Distrib. Comput. (2014) 27:21–38
DOI 10.1007/s00446-013-0196-x

Leader election for anonymous asynchronous agents in arbitrary networks

Dariusz Dereniowski · **Andrzej Pelc**

Received: 28 May 2012 / Accepted: 24 July 2013 / Published online: 3 September 2013

Abstract We consider the problem of leader election among mobile agents operating in an arbitrary network modeled as an undirected graph. Nodes of the network are unlabeled and all agents are identical. Hence the only way to elect a leader among agents is by exploiting asymmetries in their initial positions in the graph. Agents do not know the graph or their positions in it, hence they must gain this knowledge by navigating in the graph and share it with other agents to accomplish leader election. This can be done using meetings of agents, which is difficult because of their asynchronous nature: an adversary has total control over the speed of agents. When can a leader be elected in this adversarial scenario and how to do it? We give a complete answer to this question by characterizing all initial configurations for which leader election is possible and by constructing an algorithm that accomplishes leader election for all configurations for which this can be done.

Keywords Leader election · Anonymous network · Asynchronous mobile agents

D. Dereniowski (✉)
Department of Algorithms and System Modeling,
Gdansk University of Technology, Narutowicza 11/12,
80-233 Gdańsk, Poland
e-mail: deren@eti.pg.gda.pl

A. Pelc
Département d'informatique, Université du Québec en Outaouais,
Gatineau, QC J8X 3X7, Canada
e-mail: pelc@uqo.ca

1 Introduction

1.1 The model and the problem

Leader election is one of the fundamental problems in distributed computing, first stated in [38]. Each entity in some set has a Boolean variable initialized to 0 and, after the election, exactly one of these entities, called the *leader*, should change this value to 1. All other entities should know which one becomes the leader. In this paper we consider the problem of leader election among mobile agents that operate in a network. Electing a leader enables the agents to accomplish many tasks in networks more efficiently. For example, the leader may subsequently navigate in the network by itself to periodically check the functionality of network components, thus relieving other agents to accomplish other tasks. Alternatively, the leader may split some computational task among agents that will complete different parts of it in parallel, thus decreasing the total time of computation.

We assume that neither nodes of the network nor agents have labels that can be used for leader election. This assumption is motivated by scenarios where nodes and/or agents may refrain from revealing their identities, e.g., for privacy reasons. Agents may want to cooperate and may well be trustworthy, but they may be reluctant to tell who they are. Hence it is desirable to have leader election algorithms that do not rely on identities but exploit asymmetries in the initial configuration of agents due to its topology and to port labelings. From the methodological point of view, leader election in anonymous networks performed by anonymous agents is interesting because it isolates the structural properties of the network and of the initial configuration of agents as the only features that can be exploited to break symmetry. With unlabeled nodes and agents, leader election is impossible for symmetric initial configurations, e.g., in a ring in which ports at each node are 0, 1, in the clockwise direction and agents are initially situated at every node. Our goal is to answer the following question:

For which initial configurations of agents is leader election possible and how to do it when it is possible?

A network is modeled as an undirected connected graph with unlabeled nodes. It is important to note that the agents have to be able to *locally* distinguish ports at a node: otherwise, the adversary could prevent an agent from choosing a particular edge, thus making navigation in the network impossible even in the simple case of trees. This justifies a common assumption made in the literature: ports at a node of degree d have arbitrary fixed labelings $0, \ldots, d-1$. Throughout the paper, we will use the term "graph" to mean a graph with the above properties. We do not assume any coherence between port labelings at various nodes. Agents can read the port numbers when entering and leaving nodes.

At the beginning, identical agents are situated in some nodes of the graph, at most one agent at each node. The graph with bicolored nodes (black if the node is occupied, white if it is not) is called an initial configuration. Agents do not have labels and have unlimited memory: they are modeled as identical Turing machines. They execute the same deterministic algorithm.

Agents navigate in the graph in an asynchronous way which is formalized by an adversarial model used in [6, 15, 18, 19, 28] and described below. Two important notions used to specify movements of agents are the *route* of the agent and its *walk*. Intuitively, the agent chooses the route *where* it moves and the adversary describes the walk on this route, deciding *how* the agent moves. More precisely, these notions are defined as follows. The adversary initially places an agent at some node of the graph. The route is chosen by the agent and is defined as follows. The agent chooses one of the available ports at the current node. After getting to the other end of the corresponding edge, the agent learns the port number by which it enters and the degree of the entered node. Then it chooses one of the available ports at this node or decides to stay at this node. The resulting route of the agent is the corresponding sequence of edges $(\{v_0, v_1\}, \{v_1, v_2\}, \ldots)$, which is a (not necessarily simple) path in the graph.

We now describe the walk w of an agent on its route. Let $R = (e_1, e_2, \ldots)$ be the route of an agent. Let $e_i = \{v_{i-1}, v_i\}$. Let $(t_0, t_1, t_2, \ldots)$, where $t_0 = 0$, be an increasing sequence of reals, chosen by the adversary, that represent points in time. Let $w_i : [t_i, t_{i+1}] \rightarrow [v_i, v_{i+1}]$ be any continuous function, chosen by the adversary, such that $w_i(t_i) = v_i$ and $w_i(t_{i+1}) = v_{i+1}$. For any $t \in [t_i, t_{i+1}]$, we define $w(t) = w_i(t)$. The interpretation of the walk w is as follows: at time t the agent is at the point $w(t)$ of its route. This general definition of the walk and the fact that (as opposed to the route) it is designed by the adversary, are a way to formalize the asynchronous characteristics of the process. The movement of the agent can be at arbitrary speed, the adversary may sometimes stop the agent or move it back and forth, as long as the walk in each edge of the route is continuous and covers all of it.

This definition makes the adversary very powerful, and consequently agents have little control on how they move. This, for example, makes meetings between agents hard to achieve. Note that agents can meet either at nodes or inside edges of the graph.

Agents with routes R_1 and R_2 and with walks w_1 and w_2 meet at time t, if points $w_1(t)$ and $w_2(t)$ are identical. A meeting is guaranteed for routes R_1 and R_2, if the agents using these routes meet at some time t, regardless of the walks chosen by the adversary.

When agents meet, they notice this fact and can exchange all previously acquired information. However, if the meeting is inside an edge, they continue the walk prescribed by the adversary until reaching the other end of the current edge. New knowledge acquired at the meeting can then influence the choice of the subsequent part of the routes constructed by each of the agents.

Since agents do not know a priori the topology of the graph and have identical memories at the beginning, the only way to elect a leader among agents is by learning the asymmetries in their initial positions in the graph. Hence agents must gain this knowledge by navigating in the network and share it with other agents to accomplish leader election. Sharing the knowledge can be done only as a result of meetings of agents, which is difficult because of the asynchronous way in which they move.

It is not hard to see (cf. Proposition 2) that in the absence of a known upper bound on the size of the graph, leader election is impossible even for asymmetric configurations. Hence we assume that all agents know a priori a common upper bound n on the size of the graph. This is the only information about the environment available to the agents when they start the task of leader election.

Having described our model, we can now make the initial problem more precise. Call an initial configuration *eligible* if, starting from this configuration, leader election can be accomplished regardless of the actions of the adversary. Thus in order that a configuration be eligible, it is enough to have some leader election algorithm starting from it, even one dedicated to this specific configuration. Now our problem can be reformulated as follows.

Which initial configurations are eligible? Find a universal leader election algorithm that elects a leader regardless of the actions of the adversary, for all eligible configurations in graphs of size at most n, where n is known to the agents.

1.2 Our results

Assuming an upper bound n on the size of the graph, known a priori to all agents, we characterize all eligible initial configurations and construct an algorithm that accomplishes leader election for all of them. More precisely, we formulate a combinatorial condition on the initial configuration, which has the following properties. On the one hand, if this condition does not hold, then the adversary can prevent leader election starting from the given initial configuration. On the other hand, we construct an algorithm that elects a leader, regardless of the adversary, for all initial configurations satisfying the condition, in graphs of size at most equal to the given bound n.

Intuitively, leader election is possible when the initial configuration is asymmetric and when agents can learn this, regardless of the actions of the adversary. Both these requirements are contained in the necessary and sufficient condition on eligibility, which we formulate in Sect. 3. In fact, the process of learning the asymmetries by the agents is the main conceptual and technical challenge in the design and analysis of our algorithm. Agents acquire and share this knowledge as a result of meetings. The difficulty is to design the algorithm in such a way that all asymmetries be finally learned by all agents and that all agents be aware of this fact and thus capable to correctly elect the leader.

1.3 Related work

Leader election in networks was mostly studied assuming that all nodes have distinct labels and election has to be performed among nodes. This task was first studied for rings. A synchronous algorithm, based on comparisons of labels, and using $O(n \log n)$ messages was given in [30]. It was proved in [26] that this complexity is optimal for comparison-based algorithms. On the other hand, the authors showed an algorithm using a linear number of messages but requiring very large running time. An asynchronous algorithm using $O(n \log n)$ messages was given, e.g., in [42] and the optimality of this message complexity was shown in [9]. Deterministic leader election in radio networks has been studied, e.g., in [31,32,39] and randomized leader election, e.g., in [45]. In [29] the leader election problem is approached in a model based on mobile agents for networks with labeled nodes.

Many authors [3–5,8,23,35,36,43,46,48] studied various computing problems in anonymous networks. In particular, [7,48] characterize message passing networks in which leader election can be achieved when nodes are anonymous. In [47] the authors study the problem of leader election in general networks, under the assumption that labels are not unique. They characterize networks in which this can be done and give an algorithm which performs election when it is feasible. They assume that the number of nodes of the network is known to all nodes. In [25] the authors study feasibility and message complexity of sorting and leader election in rings with nonunique labels, while in [24] the authors provide algorithms for the generalized leader election problem in rings with arbitrary labels, unknown (and arbitrary) size of the ring and for both synchronous and asynchronous communication. Characterizations of feasible instances for leader election and naming problems have been provided in [10,12,13]. Memory needed for leader election in unlabeled networks has been studied in [27].

Problems involving mobile agents navigating in various environments have been extensively studied in the literature. The environment can be either the plane [14] or a network modeled by a graph. In the latter case the tasks accomplished by agents include exploration of the network [2,11] and rendezvous, in which two or more agents have to gather in the same place. Scenarios in which rendezvous was considered can be broadly divided into randomized, which are the subject of the book [1], and deterministic, which are surveyed in [41]. Deterministic rendezvous in networks was studied either assuming the possibility of marking nodes by agents [16,37], or forbidding it [17,18,20,44]. Another important dichotomy is synchronous versus asynchronous navigation of the agents. In the first case [17,20,21,44], agents traverse edges in lock-step. The asynchronous scenario was in turn studied in two variations. In the first one [33,34], agents could observe positions of other agents, but were oblivious, i.e., could not remember previously seen configurations. The second variation, which is the model used in this paper, has been previously used in [6,15,18,19,28] in the context of rendezvous between two agents. In [6,15,18,19] agents had different labels and in [28] agents were anonymous, as in our present scenario.

In the scenario of mobile agents that get aware of other agents only by meeting them, rendezvous of a pair of agents and the problem of leader election are tightly connected. Indeed, in order to elect a leader agents must meet other agents. Hence, not surprisingly, we use rendezvous of two agents, described in [18,28] as a building block for our leader election algorithm. However, it should be pointed out that leader election among many agents is a much harder task than meeting of two agents. In the latter case, the task is accomplished after meeting and the agents stop. No information transfer is necessary. By contrast, in order to perform leader election, agents must meet some agents, get information from them, separate, meet other agents, try to recognize if these other agents are those seen previously or different ones (recall that agents are anonymous, and can share only their memory content—see Sect. 2), and combine all this knowledge to finally elect a leader.

1.4 Roadmap

In Sect. 2 we formalize the description of how agents decide. This concerns both navigation decisions (on what basis the agents construct their routes) and the final decision who is the leader. We define memory states of the agents that are the basis of all these decisions. In Sect. 3 we formulate the combinatorial condition EC concerning initial configurations that is then proved to be equivalent to eligibility, and we formulate our main result. In Sect. 4 we prove two negative results concerning leader election: one saying that condition EC is necessary for eligibility and the other saying that the assumption concerning knowledge of the upper bound cannot be removed. In Sect. 5 we give our main contribution: we construct a universal algorithm electing a leader for all configurations satisfying condition EC, if agents know an upper bound on the size of the graph. Section 6 contains conclusions.

2 Memory states and decisions of agents

In this section we describe formally on what basis the agents make decisions concerning navigation in the graph (i.e., how they construct their routes) and on what basis they make the decision concerning leader election. All these decisions depend on the *memory states* of the agents. At every time t the memory state of an agent is a finite sequence of symbols defined as follows. Before an agent is woken up by the adversary, its memory state is blank: it is the empty sequence. When an agent is woken up, it perceives the degree d of its initial position, i.e., its memory state becomes the sequence (d). Further on, the memory state of an agent changes when it visits a node. It is caused by the following three types of events: entering a node by the agent, meeting other agents, and leaving a node by the agent. A change of a memory state of an agent is done by appending to its current memory state a sequence of symbols defined as follows. The change due to entering a node of degree d by port number p, consists of appending the sequence (p, d) to the current memory state of the agent. The change due to leaving a node by port q consists of appending q to the current memory state of the agent. The change due to meeting other agents is defined as follows. When entering a node v the agent considers all meetings with other agents that occurred since leaving the previous node. Suppose that the current memory states of the agents met in this time interval by agent λ were $\sigma_1, \ldots, \sigma_k$, in lexicographic order, regardless of the order of meetings in this time interval and disregarding repeated meetings corresponding to the same memory state (and thus to the same agent). Agent λ appends the sequence of symbols $([\sigma_1] \ldots [\sigma_k])$ to its current memory state. When two or more of these events occur simultaneously, for example an agent meets another agent when it enters a node, or an agent meets simultaneously several agents, then the appropriate sequences are appended to its current memory one after another, in lexicographic order. When in the previous memory state the agent made a decision to stay idle at the current node, then its memory state can change only if and when some other agent enters this node. This completes the description of how the memory states of agents evolve. Notice that after traversing an edge the action of agent λ consisting of appending a sequence of symbols $[\sigma]$ due to a meeting with an agent with current memory state σ since leaving the previous node, is performed by λ at most once. Since the number of agents is finite, this implies that, by any given moment in time, the memory state of an agent has changed only a finite number of times, and each time a finite sequence of symbols has been appended. Hence memory states are indeed finite sequences of symbols.

The decisions of agents are made always when an agent is at a node and they are of three possible types: an agent can decide to stay idle, it can decide to exit the current node by some port, or it can elect a leader and stop forever. All these decisions are based on the memory state of the agent after entering the current node and are prescribed by the algorithm. (Recall that agents execute the same deterministic algorithm.) If an agent decides to stay at a given node, then it remains idle at it until another agent enters this node. At this time the memory state of the idle agent changes, and in the new memory state the agent makes a new decision. If an agent decides to leave the current node by a given port, it walks in the edge in the way prescribed by the adversary and makes a new decision after arriving at the other end of the edge. Finally, if an agent decides to elect a leader, it either elects itself, or it decides that it is not a leader, in which case it has to give a sequence of port numbers leading from its own initial position to the initial position of the leader: this is the meaning of the requirement that every non-leader has to know which agent is the leader.

3 Feasibility of leader election

In this section we express the necessary and sufficient condition on eligibility of an initial configuration and we formulate the main result of this paper. We first introduce some basic terminology.

We will use the following notion from [48]. Let G be a graph and v a node of G. We first define, for any $l \geq 0$, the *truncated view* $\mathcal{V}^l(v)$ at depth l, by induction on l. $\mathcal{V}^0(v)$ is a tree consisting of a single node x_0. If $\mathcal{V}^l(u)$ is defined for any node u in the graph, then $\mathcal{V}^{l+1}(v)$ is the port-labeled tree rooted at x_0 and defined as follows. For every node v_i, $i = 1, \ldots, k$, adjacent to v, there is a child x_i of x_0 in $\mathcal{V}^{l+1}(v)$ such that the port number at v corresponding to edge $\{v, v_i\}$

is the same as the port number at x_0 corresponding to edge $\{x_0, x_i\}$, and the port number at v_i corresponding to edge $\{v, v_i\}$ is the same as the port number at x_i corresponding to edge $\{x_0, x_i\}$. Now node x_i, for $i = 1, \dots, k$ becomes the root of the truncated view $\mathcal{V}^l(v_i)$.

The *view* from v is the infinite rooted tree $\mathcal{V}(v)$ with labeled ports, such that $\mathcal{V}^l(v)$ is its truncation to level l, for each $l \geq 0$. For an initial configuration in which node v is the initial position of an agent, the view $\mathcal{V}(v)$ is called the view of this agent.

We will also use a notion similar to that of the view but reflecting the positions of agents in an initial configuration. Consider a graph G and an initial configuration of agents in this graph. Let v be a node occupied by an agent. A function f that assigns either 0 or 1 to each node of $\mathcal{V}(v)$ is called a *binary mapping* for $\mathcal{V}(v)$. A pair $(\mathcal{V}(v), g)$, where f is a binary mapping for $\mathcal{V}(v)$, such that $f(x) = 1$ if and only if x corresponds to an initial position of an agent, is called the *enhanced view* from v. Thus, the enhanced view of an agent additionally marks in its view the nodes corresponding to initial positions of other agents in the initial configuration.

For any route $R = (e_1, e_2, \dots, e_k)$ such that $e_i = \{v_{i-1}, v_i\}$, we denote $b(R) = v_0$ and $d(R) = v_k$, and we say that R *leads* from v_0 to v_k in G. Since nodes of G are unlabeled, agents traveling on a route are aware only of the port numbers of the edges they traverse. Hence, it will be usually more convenient to refer to these sequences of port numbers rather than to the edges of the route. Any finite sequence of non-negative integers will be called a *trail*.

We define an operator $\mathcal{T}$, that provides the trail corresponding to a given route. More formally, if $R = (\{v_1, v_2\}, \{v_2, v_3\}, \dots, \{v_{j-1}, v_j\})$ is a route in G, then define $\mathcal{T}(R) = (p_1, \dots, p_{2j-2})$ to be the trail such that p_{2i-1} and p_{2i} are the port numbers of $\{v_i, v_{i+1}\}$ at v_i and v_{i+1}, respectively, for $i = 1, \dots, j-1$. We say that a trail T is *feasible from* v in G, if there exists a route R in G such that $b(R) = v$ and $\mathcal{T}(R) = T$, and in such a case the route R is denoted by $\mathcal{R}(v, T)$.

For a sequence $A = (a_1, \dots, a_k)$ we denote by $\overline{A}$ the sequence $(a_k, a_{k-1}, \dots, a_1)$. For two sequences $A = (a_1, \dots, a_k)$ and $B = (b_1, \dots, b_r)$ we write (A, B) to refer to the sequence $(a_1, \dots, a_k, b_1, \dots, b_r)$.

For any agent λ, let $h(\lambda)$ denote its initial position. Consider two agents λ and λ'. Consider any route R leading from $h(\lambda)$ to $h(\lambda')$ and let $T = \mathcal{T}(R)$. If $T = \overline{T}$, then we say that the route R is a palindrome. For a given initial configuration, a palindrome R is called *uniform*, if for any route R' such that $\mathcal{T}(R') = \mathcal{T}(R)$, whenever $b(R')$ is occupied by an agent, then $d(R')$ is also occupied by an agent.

We are now ready to formulate our condition on an initial configuration, that will be called EC (for eligibility condition) in the sequel:

Enhanced views of all agents are different *and* (There exist agents with different views *or* There exists a non-uniform palindrome)

We now formulate our main result whose proof is the objective of the rest of the paper.

Theorem 1 *Assume that all agents are provided with an upper bound n on the size of the graph. Then an initial configuration is eligible if and only if condition* EC *holds for this configuration. Moreover, there exists an algorithm electing a leader for all eligible configurations, regardless of the actions of the adversary.*

4 The negative results

In this section we prove two negative results concerning the feasibility of leader election. Impossibility results concerning breaking symmetry by anonymous agents navigating in anonymous networks are based on general ideas going back to Angluin [3]. In our scenario we use a perfectly synchronous adversary that forces agents to move in lock-step, and we prove that for each agent there is another agent that has the same memory state at each round, and consequently leader election is impossible because at least two leaders would have to be elected. A similar argument was used in [28] in the context of rendezvous, but in the present situation we need to deal with an additional problem that was absent in [28]: showing that the two agents have the same memory state after each of them meets other agents. This will have to be argued in the proof of Lemma 2.

The first result shows that condition EC is necessary to carry out leader election, even if the graph (and hence its size) is known to the agents.

Proposition 1 *Suppose that the condition* EC *does not hold for the initial configuration. Then there exists an adversary, such that leader election cannot be accomplished for this configuration, even if the graph is known to the agents.*

The proof of Proposition 1 is split into two lemmas. Condition EC can be abbreviated as $\alpha \wedge (\beta \vee \gamma)$, where α is “Enhanced views of all agents are different”, β is “There exist agents with different views”, and γ is “There exists a non-uniform palindrome”.

Lemma 1 *Condition α is necessary for leader election.*

Proof Fix an initial configuration. Suppose that α does not hold. This means that there exist agents λ and λ' with the same enhanced view. This in turn implies that for every agent μ there exists an agent μ' that has the same enhanced view as μ. Indeed, if T is the trail corresponding to a route that leads from $h(\lambda)$ to $h(\mu)$, agent μ' is the agent whose initial position is at the end of the route corresponding to T and starting at $h(\lambda')$. For every agent μ we will call the agent μ' its twin. Consider a hypothetical leader election algorithm and the "perfectly synchronous" adversary that starts the execution of the algorithm simultaneously for all agents and moves all of them with the same constant speed. Such an adversary induces rounds which are units of time in which all agents traverse an edge. The beginning of a round coincides with the end of the previous round. Hence at the beginning and at the end of each round every agent is at a node. If agents meet inside an edge, they must meet exactly in the middle of a round in which they traverse an edge in opposite directions. We will show that the memory state of twins is identical at the end of each round. This implies that leader election is impossible, as an agent elects a leader when it is at a node, and consequently if some agent elects itself as a leader, its twin would elect itself as well, violating the uniqueness of the leader.

The invariant that the memory state of twins is identical at the end of each round is proved by induction on the round number. It holds at the beginning, due to the same degree of initial positions of twins. Suppose that after some round i the memory states of twins are identical. Consider twins μ and μ'. In round $i+1$ they exit by the same port number and enter the next node by the same port number. If in round $i+1$ they don't meet any agent in the middle of the edge, at the end of the round μ must meet agents with the same memory states as those met by μ' (if any), and hence memory states of μ and μ' at the end of round $i+1$ are identical. If in round $i+1$ agent μ meets some agents in the middle of the edge, then agent μ' must meet exactly the twins of these agents in the middle of the edge. By the inductive hypothesis, these twins have the same memory states as agents met by μ and hence again, at the end of the round the memory states of μ and μ' are identical. This concludes the proof of the lemma. Notice that the argument holds even when agents know the graph in which they operate. □

Lemma 2 *Condition $\beta \vee \gamma$ is necessary for leader election.*

Proof Suppose that $\beta \vee \gamma$ is false. This means that views of all agents are identical and every palindrome for the initial configuration (if any) is uniform. For any trail π that yields a uniform palindrome, this gives a partition of all agents into pairs (μ_π, μ'_π) of agents at the ends of routes that correspond to this trail.

Again we consider the "perfectly synchronous" adversary described in the proof of Lemma 1. There are two subcases. If there is no palindrome in the initial configuration, then we prove the following invariant, holding at the beginning of each round, by induction on the round number: the memory state of all agents is the same and there is no palindrome between agents. The invariant holds at the beginning by assumption. Suppose it holds after round i. In round $i+1$ all agents choose the same port number and enter the next node by the same port number. There are no meetings in round $i+1$. Indeed, the only meeting could be in the middle of an edge but this would mean that agents were joined by a one-edge palindrome at the beginning of round $i+1$. If a pair of agents were joined by a palindrome after round $i+1$, they would have to be joined by a palindrome longer or shorter by two edges at the beginning of the round, contradicting the inductive assumption. Hence the invariant holds by induction.

The second subcase is when there is a palindrome in the initial configuration (and hence all such palindromes are uniform). Now we prove the following invariant holding in the beginning of each round: the memory state of all agents is the same and every agent is at the end of a palindrome corresponding to the same trail. The invariant holds at the beginning by the assumption. Suppose the invariant holds after round i. In round $i+1$ all agents choose the same port number and enter the next node by the same port number. If after round i no pair of agents were at the ends of an edge with both ports equal, or they were but agents did not choose this port in round $i+1$, then no meeting occurred and the invariant carries on after round $i+1$. If after round i every pair of agents were at the ends of an edge with both ports p and the agents chose this port in round $i+1$, then meetings of agents with identical memory states occurred in pairs in the middle of each joining edge. Since meeting agents had identical memory state during the meeting, this holds also after round $i+1$ and agents are again in pairs at the ends of edges with both ports p. Thus the invariant holds at the end of round $i+1$.

Hence at the beginning of each round the memory state of all agents is the same. This implies that with the "perfectly synchronous" adversary leader election is impossible. Notice that the argument holds even when agents know the graph in which they operate. □

Now Proposition 1 follows directly from Lemmas 1 and 2.

Our second negative result shows that the assumption about the knowledge of an upper bound on the size of the graph cannot be removed from Theorem 1.

Proposition 2 *There is no algorithm that accomplishes leader election regardless of the adversary for all initial configurations satisfying condition* EC.

Proof Suppose for a contradiction that such a universal algorithm $\mathtt{A}$ exists. Consider an "almost" oriented ring of size m: ports 0,1 are in the clockwise direction at each node except one, where they are counterclockwise. This node is called *special*. The initial configuration on this ring consists of two agents: one at the special node, and one at the neighbor clockwise from it. Call this configuration C_1 (cf. Fig. 1a). This

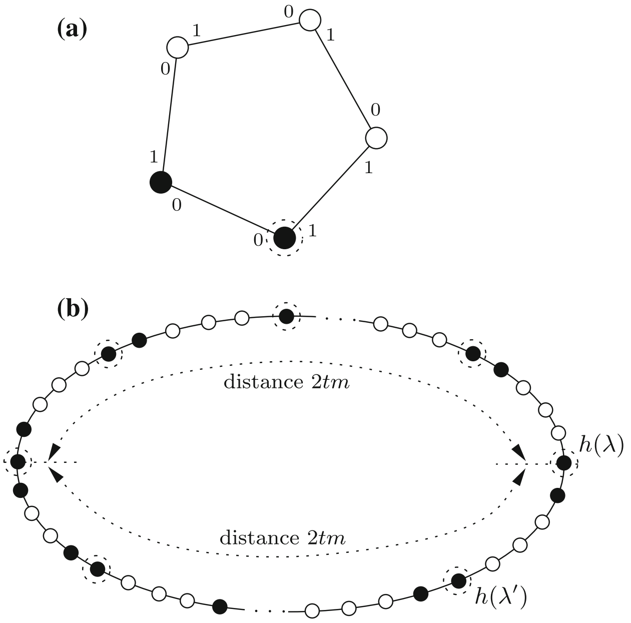


Fig. 1 **a** Configuration C_1 in a ring of size $m = 5$; **b** configuration C_2 in the corresponding ring of size $4tm$. Special nodes are *encircled*

configuration satisfies condition EC: agents have different views. Hence algorithm A must elect a leader for this configuration, regardless of the adversary. Consider a "perfectly synchronous" adversary that starts the execution of the algorithm simultaneously for all agents and moves all of them with the same constant speed. It induces rounds corresponding to edge traversals by all agents. Suppose that a leader is elected for this adversary after t rounds.

Now consider a ring of size $4tm$ in which there are $4t$ special nodes at distances m: at these nodes ports 0, 1 are in the counterclockwise direction, and in all other nodes they are in the clockwise direction. The initial configuration consists of $8t + 1$ agents. There is an agent at every special node and at every clockwise neighbor of a special node. Additionally there is an agent at the counterclockwise neighbor of one special node. Call this configuration C_2 (cf. Fig. 1b). This configuration satisfies condition EC. Indeed, due to the single group of three consecutive agents, all agents have distinct enhanced views. On the other hand, agents at special nodes have a different view from agents at clockwise neighbors of special nodes. Hence algorithm A must elect a leader for this configuration as well, regardless of the adversary. Consider the same "perfectly synchronous" adversary as before.

Consider the agent λ in the configuration C_2 that is initially situated at the special node v antipodal to the special node with both neighbors hosting agents. Consider the agent λ' initially situated at the special node v' that is clockwise from v and closest to v. In the first t rounds of the execution of A starting from configuration C_2, memory states of agents λ and λ' are identical to memory states of the agent μ initially situated at the special node of configuration C_1. This is due to the large size of the ring in configuration C_2. Hence if in configuration C_1 agent μ elects itself, then in configuration C_2 agents λ and λ' elect each of them itself as the leader after t rounds. If in configuration C_1 agent μ elects its neighbor, then in configuration C_2 agents λ and λ' elect each of them their neighbor as the leader after t rounds. In both cases two different agents are elected, which is a contradiction. □

5 The algorithm and its correctness

In this section we present an algorithm that elects a leader for all initial configurations satisfying condition EC, assuming that an upper bound on the size of the graph is known to all agents. In view of Proposition 2, this assumption cannot be removed. This upper bound, denoted by n, is an input of our algorithm.

The section is divided into three subsections. In the first subsection we provide additional terminology and notation, as well as some auxiliary results used in the algorithm and in its analysis. In the second subsection we give the intuitive overview of the algorithm and its formal description, and we provide some illustrative examples of its functioning. Finally, the third subsection is devoted to the proof that the algorithm is correct.

5.1 Additional notions and auxiliary results

Let G be a graph and let v be any node of G. For any integer $l > 0$, we define the *code* of $\mathcal{V}^l(v)$ as a sequence of integers denoted by $\xi(\mathcal{V}^l(v))$ and obtained as follows. Perform the complete depth first search traversal of $\mathcal{V}^l(v)$, starting at its root, in such a way that at each node an edge with a smaller port number is traversed prior to an edge with a larger port number. Let the ith traversed edge be from a node x_i to a node y_i, $i = 1, \dots, j$, where j is the number of edges traversed, and let p_i and p'_i be the port numbers of the edge $\{x_i, y_i\}$ at x_i and y_i, respectively. Then, the length of $\xi(\mathcal{V}^l(v))$ is $2j$, and the $(2i-1)$th and $(2i)$th elements of $\xi(\mathcal{V}^l(v))$ are p_i and p'_i, respectively, $i = 1, \dots, j$.

The following is a direct consequence of this definition.

Proposition 3 *Let u and v be any nodes of G and let $l > 0$ be an integer. $\mathcal{V}^l(u) \neq \mathcal{V}^l(v)$ if and only if $\xi(\mathcal{V}^l(u)) \neq \xi(\mathcal{V}^l(v))$.* □

Let $l > 0$ be an integer. We extend the notion of binary mappings to the truncated views. We say that f is a *binary mapping* for $\mathcal{V}^l(v)$, if f assigns either 0 or 1 to each node of $\mathcal{V}^l(v)$. If f is a binary mapping for $\mathcal{V}(v)$ (or for $\mathcal{V}^{l'}(v)$ for some $l' > 0$), then $(\mathcal{V}^l(v), f)$ (where $l \leq l'$, respectively) refers to f restricted to the nodes of $\mathcal{V}^l(v)$. Given two binary mappings f_1, f_2 for $\mathcal{V}^l(v)$, we write $f_1 \trianglelefteq f_2$

if $f_1(x) \leq f_2(x)$ for each node x of $\mathcal{V}^l(v)$. If $(\mathcal{V}(v), f)$ is the enhanced view from v and f' is a binary mapping for $\mathcal{V}^l(v)$ such that $f' \trianglelefteq f$, then the pair $(\mathcal{V}^l(v), f')$ is called a *partially enhanced view* from v. Intuitively, in a partially enhanced view only some nodes corresponding to initial positions of agents are marked.

Let $(\mathcal{V}(v), f)$ be the enhanced view from v, where v is selected so that there exists an agent λ with $h(\lambda) = v$. Then, $(\xi(\mathcal{V}^{n-1}(v)), f)$ is called the *complete identifier* of λ. The significance of the notion of a complete identifier is the following. An agent can never get the entire view or the entire enhanced view, as these are infinite objects. However, the following propositions from [40] show that to differentiate two views or two enhanced views, it is enough to consider their truncations to depth $n-1$. Thus, as stated in Corollary 1, complete identifiers identify agents with different enhanced views.

Proposition 4 ([40]) *For a n-node graph G and for all nodes u and v of G, $\mathcal{V}(u) = \mathcal{V}(v)$ if and only if $\mathcal{V}^{n-1}(u) = \mathcal{V}^{n-1}(v)$.* □

Proposition 5 ([40]) *For a n-node graph G, for all nodes u and v of G, if $(\mathcal{V}(u), f)$ and $(\mathcal{V}(v), f')$ are the enhanced views from u and v, respectively, then $(\mathcal{V}(u), f) = (\mathcal{V}(v), f')$ if and only if $(\mathcal{V}^{n-1}(u), f) = (\mathcal{V}^{n-1}(v), f')$.* □

Corollary 1 *For a n-node graph G and for any agents λ and λ', the enhanced views from $h(\lambda)$ and $h(\lambda')$ are equal if and only if the complete identifiers of λ and λ' are equal.* □

A sequence $\alpha = (\mathcal{C}, f_1, \ldots, f_j)$ is called a *label*, if the following conditions hold:

(i) $j > 0$ is an integer,
(ii) there exists a graph G' with at most n nodes and there exists a node v of G' such that $\mathcal{C}$ is the code of the truncated view to depth $3(n-1)$ from v in G',
(iii) f_i is a binary mapping for $\mathcal{V}^{3(n-1)}(v)$ for each $i = 1, \ldots, j$,
(iv) f_1 is the binary mapping for $\mathcal{V}^{3(n-1)}(v)$ that assigns 1 only to the root, and $f_i \trianglelefteq f_{i+1}$ for every index $i = 1, \ldots, j-1$.

Moreover, we say that j is the *length* of the label α, denoted by $\ell(\alpha)$. Let $\mathcal{L}_j$ be the set of all labels of length at most j. Note that if $j > 1$, then we do not impose any restrictions other than $f_{j-1} \trianglelefteq f_j$ on the binary mapping f_j. Also notice that the definition of α does not depend on G, but it depends on n.

5.2 The algorithm

In this section we give a high-level description of the algorithm and its pseudo-code formulation.

An important ingredient of the algorithm are meetings between agents during which information is exchanged. The method that guarantees that some meetings between pairs of agents will occur uses the idea of tunnels introduced in [18] and also used in [28] in the context of rendezvous of two anonymous agents. The routes $R = (e_1, \ldots, e_j)$ and R' form a *tunnel* if $R' = (e_i, e_{i-1}, \ldots, e_1, e'_1, \ldots, e'_{j'})$ for some $i \in \{1, \ldots, j\}$ and for some $j' \geq 0$. Moreover, we say that the route $(e_1, \ldots, e_i)$ is the *tunnel core with respect to* R. Note that if C is the tunnel core with respect to R, then $\overline{C}$ is the tunnel core with respect to R'.

Proposition 6 ([18]) *Let λ_i be an agent with route R_i, $i = 1, 2$. If R_1 and R_2 form a tunnel with the tunnel core $C = (e_1, \ldots, e_c)$, then λ_1 and λ_2 are guaranteed to have a meeting such that $(e_1, e_2, \ldots, e_i)$ and $(e_c, e_{c-1}, \ldots, e_i)$ are the routes of the agents traversed till the meeting, where $i \in \{1, \ldots, c\}$.* □

Informally speaking, if the routes of two agents form a tunnel, then they are guaranteed to have a meeting with the property that the routes traversed to date by the agents give (by taking one of the routes and the reversal of the other) the tunnel core.

Let $\mathcal{S}_n$ be the set of all integer sequences with terms in $\{0, \ldots, n-2\}$, whose length is even and equals at most $6(n-1)$. Then, we define

$$\mathcal{P}^n = \Big(\big(\alpha, \alpha', T\big) : \alpha, \alpha' \in \mathcal{L}_3 \quad \text{and} \quad \ell(\alpha) = \ell(\alpha') \quad \text{and} \\ T \in \mathcal{S}_n \quad \text{and} \quad (\alpha \neq \alpha' \vee (\alpha = \alpha' \wedge T = \overline{T}))\Big),$$

and let $\mathcal{P}^n_i$ be the ith triple in $\mathcal{P}^n$, $i = 1, \ldots, |\mathcal{P}^n|$.

In our leader election algorithm we will proceed in phases, and in each phase the label of each agent is fixed. (Due to the fact that the model is asynchronous, the adversary may force the agents to be in different phases in a particular point of time.) The total number of phases for each agent is 3. After the first phase each agent computes its label used in phase 2. These labels are defined in such a way that there exist two agents λ and λ' with different labels. The aim of phase 2 is that agents λ and λ' correctly identify each other's initial positions in their respective views. After phase 3 every agent can identify the initial positions of all agents in its view and hence is able to perform leader election.

The label of an agent λ used in phase p is denoted by $\alpha_p(\lambda)$, $p = 1, 2, 3$. Label $\alpha_1(\lambda)$ is computed before the start of phase 1, and $\alpha_{p+1}(\lambda)$ is computed at the end of phase p, for $p = 1, 2, 3$. Label $\alpha_4(\lambda)$ is used to elect the leader at the end of the algorithm. Each phase is divided into $|\mathcal{P}^n|$ stages. By $R_{p,s}(\lambda)$ we denote the route traversed by agent λ till the end of stage s in phase p, $p = 1, 2, 3$, $s = 1, \ldots, |\mathcal{P}^n|$. As we prove later, each agent λ starts and ends each stage at its initial position $h(\lambda)$. Let $R_{p,0}(\lambda)$ be the route of an agent λ traversed till the beginning of phase p, and hence till the end of phase $p-1$, whenever $p > 0$. Hence, $R_{1,0}(\lambda)$ is the route traversed by λ prior to the beginning of the first phase.

Now we give an informal description of Algorithm `Leader-Election`. This algorithm is executed by each agent, and λ in the pseudo-code is used to refer to the executing agent. Note that the upper bound n on the number of nodes of G is given as an input. The pseudo-code of the algorithm and pseudo-codes of its subroutines are in frames. In the informal description we refer to lines of these pseudo-codes.

First, we discuss Procedure `Initialization` that is called at the beginning of Algorithm `Leader-Election`. The agent starts by computing $\mathcal{P}^n$. This can be done knowing n, without any exploration of the graph because in order to construct $\mathcal{P}^n$ the agent needs to compute $\mathcal{S}_n$ and $\mathcal{L}_3$, and both of them (by the definition) depend only on the integer n. Then, the agent computes $\mathcal{V}^{3(n-1)}(h(\lambda))$ by performing a DFS traversal of G to the depth $3(n-1)$ (line 2). The function f^λ is set (line 3) to be the binary mapping for $\mathcal{V}^{3(n-1)}(h(\lambda))$ that assigns 0 to all nodes of $\mathcal{V}^{3(n-1)}(h(\lambda))$ except for the root. Hence, $(\mathcal{V}^{3(n-1)}(h(\lambda)), f^\lambda)$ is a partially enhanced view from $h(\lambda)$. The value of $\alpha_1(\lambda)$, that will be the label of λ in the first phase, is set to $(\xi(\mathcal{V}^{3(n-1)}(h(\lambda))), f^\lambda)$ (line 4).

```
Procedure Initialization($n$)
    Input: An upper bound $n$ on the size of $G$.
  begin
1:    Compute $\mathcal{P}^n$.
2:    Compute $\mathcal{V}^{3(n-1)}(h(\lambda))$ by performing a DFS traversal of graph $G$ to depth $3(n-1)$ that ends at $h(\lambda)$.
3:    Let $f^\lambda$ be the binary mapping for $\mathcal{V}^{3(n-1)}(h(\lambda))$ that assigns 1 only to the root of $\mathcal{V}^{3(n-1)}(h(\lambda))$.
4:    $\alpha_1(\lambda) \leftarrow (\xi(\mathcal{V}^{3(n-1)}(h(\lambda))), f^\lambda)$
  end Initialization
```

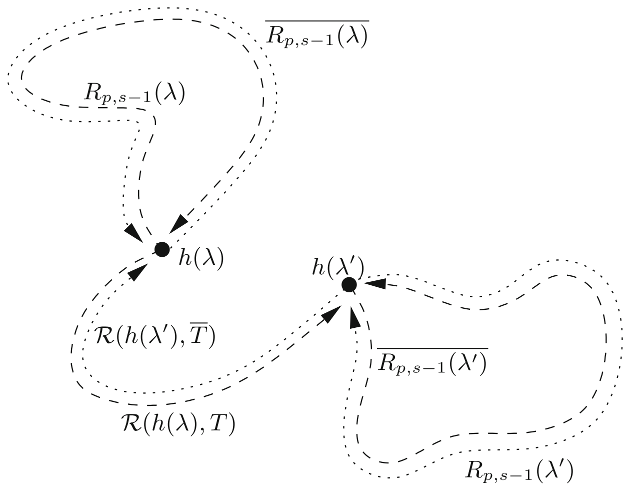
Now we informally describe the main part of Algorithm `Leader-Election`, refering to the lines of the pseudo-code given below. The pth iteration of the main 'for' loop in lines 2–13 is responsible for the traversal performed by the agent in phase p, $p \in \{1, 2, 3\}$. An internal 'for' loop in lines 3–10 is executed and its sth iteration determines the behavior of λ in stage s. The stage s of each phase 'processes' the sth element (α', α'', T) of $\mathcal{P}^n$. If $\alpha_p(\lambda) \notin \{\alpha', \alpha''\}$, then the agent λ does not move in this stage and proceeds to the next one. Otherwise let $\alpha_p(\lambda) = \alpha'$ (we describe only this case as the other one is symmetric). The agent checks in line 5 whether a certain trail (T, H) is feasible from $h(\lambda)$, and if it is not, then the stage ends. As we prove later, the verification of the feasibility of (T, H) can be done by inspecting $\mathcal{V}^{3(n-1)}(h(\lambda))$. If (T, H) is feasible from $h(\lambda)$, then λ follows the route $\mathcal{R}(h(\lambda), (T, H, \overline{T}))$ (line 6), which guarantees that:

- if the agent λ is located at $h(\lambda)$ at the beginning of stage s, then λ is located at $h(\lambda)$ at the end of stage s (see Lemma 5), and
- if the route $\mathcal{R}(h(\lambda), T)$ leads from $h(\lambda)$ to the initial position of another agent λ' and $\alpha_p(\lambda') = \alpha''$, then the routes $R_{p,s}(\lambda)$ and $R_{p,s}(\lambda')$ form a tunnel.

The agent ends phase p by updating its label. This is done by calling Function `Update-Label` in line 11, which produces a binary mapping f^λ that is used to update the label in line 12. After completing the tree phases (for $p = 1, 2, 3$) the agent calls Procedure `Choose-Leader` which completes the task of leader election.

```
Algorithm Leader-Election($n$)
    Input: An upper bound $n$ on the size of $G$.
  begin
1:    Call Initialization($n$).
2:    for $p \leftarrow 1$ to 3 do
3:        for $s \leftarrow 1$ to $|\mathcal{P}^n|$ do
4:            $(\alpha', \alpha'', T) \leftarrow \mathcal{P}^n_s$
5:            if $\alpha_p(\lambda) = \alpha'$ and $(T, \overline{\mathcal{H}(\alpha'', s-1)})$ is feasible from $h(\lambda)$ then
6:                Follow the route $\mathcal{R}(h(\lambda), (T, \overline{\mathcal{H}(\alpha'', s-1)}, \overline{T}))$.
7:            else if $\alpha_p(\lambda) = \alpha''$ and $(\overline{T}, \overline{\mathcal{H}(\alpha', s-1)})$ is feasible from $h(\lambda)$ then
8:                Follow the route $\mathcal{R}(h(\lambda), (\overline{T}, \overline{\mathcal{H}(\alpha', s-1)}, T))$.
9:            end if
10:       end for
11:       $f^\lambda \leftarrow$ Update-Label($M$), where $M$ is the memory state of $\lambda$.
12:       $\alpha_{p+1}(\lambda) \leftarrow (\alpha_p, f^\lambda)$
13:   end for
14:   Call Choose-Leader.
  end Leader-Election
```

In order to formally describe the trail H mentioned above we need the following notation. Let $\alpha' \in \mathcal{L}_3$ be a label of length $p \in \{1, 2, 3\}$, let $s \in \{1, \ldots, |\mathcal{P}^n|\}$ and let v be a node of G. We define $\mathcal{H}(\alpha', s)$ to be the trail that corresponds to the route performed till the end of stage s of phase p by an agent λ' whose label equals α' in phase p, $\alpha_p(\lambda') = \alpha'$, and whose initial position corresponds to the root of the truncated view $\mathcal{V}^{3(n-1)}(h(\lambda'))$ in α'. We prove (see Lemma 4) that, for any α and s the trail $\mathcal{H}(\alpha, s)$ can be computed on the basis of α and s. The trail H mentioned above is $\overline{\mathcal{H}(\alpha'', s-1)}$.



$\overline{R_{p,s-1}(\lambda)}$
$R_{p,s-1}(\lambda)$
$h(\lambda)$
$h(\lambda')$
$\mathcal{R}(h(\lambda'),\overline{T})$
$\overline{R_{p,s-1}(\lambda')}$
$\mathcal{R}(h(\lambda),T)$
$R_{p,s-1}(\lambda')$

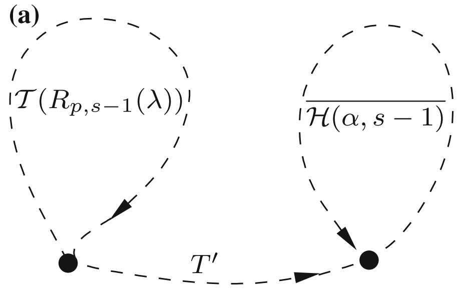
Fig. 2 The routes $(R_{p,s-1}(\lambda), \mathcal{R}(h(\lambda), T), \overline{R_{p,s-1}(\lambda')})$ (*dashed route*) and $(R_{p,s-1}(\lambda'), \mathcal{R}(h(\lambda'), \overline{T}), \overline{R_{p,s-1}(\lambda)})$ (*dotted route*)

Figure 2 illustrates the routes of a pair of agents λ and λ' that execute one iteration of the internal 'for' loop in lines 3–10 of `Leader-Election`, for the same values of p and s, such that $(\alpha_p(\lambda), \alpha_p(\lambda'), T) = \mathcal{P}^n_s$. We assume in this example that $\mathcal{R}(h(\lambda), T)$ leads from $h(\lambda)$ to $h(\lambda')$ in G. Figure 2 gives the prefixes of two routes $R_{p,s}(\lambda)$ and $R_{p,s}(\lambda')$ traversed by the two agents. The fact that the routes form a tunnel (as shown in Fig. 2) follows from Lemma 7 proven later. The routes in Fig. 2 are extended to $R_{p,s}(\lambda)$ and $R_{p,s}(\lambda')$ once λ goes from $h(\lambda')$ to its initial position $h(\lambda)$ by following the route $\mathcal{R}(h(\lambda'), \overline{T})$ in G, and λ' follows $\mathcal{R}(h(\lambda), T)$ in order to return to $h(\lambda')$, respectively. Note that the routes $R_{p,s-1}(\lambda)$, $R_{p,s-1}(\lambda')$ and $\mathcal{R}(h(\lambda), T)$ are not edge disjoint.

Although we use tunnels, we use them differently than in [18,28]. First, we are able to construct much simpler routes that form tunnels. This is due to the fact that [18,28] deals with the rendez-vous problem in finite graphs of unknown size and in infinite graphs. As argued before, for leader election we have to assume that an upper bound n on the size of the graph is known, and we take advantage of knowing n to construct 'shorter' tunnels which simplifies our analysis. Second and more importantly, it is not sufficient for our purposes to just generate a meeting for a particular pair of agents—the meetings are generated to perform the exchange of information. In particular, as a result of a meeting that occurs in a tunnel an agent should be able to determine the node (in its own view) corresponding to the initial position of the other agent. This leads us to the following concept of meetings with 'confirmation' of a trail.

Definition 1 Let $T \in \mathcal{S}_n$. Suppose that agents λ and λ' meet. We say that λ *confirms* T as a result of this meeting if

(i) λ is in stage s of phase p and λ' is in stage s' of phase p', where $p' < p$, or $p' = p$ and $s' \le s$,

(a)


$\mathcal{T}(R_{p,s-1}(\lambda))$
$\overline{\mathcal{H}(\alpha,s-1)}$
T'

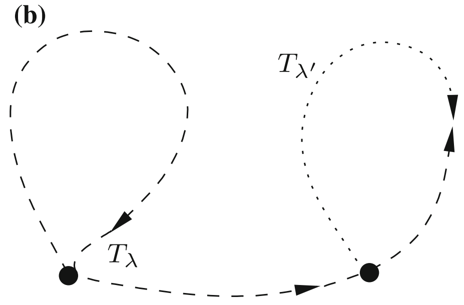
(b)


Fig. 3 **a** The trail $(\mathcal{T}(R_{p,s-1}(\lambda)), T', \overline{\mathcal{H}(\alpha', s-1)})$; **b** a meeting of λ and λ' in case when condition (iii) in Definition 1 is satisfied

(ii) $\mathcal{P}^n_s = (\alpha_p(\lambda), \alpha', T')$, where $T' = T$, or $\mathcal{P}^n_s = (\alpha', \alpha_p(\lambda), T')$, where $T' = \overline{T}$,

(iii) if T_λ and $T_{\lambda'}$ are the trails traversed by λ and λ', respectively, till the meeting, then

$$(\mathcal{T}(R_{p,s-1}(\lambda)), T', \overline{\mathcal{H}(\alpha', s-1)}) = (T_\lambda, \overline{T_{\lambda'}}).$$

As we prove in Sect. 5.3, if an agent λ confirms T as a result of a meeting with λ', then $d(\mathcal{R}(h(\lambda), T')) = h(\lambda')$.

Figure 3 depicts the equation in part (iii) of Definition 1. Figure 3a presents the trail

$$\left(\mathcal{T}(R_{p,s-1}(\lambda)), T', \overline{\mathcal{H}(\alpha', s-1)}\right)$$

that is a prefix of the trail corresponding to the route $R_{p,s}(\lambda)$ followed by the agent λ till the end of stage s of phase p. Figure 3b presents the trails T_λ (dashed line) and $T_{\lambda'}$ (dotted line) in the case of the meeting that occurs when λ is in stage s of phase p and λ' has not completed the traversal of its route till the end of stage $s-1$ of phase p.

It remains to describe Function `Update-Label` and Procedure `Choose-Leader` that are called in the Algorithm `Leader-Election`.

We start by giving intuition of the first of them. This procedure is crucial for the entire algorithm, as it takes advantage of memory state exchanges between agents that meet and permits every agent to insert initial positions of all agents in its view. This in turn allows the agents to learn asymmetries in the initial configuration and thus correctly perform leader election. Function `Update-Label` takes as an input the current memory state of an agent λ and returns a binary mapping

```
Function Update-Label($M$)
    Input: A memory state $M$ of an agent $\lambda$.
    Output: A binary mapping for $\mathcal{V}^{3(n-1)}(h(\lambda))$.
begin
 1:  Let $f^\lambda$ be the binary mapping for $\mathcal{V}^{3(n-1)}(h(\lambda))$ that assigns 1 only to the root of $\mathcal{V}^{3(n-1)}(h(\lambda))$.
 2:  Let $M_1, \ldots, M_j$ be the memory states of all agents $\lambda_1, \ldots, \lambda_j$ previously met by $\lambda$, at the times of the respective
     meetings.
 3:  for $i \leftarrow 1$ to $j$ do
 4:      Let $T' = T$ if $\lambda$ confirms $T$ and let $T' = \overline{T}$ if $\lambda_i$ confirms $T$ as a result of their meeting.
 5:      if $\lambda$ or $\lambda_i$ confirms $T$ then
 6:          $f^\lambda(x) \leftarrow 1$, where $x$ is the node of $\mathcal{V}^{3(n-1)}(h(\lambda))$ at the end of $T'$ from the root.
 7:          if the length of $\mathcal{R}(h(\lambda), T)$ is at most $n-1$ then
 8:              $f' \leftarrow$ Update-Label($M_i$)
 9:              Compute transition $\varphi$ from $\mathcal{V}^{2(n-1)}(h(\lambda))$ to $\mathcal{V}^{3(n-1)}(v)$ such that $\varphi(x)$ is the root of $\mathcal{V}^{3(n-1)}(v)$.
10:              for each $y \in \mathcal{V}^{2(n-1)}(v)$ such that $f'(\varphi(y)) = 1$ do
11:                  $f^\lambda(y) \leftarrow 1$
12:          end if
13:      end if
14:  end for
15:  return $f^\lambda$
end Update-Label
```

f^λ for its view $\mathcal{V}^{3(n-1)}(h(\lambda))$, such that $(\mathcal{V}^{3(n-1)}(h(\lambda)), f^\lambda)$ is a partially enhanced view for agent λ. Agent λ considers memory states $M_1, \ldots, M_j$ of all previously met agents at the times of the meetings. The memory state M_i, $i \in \{1, \ldots, j\}$, of an agent λ' and the memory state of λ at the time of their meeting permit the agent λ to verify whether λ or λ' confirmed T as a result of their meeting. If one of the agents confirms T, then λ takes the advantage of this fact to determine the nodes of its view corresponding to initial positions of agents. In particular, λ is able to locate a node in its own view that corresponds to the initial position of λ', because there exists a route corresponding to T and connecting the initial positions of the two agents. Afterwards, if this route is of length at most $n-1$, then λ recursively calls Function `Update-Label` for the memory state M_i (which is shorter than the current memory state of λ and thus recursion is correct). Hence, λ can compute the binary mappings corresponding to memory states of all previously met agents at the times of the meetings. Using trails between initial positions of these agents and $h(\lambda)$, as well as the obtained binary mappings, agent λ can correctly position all partially enhanced views of these agents in its own view. A call to Function `Update-Label` at the end of phase p executed by agent λ permits to compute $\alpha_{p+1}(\lambda)$.

In the formulation of Function `Update-Label` we use the following notions. Let u and v be two nodes of G. We say that a function φ assigning to each node of $\mathcal{V}^{2(n-1)}(u)$ a node of $\mathcal{V}^{3(n-1)}(v)$ is a *transition* from $\mathcal{V}^{2(n-1)}(u)$ to $\mathcal{V}^{3(n-1)}(v)$, if $\varphi(x)$ and x correspond to the same node of G for each node x of $\mathcal{V}^{2(n-1)}(u)$. For any trail T and any node v, we say that a node x at depth i in $\mathcal{V}(v)$ is at the end of T from the root, if the length of T is $2i$ and the sequence of ports corresponding to the simple path from the root of $\mathcal{V}(v)$ to x is T.

In the following example we illustrate one iteration of the ‘for’ loop in lines 3–14 of Function `Update-Label`. The graph G is given in Fig. 4a, and let $n = 4$ be an upper bound that was initially provided to each agent. The black nodes of G are the initial positions of some agents. Denote by λ_a, λ_b and λ_c the agents whose initial positions are a, b and c, respectively. Note that the views from any two nodes of G are identical in this case. However, the enhanced view from each node of G is unique. We focus on the instance of Function `Update-Label` executed by λ_c during its meeting with λ_b. For simplicity, we show only some subtrees of $\mathcal{V}^{3(n-1)}(c)$ and $\mathcal{V}^{3(n-1)}(b)$ in Fig. 4b, c, respectively. Note that the nodes of G, and therefore the nodes of any view, are unlabeled and we provide the labels only for the illustrative purpose. In this example the trail T', computed in line 4 of Function `Update-Label` equals (0, 0, 1, 1), which determines the node of the truncated view $\mathcal{V}^{3(n-1)}(c)$ that corresponds to the root of $\mathcal{V}^{3(n-1)}(b)$. The black nodes of both views correspond to the initial positions of agents that λ_c and λ_b determined prior to this meeting. The dotted arrows give the part of the transition φ that maps the nodes of $\mathcal{V}^{2(n-1)}(c)$

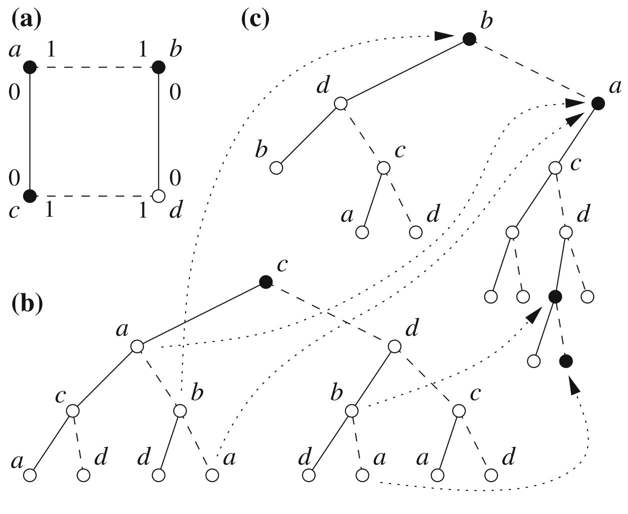


Fig. 4 **a** A graph G; **b** $\mathcal{V}^{n-1}(c)$; **c** a subtree of $\mathcal{V}^{3(n-1)}(b)$

to the black nodes of $\mathcal{V}^{3(n-1)}(b)$. It follows from the definition of the view that, in general, more than one node of $\mathcal{V}^{n-1}(h(\lambda))$ can be mapped by φ to a node of $\mathcal{V}^{3(n-1)}(v)$.

We finally present Procedure Choose-Leader that is called by Algorithm Leader-Election after the third phase. The leader is selected by an agent λ on the basis of the label $\alpha_4(\lambda) = (\xi(\mathcal{V}^{3(n-1)}(h(\lambda))), f_1^\lambda, \ldots, f_4^\lambda)$. Let x be a node at depth at most $n-1$ in $\mathcal{V}^{3(n-1)}(h(\lambda))$ and satisfying $f_4^\lambda(x) = 1$. Let S be a subtree of depth $n-1$ of $\mathcal{V}^{3(n-1)}(h(\lambda))$ rooted at x. We prove later that the pair (S, f_4^λ) is the complete identifier of some agent. Since the initial positions of all agents have been detected till the end of phase 3, the agent can determine all complete identifiers and hence elect the leader.

```
Procedure Choose-Leader
begin
1:  $\mathcal{A} \leftarrow \emptyset$
2:  for each node $x$ of $\mathcal{V}^{n-1}(h(\lambda))$ such that $f_4^\lambda(x) = 1$ do
3:      Let $f'$ be $f_4^\lambda$ restricted to the nodes of $\mathcal{V}^{n-1}(v)$, where $v$ corresponds to $x$.
4:      Compute trail $T$ such that $v = d(\mathcal{R}(h(\lambda), T))$.
5:      if $(\xi(\mathcal{V}^{n-1}(v)), f') \neq I$ for each $I$ such that $(I, T) \in \mathcal{A}$ then
6:          $\mathcal{A} \leftarrow \mathcal{A} \cup \{((\xi(\mathcal{V}^{n-1}(v)), f'), T)\}$
7:  end for
8:  Find $(I, T) \in \mathcal{A}$ such that $I = \min\{I' : (I', T') \in \mathcal{A}\}$, where min is in lexicographic order.
9:  Elect the agent whose initial position is $d(\mathcal{R}(h(\lambda), T))$ to be the leader.
end Choose-Leader
```

5.3 Correctness of the algorithm

This section is devoted to the proof that Algorithm Leader-Election correctly elects a leader whenever an initial configuration satisfies condition EC, regardless of the actions of the adversary. The proof is split into a series of lemmas. The role of the first lemma is to show that knowing $\mathcal{V}^{2n-1}(v)$ is enough to check if a trail of any length is feasible from v.

Lemma 3 *Let v be any node of G. Using $\mathcal{V}^{2n-1}(v)$, the truncated view $\mathcal{V}^l(v)$ can be computed for any positive integer l.*

Proof If $l \leq 2n-1$, then $\mathcal{V}^l(v)$ is a subtree of $\mathcal{V}^{2n-1}(v)$, so we may assume that $l > 2n-1$. We extend the view $\mathcal{V}^i(v)$ to $\mathcal{V}^{i+1}(v)$ for each $i = 2n-1, \ldots, l-1$. To this end we perform the following computation for each node x at depth $i-(n-1)$ of $\mathcal{V}^i(v)$. Let u be the node of G that corresponds to x. Note that the subtree of $\mathcal{V}^i(v)$ rooted at x and containing all descendants of x in $\mathcal{V}^i(v)$ is equal to $\mathcal{V}^{n-1}(u)$. Hence, there exists a node x' in $\mathcal{V}^{n-1}(v)$ such that x' corresponds to u, in view of the connectedness of G. This implies that there exists a node y in $\mathcal{V}^{n-1}(v)$ such that the subtree of $\mathcal{V}^{2n-1}(v)$ consisting of y and all its descendants to depth $n-1$ from y is equal to $\mathcal{V}^{n-1}(u)$. Hence, one can find any such node y of $\mathcal{V}^{n-1}(v)$. Let u' be the node of G that corresponds to y. Due to Proposition 4, $\mathcal{V}^n(u')$ is equal to $\mathcal{V}^n(u)$. Since y belongs to $\mathcal{V}^{n-1}(v)$ we obtain that $\mathcal{V}^n(u')$ is a subtree of $\mathcal{V}^i(v)$ rooted at y, and therefore we can extend the subtree $\mathcal{V}^{n-1}(u)$ by replacing in $\mathcal{V}^i(v)$ the subtree rooted at x with $\mathcal{V}^n(u')$. □

Corollary 2 *Let v be any node of G and let x be any node of $\mathcal{V}^{2n-1}(v)$. Then, the subtree of $\mathcal{V}(v)$ to depth l and rooted at x can be computed for any $l > 0$, using $\mathcal{V}^{2n-1}(v)$.* □

Corollary 3 *Let T be any trail and let v be any node of G. Using $\mathcal{V}^{2n-1}(v)$ it can be verified if T is feasible from v in G.* □

The next lemma shows that given any label α and the stage number s, it is possible to compute the trail $\mathcal{H}(\alpha, s)$ which, informally speaking, is the history of the moves of the agent with label α till this stage.

Lemma 4 *Using a label $\alpha \in \mathcal{L}_3$ and an integer $s \in \{1, \ldots, |\mathcal{P}^n|\}$, the trail $\mathcal{H}(\alpha, s)$ can be computed.*

Proof Suppose that α is of length p, $p \in \{1, 2, 3\}$. By definition, $\alpha = (\xi(\mathcal{V}^{3(n-1)}(v)), f_1, f_2, \ldots, f_p)$, where f_i is a binary mapping for $\mathcal{V}^{3(n-1)}(v)$ and v is a node of G. First note that $\mathcal{V}^{3(n-1)}(v)$ can be reconstructed from its code. Suppose that λ is an agent whose initial position is v and whose label in phase p is α, $\alpha_p(\lambda) = \alpha$.

The trail $\mathcal{H}(\alpha, s)$ can be computed by simulating the execution of Algorithm Leader-Election for the agent

λ. By its formulation, the agent λ executed Procedure `Initialization`, $p-1$ iterations of the main 'for' loop in lines 2–13 of Algorithm `Leader-Election`, and exactly s iterations of the nested 'for' loop in lines 3–10 of Algorithm `Leader-Election` in the pth iteration of the main 'for' loop. Thus results in traversing the route $\mathcal{R}(h(\lambda), \mathcal{H}(\alpha, s))$. Note that G is unknown to λ, but we will reconstruct the route by simulating edge traversals in $\mathcal{V}(v)$. (While $\mathcal{V}(v)$ is infinite, it can be reconstructed from $\mathcal{V}^{3(n-1)}(v)$ to any finite depth, using Corollary 3.)

We prove the lemma by induction on the total number of stages 'processed' by an agent. Note that if $\alpha \in \mathcal{L}_1$ and $s=0$, then the trail $\mathcal{H}(\alpha, s)$ corresponds to the route that is the DFS traversal of G to depth $3(n-1)$ and starting at v. This trail can be obtained by performing the DFS traversal of $\mathcal{V}^{3(n-1)}(v)$ that starts and ends at the root.

Now assume that $s>0$. In order to simulate the behavior of λ in any stage j, $j \in \{1, \ldots, |\mathcal{P}^n|\}$, of phase i, $i \in \{1, \ldots, p\}$, one needs to know $\alpha_i(\lambda)$. By construction, $\alpha_i(\lambda) = (\xi(\mathcal{V}^{3(n-1)}(v)), f_1, \ldots, f_i)$. The induction hypothesis and Corollary 3 imply that the second part of the condition in line 5 of Algorithm `Leader-Election` can be checked. If $\alpha_i(\lambda) \notin \{\alpha', \alpha''\}$, where $(\alpha', \alpha'', T) = \mathcal{P}^n_j$, then λ does not move in stage j of phase i. Hence, assume without loss of generality that $\alpha_i(\lambda) = \alpha'$. This implies that λ executes the instruction in line 6 of Algorithm `Leader-Election`. By the induction hypothesis, the trail $\mathcal{H}(\alpha'', j-1)$ in lines 5 and 6 can be computed on the basis of α'' and j. $\square$

We say that a route R is *closed* if $b(R) = d(R)$.

The following lemma implies that at the end of each stage each agent comes back to its initial position.

Lemma 5 *Let λ be any agent. For every $p \in \{1, 2, 3\}$ and for every $s \in \{0, 1, \ldots, |\mathcal{P}^n|\}$ the route $R_{p,s}(\lambda)$ is closed.*

Proof Denote by $R'_{p,s}(\lambda)$ the route that the agent λ follows in stage s, $s \in \{1, \ldots, |\mathcal{P}^n|\}$, of phase p, $p \in \{1, 2, 3\}$. We prove the lemma by induction on the total number of stages processed in all phases by an agent.

First note that $R_{1,0}(\lambda)$, i.e., the route of λ performed as a result of the execution of line 2 of Procedure `Initialization` is closed. Hence, it remains to prove that if $R_{p,s}(\lambda')$ is closed for each agent λ', for some $p \in \{1, 2, 3\}$ and for some $s \in \{0, \ldots, |\mathcal{P}^n|-1\}$, then $R_{p,s+1}(\lambda)$ is closed as well. Note that $R_{p,s+1}(\lambda) = (R_{p,s}(\lambda), R'_{p,s+1}(\lambda))$. Hence, $b(R'_{p,s+1}(\lambda)) = d(R_{p,s}(\lambda)) = h(\lambda)$ and therefore it is enough to argue that $R'_{p,s+1}(\lambda)$ is closed.

Let $\mathcal{P}^n_{s+1} = (\alpha', \alpha'', T)$. If $\alpha_p(\lambda) \notin \{\alpha', \alpha''\}$, then according to lines 5 and 7 of Algorithm `Leader-Election`, $R'_{p,s+1}(\lambda')$ is empty, i.e., λ does not move in stage $s+1$ of phase p. In this case the proof is completed. Otherwise, we obtain that

$$R'_{p,s+1}(\lambda) = \mathcal{R}\left(h(\lambda), (T', \overline{\mathcal{H}(\alpha, s)}, \overline{T'})\right),$$

where $T' \in \{T, \overline{T}\}$ and $\alpha \in \{\alpha', \alpha''\}$ (see lines 6 and 8 of Algorithm `Leader-Election`). Let $u = d(\mathcal{R}(h(\lambda), T'))$. Hence, $R'_{p,s+1}(\lambda)$ is closed if and only if $\mathcal{R}(u, \mathcal{H}(\alpha, s))$ is closed. However, by definition, the latter route equals $R_{p,s}(\lambda')$ for an agent λ' such that its initial position is u and $\alpha_p(\lambda') = \alpha$ (if such an agent exists). It follows from the induction hypothesis that $R_{p,s}(\lambda')$ is closed, which completes the proof of the lemma. $\square$

The next lemma shows the importance of confirmation of a trail. It implies that if an agent λ confirms T as a result of a meeting with λ', then it can correctly situate the initial position of λ' in its view.

Lemma 6 *Let $p \in \{1, 2, 3\}$, let $s \in \{1, \ldots, |\mathcal{P}^n|\}$ and let $\mathcal{P}^n_s = (\alpha', \alpha'', T)$. Suppose that the agent λ meets an agent λ', when λ is in stage s of phase p. If $\alpha_p(\lambda) = \alpha'$ and λ confirms $T' = T$, or if $\alpha_p(\lambda) = \alpha''$ and λ confirms $T' = \overline{T}$ as a result of this meeting, then $h(\lambda') = d(\mathcal{R}(h(\lambda)), T')$.*

Proof Suppose without loss of generality that $\alpha' = \alpha_p(\lambda)$. Condition (iii) in Definition 1 implies that the route $R = (R_{p,s-1}(\lambda), \mathcal{R}(h(\lambda), (T', \overline{\mathcal{H}(\alpha'', s-1)})))$ leads from $h(\lambda)$ to $h(\lambda')$ in G. Let $u = d(\mathcal{R}(h(\lambda), T'))$. By Lemma 5, both $R_{p,s-1}(\lambda)$ and $\mathcal{R}(u, \overline{\mathcal{H}(\alpha'', s-1)})$ are closed. This implies that $h(\lambda') = d(R) = d(\mathcal{R}(h(\lambda), T'))$ as required. $\square$

The following lemma shows that processing an appropriate triple (α, α', T) by two agents guarantees their meeting confirming T.

Lemma 7 *Let $p \in \{1, 2, 3\}$. Let λ and λ' be two agents such that $\mathcal{P}^n_s = (\alpha_p(\lambda), \alpha_p(\lambda'), T)$ for some $T \in \mathcal{S}_n$ and $s \in \{1, \ldots, |\mathcal{P}^n|\}$. If $d(\mathcal{R}(h(\lambda), T)) = h(\lambda')$, then prior to the first moment when one of the agents completes phase p, the agents λ and λ' have a meeting as a result of which either λ confirms T or λ' confirms $\overline{T}$.*

Proof Suppose without loss of generality that λ ends the traversal of $R_{p,s-1}(\lambda)$ at the same time or earlier than λ' ends the traversal of $R_{p,s-1}(\lambda')$. Let $R'_{p,s}(\lambda)$ be the route traversed by λ in stage s of phase p. By Lemma 5, $b(R'_{p,s}(\lambda)) = d(R_{p,s-1}(\lambda)) = h(\lambda)$. The route $R'_{p,s}(\lambda)$ is constructed as a result of the execution of lines 5–6 of Algorithm `Leader-Election` by λ. Since $\alpha'' = \alpha_p(\lambda')$ in line 6 of Algorithm `Leader-Election`, we obtain that $R'_{p,s}(\lambda)$ and $R_{p,s}(\lambda')$ form a tunnel with the tunnel core $C = (\mathcal{R}(h(\lambda), (T, \overline{\mathcal{H}(\alpha_p(\lambda'), s-1)})))$. By Proposition 6, λ and λ' will have a meeting while λ is in stage s of phase p and before λ' ends the traversal of $R_{p,s}(\lambda')$. This implies that (i) of Definition 1 holds. Moreover, (ii) of Definition 1 for $\alpha' = \alpha_p(\lambda')$ and $T' = T$ is satisfied by assumption. Let T_λ and $T_{\lambda'}$ be the trails traversed by

λ and λ', respectively, till the meeting. By Proposition 6, $(\mathcal{T}(R_{p,s-1}(\lambda)), \mathcal{T}(C)) = (T_\lambda, \overline{T_{\lambda'}})$, where C is the tunnel core. This proves that (iii) of Definition 1 holds. Hence, the agent λ confirms T as a result of the meeting. □

The role of the next lemma is to show that an agent never marks falsely an initial position of another agent in its view.

Lemma 8 *Let $\alpha_p(\lambda) = (\xi(\mathcal{V}^{3(n-1)}(h(\lambda))), f_1^\lambda, \ldots, f_p^\lambda)$ be the label of any agent λ in phase $p \in \{1,2,3\}$. If z is any node of $\mathcal{V}^{3(n-1)}(h(\lambda))$ corresponding to a node of G that is not an initial position of an agent, then $f_p^\lambda(z) = 0$.*

Proof Suppose for a contradiction that an agent λ sets $f^\lambda(z)$ to be 1, and z corresponds to a node of G that is not an initial position of an agent.

Suppose that the input memory state M of the agent λ is the shortest that satisfies this property. This assumption implies that if f' is computed by agent λ in line 8 of Function `Update-Label` for any memory state $M_i, i = 1, \ldots, j$, then $(\mathcal{V}^{3(n-1)}(v), f')$ is a partially enhanced view from v, where v corresponds to the node x at the end of T' from the root in $\mathcal{V}^{3(n-1)}(h(\lambda))$. Hence, if v is an initial position of an agent, then, due to the definition of transition, each node y from line 11 of Function `Update-Label` corresponds to an initial position of an agent.

The latter implies that the node x in line 6 of Function `Update-Label` does not correspond to the initial position of λ_i. By construction, x is at the end of T' from the root in $\mathcal{V}^{3(n-1)}(h(\lambda))$. Due to line 5 of Function `Update-Label`, either λ or λ_i confirms $T' \in \{T, \overline{T}\}$ as a result of their meeting. By Lemma 6, $d(h(\lambda), T') = h(\lambda_i)$, where T' is determined in line 4 of `Update-Label`. Thus, $h(\lambda_i)$ corresponds to the node at the end of T' from the root in $\mathcal{V}^{3(n-1)}(h(\lambda))$. The latter implies that $h(\lambda_i)$ corresponds to x, a contradiction. □

The next lemma is a companion result to Lemma 8. It says that if an agent confirms a trail T as a result of a meeting with λ', then both of them correctly mark their respective initial positions in their views.

Lemma 9 *Let $p \in \{1,2,3\}$. Let λ and λ' be two agents with labels $\alpha_{p+1}(\lambda) = (\alpha_p(\lambda), f_{p+1}^\lambda)$ and $\alpha_{p+1}(\lambda') = (\alpha_p(\lambda'), f_{p+1}^{\lambda'})$. If agent λ confirms T as a result of a meeting with agent λ' in phase p', $p' \le p$, then*

(i) $f_{p+1}^\lambda(x) = 1$, where x is at the end of T from the root in $\mathcal{V}^{3(n-1)}(h(\lambda))$ and $d(\mathcal{R}(h(\lambda), T)) = h(\lambda')$.
(ii) $f_{p+1}^{\lambda'}(x') = 1$, where x' is at the end of $\overline{T}$ from the root in $\mathcal{V}^{3(n-1)}(h(\lambda'))$ and $d(\mathcal{R}(h(\lambda'), \overline{T})) = h(\lambda)$.

Proof Suppose that λ is in some stage of phase p', $p' \le p$, when a meeting with λ' occurs as a result of which λ confirms T. Hence, one of the memory states in line 2 of Function `Update-Label` called at the end of phase p is the memory state $M_i, i \in \{1, \ldots, j\}$, of λ' at the time of the meeting. We consider the ith iteration of the 'for' loop in lines 3–14 of Function `Update-Label`, i.e., informally speaking, the iteration in which λ 'analyzes' the meeting with λ'. The agent λ determines in line 5 of Function `Update-Label` the fact that λ confirms T as a result of the meeting. Then, in line 6 of Function `Update-Label`, $f^\lambda(x)$ is set to 1, where x is at the end of T' from the root of $\mathcal{V}^{3(n-1)}(h(\lambda))$. By Lemma 6 and by the choice of T' in line 4 of Function `Update-Label`, x corresponds to $h(\lambda')$. Function `Update-Label` returns f^λ. Due to line 12 of Algorithm `Leader-Election`, $f_{p+1}^\lambda(x) = 1$.

In order to prove the second part of the lemma notice that agent λ' has also access to the memory states of λ and λ' at the time of the meeting. Hence, upon completing phase p it performs analogous computations as λ during its execution of Function `Update-Label` at the end of phase p and sets $f_{p+1}^{\lambda'}(x') = 1$. □

Let λ be an agent and let x be a node of the truncated view $\mathcal{V}^{3(n-1)}(h(\lambda))$ at the end of a trail T from the root of $\mathcal{V}^{3(n-1)}(h(\lambda))$. If the agent λ sets $f^\lambda(x) = 1$ during the execution of Function `Update-Label` and x corresponds to the initial position of some agent λ', then we say that *λ noted λ' on T*.

A label $\alpha = (\xi(\mathcal{V}^{3(n-1)}(h(\lambda))), f_1^\lambda, \ldots, f_p^\lambda)$, where $p \in \{1,2,3\}$, of an agent λ is *complete with respect to* λ' if $f_p^\lambda(x) = 1$ for each node x of $\mathcal{V}^{3(n-1)}(h(\lambda))$ that corresponds to an initial position of λ'. We say that α is *semi-complete with respect to* λ' if $f_p^\lambda(x) = 1$ for each node x that corresponds to an initial position of λ' and belongs to a level $i \le 2(n-1)$ of $\mathcal{V}^{3(n-1)}(\lambda)$.

The next lemma explains how agents confirm trails as a result of their meetings. Agents with different labels can confirm any trail in $\mathcal{S}_n$ between their initial positions, while agents with equal labels are able to confirm only 'palindromes'.

Lemma 10 *Let $p \in \{1,2,3\}$ and let λ and λ' be any two agents.*

(i) If $\alpha_p(\lambda) \ne \alpha_p(\lambda')$, then $\alpha_{p+1}(\lambda)$ is complete with respect to λ'.
(ii) If $\alpha_p(\lambda) = \alpha_p(\lambda')$, then prior to the end of its phase p the agent λ has noted λ' on each trail T such that $T \in \mathcal{S}_n$, $T = \overline{T}$ and $h(\lambda') = d(\mathcal{R}(h(\lambda), T))$.

Proof To prove (i) let $T \in \mathcal{S}_n$ be any trail such that $\mathcal{R}(h(\lambda), T)$ leads from $h(\lambda)$ to $h(\lambda')$ in G. By construction of $\mathcal{P}^n$, $\mathcal{P}_i^n = (\alpha_p(\lambda), \alpha_p(\lambda'), T)$ for some index $i \in \{1, \ldots, |\mathcal{P}^n|\}$, because $\alpha_p(\lambda) \ne \alpha_p(\lambda')$. By Lemma 7, λ confirms T or λ' confirms $\overline{T}$ as a result of a meeting that occurs when the agent is in phase p. This is done by checking the conditions (i), (ii) and (iii) in Definition 1, which can be accomplished by analyzing the memory states of λ and λ' at the time of the meeting. By Lemma 9, $f^{\lambda}(x) = 1$, where x is at the end of T in $\mathcal{V}^{3(n-1)}(h(\lambda))$, regardless of which agent confirmed a trail as a result of the meeting.

The proof of (ii) is analogous. □

Lemma 11 *There exist two agents λ and λ' such that $\alpha_2(\lambda) \neq \alpha_2(\lambda')$.*

Proof If there exist two agents λ and λ' with different views, $\mathcal{V}(h(\lambda)) \neq \mathcal{V}(h(\lambda'))$, then by Proposition 4, $\mathcal{V}^{3(n-1)}(h(\lambda)) \neq \mathcal{V}^{3(n-1)}(h(\lambda'))$. Proposition 3 implies $\alpha_1(\lambda) \neq \alpha_1(\lambda')$, and consequently $\alpha_2(\lambda) \neq \alpha_2(\lambda')$. Thus, assume in the following that the views from the initial positions of all agents are equal. In view of condition EC there exists a non-uniform palindrome. Hence, there exist two agents λ, λ' and a trail T, such that $T = \overline{T}$, $\mathcal{R}(h(\lambda), T)$ leads from $h(\lambda)$ to a node u that is the initial position of an agent λ'', and $\mathcal{R}(h(\lambda'), T)$ leads from $h(\lambda')$ to a node u' that is not the initial position of any agent. Let x and x' be the nodes of $\mathcal{V}^{3(n-1)}(h(\lambda))$ and $\mathcal{V}^{3(n-1)}(h(\lambda'))$, respectively, at the end of T from the roots. Note that x corresponds to u and x' corresponds to u'. By Lemma 10(ii), λ noted λ'' on T in phase 1 and, by Lemma 8, λ' does not note any agent on T during the entire execution of its algorithm. Hence, $f^{\lambda}(x) = 1$ and $f^{\lambda'}(x') = 0$ after the execution of Procedure `Update-Label` at the end of phase 1. Since the views $\mathcal{V}^{3(n-1)}(h(\lambda))$ and $\mathcal{V}^{3(n-1)}(h(\lambda'))$ are equal, we have $\alpha_2(\lambda) = (\xi(\mathcal{V}^{3(n-1)}(h(\lambda))), f_1^{\lambda}, f_2^{\lambda})$ and $\alpha_2(\lambda') = (\xi(\mathcal{V}^{3(n-1)}(h(\lambda'))), f_1^{\lambda'}, f_2^{\lambda'})$, where $f_2^{\lambda} \neq f_2^{\lambda'}$. Hence, $\alpha_2(\lambda) \neq \alpha_2(\lambda')$. □

The role of the next lemma is to explain indirect learning of initial positions of other agents. If two agents λ and λ' have equal labels, then they mutually situate their initial positions using a third agent λ'' with a different label as an intermediary. Such an agent exists by Lemma 11. Agents λ and λ' can correctly fill their initial positions to depth $2(n-1)$ in their views due to the fact that the intermediary λ'' has their initial positions to depth $3(n-1)$.

Lemma 12 *Let λ and λ' be two agents such that $\alpha_p(\lambda)$ is complete with respect to λ', $p \in \{1, 2, 3\}$. If λ'' is any agent such that $\alpha_p(\lambda'') \neq \alpha_p(\lambda)$, then $\alpha_{p+1}(\lambda'')$ is semi-complete with respect to λ'.*

Proof Since $\alpha_p(\lambda'') \neq \alpha_p(\lambda)$ we obtain, by the definition of $\mathcal{P}^n$, that $(\alpha_p(\lambda''), \alpha_p(\lambda), T) = \mathcal{P}_i^n$ for some $i \in \{1, \ldots, |\mathcal{P}^n|\}$, where T is such a trail that the route $R'' = \mathcal{R}(h(\lambda''), T)$ contains at most $n-1$ edges and $d(R'') = h(\lambda)$. We analyze the execution of Function `Update-Label` by λ'' at the end of phase p. By Lemma 7, λ'' confirms T or λ

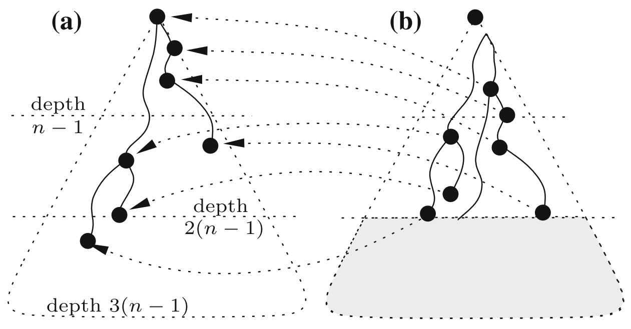


Fig. 5 **a** $\mathcal{V}^{3(n-1)}(h(\lambda))$; **b** $\mathcal{V}^{2(n-1)}(h(\lambda''))$; λ'' learns all nodes corresponding to $h(\lambda')$ to the depth $2(n-1)$ in its view, on the basis of $\mathcal{V}^{3(n-1)}(h(\lambda))$

confirms $\overline{T}$ as a result of their meeting prior to the completion of phase p by λ'', i.e., prior to the execution of Function `Update-Label` we consider. The agent λ'' learns this fact in line 5 of Function `Update-Label`. Then, the condition in line 7 of Function `Update-Label` is satisfied by assumption, and λ'' verifies this condition by checking whether the length of T does not exceed $2(n-1)$. Consequently, λ'' finds in line 9 of Function `Update-Label` a transition φ that maps the node at the end of T in $\mathcal{V}^{3(n-1)}(h(\lambda''))$ to the root of $\mathcal{V}^{3(n-1)}(h(\lambda))$, because $d(R'') = h(\lambda)$. Since R'' is of length at most $n-1$, λ'' sets $f^{\lambda''}(y) = 1$ (in the 'for' loop in lines 10–11 of Function `Update-Label`) for each node y at depth at most $2(n-1)$ in $\mathcal{V}^{3(n-1)}(h(\lambda''))$ corresponding to $h(\lambda')$. The latter is due to the fact that $\alpha_p(\lambda)$ is complete with respect to λ'. This proves that $\alpha_{p+1}(\lambda'')$ is semi-complete with respect to λ'.

See Fig. 5, where the root and some nodes that correspond to the initial position of λ' have been marked on $\mathcal{V}^{3(n-1)}(h(\lambda))$, and we show a transition that allows to determine all nodes in the view of λ'' to the depth $2(n-1)$ corresponding to $h(\lambda')$. □

In view of the next lemma, upon completion of phase 3 an agent has correctly situated all nodes corresponding to initial positions of all agents to depth $2(n-1)$ in its view.

Lemma 13 *Let λ be any agent. Then, $\alpha_4(\lambda)$ is semi-complete with respect to each agent λ'.*

Proof If $\alpha_3(\lambda) \neq \alpha_3(\lambda')$, then by Lemma 10(i), $\alpha_4(\lambda)$ is complete with respect to λ', and therefore it is semi-complete with respect to λ'. Hence, suppose that $\alpha_3(\lambda) = \alpha_3(\lambda')$, which implies $\alpha_2(\lambda) = \alpha_2(\lambda')$. Hence, Lemma 11 implies that there exists an agent λ'' such that $\alpha_2(\lambda'') \neq \alpha_2(\lambda)$. Due to Lemma 10(i), $\alpha_3(\lambda'')$ is complete with respect to λ'. Note that $\alpha_2(\lambda) \neq \alpha_2(\lambda'')$ implies $\alpha_3(\lambda) \neq \alpha_3(\lambda'')$. Lemma 12 completes the proof. □

Our final lemma says that any agent can correctly reconstruct complete identifiers of all other agents. This is due to the fact that every agent has filled initial positions of all agents to depth $2(n-1)$ in its view.

Lemma 14 *Let $\mathcal{A} = \{(I_1, T_1), \ldots, (I_a, T_a)\}$ be the set computed in Procedure* `Choose-Leader` *executed by an agent λ. Then, for every agent λ', there exists an index $i \in \{1, \ldots, a\}$ such that I_i is the complete identifier of λ' and $\mathcal{R}(h(\lambda), T_i)$ leads from $h(\lambda)$ to $h(\lambda')$ in G.*

Proof Let $\alpha_4(\lambda) = (\xi(\mathcal{V}^{3(n-1)}(h(\lambda))), f_1^{\lambda}, \ldots, f_4^{\lambda})$ be the label of λ obtained at the end of phase 3. Initially $\mathcal{A}$ is set to be empty in line 1 of Procedure `Choose-Leader`. Consider the nodes v and x and the function f' from line 3 of Procedure `Choose-Leader`. By Lemma 8, v is an initial position of an agent λ'', because $f'(x) = 1$. Thus, in view of Lemma 13, $(\xi(\mathcal{V}^{n-1}(v)), f')$ is the complete identifier of λ''. Due to lines 4–6 of Procedure `Choose-Leader`, $((\xi(\mathcal{V}^{n-1}(v)), f'), T) \in \mathcal{A}$ for some trail T such that $d(\mathcal{R}(h(\lambda), T)) = v = h(\lambda'')$. By Lemma 13, for any agent λ', there is a node in $\mathcal{V}^{n-1}(h(\lambda))$ corresponding to $h(\lambda')$. This implies that, for any agent λ', the set $\mathcal{A}$ contains (at the end of the 'for' loop in lines 2–7 of Procedure `Choose-Leader`) a pair (I_i, T_i), $i \in \{1, \ldots, a\}$, such that I_i is the complete identifier of λ' and $\mathcal{R}(h(\lambda), T_i)$ leads from $h(\lambda)$ to $h(\lambda')$ in G. □

Theorem 2 *If the condition* EC *is satisfied for an initial configuration, then Algorithm* `Leader-Election` *correctly elects a leader regardless of the actions of the adversary.*

Proof Let λ^* be the agent whose complete identifier I^* is lexicographically smallest among the complete identifiers of all agents. By condition EC (more precisely by its part saying that all enhanced views are different) and by Corollary 1, agent λ^* is unique.

Each agent λ computes $\alpha_4(\lambda)$ as a result of the execution of Algorithm `Leader-Election`. By Lemma 14, λ computes (in lines 1–7 of Procedure `Choose-Leader`), for each agent λ', the complete identifier of λ' and a trail T such that the route $\mathcal{R}(h(\lambda), T)$ leads from $h(\lambda)$ to $h(\lambda')$ in G. By Corollary 1, the complete identifiers uniquely distinguish the agents. Using the lexicographic order $\preceq$ on the set of all complete identifiers, the agent λ finds in line 8 of Procedure `Choose-Leader` the complete identifier $I = (\mathcal{V}^{n-1}(u), f)$ such that $(I, T) \in \mathcal{A}$ and $I \preceq I'$ for each I' such that $(I', T') \in \mathcal{A}$ for some trail T'. By definition, $I = I^*$. Then, λ decides in line 9 of Procedure `Choose-Leader` that the agent with the initial position $d(\mathcal{R}(h(\lambda), T))$ is the leader. This is agent λ^*. □

Theorem 2, together with Proposition 1 implies our main result which is Theorem 1 from Sect. 3.

6 Conclusion

We characterized all initial configurations of agents for which leader election is possible and we constructed a universal algorithm electing a leader for all such configurations, assuming that agents know an upper bound on the size of the graph. We observed that the latter assumption cannot be removed. In this paper we focused on the feasibility of leader election under a very harsh scenario in which the adversary controls the speed and the way in which agents move along their chosen routes. This adversarial scenario captures the totally asynchronous nature of mobile agents.

While we gave a complete solution to the problem of feasibility of leader election, we did not try to optimize the efficiency of the algorithm, e.g., in terms of its cost, i.e., of the total or of the maximum number of edge traversals performed by the mobile agents. In fact, any kind of such optimization appears to be quite challenging. It is clear that in order to elect a leader agents have to meet. Already the much simpler problem of optimizing the cost of meeting of two agents in our asynchronous model is open, both when agents have different labels [18] and when they are anonymous, as in our present scenario [28]. In particular, in the latter paper the authors asked if rendezvous of two agents can be accomplished (whenever it is feasible) at a cost polynomial in the size of the graph. Recently, an asynchronous rendezvous algorithm for labeled agents, working at a cost polynomial in the size of the graph and in the length of the shorter label, has been obtained in [22].

Leader election can be used to subsequently accomplish other tasks in networks, e.g., gathering all agents in one node. In our context this can be done as follows. As soon as an agent elects itself as leader, it goes back to its initial position. Every other agent, after electing a leader, goes to the initial position of the leader. (Recall that at the time of electing a leader each agent knows a path from its initial position to the initial position of the leader, and each agent remembers a path to its own initial position at all times.) In this way all agents will eventually gather at the initial position of the leader. Since the leader, at the time of electing itself, knows that it has seen all agents and that it distinguished all distinct agents, it may have counted them. The leader waits until all agents come to join it and declares that gathering is completed.

Acknowledgments This work was done during the visit of Dariusz Dereniowski at the Research Chair in Distributed Computing of the Université du Québec en Outaouais. This author was partially supported by the Polish Ministry of Science and Higher Education (MNiSW) Grant N N516 196437 and by a scholarship for outstanding young researchers founded by the Polish Ministry of Science and Higher Education. Andrzej Pelc was partially supported by NSERC discovery grant and by the Research Chair in Distributed Computing at the Université du Québec en Outaouais.

References

1. Alpern, S., Gal, S.: The Theory of Search Games and Rendezvous. International Series in Operations Research and Management Science. Kluwer, Dordrecht (2003)
2. Ambuhl, C., Gasieniec, L., Pelc, A., Radzik, T., Zhang, X.: Tree exploration with logarithmic memory. ACM Trans. Algorithms **7**, paper 17 (2011)
3. Angluin, D.: Local and global properties in networks of processors. In: Proceedings of the 12th Annual ACM Symposium on Theory of Computing (STOC 1980), pp. 82–93
4. Attiya, H., Snir, M., Warmuth, M.: Computing on an anonymous ring. J. ACM **35**, 845–875 (1988)
5. Attiya, H., Snir, M.: Better computing on the anonymous ring. J. Algorithms **12**, 204–238 (1991)
6. Bampas, E., Czyzowicz, J., Gasieniec, L., Ilcinkas, D., Labourel, A.: Almost optimal asynchronous rendezvous in infinite multidimensional grids. In: Proceedings of the 24th International Symposium on Distributed Computing (DISC 2010), LNCS 6343, pp. 297–311
7. Boldi, P., Shammah, S., Vigna, S., Codenotti, B., Gemmell, P., Simon, J.: Symmetry breaking in anonymous networks: characterizations. In: Proceedings of the 4th Israel Symposium on Theory of Computing and Systems, (ISTCS 1996), pp. 16–26
8. Boldi, P., Vigna, S.: Computing anonymously with arbitrary knowledge. In: Proceedings of the 18th ACM Symposium on Principles of Distributed Computing (PODC 1999), pp. 181–188
9. Burns, J.E.: A Formal Model for Message Passing Systems. Technical Report TR-91. Computer Science Department, Indiana University, Bloomington (1980)
10. Chalopin, J.: Local computations on closed unlabelled edges: the election problem and the naming problem. In: Proceedings of the 31st Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM 2005), pp. 82–91
11. Chalopin, J., Das, S., Kosowski, A.: Constructing a map of an anonymous graph: applications of universal sequences. In: Proceedings of the 14th International Conference on Principles of Distributed Systems (OPODIS 2010), pp. 119–134
12. Chalopin, J., Mazurkiewicz, A.W., Métivier, Y.: Labelled (hyper)graphs, negotiations and the naming problem. In: Proceedings of the 4th International Conference on Graph Transformations (ICGT 2008), pp. 54–68
13. Chalopin, J., Métivier, Y.: Election and local computations on edges. In: Proceedings of the Foundations of Software Science and Computation Structures (FoSSaCS 2004), pp. 90–104
14. Cieliebak, M., Flocchini, P., Prencipe, G., Santoro, N.: Solving the robots gathering problem. In: Proceedings of the 30th International Colloquium on Automata, Languages and Programming (ICALP 2003), pp. 1181–1196
15. Collins, A., Czyzowicz, J., Gasieniec, L., Labourel, A.: Tell me where I am so I can meet you sooner. In: Proceedings of the 37th International Colloquium on Automata, Languages and Programming (ICALP 2010), pp. 502–514
16. Czyzowicz, J., Dobrev, S., Kranakis, E., Krizanc, D.: The power of tokens: rendezvous and symmetry detection for two mobile agents in a ring. In: Proceedings of the 34th Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM 2008), pp. 234–246
17. Czyzowicz, J., Kosowski, A., Pelc, A.: How to meet when you forget: log-space rendezvous in arbitrary graphs. Distrib. Comput. **25**, 165–178 (2012)
18. Czyzowicz, J., Labourel, A., Pelc, A.: How to meet asynchronously (almost) everywhere. ACM Trans. Algorithms **8**, article 37 (2012)
19. De Marco, G., Gargano, L., Kranakis, E., Krizanc, D., Pelc, A., Vaccaro, U.: Asynchronous deterministic rendezvous in graphs. Theor. Comput. Sci. **355**, 315–326 (2006)
20. Dessmark, A., Fraigniaud, P., Kowalski, D., Pelc, A.: Deterministic rendezvous in graphs. Algorithmica **46**, 69–96 (2006)
21. Dieudonné, Y., Pelc, A.: Anonymous meeting in networks. In: Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2013), pp. 737–747
22. Dieudonné, Y., Pelc, A., Villain, V.: How to meet asynchronously at polynomial cost. In: Proceedings of the 32nd Annual ACM Symposium on Principles of Distributed Computing (PODC 2013), pp. 92–99
23. Diks, K., Kranakis, E., Malinowski, A., Pelc, A.: Anonymous wireless rings. Theor. Comput. Sci. **145**, 95–109 (1995)
24. Dobrev, S., Pelc, A.: Leader election in rings with nonunique labels. Fundamenta Informaticae **59**, 333–347 (2004)
25. Flocchini, P., Kranakis, E., Krizanc, D., Luccio, F.L., Santoro, N.: Sorting and election in anonymous asynchronous rings. J. Parallel Distrib. Comput. **64**, 254–265 (2004)
26. Fredrickson, G.N., Lynch, N.A.: Electing a leader in a synchronous ring. J. ACM **34**, 98–115 (1987)
27. Fusco, E., Pelc, A.: How much memory is needed for leader election. Distrib. Comput. **24**, 65–78 (2011)
28. Guilbault, S., Pelc, A.: Asynchronous rendezvous of anonymous agents in arbitrary graphs. In: Proceedings of the 15th International Conference on Principles of Distributed Systems (OPODIS 2011), LNCS 7109, pp. 162–173
29. Haddar, M.A., Kacem, A.H., Métivier, Y., Mosbah, M., Jmaiel, M.: Electing a leader in the local computation model using mobile agents. In: Proceedings of the 6th ACS/IEEE International Conference on Computer Systems and Applications (AICCSA 2008), pp. 473–480
30. Hirschberg, D.S., Sinclair, J.B.: Decentralized extrema-finding in circular configurations of processes. Commun. ACM **23**, 627–628 (1980)
31. Jurdzinski, T., Kutylowski, M., Zatopianski, J.: Efficient algorithms for leader election in radio networks. In: Proceedings of the 21st ACM Symposium on Principles of Distributed Computing (PODC 2002), pp. 51–57
32. Kowalski, D., Pelc, A.: Leader election in ad hoc radio networks: a keen ear helps. J. Comput. Syst. Sci. **79**, 1164–1180 (2013)
33. Klasing, R., Kosowski, A., Navarra, A.: Taking advantage of symmetries: gathering of asynchronous oblivious robots on a ring. In: Proceedings of the 12th International Conference on Principles of Distributed Systems (OPODIS 2008), pp. 446–462
34. Klasing, R., Markou, E., Pelc, A.: Gathering asynchronous oblivious mobile robots in a ring. Theor. Comput. Sci. **390**, 27–39 (2008)
35. Kranakis, E.: Symmetry and computability in anonymous networks: a brief survey. In: Proceedings of the 3rd International Conference on Structural Information and Communication Complexity, pp. 1–16 (1997)
36. Kranakis, E., Krizanc, D., van der Berg, J.: Computing boolean functions on anonymous networks. Inf. Comput. **114**, 214–236 (1994)
37. Kranakis, E., Krizanc, D., Santoro, N., Sawchuk, C.: Mobile agent rendezvous in a ring. In: Proceedings of the 23rd International Conference on Distributed Computing Systems (ICDCS 2003), pp. 592–599

38. Le Lann, G.: Distributed systems—towards a formal approach. In: Proceedings of the IFIP Congress, pp. 155–160. North Holland (1977)
39. Nakano, K., Olariu, S.: Uniform leader election protocols for radio networks. IEEE Trans. Parallel Distrib. Syst. **13**, 516–526 (2002)
40. Norris, N.: Universal covers of graphs: isomorphism to depth $N-1$ implies isomorphism to all depths. Discret. Appl. Math. **56**, 61–74 (1995)
41. Pelc, A.: Deterministic rendezvous in networks: a comprehensive survey. Networks **59**, 331–347 (2012)
42. Peterson, G.L.: An $O(n \log n)$ unidirectional distributed algorithm for the circular extrema problem. ACM Trans. Program. Lang. Syst. **4**, 758–762 (1982)
43. Sakamoto, N.: Comparison of initial conditions for distributed algorithms on anonymous networks. In: Proceedings of the 18th ACM Symposium on Principles of Distributed Computing (PODC 1999), pp. 173–179
44. Ta-Shma, A., Zwick, U.: Deterministic rendezvous, treasure hunts and strongly universal exploration sequences. In: Proceedings of the 18th ACM-SIAM Symposium on Discrete Algorithms (SODA 2007), pp. 599–608
45. Willard, D.E.: Log-logarithmic selection resolution protocols in a multiple access channel. SIAM J. Comput. **15**, 468–477 (1986)
46. Yamashita, M., Kameda, T.: Computing on anonymous networks. In: Proceedings of the 7th ACM Symposium on Principles of Distributed Computing (PODC 1988), pp. 117–130
47. Yamashita, M., Kameda, T.: Electing a leader when processor identity numbers are not distinct. In: Proceedings of the 3rd Workshop on Distributed Algorithms (WDAG 1989), LNCS 392, pp. 303–314
48. Yamashita, M., Kameda, T.: Computing on anonymous networks: part I—characterizing the solvable cases. IEEE Trans. Parallel Distrib. Syst. **7**, 69–89 (1996)