Near-Linear Time Leader Election in Multiagent Networks

Ajay D. Kshemkalyani
University of Illinois at Chicago
Chicago, USA
ajay@uic.edu

Manish Kumar
Indian Institute of Technology Madras
Chennai, India
manishsky27@gmail.com

Anisur Rahaman Molla
Indian Statistical Institute Kolkata
Kolkata, India
molla@isical.ac.in

Gokarna Sharma
Kent State University
Kent, USA
gsharma2@kent.edu

ABSTRACT

Leader election is a fundamental and widely studied problem in distributed computing, traditionally explored in the message-passing model, where each node in a distributed network graph represents a static computational device that communicates with others by exchanging messages. This paper studies leader election in the agent-based network, in which the computational devices are modeled as relocatable or mobile agents that explore a graph and perform computations. Each node in the graph serves as a container or host for these mobile agents, and communication occurs between agents when they move to the same node. We consider the scenario where n agents are arbitrarily placed on the nodes of an anonymous, arbitrary n-node, m-edge graph G. The goal is for the agents to elect a leader such that one agent is designated as the leader, knowing it is the leader, while all other agents recognize they are not the leader. The objective is to minimize the time to elect the leader and memory usage per agent. Following the literature, we consider the synchronous setting where each agent performs its operations synchronously with others and hence the time complexity is measured in rounds. There exists a deterministic solution that elects a leader in $O(m)$ rounds with $O(n \log n)$ bits of memory at each agent in the agent-based network. In this paper, we present a deterministic algorithm that elects a leader in $O(n \log^2 n)$ rounds with each agent using only $O(\log n)$ bits. This is a significant improvement, as the memory usage is optimal and the time complexity is almost linear in n (up to $O(\log^2 n)$ factor). Additionally, leveraging this leader election result, we provide improved time and/or memory bounds for four core distributed graph problems: minimum spanning tree, gathering, maximal independent set, and minimal dominating set.

KEYWORDS

Distributed algorithms; mobile agents; local communication; leader election; MST; MIS; gathering; minimal dominating set; time and memory complexity

ACM Reference Format:
Ajay D. Kshemkalyani, Manish Kumar, Anisur Rahaman Molla, and Gokarna Sharma. 2025. Near-Linear Time Leader Election in Multiagent Networks.

Proc. of the 24th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2025), Y. Vorobeychik, S. Das, A. Nowé (eds.), May 19 – 23, 2025, Detroit, Michigan, USA. © 2025 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org).

Model	Devices	Local computation	Device storage	Neighbor communication
Message-passing	Static	Unlimited	Unrestricted	Messages
Agent-based	Mobile	Unlimited	Limited	Relocation

Table 1: Comparing message-passing and agent-based models.

In *Proc. of the 24th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2025), Detroit, Michigan, USA, May 19 – 23, 2025*, IFAAMAS, 9 pages.

1 INTRODUCTION

The well-studied *message-passing model* [20] of distributed computing assumes an underlying distributed network represented as an undirected graph $G = (V, E)$, where each vertex/node corresponds to a *computational device* (such as a computer or a processor), and each edge corresponds to a bi-directional communication link. Each node $v \in G$ has a distinct $\Theta(\log n)$-bit identifier, $n = |V|$. The structure of G (topology, latency) is assumed to be not known in advance but it is assumed that each node knows its neighboring nodes. The nodes interact with one another by sending messages (hence the name message-passing model). The computation proceeds according to synchronized *rounds*. In each round, each node v can perform unlimited local computation and may send message (possibly distinct) to each of its neighbors. Additionally, each node v is assumed to have no storage constraint.

In this paper, we abstract away from the message-passing model and consider the *agent-based* distributed computing model where the computational devices are modeled as *relocatable or mobile computational devices* (which we call mobile agents). Instead of vertex/node being a *static* device in the message-passing model, the vertices/nodes serve as *containers* for the devices in the agent-based model. An agent can move from one node to a neighboring node via the edges of the graph. This gives two major differences between these models (see Table 1).

Difference I. The graph nodes do not have identifiers, computation ability, and storage, but the devices (aka agents) are assumed to have distinct $O(\log n)$-bit identifiers, computation ability, and (limited) storage.
Difference II. The devices (agents) cannot send messages to other devices except the ones co-located at the same node. To send a message to a device positioned at a neighboring node, a device needs to relocate to the neighbor.

Difference II is the major problem for the agent-based model, since when a device relocates to a neighbor for communication, the device at that neighbor might relocate to another neighbor.

With advancements in technology across automated systems such as robots, drones, UAVs, and self-driving cars, research on computing with mobile agents has become increasingly essential. The agent-based distributed computation is potentially be useful and applicable in scenarios like private networks in the military or sensor networks in inaccessible terrain where direct access to the network is possibly obstructed, but small battery-powered relocatable computational devices can learn network structures and their properties for overall network management. Additionally, the device relocation may help avoiding communication link compromise. Some examples include search and rescue (SAR) operations [21, 24–26], underwater navigation [5], network-centric warfare [15], social network modeling [27], and social epidemiology [7].

In this paper, we study leader election in the agent-based model. Leader election is one of the fundamental and well-studied problems in distributed computing due to its applications in numerous problems, such as resource allocation, reliable replication, load balancing, synchronization, membership maintenance, crash recovery, etc. The problem of leader election in the agent-based model requires a set of agents operating in the distributed network G to elect a unique leader among themselves, i.e., exactly one agent (at a node) must output the decision that it is the leader.

The performance of an algorithm designed in the agent-based model is measured w.r.t. two metrics that are fundamental to the agent-based model: *time complexity* of a solution and *storage requirement* per agent. *Deterministic* algorithms are typically sought since they may be more suitable for relocatable devices.

Contributions. Kshemkalyani *et al.* [10] recently provided the first deterministic algorithm for leader election in the agent-based model, which elects a leader in $O(m)$ time with $O(n \log n)$ bits at each agent. The algorithm achieves so without relying on any knowledge (neither exact nor an upper bound) on graph parameters, such as n (the network size and also the number of agents), Δ (the maximum degree of G), and D (diameter of G). Notice that G is anonymous meaning that graph nodes do not have identifiers (only the agents have unique identifiers). In this paper, along the line of the result of [10], we prove the following theorem.

Theorem 1.1 (**leader election**). *Given any initial configuration of n agents with unique identifiers positioned arbitrarily on the nodes of the n-node graph G with no node identifiers, there is a deterministic algorithm that elects one agent at a graph node as a leader in $O(n \log^2 n)$ rounds with $O(\log n)$ bits per agent, without agents knowing any graph parameter a priori. (***Section 3***)*

Theorem 1.1 is a significant improvement over the only previously known result for leader election in the agent-based model due to Kshemkalyani *et al.* [10]. In fact, the memory complexity is optimal since any algorithm designed in the agent-based model with n agents needs $\Theta(\log n)$ bits per agent [1, 11]. The time complexity is optimal within an $O(\log^2 n)$ factor since $\Theta(n)$ is the time lower bound for the kind of problems that ask the n agents to be on n nodes if they were not positioned that way initially [1, 11].

Besides its own merit regarding improved time and memory complexities for an important problem, this leader election result became central in improving the time and/or memory complexities of many other fundamental distributed graph problems. We consider explicitly four problems, namely, minimum spanning tree (MST), maximal independent set (MIS), minimal dominating set (MDS), and gathering and establish improved results on time and/or memory (the results are in **Section 4**). Table 2 lists and compares all the established results with the previous results.

Algorithm	Knowledge	Time complexity	Memory per agent
Leader election			
Section 3	–	$O(n \log^2 n)$	$O(\log n)$
[10]	–	$O(m)$	$O(n \log n)$
MST			
Section 4	–	$O(m + n \log^2 n)$	$O(\Delta \log n)$
[10]	–	$O(m + n \log n)$	$O(n \log n)$
MIS			
Section 4	–	$O(n \log^2 n + n\Delta)$	$O(\log n)$
[10]	–	$O(n\Delta)$	$O(n \log n)$
[18]	n, Δ	$O(n\Delta \log n)$	$O(\log n)$
MDS			
Section 4	–	$O(n \log^2 n + m)$	$O(\log n)$
[10]	–	$O(m)$	$O(n \log n)$
[4]	n, Δ, m, γ	$O(\gamma\Delta \log n + n\gamma + m)$	$O(\log n)$
Gathering			
Section 4	–	$O(n \log^2 n)$	$O(\log n)$
[10]	–	$O(m)$	$O(n \log n)$
[16]	n	$O(n^3)$	$O(M + m \log n)$

Table 2: Comparing previous and developed results for five problems in a graph G in the agent-based model. M is the memory required for the Universal Exploration Sequence (UXS) [23], γ is the number of clusters of agents in the initial configuration, and n, m, Δ, respectively, are the number of nodes/agents, number of edges, and maximum degree of G. '–' means no a priori knowledge of graph parameters.

Establishing Theorem 1.1 needed a fast solution for the problem of *dispersion* which can be defined as follows: If n agents are not on n nodes (i.e., one per node) initially, the goal is to relocate themselves autonomously to be positioned on such configuration. The leader election procedure can then be run. Kshemkalyani *et al.* [10] solved dispersion through a $O(m)$ time $O(n \log n)$ bits per agent algorithm, which became the dominating cost for their leader election algorithm. A candidate algorithm to solve dispersion faster with less memory is due to [22] in which dispersion is achieved in $O(n \log^2 n)$ time with $O(\log n)$ bits per agent. However this algorithm is non-terminating without prior knowledge on n, meaning that the agents do not know whether dispersion is finished without knowing n. Since it is not known when dispersion finishes, the agents cannot start leader election procedure. Our main contribution lies in making this dispersion algorithm terminating without knowing n and keeping intact both time and memory complexities.

One might suggest to solve first gathering instead of dispersion for leader election. Once all agents are gathered at a node, an agent (with smallest/largest identifier) can be picked as a leader since the agents have unique identifiers. Although the idea seems promising, solving gathering turned out to be expensive in terms of both time and memory complexities. The best-known gathering method [16] requires $O(n^3)$ time and $O(M + m \log n)$ memory, where M denotes the memory needed for the Universal Exploration Sequence

(UXS) [23]. Additionally, these gathering algorithms assume a priori knowledge on n. Our dispersion based leader election algorithm provided better time/memory complexities being oblivious to n.

Overview of Techniques. Our proposed technique has two aspects: dispersion and leader election. Our technique on leader election extends the technique on Kshemkalyani *et al.* [10], whereas our technique in dispersion is based on Sudo *et al.* [22]. We organize the entire dispersion and leader election process into 21 independent slots, ensuring that each slot operates separately without interference from others. Within these slots, agents first compete for the role of a local leader, and then, based on priority, one of the local leaders is elected as the unique global leader. The protocol guarantees the presence of at least one local leader. Given the agents' general configuration, it is crucial to ensure that a local leader verifies its highest priority status to become the global leader, with the solution to dispersion serving as a method to assess this priority.

We propose a two-stage approach to elect a unique (global) leader. In the first stage, (at least) a local leader is elected, and in the second stage, one of the local leaders is chosen as the unique global leader. After being elected as a local leader in the first stage, each local leader can proceed to run the same *global election* procedure in the second stage to compete for the role of global leader. However, during the first stage, an agent becomes a 'local leader' using one of two procedures. If an agent starts alone, it follows a *singleton election* procedure to become a local leader. If an agent starts with others, the minimum ID agent among them becomes the local leader. The singleton election procedure executed by an agent a at node w involves visiting the neighbors of w, potentially multiple times. This procedure selects agent a at node w as a local leader if and only if all of w's neighbors initially have a singleton agent positioned. The proposed technique guarantees that starting from any initial configuration, at least one agent becomes a local leader.

Once an agent becomes a local leader, it initiates the global election procedure to attempt to become the global leader. This procedure involves verifying whether the local leader agent can traverse all the edges of the graph G. If the local leader successfully traverses all the edges, it returns to the node where it initially became a local leader (referred to as the *home node* of that local leader) and declares itself as the global leader. We demonstrate that after an agent declares itself as the global leader, no other local leader agent can become a global leader, ensuring the uniqueness of the global leader.

To become the global leader, each local leader, say a, interacts with one of its neighboring agents, say b, which oscillates between two nodes: its home node and the node occupied by a. Agent b retains all information related to both nodes, introducing a new technique developed in this work where an agent can effectively manage information from multiple nodes. A local leader leaves its home node to evaluate its priority on a global scale, placing any lower-priority agent (referred to as a 'zombie') during the DFS traversal (with HEO-DFS technique discussed in [22]), if accompanied. During this process, agent a constructs its own DFS tree, maintaining its leadership as long as it holds the highest priority. Within this DFS tree, agents record information about their first child, next sibling, and parent with a couple of other pointers, keeping memory usage limited to $O(\log n)$ bits. This allows the tree to be traversed efficiently with the information needed by the global leader to announce its election as the global leader and place another zombie agent at an empty node if required. This memory-efficient tree construction is a novel technique developed in this paper.

Since multiple local leaders may be elected in the first stage, several global election procedures can run simultaneously. Each local leader is associated with a group identifier in the form of $(a.level, a.leader)$, where $a.leader = a$.ID and $a.level$ is the level of agent a (initially, set to 1). A zombie or settler does not initially belong to any group; however, if it accompanies a leader, it becomes part of that leader's HEO-DFS. The group identifier $(a.level, a.leader)$ is considered to have higher priority if it satisfies the condition $(a.level < b.level) \vee (a.level = b.level \wedge a.leader < b.leader)$.

Once a leader is elected, we leverage it to solve four other fundamental problems: minimum spanning tree (MST), maximal independent set (MIS), minimal dominating set (MDS), and gathering. Our approach improves existing results on time and/or memory.

Related Work. Leader election is a well-studied problem in distributed computing, especially in the message-passing model, see [2, 8, 12–14, 19]. In the agent-based model, leader election was studied recently for the first time by Kshemkalyani *et al.* [10]. They provided a deterministic algorithm with time complexity $O(m)$ rounds and memory complexity $O(n \log n)$ bits per agent, without agents knowing graph parameters (such as n, m, Δ). They also provided a deterministic algorithm for MST, solving it in $O(m+n \log n)$ rounds with $O(n \log n)$ bits per agent. Their leader election result served as a crucial subroutine in their MST algorithm.

For MIS in the message-passing model, the best-known deterministic algorithm has time complexity $O(2^{\sqrt{\log n}})$ [3, 17]. For MDS, Deurer *et al.* [6] gave two deterministic $(1+\epsilon)(1+\log(\Delta+1))$-approximation algorithms with time complexity, respectively, $O(2^{O(\sqrt{\log(n)\log(\log(n))})})$ and $O(\Delta \text{polylog}(\Delta) + \text{polylog}(\Delta)\log^{\star}(n))$, where $\epsilon > \frac{1}{\text{polylog}(\Delta)}$. Both MIS and MDS were solved in the agent-based model in [4, 18] with time and memory complexities reported in Table 2 assuming that agents have a priori knowledge on n, Δ (additionally m, γ for MDS). In [10], the a priori knowledge assumption was lifted for both MIS and MDS.

Gathering is relatively well-studied in the agent-based model, however with the assumption of known n. The recent results are [16, 23]. [23] provided a $\tilde{O}(n^5 \log \beta)$ time solution to gather $k \leq n$ agents in arbitrary graph G, where $\tilde{O}$ hides polylog factors and β is the smallest identifier among agents. [16] provided improved time complexities for large values of k (not knowing k but n): (i) $O(n^3)$ rounds, if $k \geq \lfloor \frac{n}{2} \rfloor + 1$ (ii) $\tilde{O}(n^4)$ rounds, if $\lfloor \frac{n}{2} \rfloor + 1 \leq k < \lfloor \frac{n}{3} \rfloor + 1$, and (iii) $\tilde{O}(n^5)$ rounds, if $\lfloor \frac{n}{3} \rfloor + 1 > k$. The memory complexity is $O(M + m \log n)$ bits per agent, where M is the memory required to implement the universal traversal sequence (UXS) as defined in [23]. Kshemkalyani *et al.* [10] provided $O(n\Delta)$ time $O(n \log n)$ bits per agent gathering algorithm without agents needing to know n a priori (see Table 2 for comparison).

2 MODEL

The network is considered as a connected, undirected graph $G = (V, E)$ with $|V| = n$ nodes, $|E| = m$ edges, diameter D, and maximum

degree Δ. Each node $v_i \in V$ has δ_i ports, one for each incident edge, labeled $[0, \ldots, \delta_i - 1]$. There are n agents $Q = \{a_1, \ldots, a_n\}$ initially positioned on the nodes of G. Each agent has a unique ID in the range $[1, n^{O(1)}]$. The topology of G is unknown to the agents. Additionally, agents are not aware of any graph parameter, such as network size n, maximum degree Δ, diameter D, etc. In an initial configuration, a graph node may have zero, one, or multiple agents positioned. The agents are always positioned on the nodes of G, except while they are migrating between nodes. An agent can communicate while at a node (not while it is migrating). An agent can communicate only to other agents at the same node, i.e., two agents at two different nodes cannot communicate.

Agents can traverse edges from node v to node u along the edge e_{vu}. Borrowing an assumption from the message-passing model, e.g., [9], we assume that an agent can cross an edge in one round, regardless of edge weight, even when G is weighted. Upon exiting v along port p_{vu}, the agent knows the corresponding port p_{uv} upon entering node u. Additionally, if G is weighted, the agent learns the weight $w(e_{vu})$ of edge e_{vu} when it arrives at u. There is no assumed relationship between the port numbers on the two ends of an edge. We do not put limit on how many agents can traverse an edge simultaneously, meaning the agent-based model does not impose congestion constraint on edge traversal.

The agents operate in a synchronous setting, meaning that the computation proceeds in rounds and time complexity is measure with respect to the number of rounds until a solution. All agents are active in each round. In each round, an agent at a node can perform local computation based on its stored information and the port labels at the node. The computation results in $\perp$ or a port to be taken. Before moving, an agent may update the storage of another agent staying at the current node. An agent exiting a node at a round always reaches a neighboring node by the end of that round. The storage complexity is measured w.r.t. the amount of memory (in bits) used by each agent until a solution.

At any round, the agents' distribution on G can be as follows:

- *dispersed* — n agents are on n nodes of G,
- *rooted* — n agents are on a single node of G, or
- *general* — a configuration that is neither dispersed nor rooted.

We refer to the initial configuration as dispersed, rooted, or general, depending on the agents' initial positions; the agents do not know a priori whether the initial configuration is dispersed, rooted, or general. Knowing the nature of initial configuration would permit agents to run specific procedures. An agent is called a *singleton* if it is alone on a node, and *non-singleton* (or multiplicity) otherwise.

3 LEADER ELECTION

In this section, we discuss our deterministic leader election algorithm, which guarantees the selection of a single global leader starting from any initial configuration (dispersed, rooted, or general) of n agents on an n-node graph G. Moreover, if the agents initially start in rooted or general configurations, they end up in a dispersed configuration once leader election procedure is complete.

Initially, each graph node may contain zero, one, or multiple agents, all of which are considered "candidates" for the leadership role. A candidate must first attain the status of a "local leader" before competing for the role of "global leader." Any candidate that fails to become a "local leader" (as well as any "local leader" that fails to become a "global leader") transitions into a "settler" if it has its home node (which is the node where the agent becomes the candidate/local leader) otherwise becomes the "zombie". We run our algorithmic steps in slots, where each slot is independent. Which part of the algorithm runs in what slot and which agent initiates that computation is provided in Table 3.

Slot number	Role	Initiative	Pseudocode
Slot 1–3	Global Leader Election	Candidate	1, 2, and 3
Slot 4	Local Leader Election	Local Leaders	4
Slot 5	Settle, increment level, etc.	Local Leaders	4
Slot 6	Move to join Probe()	Settlers	6
Slot 7–15	Probe()	Local Leaders	5
Slot 16–19	Chase for Local Leaders	Zombies	7
Slot 20–21	Move forward/backward	Local Leaders	4

Table 3: Slot assignments with respect to the pseudocode.

Terminology	Meaning
n	Number of nodes in the graph
m	Number of edges in the graph
u, v	Node in the graph
a, b	Agent in the graph
$a.ID$	ID of the agent a
Δ	Maximum degree of the graph
$N(v)$	Neighbor of the node v
$p_v(u)$	Port at v leading to u
$N(v, i)$	Node $u \in N(v)$ such that $p_v(u) = i$

Table 4: An overview of general notations used in the paper.

It is depicted in Algorithm 1 (Lines 2–7), if an agent is initially singleton then it runs $Singleton\_Election(a)$ procedure to become a local leader or settler. On the other hand, if it is not singleton, the minimum ID agent becomes the local leader and all others become zombies. After the agent becomes the local leader, say a at node w, it informs all its neighbors that it is a local leader if it becomes the leader through $Singleton\_Election(a)$ and as described in Lines 13–15 of Algorithm 1, that local leader asks the port-1 agent across the edge to oscillate between its home node w and b's home node. b stores all the information corresponding to the agent a (its local leader). a starts the DFS until it has the highest priority as described in Algorithms 4–7. We define a relationship $\prec$ between any two non-zombie agents a and b using the group identifiers $(a.level, a.local\_leader)$ as follows: $a \prec b \iff (a.level < b.level) \vee (a.level = b.level \wedge a.local\_leader < b.local\_leader)$. If $a \prec b$ then we consider b is stronger than a. Initially, the level is set to 1. When a leader a encounters a zombie z with the same level, a increments its level by one, and z resets its level to zero. This "level up" changes the identifier of a's group, i.e., from $(a.ID, i)$ to $(a.ID, i + 1)$ for some i. At this point, a loses all nodes from its territory (agent having the same level as a and their leader is a). In other words, each time a leader a increases its level, it restarts its HEO-DFS from the beginning. Note that this "level up" event also occurs when two leaders a, b ($b \prec a$) with the same level meet (and there is no stronger agent at the location). b becomes a zombie after it finds a stronger (based on $\prec$) leader a or vice-versa. We have Lemma 3.4 that states that this "level up" does not occur more than $\log n + 1$ times. Settling of zombie, oscillating agent if required, and "level up" is discussed in Slot 5 with initialization of other required terminology (Tables 4 and 5).

Terminology	Meaning
$v(a)$	Node where agent a resides
$\psi(w)$	Agent at home node w
$A_L(w), A_S(w)$ and $A_Z(w)$	Set of local leaders, settlers, and zombies
$a \prec b$	$(a.level < b.level) \vee$ $(a.level = b.level \wedge a.level < b.leader)$
$\psi(w).next$	Next unsettled node in $N(w)$
$\psi(w).done$ = **true**	Probing is done at node w
$\psi(w).parent$	Parent of node w
$a.init\_alone \leftarrow$ **true**	Agent a is alone at $v(a)$, initially
$a.status$	$\in \{candidate, settler,$ $local\_leader, global\_leader\}$
$\psi(w).help \neq \perp$	Help for probing is required
$a.InitProbe$ = **true**	Agent a requires probing
$\psi(w).checked = \ell$	$N(w,0), N(w,1), \ldots, N(w,\ell)$ are settled
$a.pin$	Incoming port of the agent a
$a.level$	Level of agent a; initially, set to 1
$z.level_L$	Location level
$z.level_S$	Swarm level
$\psi(w).firstchild$	First Child for the agent w during DFS
$\psi(w).nextsibling$	Next sibling during DFS at node w

Table 5: Overview of the notations used in pseudocode.

In Slot 6, the settler moves for HEO-DFS (as discussed in Sudo *et al.* [22]). There might be the case that leader a has encountered a stronger leader b and a has become a zombie. In that case, b would set $a.help = \perp$ in Slot 7. Notice that if more than two neighbors have requested help, in that case, Slot 6 reaches out for help to the one that is the stronger leader, and the other leader becomes a zombie. Slot 8 deals with the fact that when probing (finding the appropriate port for DFS) is done while Slots 9–10 begin the probing and Slot 11 figures out whether nodes are occupied with the agents or not.

Similarly, Slots 16–19 address the issues of the weak and strong zombies. We give zombies different chasing speeds as follows. First, we classify zombies based on two variables $level_L$ and $level_S$ that each zombie manages. For any zombie z, we call $z.level_L$ and $z.level_S$ the location level and swarm level of z. When a leader z becomes a zombie, it initializes both $z.level_L$ and $z.level_S$ with its level, i.e., $z.level$. Thereafter, a zombie z copies the level of $\psi(v(z))$ to $z.level_L$ and updates $z.level_S$ to be $\max\{b.level \mid b \in A_Z(v(z))\}$ in every $O(1)$ rounds. Since a zombie only chases a leader with an equal or greater level, $z.level_S \leq z.level_L$ always holds. We say that a zombie z is strong if $z.level_S = z.level_L$; z is weak otherwise. Then, we exploit the assumption that the agents are synchronous and let weak zombies move twice as frequently as strong zombies to chase a leader. As we prove in Lemmas 3.5–3.7, this difference in chasing speed results in a desirable property that $\min(\{a.level \mid a \in A_{AL}\} \cup \{z.level_L \mid z \in A_Z\})$ is monotonically non-decreasing and increases by at least one in every $O(n \log n)$ rounds, where A_{AL}, A_Z, respectively, are the set of active local leaders and zombies in the whole graph G until $A_{AL} \cup A_Z$ becomes empty. Thus, by Lemma 3.4, $A_{AL} \cup A_Z$ becomes empty and dispersion is achieved in $O(n \log^2 n)$ steps. Additionally, there might exist a neighboring agent oscillating between the local leader node and its home node. Therefore, whenever an agent moves to some other node it waits for one round to make sure that there does not exist any other agent oscillating between two neighboring nodes. Other than waiting for one round, a detailed description of these slots can be found in Sudo *et al.* [22].

Slots 12–15 discuss the case of helping agents not finding the settler agent in their neighbors. In that case, the local leader updates

Algorithm 1: Local Leader election for agent a

States: Initially, each agent a positioned at node $w = v(a)$ has $a.status \leftarrow candidate$. Similarly, $v = v(b)$. We call an explorer a leader if $a.local\_leader = a.ID$, otherwise a zombie.

1 /****** Slots 1 to 3 begins ******/
2 **if** $a.status = candidate$ **then**
3 **if** $a.init\_alone = True$ **then**
4 $Singleton\_Election(a)$
5 **if** $a.init\_alone = false$ *and* $\{a\} \neq A(w)$ **then**
6 **if** *a is the minimum ID agent at* $v(a)$ **then**
7 $a.status \leftarrow local\_leader$ and $A(w) \setminus \{\psi(w), a.local\_leader\}$ become the zombie. `// A(w) is the set of agents at node w.`
8 **if** $a.status = local\_leader$ **then**
9 **if** *a became local leader through* $Singleton\_Election(a)$ **then**
10 Inform all neighbors that a is a local leader.
11 **else if** *there is no agent across port-1* **then**
12 Local leader a places an agent there as a settler.
13 Ask the port-1 neighbor, say $\psi(v)$, to oscillate between v and w and keep the information that a is the local leader.
14 Agent $\psi(v)$ stores all other information corresponding to a until an agent settles at node $\psi(w)$.
15 Agent a does not settle at node w until it completes the DFS if a has the highest priority. `// A local leader a settles at some other agent position where an agent oscillates only if that place does not belong to the highest priority local leader.`
16 **if** *a completed the DFS and* $a.status = local\_leader$ **then**
17 Agent a becomes the $global\_leader$ and waits for n phases (collection of slots). `// Gathering of the local leader which became a zombie.`
18 Agent a traverses the whole graph by using DFS with the help of the "first child" port and "next sibling" port, informs all the settler agents, and settles the zombies along the way if needed.

the latest child with information about the next sibling as well as moves to the next available nodes to settle the agent as per HEO-DFS. On the other hand, Slots 20 and 21 update the pointers based on the forward/backward move. Finally, a local leader becomes the global leader after completing the DFS traversal and informs all the agents of it being a global leader by DFS traversal of G that was constructed during the HEO-DFS traversal with the help of parent, first child, latest child, and next sibling. If any zombie also accompanies the global leader that would be placed at any available empty node during the DFS traversal. A summary of these slots and their operations are provided in Table 3.

3.1 Analysis of the Algorithm

Lemma 3.1. *In Singleton_Election (Algorithm 2) run by agent a at node w, if there is a neighboring agent b positioned on the neighbor node v such that $\delta_w = \delta_v$ and δ_w, δ_v both being the minimum, a meets b in $O(\delta_w \log^2 n)$ rounds, running Neighbor_Exploration_with_Padding (Algorithm 3).*

Proof. Given that agent IDs are drawn from the interval $[1, n^{O(1)}]$, each agent's ID size is at most $c \cdot \log n$, for some constant c. Therefore, for any two agents a and b, there are two possible scenarios for their ID sizes: either their IDs have the same number of bits, or they differ in the number of bits.

Let us first consider the case where a and b have an equal number of bits, say $\mathcal{B}$. Since the IDs are unique, they must differ by at least

Algorithm 2: *Singleton_Election*(a)

$\delta_w \leftarrow$ degree of node w.
$N(v(a)) \leftarrow$ neighbors of node $v(a)$.
a visits neighbors in $N(w)$ (in increasing port numbers) one by one and stays there for two rounds.
while $a.status = candidate$ **do**
  if $\exists$ *neighbor* $v(b), \delta_w > \delta_v$ *or has status local_leader or* b *knows local_leader or (* $\exists$ *neighbor* $v(b), \delta_w = \delta_v$ *such that* $a.ID < b.ID$*)* **then**
    $a.status \leftarrow settler$
  else if $\exists$ *neighbor* $v, \delta_w = \delta_v$ *and* v *is empty* **then**
    $Neighbor\_Exploration\_with\_Padding(a)$
  else if $\forall$ *neighbor* $v, \delta_v > \delta_w$ *but* $\exists$ *(at least) a neighbor* v' *which is empty* **then**
    a visits the empty neighbors in the interval of $3\delta_{v'}$ rounds starting from $v(a)$ and ending at $v(a)$. `// Stay for one round at the neighboring node and two rounds spent in oscillation.`
    if *an agent* v' *is found at* $v(v')$ *and* v' *knows* $local\_leader$ *or has status* $local\_leader$ **then**
      $a.status \leftarrow settler$
  if $\forall$ *neighbor* $v, \delta_w < \delta_v$ *and all neighboring agents were initially singletons and no neighbor has status local_leader and (*$\forall$ *neighbor* $v(b)$*, if* $\delta_w = \delta_v$ *and* $b.ID < a.ID$*)* **then**
    $a.status \leftarrow local\_leader$

Algorithm 3: *Neighbor_Exploration_with_Padding*(a)

$\mathcal{B} \leftarrow$ number of bits in the ID of a
$\mathcal{B} + 2\mathcal{B}^2 \leftarrow$ number of bits in the ID of a after padding a sequence of '10' bits $\mathcal{B}^2$ times to the LSB in the original $\mathcal{B}$-bit ID.
Starting from MSB and ending on LSB, if the bit is '1' visit the $N(w)$ one by one and stay for two rounds at the neighboring node which finishes in $3\delta_w$ rounds. If the bit is '0' then $\psi(w)$ stay at w for $3\delta_w$ rounds.
for $3\delta_w(\mathcal{B} + 2\mathcal{B}^2)$ *rounds agent* a *explores* $N(w)$ *based on padding* **do**
  if *agent* a *meets an agent* b *which knows* $local\_leader$ *or (*$\exists$ *neighbor* $\psi(v), \delta_w = \delta_v$ *such that* $b.ID > a.ID$*)* **then**
    $a.status \leftarrow settler$.
  else if $\delta_u = \delta_v$ *and* $a.ID > b.ID$ **then**
    $a.status \leftarrow local\_leader$.
if $\exists$ *neighbor* $v, \delta_w = \delta_v$ *and* v *is empty* **then**
  $a.status \leftarrow settler$.

one bit. That is, if one agent has a '1' in the β-th position from the most significant bit (MSB), the other agent must have a '0' in that same position. Agents move to explore their neighbors when they have a '1' in the current bit and remain stationary when the bit is '0'. As a result, agent a will find b within $3 \cdot \delta_w \cdot \mathcal{B}$ rounds, where δ_w is the degree of a. Since there is at least one bit where their IDs differ, the total time required for a to meet b is $O(\delta_w \log n)$ rounds.

Now, consider the scenario where the IDs have unequal numbers of bits. Suppose a has $\mathcal{B}$ bits and b has $\mathcal{D}$ bits, with $\mathcal{B} \neq \mathcal{D}$. Without loss of generality, let $\mathcal{B} > \mathcal{D}$, i.e., $\mathcal{B} = \mathcal{D} + c_1$, where $\mathcal{D}, c_1 \geq 1$. After padding the ID of a, the total number of bits becomes $\mathcal{B} + 2\mathcal{B}^2$. To compute this explicitly, we have:

$$\mathcal{B} + 2\mathcal{B}^2 = (\mathcal{D} + c_1) + 2(\mathcal{D} + c_1)^2 = 2\mathcal{D}^2 + 2c_1^2 + 4 \cdot \mathcal{D} \cdot c_1 + \mathcal{D} + c_1.$$

Similarly, after padding, the total number of bits in the ID of b becomes $\mathcal{D} + 2\mathcal{D}^2$. Thus, the difference in the number of bits between a and b after padding is: $2c_1^2 + 4 \cdot \mathcal{D} \cdot c_1 + c_1$. Since $\mathcal{D}, c_1 \geq 1$, this difference is at least 7 bits. Furthermore, out of these 7 bits, at least 3 are '1's, during which a can explore its neighbors, while b

Algorithm 4: The behavior of local leader a

while *a does not know the leader and another* $2n$ *rounds do not pass* **do**
  /********** Slot 4 begins **********/
  Let $w = v(a)$
  if $\exists b \in A_L(w) \cup A_S(w) : a \prec b$ **then**
    $a.level_L \leftarrow a.level_S \leftarrow a.level$
    $a.local\_leader \leftarrow b.local\_leader$ `// a becomes a zombie and stops Algorithm 4`
  /********** Slot 5 begins **********/
  if $A(w) \neq \{a\}$ `// a is an active leader if` $A(w) \neq \{a\}$.
  then
    if $\psi(w) = \perp \vee \psi(w) \prec a$ **then**
      if $\psi(w) = \perp$ **then**
        Agent a waits for 1 round.
        if $\psi(w) = \perp$ *even after one round or oscillating agent covering lower priority leader's node* **then**
          Settle one zombie from $A_Z(w)$ at w.
          The oscillating agent, if any, settles one zombie other than node w and stops oscillation.
      $\psi(w).parent \leftarrow a.parent$ `// initially,` $a.parent = \perp$.
    if $\exists b \in A_Z(w) : a.level = b.level$ **then**
      $(a.level, b.level) \leftarrow (a.level + 1, 0)$
      $\psi(w).parent \leftarrow \perp$
      $a.InitProbe \leftarrow$ **true** `// Dispersion restart from` w.
    $(\psi(w).local\_leader, \psi(w).level) \leftarrow (a.ID, a.level)$
    if $a.InitProbe =$ ***true*** `// Initially,` $a.InitProbe =$ **true** **then**
      $(\psi(w).next, \psi(w).$checked, $\psi(w).$help, $\psi(w).done,$ $\psi(w).firstchild, \psi(w).latestchild) \leftarrow (\perp, -1, \perp$, **false**, $\perp, \perp)$
      $a.InitProbe \leftarrow$ **false**
    $Probe(a)$ `// See Algorithm 5`
    if $\psi(w).done =$ ***true*** **then**
      /********** Slot 20 begins **********/
      if $\psi(w).next = \perp$ **then**
        $\psi(w).next \leftarrow \psi(w).parent$ `// For backward move`
        Agent a moves to $\psi(w).latestchild$ and updates the nextsibling to $\perp$. `// The Latest child does not have the next sibling.`
        Agent a returns back to w.
        $\psi(w).latestchild \leftarrow \perp$
      All agents in $A(w) \backslash \{\psi(w)\}$ move to $N(w, \psi(w).next)$
      /********** Slot 21 begins **********/
      $a.parent \leftarrow a.$pin
      $a.InitProbe \leftarrow$ **true** `// Reached at the next node for dispersion. To figure out the next port for DFS Probing would be required.`

remains stationary at its position v after completing its exploration in $\delta_v(d + 2\mathcal{D}^2)$ rounds. Consequently, a has at least 3 opportunities to encounter b at node v.

Therefore, the round complexity is $O(\delta_w(\mathcal{B} + 2\mathcal{B}^2)) = O(\delta_w \log^2 n)$, as $\mathcal{B} \leq c \cdot \log n$. The lemma follows. □

Lemma 3.2. *An initially singleton agent* a *at node* w *running Singleton_Election (Algorithm 2) either becomes a local_leader or a settler within* $O(n \log^2 n)$ *rounds.*

We omit the proof of this lemma due to space constraints.

Lemma 3.3. *The Line 18 (in Algorithm 1) takes* $O(n)$ *rounds to inform all the agents in the graph.*

Algorithm 5: *Probe*(a)

```
1  /****************************** Slot 7 begins ***********************/
2  Let w = v(a)
3  ψ(w).next ← { min P,  if P ≠ φ
               { ⊥,     otherwise
4  where P = [0, ψ(w).checked] \ {b.pin | b ∈ A_S(w)\ψ(w)}
5  b.help ←⊥ for all b ∈ A_S(w) with b ≺ a.
6  Let all agents b ∈ A_S(w) with b ≺ a go back to their homes.
7  if ψ(w).next ≠⊥ ∨ ψ(w).checked = δ_w − 1 then
8      /****************************** Slot 8 begins *********************/
9      Execute b.help ←⊥ for each b ∈ A_S(w)\{ψ(w)}
10     Let all agents in A_S(w)\{ψ(w)} go back to their homes.
11     ψ(w).done ← true
12 else
13     /****************************** Slots 9 to 10 begins ******************/
14     Let {a_1, a_2, ..., a_x} be the set of agents in A_S(w)\{ψ(w)}
15     Let δ' = min(x, δ_w − 1 − ψ(w).checked)
16     Let u_i = N(w, i + ψ(w).checked) for i = 1, 2, ..., δ'
17     for each a_i ∈ {a_1, a_2, ..., a_δ'} in parallel do
18         a_i moves to u_i and stays for a round.   // Neighboring node
             might have an oscillating agent.
19         /****************************** Slot 11 begins ******************/
20         if (a_i.local_leader, a_i.level) =
             (ψ(u_i).local_leader, ψ(u_i).level) and a_i.found ≠ ψ(w)
             // ψ(w) does not consider his home node as empty
             node.
21          then
22             a_i.found ← true
23             ψ(u_i).help ← a.pin
24         else
25             a_i.found ← false
26         Move to N(u_i, a_i.pin)
27     /****************************** Slots 12 to 15 begins ******************/
28     ψ(w).checked ← ψ(w).checked + δ'
29     Let all agents in A_S(w)\{ψ(w)} go back to their homes.
30     if ∃i ∈ [1, δ'] : a_i.found = false then
31         ψ(w).next ← i + ψ(w).checked
32         if ψ(w).firstchild =⊥ then
33             ψ(w).firstchild ← i + ψ(w).checked
34             ψ(w).latestchild ← i + ψ(w).checked
35         else
36             Agent a moves to ψ(w).latestchild and update the
                 nextsibling at N(w, ψ(w).latestchild).
37             Agent a returns back to w.
38             ψ(w).latestchild ← i + ψ(w).checked
```

Algorithm 6: The behavior of a *settler* s in Slot 6

```
1  /****************************** Slot 6 begins ***********************/
2  if s.help ≠⊥ then
3      Move to N(v(s), s.help)
```

Algorithm 7: The behavior of a *zombie* z in Slots 16 and 19

```
1  /****************************** Slot 16 to 17 begins ***********************/
2  (z.level_L, z.level_S) ← (ψ(w).level, max{z'.level | z' ∈ A_Z(v(z))})
3  if A_L(v(z)) ≠ φ and z is a weak zombie then
4      Move to N(v(z), ψ(v(z)).next)
5  /****************************** Slot 18 to 19 begins ***********************/
6  if A_L(v(z)) = φ then
7      Move to N(v(z), ψ(v(z)).next)
```

Proof. In Line 18 of Algorithm 1, agent a traverses the DFS tree using the "first child" and "next sibling" ports. The agent a performs DFS traversal, utilizing these ports to navigate the tree. If an agent is not settled at any node, a waits for a round and then settles the zombie with the minimum ID, provided it is not $\psi(w)$, the home node of agent a. This process requires at most $3n$ rounds: up to $2n$ rounds to traverse all edges twice and an additional n rounds for waiting at nodes where agents may oscillate. Hence, the overall round complexity is $O(n)$. □

Lemma 3.4. *The level of an agent is always at most* $\log n + 1$.

Proof. A level-up event involves one *local_leader*, a, and one zombie, b, both at the same level. After the event, zombie b is assigned level 0 and will never initiate a level-up event again, as the *local_leader*'s level is non-decreasing, starting from level 1. Consequently, for any $i \geq 1$, the maximum number of agents that can reach level i is $\lfloor n/2^{i-1} \rfloor$, which proves the lemma. □

Lemma 3.5. *The location level of a zombie is monotonically non-decreasing.*

Proof. Neither a *local_leader* nor a settler decreases its level in Dispersion. When a zombie z does not accompany a *local_leader*, it chases a *local_leader* through port $\psi(v(z)).next$. This port $\psi(v(z)).next$ is updated only if a *local_leader* makes a forward or backward move from $v(z)$, and the *local_leader* updates the level of $\psi(N(v(z), \psi(v(z)).next))$ if it is smaller than its level. Thus, a zombie never decreases its location level by chasing a *local_leader*. When a zombie z accompanies a *local_leader*, the *local_leader* copies its level to $\psi(v(z)).level$ in slot 5, which is copied to $z.level_L$ in slots 12-15. The *local_leader* that z accompanies may change but does not change to a weaker *local_leader*. Thus, a zombie never decreases its location level when accompanying a *local_leader*. □

Lemma 3.6. *The number of weak zombies with a location level* $i \geq 0$ *is monotonically non-increasing starting from any configuration where* $min(\{a.level \mid a \in A_{AL}\} \bigcup \{z.level_L \mid z \in A_Z\}) = i$.

Proof. Let C be a configuration where $\min(\{a.\text{level} \mid a \in A_{AL}\} \cup \{z.level_L \mid z \in A_Z\}) = i$. When a *local_leader* at level i turns into a zombie, its location level becomes i (Line 5). As a result, a *local_leader* at level i may become a strong zombie with location level i, but it will never become a weak zombie at the same location level. A zombie's swarm level only decreases when it accompanies a *local_leader* (and this *local_leader* settles another zombie). Therefore, a strong zombie with location level i that does not accompany a *local_leader* cannot turn into a weak zombie without first increasing its location level. Additionally, starting from C, a strong zombie with location level i must increase its location level upon encountering a *local_leader* in slot 4. Thus, the number of weak zombies at location level i decreases monotonically. □

Lemma 3.7. *In Lemma 3.6, location level* i *is monotone non-decreasing and increases by at least one in every* $O(n \log n)$ *time steps unless* $A_{AL} \bigcup A_Z$ *becomes empty.*

Proof. Let i be an integer such that $i \geq 0$ and C be a configuration where $\min(\{a.level \mid a \in A_{AL}\} \cup \{z.level_L \mid z \in A_Z\}) = i$.

It is sufficient to show that *local_leader*s with level i and zombies with location level i disappear within $O(n \log n)$ time steps, starting from C. Consider an execution starting from configuration C. By Lemma 3.6, no weak zombie with location level i is newly created during this execution. Let z represent any weak zombie at location level i that is not accompanied by a *local_leader* in configuration C. Over every 21 time slots, z moves twice, whereas both a strong zombie and a *local_leader* only move once, excluding movement related to probing. Consequently, z either catches up to a strong zombie and becomes strong reaches a *local_leader* with level i, or increases its location level within $O(n)$ time steps. When z reaches a *local_leader*, it either joins the *local_leader*'s HEO-DFS or the *local_leader* turns into a zombie. In the latter case, z becomes a strong zombie. Therefore, within $O(n \log n)$ time steps, z either settles or becomes a strong zombie (aligned with the current *local_leader*). As a result, the number of weak zombies with location level i is reduced to zero in $O(n \log n)$ time steps.

Once all weak zombies at location level i are gone, no waiting *local_leader* at level i will resume its HEO-DFS without first increasing its level, since no weak zombies remain at that level. Consequently, every active *local_leader* at location level i either turns into a zombie with location level at least $i + 1$ or becomes a waiting *local_leader* within $O(n \log n)$ time steps. Therefore, active *local_leader*s at location level i also disappear within $O(n \log n)$ time steps. From this point onward, no *local_leader* will move within the territory of a group with level i or lower. As a result, every strong zombie with location level i either increases its location level or catches up to a waiting *local_leader*. Since the waiting *local_leader*'s level is at least i, catching up also results in z increasing its level by at least one. □

Proof of Theorem 1.1: Lemmas 3.1–3.7 support the round complexity $O(n \log^2 n)$. From Algorithms 1–7, it follows that each agent a uses the constant amount of information to track its status – $a.ID$, status as a candidate, settler, zombie, local leader, global leader, and other pointers of a parent, first child, latest child, and next sibling along with some initial probes. All of these require $O(\log n)$ bits of memory. In Algorithm 1, Line 18 ensures that the highest priority agent completes the DFS traversal in $O(n)$ rounds. Since agents have unique priority, a unique agent is elected as a leader. □

4 APPLICATIONS TO OTHER PROBLEMS

We use leader election Theorem 1.1 and provide improved solutions to MST, MIS, MDS, and Gathering in the agent-based model (see Table 2 for result summary and comparison). All these problems assume n agents initially located arbitrarily on the nodes of G.

Gathering. The problem of gathering asks the agents to relocate autonomously to position them on a node of G not fixed a priori. We establish the following result.

Theorem 4.1. *There is a deterministic algorithm that solves gathering in G in $O(n \log^2 n)$ rounds with $O(\log n)$ bits at each agent, without agents knowing any graph parameter a priori.*

This is an improvement over the state-of-the-art result of [10] which solves gathering in $O(m)$ time with $O(n \log n)$ bits at each agent. Our solution is as follows: first elect a leader using Algorithm 1 in $O(n \log^2 n)$ rounds with $O(\log n)$ bits at each agent, then ask the leader to re-traverse the DFS tree built during leader election to collect the agents to its root node which finishes in $O(n)$ time with $O(\log n)$ bits per agent.

MIS. The problem of maximal independent set (MIS) asks the agents to relocate autonomously to find a subset $S \subset V$ of nodes such that S forms an MIS of G. We establish the following result.

Theorem 4.2. *There is a deterministic algorithm that finds an MIS of G in $O(n \log^2 n + n\Delta)$ rounds with $O(\log n)$ bits per agent, without agents knowing any graph parameter a priori.*

This is an improvement on memory over the state-of-the-art result of [10] which finds a MIS in $O(n\Delta)$ rounds with $O(n \log n)$ bits at each agent. Our solution is as follows: first solve gathering as above and then use the technique of Pattanayak *et al.* [18], which starting from a gathered configuration, finds an MIS of G in $O(n\Delta)$ rounds with $O(\log n)$ bits at each agent.

MDS. A *dominating set* of G is a subset $DS \subset V$ of nodes such that if $v \notin DS$, v has a neighbour in DS. We establish the following result.

Theorem 4.3. *There is a deterministic algorithm that finds an MDS of G in $O(n \log^2 n + m)$ rounds with $O(\log n)$ bits at each agent, without agents knowing any graph parameter a priori.*

This is an improvement on memory over the state-of-the-art result of [10] which finds an MDS in $O(m)$ rounds with $O(n \log n)$ bits at each agent. Our solution is as follows: first solve gathering as above and then use the technique of Chand *et al.* [4], which starting from a gathered configuration, finds an MDS of G in $O(m)$ rounds with $O(\log n)$ bits at each agent.

MST. An MST of G is a spanning tree that includes all vertices in V and has the minimum possible total edge weight, with no cycles. We establish the following result.

Theorem 4.4. *There is a deterministic algorithm that finds an MST of G in $O(m + n \log^2 n)$ rounds with $O(\Delta \log n)$ bits at each agent, without agents knowing any graph parameter a priori.*

This is an improvement on memory over the state-of-the-art result of [10] which finds an MST in $O(m + n \log n)$ rounds with $O(n \log n)$ bits at each agent. Our solution is as follows: first elect a leader as in Algorithm 1 and then use the MST technique of Kshemkalyani *et al.* [10].

5 CONCLUDING REMARKS

In this paper, we have studied leader election, a fundamental and widely-studied problem, in the agent-based model which extends the message-passing model in a new direction. Specifically, we have developed a deterministic algorithm that elects an agent at a graph node as a leader in $O(n \log^2 n)$ rounds with only $O(\log n)$ bits at each agent, without agents knowing any graph parameter a priori. This result is interesting and significant from three aspects: (i) It is a substantial improvement over the best previously known $O(m)$ time $O(n \log n)$ bits/agent result, (ii) The memory bound is optimal and time bound is optimal within an $O(\log^2 n)$ factor, and (iii) The result became central in improve time and/or memory complexities of many other fundamental distributed graph problems, such as MIS, MDS, MST, and gathering. For the future work, it would be interesting to remove $O(\log^2 n)$ factor from time to obtain simultaneously time-and-memory-optimal leader election algorithm.

REFERENCES

[1] John Augustine and William K. Moses Jr. 2018. Dispersion of Mobile Robots: A Study of Memory-Time Trade-offs. In *ICDCN*. 1:1–1:10.

[2] Baruch Awerbuch. 1987. Optimal Distributed Algorithms for Minimum Weight Spanning Tree, Counting, Leader Election and Related Problems (Detailed Summary). In *STOC*, Alfred V. Aho (Ed.). ACM, 230–240.

[3] Baruch Awerbuch, Andrew V Goldberg, Michael Luby, and Serge A Plotkin. 1989. Network decomposition and locality in distributed computation. In *FOCS*, Vol. 30. Citeseer, 364–369.

[4] Prabhat Kumar Chand, Anisur Rahaman Molla, and Sumathi Sivasubramaniam. 2023. Run for Cover: Dominating Set via Mobile Agents. In *ALGOWIN*. Springer, 133–150.

[5] Yang Cong, gu Changjun, Tao Zhang, and Yajun Gao. 2021. Underwater Robot Sensing Technology: A Survey. *Fundamental Research* 1 (03 2021).

[6] Janosch Deurer, Fabian Kuhn, and Yannic Maus. 2019. Deterministic distributed dominating set approximation in the CONGEST model. In *PODC*. 94–103.

[7] Abdulrahman El-Sayed, Peter Scarborough, Lars Seemann, and Sandro Galea. 2012. Social network analysis and agent-based modeling in social epidemiology. *Epidemiologic perspectives & innovations : EP+I* 9 (02 2012).

[8] Robert G. Gallager, Pierre A. Humblet, and Philip M. Spira. 1983. A Distributed Algorithm for Minimum-Weight Spanning Trees. *ACM Trans. Program. Lang. Syst.* 5, 1 (jan 1983), 66–77.

[9] Juan A. Garay, Shay Kutten, and David Peleg. 1993. A Sub-Linear Time Distributed Algorithm for Minimum-Weight Spanning Trees (Extended Abstract). In *FOCS*. IEEE Computer Society, 659–668.

[10] Ajay D. Kshemkalyani, Manish Kumar, Anisur Rahaman Molla, and Gokarna Sharma. 2024. Agent-based Leader Election, MST, and Beyond. In *DISC*. LIPIcs, 50:1–50:7.

[11] Ajay D. Kshemkalyani, Anisur Rahaman Molla, and Gokarna Sharma. 2019. Fast Dispersion of Mobile Robots on Arbitrary Graphs. In *ALGOSENSORS*. 23–40.

[12] Manish Kumar and Anisur Rahaman Molla. 2023. On the Message Complexity of Fault-Tolerant Computation: Leader Election and Agreement. *IEEE Trans. on Parallel and Dist. Syst.* 34, 4 (2023), 1115–1127.

[13] Shay Kutten, Gopal Pandurangan, David Peleg, Peter Robinson, and Amitabh Trehan. 2015. On the Complexity of Universal Leader Election. *J. ACM* 62, 1 (2015), 7:1–7:27.

[14] Gérard Le Lann. 1977. Distributed Systems - Towards a Formal Approach. In *IFIP Congress*, Bruce Gilchrist (Ed.). North-Holland, 155–160.

[15] Jaeyeong Lee, Sunwoo Shin, Moonsung Park, and Chongman Kim. 2018. Agent-Based Simulation and Its Application to Analyze Combat Effectiveness in Network-Centric Warfare Considering Communication Failure Environments. *Mathematical Problems in Engineering* 2018 (12 2018), 1–9.

[16] Anisur Rahaman Molla, Kaushik Mondal, and William K. Moses Jr. 2023. Fast Deterministic Gathering with Detection on Arbitrary Graphs: The Power of Many Robots. In *IPDPS*. IEEE, 47–57.

[17] Alessandro Panconesi and Aravind Srinivasan. 1996. On the complexity of distributed network decomposition. *Journal of Algorithms* 20, 2 (1996), 356–374.

[18] Debasish Pattanayak, Subhash Bhagat, Sruti Gan Chaudhuri, and Anisur Rahaman Molla. 2024. Maximal Independet Set via Mobile Agents. In *ICDCN*. ACM, 74–83.

[19] David Peleg. 1990. Time-optimal leader election in general networks. *J. Parallel Distrib. Comput.* 8, 1 (jan 1990), 96–99.

[20] David Peleg. 2000. *Distributed Computing: A Locality-sensitive Approach*. SIAM, Philadelphia, PA, USA.

[21] Mohamed Rihan, Mahmoud Selim, Chen Xu, and Lei Huang. 2019. D2D Communication Underlaying UAV on Multiple Bands in Disaster Area: Stochastic Geometry Analysis. *IEEE Access* (2019).

[22] Yuichi Sudo, Masahiro Shibata, Junya Nakamura, Yonghwan Kim, and Toshimitsu Masuzawa. 2024. Near-linear Time Dispersion of Mobile Agents. In *DISC*. LIPIcs, 38:1–38:22.

[23] Amnon Ta-Shma and Uri Zwick. 2014. Deterministic Rendezvous, Treasure Hunts, and Strongly Universal Exploration Sequences. *ACM Trans. Algorithms* 10, 3 (2014), 12:1–12:15.

[24] Grzegorz Wilk-Jakubowski, Radoslaw Harabin, and Stanislav Ivanov. 2022. Robotics in crisis management: A review. *Technology in Society* (2022).

[25] Minhao Wu. 2023. Robotics Applications in Natural Hazards. *Highlights in Science, Engineering and Technology* (2023).

[26] Nan Zhao, Weidang Lu, Min Sheng, Yunfei Chen, Jie Tang, F. Yu, and Kai-Kit Wong. 2019. UAV-Assisted Emergency Networks in Disasters. *IEEE Wireless Communications* (2019).

[27] Chengxiang Zhuge, Chunfu Shao, and Binru Wei. 2018. An Agent-based Spatial Urban Social Network Generator: A Case Study of Beijing, China. *Journal of Computational Science* 29 (09 2018).