so that the gathering time will be at least $cD \log L$, for some constant c. This shows that the complexity $O(D \log L)$ is tight.

Finally, we show that, for any $x > 2$ and any $R > 1$, there is an algorithm that gathers all $x \cdot R$ agents partitioned in arbitrary R teams of size x, in time $O(D)$, which is clearly optimal.

Solving the gathering problem for agents that are finite automata navigating in an infinite environment requires new methodological tools. Traditional gathering techniques in graphs are *count driven*: agents make decisions based on counting steps. Since distances between agents may be unbounded, agents have to count unbounded numbers of steps. When agents are finite automata, counting unbounded numbers of steps is impossible[3], hence we must use different methods. In all our gathering algorithms, changes of the agents' behavior are triggered not by counting steps but by events which are meetings between agents during which they interact. Hence our new technique is *event driven*. Designing the behavior of the agents based on meeting events, so as to guarantee gathering regardless of the adversary's decisions is our main methodological contribution.

While we assume that agents operate in the infinite line, all our positive results remain true also for arbitrary bounded lines and for arbitrary rings.

Related work. Gathering mobile agents in graphs, also called *rendezvous* if there are only two agents, is a well studied topic in the distributed computing literature.

In the majority of the papers on gathering, it is assumed that nodes do not have distinct identities, and agents cannot mark nodes: we follow these assumptions in the present paper. However, departures from this model exist: rendezvous was also considered in graphs whose nodes are labeled [8, 15], or when marking nodes by agents is allowed [14]. Randomized rendezvous was surveyed in [1] and deterministic rendezvous – in [18].

Most of the literature on rendezvous considered finite graphs and assumed the synchronous scenario, where agents move in rounds. In [19], the authors gave rendezvous algorithms with time polynomial in the size of the graph and the length of agents' labels. Gathering many agents in the presence of Byzantine agents that can behave arbitrarily was studied in [6].

In the above cited papers, agents are modeled as Turing machines and their memory is unbounded. Other studies concern the minimum amount of memory that agents must have in order to accomplish rendezvous [9, 13]. In this case, agents are modeled as state machines and the number of states is a function of the size of graphs in which they operate.

Several authors studied synchronous rendezvous in infinite graphs. In all cases, agents had distinct identities and were modeled as Turing machines. In [8], the authors considered rendezvous in infinite trees and grids, under the assumption that the agents know their

[3] This is the reason for our impossibility result in case of teams of size 1.

location in the graph (then the initial location can serve as a label). In [3], rendezvous in infinite trees was investigated. Rendezvous in arbitrary graphs, finite or infinite, was studied in [4, 16]. Rendezvous in the oriented grid was investigated, e.g., in [2, 4, 5, 8].

In several papers, asynchronous gathering was studied in the plane [7, 12] and in graphs [5, 2, 11]. In the plane, agents are modeled as moving points, and it is usually assumed that they can see the positions of other agents. In graphs, an agent chooses the edge to traverse, but the adversary controls the speed of the agent. Then rendezvous at a node cannot be guaranteed even in the two-node graph, hence the agents are permitted to meet inside an edge. For asynchronous rendezvous, the optimization criterion is the cost, i.e., the total number of edge traversals. In [2], the authors designed almost optimal algorithms for asynchronous rendezvous in infinite multidimensional grids, assuming that an agent knows its position in the grid. In [5, 11] this assumption was replaced by a weaker assumption that agents have distinct identities. In [5], a polynomial-cost algorithm was designed for the infinite oriented two-dimensional grid, and in [11] – for arbitrary finite graphs.

2 Preliminaries

In this section, we introduce the notions, notations and basic facts used throughout the paper. All proofs omitted in this section are deferred to the full version of this paper.

For any port labeling of a line, we will use two notions describing the two directions of the line. For any node, we define the *plus direction* and the *minus direction* as the directions corresponding to port 1 and port -1, respectively. We also define the *left* and *right* directions of any line. For the oriented line, the left (resp. right) direction is the minus (resp. plus) direction. For any other port labeling, these directions are chosen arbitrarily. Hence, in the oriented line, agents can identify directions left and right. For an arbitrary port labeling, agents cannot identify them, so we will use these expressions only in comments and in the analysis. For any two nodes v and v' in a line, we define their distance $dist(v, v')$ as the number of edges between them.

Agents are identified with their labels. The node at which a team of agents are woken up is referred to as the *base* of the agents in the team. For any port labeling of a line, and for any base of a single agent, navigating in the line, we define the *trajectory* of this agent as the infinite sequence of terms $-1, 0, 1$ with the following meaning. The ith term of the trajectory is 0 if the agent stays put in the ith round, it is -1 if the agent takes port -1 in the ith round, and it is 1 if the agent takes port 1 in the ith round.

We say that a trajectory of an agent has a *period* of length τ if there exists an integer $t \geq 1$, such that, for any fixed $0 \leq i < \tau$, the $(t + j\tau + i)$th term of the trajectory is the same for all $j \geq 0$. The sequence of terms $-1, 0, 1$ corresponding to indices $t+j\tau, t+j\tau+1, \ldots, t+j\tau+\tau-1$ is called a period of this trajectory, and the sequence of terms $-1, 0, 1$ corresponding to indices $1, \ldots, t-1$ is called a *prefix* corresponding to this period. A trajectory that has a period is called *periodic*.

Apart from the port labeling that yields the oriented line (port numbers at both extremities of each edge are different), we consider another important port labeling, called *homogeneous*, in which, for every edge h, port numbers at both extremities of h are equal. A line with this port labeling will be called homogeneous.

▶ **Proposition 1.** *The trajectory of any agent navigating either in the oriented or in the homogeneous line and starting at any node of it is periodic.*

The notion of a period permits us to define three important notions concerning the trajectory of an agent navigating in the oriented or in the homogeneous line: boundedness, the progress direction and the speed. We first define these notions for an agent in the oriented

line. Consider any starting node (base) of the agent, and consider a period of length τ of its trajectory α. Let v and v' be nodes where the agent is situated at the beginning of two consecutive periods. There are three cases: v' is left of v, $v' = v$, and v' is right of v. For the oriented line, this is independent of the base of the agent. It is easy to see that:

- if v' is left (resp. right) of v then the agent visits all nodes of the line left (resp. right) of the base and only a finite number $r(\alpha)$ (resp. $l(\alpha)$) of nodes right (resp. left) of the base ($r(\alpha)$ and $l(\alpha)$ are independent of the base); in this case we say that the trajectory of the agent is *left-progressing* (resp. *right-progressing*);
- if $v' = v$ then the agent visits only a finite number of nodes: at most $b(\alpha)$ nodes on each side of the base ($b(\alpha)$ is independent of the base); in this case we say that the trajectory of the agent is *bounded*;

If the trajectory of the agent is left- or right-progressing, its *speed* is defined as $dist(v, v')/\tau$. The definition of the speed does not depend on the choice of the period of the trajectory.

In the case of the homogeneous line, the situation is slightly different for two reasons. First, the progress direction of a trajectory depends on the base. Hence we will use notions *minus progressing* and *plus progressing*, where the directions are defined with respect to the base. Second (unlike in the oriented line), it is possible that the node v' at the beginning of a period is different from the node v at the beginning of the preceding period but after two consecutive periods these nodes are the same. This happens, e.g., when the period is $(1, 1, -1)$. Hence, in order to define progress direction and boundedness properly, we consider *double periods*: a period Q is double, if it is the concatenation $P * P$, for some period P. For double periods, the above issue disappears.

Consider any base u of the agent, and consider a double period of length τ of its trajectory α. Let v and v' be nodes where the agent is situated at the beginning of two consecutive periods. We use the plus and minus direction with respect to u. There are three cases: v' is in minus direction from v, $v' = v$, and v' is in plus direction from v. We have:

- if v' is in minus (resp. plus) direction from v then the agent visits all nodes of the line in minus (resp. plus) direction from u and only a finite number $r(\alpha)$ (resp. $l(\alpha)$) of nodes in plus (resp. minus) direction from u ($r(\alpha)$ and $l(\alpha)$ are independent of the base); in this case we say that the trajectory of the agent is *minus-progressing* (resp. *plus-progressing*);
- if $v' = v$ then the agent visits only a finite number of nodes: at most $b(\alpha)$ nodes on each side of the base ($b(\alpha)$ is independent of the base); in this case we say that the trajectory of the agent is *bounded*;

If the trajectory of the agent is minus- or plus-progressing, its *speed* is defined as $dist(v, v')/\tau$, for any double period. Similarly as before, the definition of the speed does not depend on the choice of the double period of the trajectory. Notice that in the case of the homogeneous line, for a fixed base of an agent, we can still use the expression "left-" or "right-progressing" with respect to its trajectory, although, unlike for the oriented line, these notions also depend on the base and not only on the trajectory.

The following proposition bounds the length of a prefix and of a period of trajectories.

▶ **Proposition 2.** *For any periodic trajectory α, denote by σ the length of the shortest period of α and by π the length of the prefix preceding the first period of length σ. Then $\pi + \sigma \leq 3H$.*

We use the following fact holding both for the oriented and the homogeneous line. Consider two agents a and b with bases u and v, respectively. We say that agent b *follows* agent a, if either the trajectories of both of them are left-progressing and u is left of v, or the trajectories of both of them are right-progressing and u is right of v. The following proposition says intuitively that, if the initial distance between the agents is sufficiently large, then:

Table 1 The four modes used in Algorithm `Oriented` and their corresponding actions.

Mode	Action
`right_slow`	Move one step right, stay put for one round, and repeat
`right_fast`	Move one step right in each round
`left_fast`	Move one step left in each round
`sentinel`	Stay put

(i) if the follower is not faster than the followed agent, then agents can never meet, and
(ii) if the difference between the speeds of the agents is small, then they cannot meet soon.

▶ **Proposition 3.** *Consider two agents, a and b, with bases u and v, respectively, either in the oriented line or in the homogeneous line, such that $D = dist(u, v) > 72H^2 + 6H$ and agent b follows agent a. Let $z = V(b) - V(a)$, where $V(a)$ and $V(b)$ denote the speeds of the trajectories of a and b, respectively.*

i) *If $z \leq 0$, then agents a and b can never meet;*
ii) *If $z > 0$, then agents a and b cannot meet before round $3H+36H^2+\lceil(D-6H-72H^2)/z\rceil$.*

3 The Impossibility Result

In this section we show that, if the team size is $x = 1$, then gathering is impossible for some port labeling of the line. This port labeling is that of the oriented line. The proof of the following theorem is deferred to the full version of this paper.

▶ **Theorem 4.** *Consider an arbitrary set of agents in teams of size 1, i.e., $x = 1$. The adversary can place these agents at distinct nodes of the oriented line in such a way that no pair of agents will ever meet.*

4 The Oriented Line

In this section, we consider gathering of R teams of agents in the oriented line. In view of Theorem 4, if the size x of each team is one, then the adversary can prevent gathering. So, we assume that $x > 1$, throughout this section. Due to space limitation, we only present the high-level idea of the algorithm, while its detailed description and the proof of its correctness and complexity are deferred to the full version of this paper.

We now describe Algorithm `Oriented` that accomplishes gathering of arbitrary R teams of $x > 1$ agents in the oriented line, in optimal time $O(D)$, where D is the distance between the bases of the most distant teams.

Modes. In each round of the algorithm execution, each agent is in some mode, encoded in the state of the agent. In each mode, an agent performs some action. All modes and their corresponding actions, used in Algorithm `Oriented`, are listed in Table 1.

The high-level idea of the algorithm. After waking up, the agent with the smallest label in each team assigns itself mode `right_slow` which means that it moves right with speed one-half, and the other $x - 1$ agents in the team assign themselves mode `sentinel` and stay put at their base. An agent a in mode `right_slow` (except the agent from the rightmost team) will meet agents in mode `sentinel`. Each time such a meeting happens, agent a counts the total number of agents in mode `sentinel` it has seen so far. Notice that among

all the agents in mode `right_slow`, only the agent b from the leftmost team will meet $(x-1)(R-1)$ agents in mode `sentinel` (except `sentinel` agents from its own team). After meeting $(x-1)(R-1)$ agents in mode `sentinel`, agent b switches to mode `right_fast` which means that it moves right with speed 1. It is the only agent that ever switches to mode `right_fast`. Speed 1 allows it to meet agents in mode `right_slow` that it follows. Each time agent b meets an agent in mode `right_slow`, agent b counts the total number of agents in mode `right_slow` it has seen so far, and the agent in mode `right_slow` switches to mode `sentinel`. After meeting $R-1$ agents in mode `right_slow`, agent b switches to mode `left_fast`. Let c be the $(R-1)th$ agent in mode `right_slow` that agent b met. At this meeting, agent c also switches to mode `left_fast`.

So, both agents b and c move together left with speed 1. At this time, all the other $(x \cdot R - 2)$ agents are in mode `sentinel`, left of agents b and c. Then, each time a meeting between agents in mode `left_fast` and agents in mode `sentinel` happens, all the agents in mode `sentinel` switch to mode `left_fast` and move together with the other agents in mode `left_fast`. In the end, all $x \cdot R$ agents gather at the base u of the leftmost team. They know that this happened, by counting the number of agents at u, and transit to state `STOP`. The following theorem states the correctness and complexity of Algorithm `Oriented`. Its proof is deferred to the full version of this paper.

▶ **Theorem 5.** *Algorithm `Oriented` gathers R teams of $x > 1$ agents each, in the oriented line in $7D$ rounds, where D denotes the distance between the bases of the most distant teams.*

5 The Unoriented Line

In this section, we consider gathering in the unoriented line, in which the port labeling at each node is arbitrary. We will use the expressions "left" and "right", to denote the two directions of the line but we need to keep in mind that agents cannot identify these directions, so we will use these expressions only in comments and in the analysis.

It follows from Section 3 that, if the team size is $x = 1$, then gathering is impossible for some port labeling, namely that of the oriented line. This section is devoted to showing that, in the unoriented line, the optimal gathering time is $\Theta(D \log L)$, for $x = 2$, and it is $\Theta(D)$, for $x > 2$, where D is the distance between the bases of the most distant teams. We first consider the case $x = 2$.

5.1 Teams of size 2

Results of this section are by far the most technically difficult. The section is organized as follows. First we prove that, regardless of the finite deterministic automaton used, and even for two teams of size 2, the adversary can choose a particular port labeling of the line, namely the homogeneous port labeling, and it can choose the bases at an arbitrarily large distance D and a composition of teams in such a way that the time of gathering is at least $cD \log L$, for some positive constant c. Then we show, for any number $R > 1$ of teams of size 2, an algorithm that always guarantees gathering in time $O(D \log L)$. In view of the above lower bound, this complexity is optimal. Proofs omitted in this section are deferred to the full version of this paper.

5.1.1 The lower bound

Consider the homogeneous line. Let $C = \lfloor L/2 \rfloor$. We will consider the following C teams of size 2: $\{1, 2\}$, $\{3, 4\}$, ... , $\{2C-1, 2C\}$. For each of those teams $\{a, a+1\}$, there are fixed trajectories $\alpha(a)$ and $\alpha(a+1)$ corresponding to agents a and $a+1$, respectively, yielded by

the finite deterministic automaton used. Each of the above teams is called *one-way* if either both corresponding trajectories are plus-progressing, or both are minus-progressing, or at least one of them is bounded.

▶ **Lemma 6.** *If there exist two one-way teams $\{a, a+1\}$ and $\{b, b+1\}$, for odd $a, b \leq 2C-1$, then there exist arbitrarily large integers D such that the adversary can place these teams at two bases at distance D, so that gathering will never happen.*

In view of Lemma 6, we may assume that there is at most one one-way team. Hence there exist $C-1$ teams $\{a, a+1\}$, for odd $a \leq 2C-1$ that are not one-way. Call these teams *canonical*. By definition, for every canonical team, one of the corresponding trajectories is plus-progressing and the other is minus-progressing. Call the plus-progressing (resp. minus-progressing) trajectory the *plus-trajectory* (resp. the *minus-trajectory*) of the team. Agents corresponding to these trajectories are called the *plus-agent* (resp. *minus-agent*) of the team.

Consider two agents a and b with bases u and v, respectively, such that u is left of v. We say that the agents are *diverging*, if the trajectory of a is left-progressing and the trajectory of b is right-progressing.

▶ **Lemma 7.** *If the distance between the bases u and v of diverging agents is larger than $6H$ then these agents can never meet.*

We will use the following lemma that is a direct consequence of [10, Theorem 3.1]. [4]

▶ **Lemma 8.** *Consider a set of g agents, each of which is assigned a trajectory. There exist two agents in this set, such that if they start at two nodes of a line at distance D and follow their trajectories then the first meeting between them occurs after at least $\frac{1}{6}D \log g$ rounds.*

The main result of this subsection is the following theorem.

▶ **Theorem 9.** *For any finite deterministic automaton formalizing the agents, there exists a positive constant c such that, for arbitrarily large D, the adversary can choose two canonical teams and their bases at distance D on the homogeneous line, so that gathering takes time at least $cD \log L$.*

Proof. Let $p = \lfloor L^{1/3} \rfloor$. We consider the partition of the interval $(0, 1]$ into subintervals $I_1, \ldots, I_p$, where $I_i = (\frac{i-1}{p}, \frac{i}{p}]$, for $i = 1, \ldots, p$. For any $i, j \in \{1, \ldots, p\}$, denote $\Sigma_{i,j} = I_i \times I_j$. Hence, $\Sigma_{i,j}$ form a partition of the square $(0, 1] \times (0, 1]$ into p^2 squares. We assign each canonical team to one of these squares as follows: a canonical team is assigned to square $\Sigma_{i,j}$, if the speed of the minus-trajectory of the team is in I_i and the speed of the plus-trajectory of the team is in I_j. Since there are p^2 squares and $C - 1 = \lfloor L/2 \rfloor - 1$ canonical teams, there is at least one square $\Sigma_{i,j}$ to which at least $g = p/2$ canonical teams are assigned. Let $\Sigma = \Sigma_{i,j}$ denote any such square (there can be many of them).

We now describe the decisions of the adversary. Let D be any odd integer larger than $2 \cdot (72H^2 + 6H)$. There are two cases.

Case 1. $i \geq j$. Consider the plus-agents of the teams assigned to Σ. There are at least g of them. By Lemma 8, there exist two of them, p_1 and p_2, such that if they are placed at any two nodes at distance D and follow their trajectories, then they cannot meet before $\frac{1}{6}D \log g$ rounds. Now the adversary makes its first choice: it chooses canonical teams to which p_1 and p_2 belong. These are teams $\{p_1, q_1\}$ and $\{p_2, q_2\}$, where q_i denote the minus

[4] [10, Theorem 3.1] holds even in a ring and even if agents are Turing machines.

agents in each team. (Note that we do not know which of the agents has an even label and which has an odd label in each team). It remains to choose the bases. As u, the adversary chooses any node in the line, such that port 1 is in the right direction, and chooses as v the (unique) node at distance D right of u. Finally the adversary chooses u as the base of team $\{p_1, q_1\}$ and chooses v as the base of team $\{p_2, q_2\}$.

Case 2. $i < j$. Now the decisions of the adversary are symmetric with respect to Case 1. This time, consider the minus-agents of the teams assigned to Σ. There are at least g of them. By Lemma 8, there exist two of them, q_1 and q_2, such that if they are placed at any two nodes at distance D and follow their trajectories, then they cannot meet before $\frac{1}{6}D \log g$ rounds. The adversary chooses canonical teams to which q_1 and q_2 belong. These are teams $\{p_1, q_1\}$ and $\{p_2, q_2\}$, where p_i denote the plus agents in each team. It remains to choose the bases. As u the adversary chooses any node in the line such that port -1 is in the right direction, and chooses as v the (unique) node at distance D right of u. Finally the adversary chooses u as the base of team $\{p_1, q_1\}$ and chooses v as the base of team $\{p_2, q_2\}$ (here nothing changes).

▷ Claim 10. There is a positive constant c such that, for the above choices of the adversary, the first meeting between agents of different teams occurs after at least $cD \log L$ rounds.

Proof. In order to prove the claim, we consider Case 1. The argument in Case 2 is similar.

The choice of teams $\{p_1, q_1\}$ and $\{p_2, q_2\}$ guarantees that the meeting between agents p_1 and p_2 requires at least $\frac{1}{6}D \log g = \frac{1}{6}D \log \frac{p}{2}$ rounds, as $g = p/2$.

Agent q_1, based at u, is a minus-agent and its trajectory is left-progressing; agent q_2, based at v, is a minus-agent and its trajectory is right-progressing. Hence, agents q_1 and q_2 are diverging. As $D > 6H$, agents q_1 and q_2 can never meet, in view of Lemma 7.

Observe that agent p_2 follows agent q_1. First, suppose that $i > j$. The speed $V(q_1)$ of the trajectory of q_1 is larger than the speed $V(p_2)$ of the trajectory of p_2. As $D > 2 \cdot (72H^2 + 6H)$, agent p_2 can never meet agent q_1, in view of Proposition 3 (i). Next, suppose that $i = j$. Then $|V(p_2) - V(q_1)| < 1/p$, because $V(q_1)$ and $V(p_2)$ are both in the interval I_i. If $V(p_2) - V(q_1) \leq 0$, then agent p_2 can never meet agent q_1, in view of Proposition 3 (i). Otherwise, as $D > 2 \cdot (72H^2 + 6H)$, Proposition 3 (ii) implies that the meeting between p_2 and q_1 cannot be achieved before round $3H + 36H^2 + \lceil (D - 6H - 72H^2)p \rceil > \frac{D}{2}p \in \omega(D \log p)$. Symmetrically, agent p_1 follows agent q_2. Using similar arguments, agents p_1 and q_2 either never meet or meet after time $\omega(D \log p)$. It follows that for a sufficiently small constant c', any meeting between agents p_1 and q_2 or p_2 and q_1 cannot happen before round $c'D \log p$. Hence, for a sufficiently small constant c, the first meeting between agents of different teams occurs after at least $cD \log L$ rounds. ◁

Claim 10 concludes the proof of the Theorem. ◀

5.1.2 The algorithm

We now describe Algorithm `Small Teams Unoriented` that guarantees gathering teams of size 2 in an unoriented line, in time $O(D \log L)$. More precisely, the algorithm accomplishes gathering of arbitrary R teams of size 2 in time $O(D \log L)$, where D is the distance between the bases of the most distant teams, and the port labeling of the line is arbitrary.

In our algorithm, we will use the following procedure `Dance` $(string, p)$. Its parameter *string* is a finite binary sequence, and its parameter p is one of the possible ports -1 or 1. Intuitively, procedure `Dance` $(string, p)$ is an infinite procedure divided into phases of k rounds each, where k is the length of the binary sequence *string*. In each phase, the agent

Table 2 The seven modes in which an agent with label ℓ can be, where $\alpha = Tr(\ell)$ and $\beta = (1110)$.

Mode	Action
$(-1)-$`slow`	`Dance` $(\alpha, -1)$
$(+1)-$`slow`	`Dance` $(\alpha, +1)$
$(-1)-$`fast`	`Dance` $(\beta, -1)$
$(+1)-$`fast`	`Dance` $(\beta, +1)$
`idle`	Stay put
`hibernate`	Stay put
`sentinel`	Stay put

first chooses the edge h on which the dance will be executed, and then performs the dance itself on edge h. In the first phase, h corresponds to port p, and in each subsequent phase, h is the edge incident to the current node, which is different from the edge on which the dance in the previous phase was executed. The agent processes k bits of the sequence *string* in k consecutive rounds as follows: if the bit is 1, then traverse edge h; otherwise, stay idle. The pseudo-code of procedure `Dance` is deferred to the full version of this paper.

While procedure `Dance` is formulated as an infinite procedure, it will be interrupted in some round of the execution of the algorithm, and a new procedure `Dance` with different parameters will be started.

Label transformation. Recall that labels of all agents are different integers from the set $\{1, \ldots, L\}$. Let `len` denote $\lfloor \log L \rfloor + 1$. Consider a label $\ell \in \{1, \ldots, L\}$. We represent it by the unique binary sequence $Bin(\ell) = (b_1 b_2 \cdots b_{\texttt{len}})$ which is the binary representation of ℓ, with a string of zeroes possibly added as a prefix, to get length `len`.

Let $z = 2 \cdot \texttt{len} + 4$. We now define the transformed label obtained from ℓ. This is the binary sequence $Tr(\ell)$ of length z defined as follows: In $Bin(\ell) = (b_1 b_2 \cdots b_{\texttt{len}})$, replace each bit 1 by the string 11, replace each bit 0 by the string 00, add the string 10 as a prefix and the string 00 as a suffix. Due to the added prefix string 10, $Tr(\ell)$ always contains an odd number of bits 1. For example, if L is 15 and ℓ is 3, then $Bin(\ell)$ is the binary sequence (0011), and $Tr(\ell)$ is the binary sequence (100000111100).

Modes. In each round of the algorithm execution, each agent is in some *mode*, encoded in the state of the agent. In each mode, an agent performs some action. Four of these modes instruct the agent to execute procedure `Dance` with various parameters, and three other modes instruct it to stay put. Modes are changed at meetings of the agents, called events. In Table 2, we list seven modes in which an agent can be.

We now explain the roles that are played by the seven modes. An agent in any of the modes `idle`, `hibernate`, or `sentinel` stays at the current node until a new event happens at this node. Mode `hibernate` is used only in the first z rounds of the algorithm, while modes `idle` and `sentinel` are used in the remaining rounds. In particular, mode `sentinel` is only assigned to the leftmost and rightmost agents, once these agents identify themselves as such.

Modes $(-1)-$`slow` and $(+1)-$`slow` will be called *slow* modes, and modes $(-1)-$`fast` and $(+1)-$`fast` will be called *fast* modes. For simplicity, we call an agent in slow (resp. fast) mode a slow (resp. fast) agent. In view of procedure `Dance`, a slow or fast agent dances along the same edge in each phase. We call the edge endpoint, at which a phase starts, the *dancing source* of the agent, in this phase. Let ℓ denote the label of an agent, and let $\alpha = Tr(\ell)$ and $\beta = (1110)$. As shown in Table 2, a slow or a fast agent executes procedure

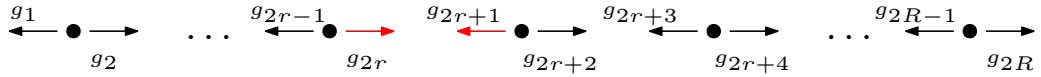

Figure 1 Agents $g_1, \ldots, g_{2R}$ on the unoriented line. Agents in red indicate a pair of neighbors.

`Dance` with binary sequences α and β, respectively. Observe that the sequences α and β used in modes slow and fast have an odd number of bits 1. Hence, after each phase, the dancing source of a slow or fast agent is changed to be the other endpoint of this edge. During an execution of procedure `Dance`, the way that the dancing source changes looks like an object moving on the unoriented line exactly one step in the same direction in each phase. We call the direction, in which the dancing source of an agent executing `Dance` moves, the *progress direction* of the agent. Although the port numbers at each node are assigned arbitrarily by the adversary and a slow or fast agent cannot tell which direction is left or right, it is always aware of its progress direction. Indeed, consider an agent g that arrives at node u in some round of the execution of `Dance`. The agent knows if, in this phase, u is the dancing source or not. If u is the dancing source of g, then the port via which g arrives at u, indicates its progress direction; otherwise, its progress direction is indicated by the other port at u. This is true, because sequences α and β start with bit 1, end with bit 0, and contain an odd number of bits 1.

An agent in mode $(-1)-$`slow` or mode $(-1)-$`fast` (resp. mode $(+1)-$`slow` or mode $(+1)-$`fast`) leaves the current node via port -1 (resp. 1), in the first round of the execution of procedure `Dance`. This is because the second parameter in the procedure is -1 (resp. 1) and sequences α and β both start with bit 1. We will prove that two slow agents, moving towards each other while executing procedure `Dance`, will meet in a phase, when their dancing sources are at a distance either 1 or 2.

Meetings of a fast and a slow agent happen for a different reason. When a fast agent meets a slow agent progressing in the same direction, we will say that the fast agent *catches* the slow agent. In view of definitions of sequences α and β, each phase of procedure `Dance` in a slow mode lasts $z = |Tr(\ell)| > 4$ rounds, while each phase in a fast mode lasts only four rounds. This means that the dancing source of a slow agent changes every z rounds, while the dancing source of a fast agent changes every four rounds. Observe that any 4-bit sub-segment of α is different from β. Due to this observation and the difference of the dancing source changing frequency for slow and fast agents, we will show that a fast agent following a slow agent progressing in the same direction always catches it.

The high-level idea of the algorithm. In the wake-up round, agents are at their bases in teams of size 2. Since they can see each other's labels, they can compare them and assign modes as follows: the agent with smaller label assigns itself mode $(-1)-$`slow` and the agent with larger label assigns itself mode $(+1)-$`slow`. Conceptually, we number agents $g_1, \ldots, g_{2R}$ as follows (see Fig. 1).

Consider the r-th team from the left, where $r = 1, \ldots, R$. Agent g_{2r-1} is the agent from the r-th team that first moves left, and agent g_{2r} is the agent from the r-th team that first moves right. Agent g_{2r-1} will be called the left agent of the r-th team, and agent g_{2r} will be called the right agent of the r-th team. Notice that an agent does not know in which team it is, and it does not know if it is the left or the right agent in a team, due to the adversarial port labeling of the undirected line. Let's call agents g_{2r} and g_{2r+1} *neighbors*, for any $r = 1, \ldots, R-1$.

Our algorithm guarantees that in some round of its execution, all neighbors will meet in pairs. After such a meeting, both agents switch to mode $(+1)-$`fast`, however this change might not happen immediately at the meeting. If the bases of the r-th team and the $(r+1)$-th team are at a distance exactly one, then both neighbors from these teams switch to mode `hibernate` at their meeting and then switch to mode $(+1)-$`fast` in round z. If the distance between their bases is exactly two, then both neighbors maintain their slow modes until round z and switch to mode $(+1)-$`fast` in this round. (Note that, in both above cases, the earliest round when both neighbors meet could be earlier than round z).

Otherwise, when the distance between bases is more than two, then both neighbors switch to mode $(+1)-$`fast` immediately after the meeting. After changing to a fast mode, an agent swings (possibly switching from mode $(+1)-$`fast` to $(-1)-$`fast` or vice versa) between agent g_1 and agent g_{2R} both of which are still in slow modes. Therefore, it is crucial to identify these agents. To this end, each agent e in a slow mode keeps a *bag* to which it adds labels of fast agents that have caught it. We will show that only bags of g_1 and g_{2R} can have size $(2R-2)$ and that, in some round, they will have this size. This is how the pair of agents g_1 and g_{2R} is identified. At this time these agents switch mode to `sentinel`.

However, an agent in mode `sentinel` does not know whether it is g_1 or g_{2R}. We will distinguish g_1 from g_{2R} indirectly, using fast agents. The $2R-2$ fast agents swing from one agent in mode `sentinel` to the other, gather their labels, compare them, go to the agent with the smaller label and change state to `idle`. Without loss of generality, suppose that the label of agent g_1 is smaller than that of g_{2R}. As a result, all $2R-2$ fast agents meet g_1 and stay with it in mode `idle`; in other words, $2R-1$ agents gather at the same node in some round. When this happens, each of these $2R-1$ agents switches to a fast mode and progresses in the direction opposite to their last move. Hence they will meet agent g_{2R} which stays put in mode `sentinel`. In the end, all $2R$ agents gather at the same node where g_{2R} stays, and transit to state STOP. The detailed description of Algorithm `Small Teams Unoriented` is deferred to the full version of this paper.

The complexity of the algorithm is stated as Theorem 11.

▶ **Theorem 11.** *Suppose that agents have distinct labels drawn from the set $\{1,\cdots,L\}$, that teams are of size 2, and that the distance between the bases of the most distant teams is D. Algorithm `Small Teams Unoriented` gathers all agents at the same node of an unoriented line in time $O(D\log L)$.*

Due to space limitation, we present a sketch proof for Theorem 11. Without loss of generality, assume that the label of g_1 is smaller than the label of g_{2R}. First, we prove that agent g_{2r} meets its neighbor g_{2r+1}, for each $1\leq r<R$. After the meeting, g_{2r} and g_{2r+1} become fast agents and swing between g_1 and g_{2R}. Then, we show that only g_1 and g_{2R} will change (possibly in different rounds) to mode `sentinel`. As agents $g_2,\ldots,g_{2R-1}$ swing between g_1 and g_{2R}, and g_1 is in mode `sentinel`, agents $g_2,\ldots,g_{2R-1}$ will meet g_1 and stay with it in mode `idle`. Next, we prove that by the earliest round when all $2R-2$ fast agents gather at the same node where g_1 stays, agent g_{2R} has already changed to mode `sentinel`. As a result, all $2R-1$ agents, apart from g_{2R}, become fast agents, progress towards g_{2R} and eventually meet g_{2R}, while g_{2R} stays put in mode `sentinel`. Then gathering is achieved and all agents transit to state STOP. To establish the complexity of the algorithm, let t_1 denote the earliest round in which agent g_i becomes a fast agent after meeting its neighbor, for all $1<i<2R$. Let t_2 denote the earliest round in which both g_1 and g_{2R} switched to mode `sentinel`. Let t_3 denote the earliest round in which all $2R-1$ agents, apart from g_{2R}, have gathered at the same node. Let t_4 denote the earliest round in which all $2R$ agents gather at the same node. Our goal is to prove that all rounds $t_1,\ldots,t_4$ exist; furthermore, we will show that $t_1<t_2\leq t_3<t_4\in O(D\log L)$.

Table 3 The five modes in Algorithm `Large Teams Unoriented` and the corresponding actions.

Mode	Action
$(-1)-$`slow`	`Proceed` $(-1, true)$
$(+1)-$`slow`	`Proceed` $(1, true)$
$(-1)-$`fast`	`Proceed` $(-1, false)$
$(+1)-$`fast`	`Proceed` $(1, false)$
`sentinel`	Stay put

5.2 Teams of size larger than 2

It remains to consider the gathering problem in any unoriented line for teams of size larger than 2. Using more than two agents in each team, we design Algorithm `Large Teams Unoriented`, faster than that for teams of size 2: our gathering algorithm for larger teams works in optimal $O(D)$ rounds. In this section, we only present the high-level idea of the algorithm, while the omitted details are deferred to the full version of this paper.

Algorithm `Large Teams Unoriented` uses procedure `Proceed` $(p, slow)$. The procedure has two parameters: p is a port number drawn from $\{-1, 1\}$, and *slow* is a Boolean value. Like procedure `Dance`, `Proceed` $(p, slow)$ is an infinite procedure, divided into phases. Assume that an agent a starts performing `Proceed` $(p, slow)$ at node u. If *slow* is false, then each phase has one round. In the first phase, the agent moves to the neighbor u' of u corresponding to port p at u. Let w denote the node that the agent reached in the $(i-1)$-th phase, for $i > 1$. In the i-th phase, the agent moves to the neighbor w' of w such that $dist(w', u)$ is i. Intuitively, the agent moves away from u with speed one in the direction of port p at node u.

If *slow* is true, each phase consists of two consecutive rounds. In the first phase, the agent moves to the neighbor u' of u corresponding to port p at u, and stays at u' for one round. For any phase $i > 1$, let w denote the node that the agent reached in the $(i-1)$-th phase. In the i-th phase, the agent moves to the neighbor w' of w such that $dist(w', u)$ is i and stays at w' for one round. Intuitively, the agent moves away from u with speed one-half in the direction of port p at u. The pseudo-code of the procedure is deferred to the full version of this paper.

Modes. Similarly to the previous two algorithms, during the execution of Algorithm `Large Teams Unoriented`, each agent is in one of five modes, encoded in the states of the automaton. Each mode corresponds to a different action, as shown in Table 3. Hence, there are five different actions an agent can choose to take. Four of them are to call procedure `Proceed` (with different parameters), and the fifth is to stay put, waiting for future events.

We call agents in modes `right_fast` and `left_fast` *fast* agents, agents in modes `left_slow` and `right_slow` *slow* agents, and agents in mode `sentinel` *sentinel* agents.

The high-level idea of the algorithm. After wake-up, in each team, the agent with the smallest label assigns itself mode $(-1)-$`slow`, the agent with the second smallest label assigns itself mode $(+1)-$`slow`, and all the other $(x-2)$ agents assign themselves mode `sentinel`. Notice that, in each team, the agents with the two smallest labels proceed in different directions, and the agent with the largest label assigns itself mode `sentinel`.

Slow agents move along the line with speed one-half. For an agent, we call any base other than its own a *foreign base*. Each time a slow agent meets $(x-2)$ agents in mode `sentinel`, it reaches a foreign base. A slow agent can tell how many foreign bases it has reached, by

counting the total number of sentinel agents it has seen so far. During the execution of the algorithm, only two of the slow agents can reach $(R-1)$ foreign bases. These two slow agents are from the two most distant teams. Let u denote the $(R-1)$-th foreign base that a slow agent a reaches. By the round when agent a reaches u, it has met $(x-2)(R-1)$ agents in mode `sentinel`. Agent a switches to mode `sentinel`. The largest label ℓ_1 among agents from the team of a is compared to the largest label ℓ_2 among agents based at u. (These labels are coded in the states of agents currently collocated at u). If $\ell_1 < \ell_2$, then the agent with label ℓ_2 becomes a fast agent. This agent, called e, has the largest label among all agents in the two most distant teams. e is the first agent to become fast. When this happens, all the $x \cdot R$ agents are distributed on the line as follows: there are $(x-2)$ sentinel agents at node u, along with the fast agent e; there are only $(R-1)$ slow agents, on one side of u, progressing away from u; there are $(x-2)(R-1)+1$ sentinel agents and $(R-1)$ slow agents on the other side of u, and these slow agents are progressing away from u as well. Agent e progresses in the direction (with respect to u) where only slow agents are located.

As a fast agent, agent e moves with speed one. The fast speed allows agent e to meet all the $(R-1)$ slow agents that it follows. Each time agent e meets a slow agent, it increments the total number of slow agents it has seen since becoming fast, and the slow agent switches to the same mode as that of e and moves together with agent e from now on. Let v denote the node where agent e meets the $(R-1)$-th slow agent. In the round when agent e reaches node v, there are R agents at this node, including agent e itself. In this round, all these R agents become fast agents and move in the direction that agent e comes from. At this point, all the $x \cdot R$ agents are distributed on the line as follows: there are R fast agent at node v; there are no agents at all on one side of v; there are $(x-1)R-(R-1)$ agents in mode `sentinel` and $R-1$ slow agents on the other side of v, and these slow agents move away from v. Hence, all R fast agents follow these $R-1$ slow agents. Since then, each time a meeting happens at some node w, all the non-fast agents at node w (i.e., slow agents and sentinel agents) switch to the mode shared by all fast agents at node w. In this way, all the agents at node w move together, after the meeting. When a meeting happens at a node where $x \cdot R$ agents gather, all the agents transit to state `STOP`. These agents know when gathering occurs by counting the number of agents collocated at each meeting. The detailed description of the algorithm is deferred to the full version of this paper. Theorem 12 states the correctness and complexity of the algorithm.

▶ **Theorem 12.** *Algorithm* `Large Teams Unoriented` *gathers R teams of $x > 2$ agents each, on an unoriented line in $14D$ rounds, where D denotes the maximum distance between bases.*

6 Conclusion

We gave a complete solution of the feasibility and complexity problem of gathering teams of agents modeled as deterministic finite automata on oriented and unoriented lines, showing differences that arise from the orientation feature. To the best of our knowledge, this is the first time gathering of deterministic finite automata that cannot communicate remotely is considered in an infinite environment. It is impossible to design a finite automaton that explores infinite lines, and hence we had to invent new gathering techniques. A natural generalization of our results would be to study gathering of teams of automata in arbitrary (connected) infinite graphs. In this paper we assumed that all teams of agents are of equal size. Generalizing our results to teams of possibly different sizes is another open problem.

References

1 Steve Alpern and Shmuel Gal. *The Theory of Search Games and Rendezvous*, volume 55 of *International series in operations research and management science*. Springer New York, NY, 2003. doi:10.1007/b100809.

2 Evangelos Bampas, Jurek Czyzowicz, Leszek Gąsieniec, David Ilcinkas, and Arnaud Labourel. Almost optimal asynchronous rendezvous in infinite multidimensional grids. In *Proc. 24th International Symposium on Distributed Computing (DISC 2010)*, volume 6343, pages 297–311. Springer Berlin Heidelberg, 2010. doi:10.1007/978-3-642-15763-9_28.

3 Subhash Bhagat and Andrzej Pelc. Deterministic rendezvous in infinite trees. *CoRR*, abs/2203.05160, 2022. doi:10.48550/arXiv.2203.05160.

4 Subhash Bhagat and Andrzej Pelc. How to meet at a node of any connected graph. In *Proc. 36th International Symposium on Distributed Computing (DISC 2022)*, volume 246 of *LIPIcs*, pages 11:1–11:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.DISC.2022.11.

5 Sébastien Bouchard, Marjorie Bournat, Yoann Dieudonné, Swan Dubois, and Franck Petit. Asynchronous approach in the plane: a deterministic polynomial algorithm. *Distributed Comput.*, 32(4):317–337, 2019. doi:10.1007/S00446-018-0338-2.

6 Sébastien Bouchard, Yoann Dieudonné, and Anissa Lamani. Byzantine gathering in polynomial time. *Distributed Comput.*, 35(3):235–263, 2022. doi:10.1007/S00446-022-00419-9.

7 Mark Cieliebak, Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro. Distributed computing by mobile robots: Gathering. *SIAM J. Comput.*, 41(4):829–879, 2012. doi:10.1137/100796534.

8 Andrew Collins, Jurek Czyzowicz, Leszek Gasieniec, Adrian Kosowski, and Russell A. Martin. Synchronous rendezvous for location-aware agents. In *Proc. 25th International Symposium on Distributed Computing (DISC 2011)*, volume 6950 of *Lecture Notes in Computer Science*, pages 447–459. Springer, 2011. doi:10.1007/978-3-642-24100-0_42.

9 Jurek Czyzowicz, Adrian Kosowski, and Andrzej Pelc. How to meet when you forget: log-space rendezvous in arbitrary graphs. *Distributed Comput.*, 25(2):165–178, 2012. doi:10.1007/S00446-011-0141-9.

10 Anders Dessmark, Pierre Fraigniaud, Dariusz R. Kowalski, and Andrzej Pelc. Deterministic rendezvous in graphs. *Algorithmica*, 46(1):69–96, 2006. doi:10.1007/S00453-006-0074-2.

11 Yoann Dieudonné, Andrzej Pelc, and Vincent Villain. How to meet asynchronously at polynomial cost. *SIAM J. Comput.*, 44(3):844–867, 2015. doi:10.1137/130931990.

12 Paola Flocchini, Giuseppe Prencipe, Nicola Santoro, and Peter Widmayer. Gathering of asynchronous robots with limited visibility. *Theor. Comput. Sci.*, 337(1-3):147–168, 2005. doi:10.1016/J.TCS.2005.01.001.

13 Pierre Fraigniaud and Andrzej Pelc. Delays induce an exponential memory gap for rendezvous in trees. *ACM Trans. Algorithms*, 9(2):17:1–17:24, 2013. doi:10.1145/2438645.2438649.

14 Evangelos Kranakis, Nicola Santoro, Cindy Sawchuk, and Danny Krizanc. Mobile agent rendezvous in a ring. In *Proc. 23rd International Conference on Distributed Computing Systems (ICDCS 2003)*, Lecture Notes in Computer Science, pages 592–599. IEEE Computer Society, 2003. doi:10.1109/ICDCS.2003.1203510.

15 Avery Miller and Andrzej Pelc. Fast deterministic rendezvous in labeled lines. In *Proc. 37th International Symposium on Distributed Computing (DISC 2023)*, volume 281 of *LIPIcs*, pages 29:1–29:22. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.DISC.2023.29.

16 Debasish Pattanayak and Andrzej Pelc. Deterministic treasure hunt and rendezvous in arbitrary connected graphs. *CoRR*, abs/2310.01136, 2023. doi:10.48550/arXiv.2310.01136.

17 Debasish Pattanayak and Andrzej Pelc. Computing functions by teams of deterministic finite automata. *arXiv preprint*, 2023. doi:10.48550/arXiv.2310.01151.

18 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. doi:10.1007/978-3-030-11072-7_17.

19 Amnon Ta-Shma and Uri Zwick. How to meet asynchronously at polynomial cost. *ACM Trans. Algorithms*, 10(3):12:1–12:15, 2014. doi:10.1145/2601068.