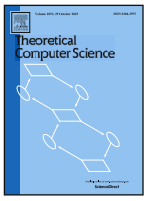
Theoretical Computer Science 1059 (2026) 115578

Contents lists available at ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs



Faster leader election via mobile agents and its applications ⋆

Ajay D. Kshemkalyani [a], Manish Kumar [b,1,*], Anisur Rahaman Molla [c,2], Debasish Pattanayak [d], Gokarna Sharma [e]

[a] Department of Computer Science, University of Illinois Chicago, Chicago, IL, 60607, USA
[b] Department of Computer Science & Engineering, IIT Madras, Chennai, 600036, India
[c] Cryptology and Security Research Unit, Indian Statistical Institute, Kolkata, 700108, India
[d] Department of Computer Science & Engineering, IIT Indore, Indore, 452020, India
[e] Department of Computer Science, Kent State University, Kent, OH, 44242, USA

ARTICLE INFO

Section Editor: Dr. Pinyan Lu
Handling Editor: Katie Harris

Keywords:
Distributed algorithms
Mobile agents
Local communication
Leader election
MST
MIS
Gathering
Minimal dominating sets
Time and memory complexity
Graph parameters

ABSTRACT

Leader election is a critical and extensively studied problem in distributed computing. This paper introduces the study of leader election using mobile agents. Consider n agents initially placed arbitrarily on the nodes of an arbitrary, n-node, m-edge graph G. These agents move autonomously across the nodes of G and elect one agent as the leader such that the leader is aware of its status as the leader, and the other agents know they are not the leader. The goal is to minimize both time and memory usage.

We study the leader election problem in a synchronous setting where each agent performs operations simultaneously with the others, allowing us to measure time complexity in terms of rounds. We assume that the agents have prior knowledge of the number of nodes n and the maximum degree of the graph Δ.

We first elect a leader deterministically in $O(n \log^2 n + D\Delta)$ rounds with each agent using $O(\log n)$ bits of memory, where D is the diameter of the graph. Leveraging this leader election result, we then present a deterministic algorithm for constructing a minimum spanning tree of G in $O(m + n \log n)$ rounds, with each agent using $O(\Delta \log n)$ bits of memory.

Furthermore, using the same leader election result, we improve time and memory bounds for other key distributed graph problems, including gathering, maximal independent set, and minimal dominating set. For all the aforementioned problems, our algorithm remains memory optimal.

Finally, we also perform simulations to validate our theoretical results and show the performance trends of our algorithm under different graph topologies.

1. Introduction

Leader election is one of the fundamental and well-studied problems in distributed computing due to its significance in various applications, including resource allocation, reliable replication, load balancing, synchronization, membership management, and crash

⋆ A preliminary version of this article appeared in the Proceedings of ICDCIT 2025 [1].

* Corresponding author.
E-mail addresses: ajay@uic.edu (A.D. Kshemkalyani), manishsky27@gmail.com (M. Kumar), molla@isical.ac.in (A.R. Molla), debasish@iiti.ac.in (D. Pattanayak), gsharma2@kent.edu (G. Sharma).

[1] The work of M. Kumar is supported by the Centre for Cybersecurity, Trust and Reliability (CyStar), IIT Madras.
[2] The work of A. R. Molla was supported, in part, by ISI DCSW Project, file no. H5413.

https://doi.org/10.1016/j.tcs.2025.115578
Received 14 April 2025; Received in revised form 25 August 2025; Accepted 11 October 2025
Available online 22 October 2025

Table 1
Summary of our results - leader election, MST, MIS, MDS and gathering.

Summarizing Results of This Paper			
Algorithm	Knowledge	Time	Memory (optimal)
Leader Election	n, Δ	$O(n \log^2 n + D\Delta \log n)$	$O(\log n)$
MST	Leader, Dispersed	$O(m + n \log n)$	$O(\Delta \log n)$
MIS	Leader, Dispersed	$O(n \log \Delta)$	$O(\log n)$
MDS	Leader, Dispersed	$O(n \log \Delta)$	$O(\log n)$
Gathering	Leader, Dispersed	$O(n \log \Delta)$	$O(\log n)$

recovery. It can also be viewed as a form of symmetry breaking, where a single designated process or node (the leader) is responsible for making the critical decisions. In this paper, we explore the graph-level task of leader election in a distributed network under the agent-based model. In the agent-based model, the leader election problem involves a group of agents operating within a distributed network, where the goal is for exactly one agent to declare itself as the leader.

The agent-based model is particularly valuable in scenarios like private military networks or sensor networks in remote areas, where direct access to the network may be challenging, but small, battery-powered, relocatable devices can traverse the environment to gather information about the network's structure for management purposes. Significant applications of the agent-based model in network management can be found in areas such as underwater navigation [2], network-centric warfare in military systems [3], social network modeling [4], and the study of social epidemiology [5]. Moreover, limited storage capacity reduces the risk of sensitive network information being exposed if a device is compromised. Additionally, mobility in these devices enhances communication security by mitigating concerns over message interception.

Recent applications of the agent-based model have emerged in various areas. A notable example is the work by Martinkus et al. [6], which introduces AgentNet-a graph neural network (GNN) architecture where a group of relocatable (neural) devices, referred to as neural agents, 'walk' across the graph and collaboratively classify graph-level tasks such as detecting triangles, cliques, and cycles. In this model, neural agents can gather information from the nodes they occupy, neighboring nodes they visit, and other co-located devices. Moreover, a recent study [7] demonstrated that a fundamental graph-level task can be solved in the agent-based model using a deterministic algorithm within $O(\Delta \log n)$ rounds, with each device requiring only $O(\Delta \log n)$ bits of storage.

We present a deterministic algorithm for leader election that offers provable guarantees on two key performance metrics in the agent-based model: the solution's *time complexity* and the storage requirements for each agent. We focus specifically on deterministic algorithms, as they may be more appropriate for relocatable devices. In the agent-based model, storage requirements are treated as a primary performance metric alongside time complexity. The objective in the agent-based model is to minimize storage usage, ideally keeping it as small as the size of a device identifier, typically $O(\log n)$ bits per device. The limited storage prevents relocatable devices from first traversing the graph to learn the topology and then performing graph computation as a separate step. Our goal is to develop an algorithm that is storage-optimal.

Building on our deterministic leader election algorithm, which guarantees both time and storage efficiency, we present improved algorithms for several fundamental distributed graph problems, including minimum spanning tree (MST), maximal independent set (MIS), minimal dominating set (MDS), and gathering. A summary of these results, detailing their time and memory complexities along with the required knowledge, is provided in Table 1. Our leader election algorithm requires prior knowledge of the number of nodes/agents, denoted as n, and the maximum degree of the graph, Δ. We assume that the number of agents equals the number of nodes in the graph, which directly applies to graph problems such as MST, MIS, MDS, and gathering. Furthermore, the results presented in Table 1, aside from the leader election algorithm, assume that the leader is known-that is, each agent is aware of whether they are the leader. Additionally, our leader election process disperses the agents, so we assume each agent is positioned at a distinct node. The aforementioned problem has been studied by Kshemkalyani et al. [8], where no prior knowledge of global parameters was assumed. In contrast, our approach, with prior knowledge of the number of nodes and the maximum degree of the graph, achieves faster and memory-optimal results. A comparative analysis of our results is presented in Table 2.

1.1. Our results

We examine the leader election problem in a synchronous setting where all agents operate simultaneously, allowing us to measure time complexity in terms of rounds. We assume that the agents have prior knowledge of both the number of nodes n and the maximum degree Δ of the graph. We show the following main results.

1. **Leader Election -** Given any configuration of n agents positioned initially arbitrarily on the nodes of a n-node graph G, there is a deterministic algorithm that elects a leader in $O(n \log^2 n + D\Delta \log n)$ rounds with $O(\log n)$ bits of memory per agent for a known value of n and Δ.
2. **Minimum Spanning Tree (MST) -** Given any configuration of n agents positioned initially arbitrarily on the nodes of a n-node graph G, there is a deterministic algorithm that finds an MST of G in $O(m + n \log^2 n + D\Delta \log n)$ rounds with $O(\Delta \log n)$ bits per agent for a known value of n and Δ.
3. **Maximal Independent Set (MIS) and Minimal Dominating Set (MDS) -** Given any configuration of n agents positioned initially arbitrarily on the nodes of a n-node graph G, there is a deterministic algorithm that finds an MIS of G in $O(n \log^2 n + D\Delta \log n)$ rounds

Table 2
Summary of previous and our results in the agent-based model. M is the memory required for the Universal Exploration Sequence (UXS) [13] and γ is the number of clusters of agents in the initial configuration. '-' means no prior knowledge of any parameter is required.

Comparative Analysis of the Closely Related Work			
Algorithm	Knowledge	Time	Memory
Leader Election			
Section 4	n, Δ	$O(n \log^2 n + D\Delta \log n)$	$O(\log n)$
Kshemkalyani et al. [8]	–	$O(m)$	$O(n \log n)$
MST			
Section 5	n, Δ	$O(m + n \log^2 n + D\Delta \log n)$	$O(\Delta \log n)$
Kshemkalyani et al. [8]	–	$O(m + n \log n)$	$O(n \log n)$
MIS			
Section 5	n, Δ	$O(n \log^2 n + D\Delta \log n)$	$O(\log n)$
Pattanayak et al. [10]	n, Δ	$O(n\Delta \log n)$	$O(\log n)$
Kshemkalyani et al. [8]	–	$O(n\Delta)$	$O(n \log n)$
MDS			
Section 5	n, Δ	$O(n \log^2 n + D\Delta \log n)$	$O(\log n)$
Chand et. al [11]	n, Δ, m, γ	$O(\gamma\Delta \log n + n\gamma + m)$	$O(\log n)$
Kshemkalyani et al. [8]	–	$O(m)$	$O(n \log n)$
Gathering			
Section 5	n, Δ	$O(n \log^2 n + D\Delta \log n)$	$O(\log n)$
Molla et al. [12]	n	$O(n^3)$	$O(M + m \log n)$
Kshemkalyani et al. [8]	–	$O(m)$	$O(n \log n)$

with $O(\log n)$ bits per agent for a known value of n and Δ. Every MIS is an MDS, and the MDS bound follows directly from the MIS result.

4. **Gathering -** Given any configuration of n agents positioned initially arbitrarily on the nodes of a n-node graph G, there is a deterministic algorithm that collects all n agents to a node in G not fixed a priori in $O(n \log^2 n + D\Delta \log n)$ rounds with $O(\log n)$ bits per agent for a known value of n and Δ.

Furthermore, we also carried out simulations of leader election across various graph topologies to validate our theoretical findings and illustrate the algorithm's performance trends under different structural properties.

The rest of the paper is organized as follows.

Paper Organization: Section 2 states our distributed computing model. Section 3 discusses the closely related work. Section 4 presents the leader election algorithm, which is the main contribution of the paper. Section 5 explores some graph-related applications of the leader election with improved time/memory complexity. Section 6 presents the simulation results of our leader election algorithm. Lastly, Section 7 concludes the paper with some interesting future work.

2. Computational model

We model the network as a connected, undirected graph $G = (V, E)$ with $|V| = n$ nodes and $|E| = m$ edges. Each node $v_i \in V$ has δ_i ports corresponding to each incident edge, labeled $[1, \ldots, \delta_i]$. We consider a set $\mathcal{R} = \{r_1, r_2, \ldots, r_n\}$ of n agents, initially positioned on the nodes of G[1] The agents have unique IDs within the range $[1, n^c]$, for some constant c, and agents know the value of c.

Initially, a node may host zero, one, or multiple agents. An agent at a node can communicate with other agents present at the same node but cannot communicate with agents at different nodes (this is the *local communication* model). An agent can move from node v to a neighboring node u via the edge e_{vu}. Following standard message-passing models, such as in [9], we assume that an agent can traverse an edge within a single round, regardless of the edge's weight, even in weighted graphs. When an agent moves from v to u along port p_{vu}, it learns the corresponding port p_{uv} upon arrival at u. Furthermore, at any node v, the agent is aware of the weight $w(e)$ (if G is weighted) of the edge e_{vu} connecting v to its neighbor u. We assume no correlation between the two port numbers of an edge. Multiple agents can traverse an edge simultaneously, as the agent-based model imposes no restrictions on how many agents may move across an edge at the same time.

The agents operate in a synchronous setting, similar to the standard $\mathcal{CONGEST}$ model: In each round, an agent r_i at node v_i can perform local computations based on its stored information and the (unique) port labels at its current node, and then decide to either remain at the node or move to a neighboring node via a port. Before leaving, the agent may write information to the storage of another agent at the same node. An agent exiting a node always reaches another node by the end of the round (agents are never positioned on edges at the end of a round).

[1] Certain graph problems may be solvable with $k < n$ agents, but not all, such as the Minimum Spanning Tree (MST), where computing the MST with $k < n$ agents will yield the MST of a subgraph of G.

The agents are assumed to wake up simultaneously at the beginning of the execution. The time complexity is measured in the number of rounds required to reach a solution, while the memory complexity is the number of bits stored by each agent during the execution.

At any round, the agent distribution on G may satisfy one of the following configurations:

- *Dispersed*: each of the n agents occupies a distinct node,
- *Rooted*: all n agents are gathered at a single node,
- *General*: the configuration is neither dispersed nor rooted.

3. Related work

The leader election problem was first stated by Le Lann [14] in the context of token ring networks, and since then it has been central to the theory of distributed computing. Gallager, Humblet, and Spira [15] provided a deterministic algorithm for any n-node graph G with time complexity $O(n \log n)$ rounds and message complexity $O(m + n \log n)$. Awerbuch [16] provided a deterministic algorithm with time complexity $O(n)$ and message complexity $O(m + n \log n)$. Peleg [17] provided a deterministic algorithm with optimal time complexity $O(D)$ and message complexity $O(mD)$. Recently, an algorithm was given in [18] with message complexity $O(m)$ but no bound on time complexity, and another algorithm with $O(D \log n)$ time complexity and $O(m \log n)$ message complexity. Additionally, it was shown in [18] that the message complexity lower bound of $\Omega(m)$ and time complexity lower bound of $\Omega(D)$ for deterministic leader election in graphs. Leader election was not studied in the agent-based model before, except for [8]. See Table 2.

For MST, the algorithm in Gallager, Humblet, and Spira [15] is the first deterministic algorithm in the message-passing model with time complexity $O(n \log n)$ and message complexity $O(m + n \log n)$. Time was improved to $O(n)$ in [16] and to $O(\sqrt{n} \log^* n + D)$ in [9,19]. Furthermore, a time lower bound of $\Omega(\sqrt{n}/\log n + D)$ was given in [20]. MST was not studied in the agent-based model before, except for [8]. See Table 2.

For the MIS problem in the message-passing model, the best-known deterministic distributed algorithms initially had a time complexity of $O(2^{\sqrt{\log n}})$ [21,22], which was later improved to $O(\log^6 n)$ [23]. For MDS, Deurer et al. [24] gave two algorithms with an optimal approximation ratio of $(1+\epsilon)(1+\log(\Delta+1))$ running respectively in $O(2^{O(\sqrt{\log(n)\log(\log(n))})})$ and $O(\Delta \operatorname{polylog}(\Delta) + \operatorname{polylog}(\Delta) \log^\star(n))$ rounds for $\epsilon > \frac{1}{\operatorname{polylog}(\Delta)}$ since every MIS is an MDS, therefore, the result of MIS [23] also holds for MDS. MIS and MDS were solved in the agent-based model in [10,11] with time and memory complexities reported in Table 2 assuming n, Δ (additionally m, γ for MDS) are known to agents a priori. Kshemkalyani et al. [8] improved these results w.r.t. time/memory complexities as well as lifted the parameter assumptions, see Table 2.

Gathering is a very old problem and has been studied extensively in the agent-based model. The recent results are [12,13] (detailed literature in [12]). Ta-Shma and Zwick [13] provided a $\tilde{O}(n^5 \log \beta)$ time solution to gather $k \leq n$ agents in arbitrary graphs, where $\tilde{O}$ hides polylog factors and β is the smallest label among agents. Molla et al. [12] provided improved time bounds for large values of k assuming n is known but not k: (i) $O(n^3)$ rounds, if $k \geq \lfloor * \rfloor \frac{n}{2} + 1$ (ii) $\tilde{O}(n^4)$ rounds, if $\lfloor * \rfloor \frac{n}{2} + 1 \leq k < \lfloor * \rfloor \frac{n}{3} + 1$, and (iii) $\tilde{O}(n^5)$ rounds, if $\lfloor * \rfloor \frac{n}{3} + 1 > k$. Each agent requires $O(M + m \log n)$ bits of memory, where M is the memory required to implement the universal traversal sequence (UXS) [13]. Kshemkalyani et al. [8] lifted the assumption of known n and improved time bound for $k = n$ case from $O(n^3)$ to $O(m)$ and additionally the memory bound.

Finally, there is a model, called *whiteboard* [25], related to the agent-based model. The whiteboard model considers (limited) storage at the network nodes in addition to (limited) storage per device. Other aspects remain the same as in the agent-based model, for example, devices have identifiers, computation ability, and communication through relocation. It is immediate that any solution designed in the agent-based model works in the whiteboard model without any change, but the opposite may not be true.

4. Leader election

In this section, we present our deterministic leader election algorithm starting from any initial configuration (dispersed, rooted, or arbitrary) of n agents on an n-node graph G, ensuring that one agent is elected as a leader. Initially, we disperse the agents on the graph with the help of the algorithm discussed in [26]. After dispersing the agents, we elect the leader. The running time of our algorithm depends on the starting configuration, increasing from the dispersed configuration to the rooted configuration, and finally to the general configuration. Our primary goal is to minimize the round complexity by keeping the optimal memory (in bits) per agent.

4.1. Challenges and high-level idea for leader election

We are given a set $\mathcal{R}$ of n agents with unique IDs positioned initially arbitrarily on the nodes of an n-node, m-edge graph G such that n and Δ are known to the agents. We also assume that the range $[1, n^c]$ of the IDs is known to the agents for any constant $c > 1$. We aim to elect a unique leader; the pseudocode is given in Algorithm 1. Initially, we disperse all the agents (if not dispersed already) with a well-known algorithm [26]. Then each agent iteratively passes the information of a higher known ID to its neighbors, and finally, the agent with the highest ID agent becomes the leader. The algorithm (after dispersion) uses $O(D\Delta \log n)$ rounds and $O(\log n)$ bits of memory per agent. Our leader election following dispersion presents three key challenges to the aforementioned complexity: (i) The graph diameter is unknown, making it difficult to terminate the algorithm. (ii) Since agents are mobile entities, therefore,

meeting neighboring agents is challenging. (iii) The algorithm assumes each agent has only $O(\log n)$ bits of memory, making it difficult to track all the information passed by neighboring agents.

On a high level, we deal with all the above challenges in the following ways (in order): (i) Based on a voting mechanism, our algorithm ensures that the agents have passed the information (about the highest ID) from one end of the graph to another. (ii) Based on ID, our algorithm makes sure that each neighbor meets all of its neighbors in a definite interval of time/rounds. (iii) Based on the child-parent relationship, our algorithm makes sure only each child keeps track of its parent rather than the parent keeps track of all its children, which requires $\Delta \log n$ memory.

Algorithm 1: Leader election.

Input : A set $\mathcal{R}$ of n agents with unique IDs positioned initially arbitrarily on the nodes of an n-node, m-edge graph G such that n and Δ are known to the agents.

Output: An agent in $\mathcal{R}$ is elected as a leader with status *leader*.

```
1  Disperse the agents in O(n log^2 n) rounds with the help of an arbitrary dispersion algorithm in [26].▷ Takes O(n log^2 n) rounds.
2  A phase consists of O(Δ log n) rounds, in which agent r_u meets all its neighbors, follows from Lemma 1.
3  Each agent r_u does nothing for a phase of cΔ log n rounds.
4  while agent r_u is unaware about the leader do
5      if agent r_u has not sent its highest ID known from the previous phase then
6          Agent r_u sends its highest known ID to all the neighbors
7      if r_u receives higher ID from neighbour r_v then
8          r_u considers r_v as its parent and becomes the child of r_v
9          if r_v remains parent of r_u for next two phases then
10             r_u sends "undecided" vote to r_v for next two phases // Initialization of ''undecided'' vote.
11     if r_u received a "yes" vote and did not receive a vote "undecided" from all its children (if any), in the previous phase, after two
         phases of proposal then
12         r_u sends the vote "yes" to its parent (if any)
13     else
14         r_u sends the vote "undecided" to its parent (if any)
15     if r_u does not have child then
16         r_u keeps sending a vote "yes" to its parent until it gets to know about the leader // r_u did not receive a vote
               from any neighbor.
17     if r_u received its own proposed ID from all its children with a "yes" vote in the previous phase then
18         r_u becomes the leader and informs its ID to all neighbors
19     if r_u knows the leader ID from the previous phase then
20         r_u informs the leader ID to all its neighbors
```

4.2. Detailed description of the algorithm

We disperse the agents in $O(n \log^2 n)$[2] rounds with the help of the dispersion algorithm for the general configuration, given in [26]. Their algorithm does not have a termination time; therefore, with the help of a known parameter of n all the agents get to know the dispersion time. After dispersing all the agents, the algorithm operates in phases of $2\Delta \log n$ rounds each. In a phase, each agent meets all its neighbors in $O(\Delta \log n)$ time. Each agent tries to figure out the highest ID agent — in phase k, the highest ID in a k-hop neighborhood is identified. The highest ID agent eventually becomes the leader.

Meeting with Neighboring Agents: The approach is based on the ID of the agent, whose size is $O(\log n)$, i.e., $c \log n$ since the highest ID is n^c (n is known parameter for the Algorithm 1) and the known value of Δ. Each agent r_u keeps the ID length $c \log n$; if ID is shortened by x bits, then x number of bits with value 0 is prepended as MSB (Most Significant Bit). Starting from the LSB (Least Significant Bit) of an agent r_u's ID, each agent r_u decides whether it will visit its neighbor or not. If LSB is 1 then the agent r_u visits its neighboring nodes one by one based on port numbering (in increasing order) which takes $2\delta_{r_u}$ rounds (two rounds to visit a neighboring node — back and forth) to explore all the neighbors, where δ_{r_u} is the degree for the agent r_u. If LSB is 0 for an agent r_u, then agent r_u waits for 2Δ rounds at its node. After 2Δ rounds, r_u decides based on the next bit in its ID whether it would explore its neighboring nodes or stay at its node. Notice that if $\delta_{r_u} < \Delta$ then r_u completes visiting its neighboring agents in less than 2Δ rounds for bit value 1 in its ID. Therefore, r_u stays at its node for the remaining rounds ($2\Delta - 2\delta_{r_u}$). Lemma 1 shows that agent r_u meets with all its neighbors at least once in $O(\Delta \log n)$ rounds.

[2] Our approach uses the algorithm as it is and considers the constant as in [26] throughout the paper.

Election of the highest ID agent as leader: After dispersing every agent such that each node possesses only one agent, each agent r_u does nothing for a phase of $O(\Delta \log n)$ rounds. We consider *Meeting with Neighboring Agents* as a phase that takes $2\Delta \log n$ rounds. In this phase, we keep our algorithm deliberately ideal to simplify the flow of the algorithm, i.e., to pass the updated information in the next phase that was received in the prior phase.

The following statements and conditions are executed by agent r_u until r_u gets to know the leader and informs all its neighbors:

In the very first phase, each agent already has its own ID, which would be passed during the next phase. Each agent r_u passes its ID to neighboring agents if that is the highest ID known to r_u. Therefore, if agent r_u has not sent (earlier) its highest ID known from the prior phase, then agent r_u sends its highest known ID to all the neighbors.

If r_u receives a higher ID from neighbor r_v then r_u considers r_v as its parent and becomes the child of r_v. This is required since, due to memory limitation, r_u can not keep track of all its neighbors to whom it passes the higher ID. Therefore, each receiving agent (child) keeps track of the sender (parent), which is not more than 1 in numbers. In case of a tie — (i) more than one agent sends the same ID which is not less than the earlier known ID, an agent r_u gives priority to the agent that was the parent in the prior phase. Notice that this condition is more than breaking a tie, since in another case — giving priority to the latest sending agent, an agent might keep changing its parent forever. (ii) If there is more than one agent that proposes the same ID in the same phase then priority (to be considered as parent) is given to the higher ID proposer. Additionally, if r_v remains the parent of r_u for the next two phases, then r_u sends an "undecided" vote to r_v for the next two phases. In this way, the initialization of the "undecided" vote takes place. The vote "undecided" is sent to the parent by its child if it received the "undecided" vote from any of its children, or r_v becomes the parent two phases before. In these two phases, r_u's children (if any) receive the ID and send their vote. Therefore, r_u sends an "undecided" vote in the first two phases irrespective of its children's vote. From the next phase onwards, r_u receives its children's vote and considers them for voting to its parent.

In case r_u received a "yes" vote from all its children (if any) and did not receive the vote "undecided", in the previous phase, after two phases of the proposal, then r_u sends the vote "yes" to its parent (if any). An agent r_u considers a neighbor its child only if it sends the vote either "yes" or "undecided". An agent r_u sends a "yes" vote to its parent only if it has received a "yes" vote from all its children (if any) and it has not received any "undecided" vote from its neighbors. Otherwise, r_u sends the vote "undecided" to its parent (if any). Notice that if agent r_u does not possess any child, in that case, r_u does not receive any "yes" or "undecided" vote. This implies that r_u does not have any children and r_u initiates the vote "yes" to its parent after two phases of passing the highest ID known to its neighbors. A child always responds with a vote of "yes" or "undecided". If r_u does not have a child, then r_u keeps sending a vote "yes" to its parent until it learns about the leader.

Whenever r_u receives its own proposed ID from all its children with a "yes" vote in the prior phase, then r_u becomes the leader and informs its ID to all neighbors. If r_u knows the leader ID from the prior phase, in that case, r_u informs the leader ID to all its neighbors.

Notice that the knowledge of n (the number of agents) and Δ (the maximum degree) is indispensable for the proposed algorithm, as they govern phase termination and synchronization. Specifically, n enables agents to bound the dispersion phase by $O(n \log^2 n)$ rounds and transition consistently to election (necessary for the execution of Line 1 of Algorithm 1), while Δ ensures correct execution of the *Meeting with Neighboring Agents* protocol, where a 2Δ-round wait guarantees meeting. In the absence of n or Δ, dispersion termination and synchronization would fail, respectively, thereby invalidating the election process.

Now, we present the above claims formally. In Lemma 1, we show the number of rounds required to meet any two neighbors. Lemma 2 shows the time and memory complexity required by Algorithm 1. Further, Lemma 3 shows the correctness of the Algorithm 1 such that it elects a unique leader.

Lemma 1. *After dispersion, every agent requires at most $O(\Delta \log n)$ rounds to meet all its neighbors.*

Proof. To prove our lemma, we prove the results for agent r_u and any neighboring agent r_v. Each agent r_u has a unique ID with length $O(\log n)$ bits. Therefore, there exists at least one index between r_u and any neighboring agent r_v such that their bits are different. Furthermore, each agent explores its neighboring nodes in the interval of 2Δ rounds based on the next bit (starting from the LSB). This implies that during the different bits for a certain index, either r_u would visit r_v or vice versa. Henceforth, each r_u meets all its neighbors in $O(\Delta \log n)$ rounds. Hence, the lemma. □

Lemma 2. *Algorithm 1 takes $O(n \log^2 n + D\Delta \log n)$ rounds and $O(\log n)$ memory to elect the leader.*

Proof. For round complexity, Line 1 takes $O(n \log^2 n)$ rounds to disperse all the n agents [26]. Furthermore, Lemma 1 takes $O(\Delta \log n)$ rounds to meet its neighbor. Therefore, to pass the information one hop away takes $O(\Delta \log n)$ rounds. The farthest agent is D hops away from any other agent which takes overall $O(D\Delta \log n)$ rounds to know the highest ID agent in the graph. Then it takes $O(D\Delta \log n)$ rounds for the "yes" votes to propagate back to the highest ID agent. After that, it takes $O(D\Delta \log n)$ rounds to inform all the agents in the graph about the highest ID agent, i.e., the leader. This shows the overall round complexity for leader election is $O(n \log^2 n + D\Delta \log n)$. In case of memory complexity, Line 1 takes $O(\log n)$ memory [26]. On the other hand, after dispersion, knowing the highest ID also takes $O(\log n)$ bits to store the information of ID, parent, highest ID known, and Leader ID. Hence, the lemma. □

Lemma 3. *Algorithm 1 correctly elects a unique agent as the leader.*

Proof. For the sake of contradiction, let us suppose Algorithm 1 elects more than one leader, namely, ℓ_1 and ℓ_2 such that $\ell_1 > \ell_2$, i.e., ℓ_1's ID is higher than ℓ_2. In such a case, ℓ_2's children voted "yes" for ℓ_2 and so did their children and their children's children, and so on. Therefore, ℓ_1, which is an agent situated at the graph, also voted for ℓ_2, which is not possible. Hence, there exists a unique leader with the highest ID. □

From the above discussion, we have the following result.

Theorem 1. *Given any configuration of n agents positioned initially arbitrarily on the nodes of a n-node graph G, there is a deterministic algorithm that elects a leader in $O(n \log^2 n + D\Delta \log n)$ rounds with $O(\log n)$ bits of memory per agent for a known value of n and Δ.*

Remark 1. For the rooted configuration, dispersion takes $O(n \log n)$ rounds in Line 1 of Algorithm 1 as shown in [26]. Therefore, round complexity becomes $O(n \log n + D\Delta \log n)$. On the other hand, for the case of an already dispersed configuration, round complexity is $O(D\Delta \log n)$. Furthermore, memory complexity remains unchanged in both configurations, i.e., $O(\log n)$ bits per agent.

5. Applications to additional graph problems

We leverage the leader election result (Theorem 1) to enhance existing solutions for MST, MIS, MDS, and Gathering problems in the agent-based model. The improvements over previous results (discussed in Table 2) are achieved either by optimizing time or memory. All our solutions are memory-optimal, with most offering significant reductions in round complexity as well.

5.1. Minimum spanning tree

A **Minimum Spanning Tree (MST)** of a graph $G = (V, E)$ with n nodes is a subgraph $T = (V, E')$ such that $w(e)$ represents the weight of an edge e and for the MST T:

$$T = \arg\min_{T' \subseteq G} \sum_{e \in T'} w(e)$$

subject to the constraints that T' is connected, acyclic, and spans all n nodes in G.

Kshemkalyani et al. [8] constructed the MST in $O(m + n \log n)$ rounds and $O(\Delta \log n)$ bits per agent from a given dispersed configuration in the presence of a leader. From Sudo et al. [26], we can disperse all the n agents in $O(n \log^2 n)$ rounds and $O(\log n)$ memory. Their protocol does not have termination [26], therefore, knowledge of n provides the dispersed configuration in $O(n \log^2 n)$ rounds. From the above discussion and Theorem 1, we have the following result.

Theorem 2 (MST). *Given any configuration of n agents positioned initially arbitrarily on the nodes of a n-node graph G, there is a deterministic algorithm that finds an MST of G in $O(m + n \log^2 n + D\Delta \log n)$ rounds with $O(\Delta \log n)$ bits per agent.*

Remark 2. In an MST, an agent might have to track all its neighboring edges as the MST edge that requires $O(\Delta \log n)$ bits of memory. Therefore, Theorem 2 is memory optimal.

5.2. Maximal independent set (MIS)

Suppose n agents are initially located arbitrarily on the nodes of an n-node anonymous graph $G = (V, E)$. The goal in the maximal independent set (MIS) problem is to relocate the agents autonomously to find a subset $S \subset V$ of nodes such that S forms an MIS of G.

Description of the MIS Algorithm: We construct the MIS from a dispersed configuration and known leader in Algorithm 2. In this, the leader becomes the MIS agent and converts all its neighbors non-MIS in $O(\log \Delta)$ rounds with the help of the Helping-Each-Other (HEO) technique introduced by [26]. In this, each agent r_u having the token visits its neighbor by port 1 and comes back with the agent situated across port 1. Now, these two agents visit ports 2 and 3 individually and come back with one agent each. Similarly, after every 2 rounds the number of agents becomes double, and for the degree δ_{r_u}, it takes $2 \log \delta_{r_u}$ rounds to gather all the neighboring agents at r_u's node. During the MIS algorithm, we run a variant of Depth-First-Search (DFS) with the help of a token. We run the MIS algorithm until some neighbor of agent r_u's remains unexplored. We call an agent r_u as unexplored if it never held the token. Specifically, if the leader's neighbor remains unexplored, we run the following protocol for any agent r_u. If agent r_u has the token and it is neither an MIS nor a non-MIS agent, in that case, r_u collects all the agents using HEO techniques and figures out the status of all other agents. If none of the neighboring agents is an MIS agent only then r_u becomes the MIS agent. Otherwise, r_u becomes a non-MIS agent. If agent r_u has the token and it is either an MIS or a non-MIS agent and there exists any unexplored port then r_u passes the token to the next unexplored port (in increasing order) and becomes the parent of the node across the port. Otherwise, r_u sends the token to the parent node. In this protocol, each agent stores its own ID and parent's port number. It requires $O(\log n)$ memory. Furthermore, the token reaches all the n nodes and takes at most $O(\log \Delta)$ rounds to run HEO. Therefore, the overall time complexity to run the MIS algorithm is $O(n \log \Delta)$ rounds and $O(\log n)$ memory.

Below, we show the correctness of Algorithm 2.

Lemma 4. *Algorithm 2 generates an MIS.*

Proof.

We prove the lemma by considering two cases: (i) The token is passed to each agent r_u. (ii) the set S computed by the algorithm is a valid Maximal Independent Set.

We prove both cases by contradiction. For case (i), let us assume the token is not passed to agent r_u. Then, it is also not passed to r_u's parent, grandparent, and so forth, due to Line 13 of Algorithm 2. This implies that the token was not passed by the leader, contradicting Line 2.

Algorithm 2: Maximal independent set.

Input : A set $\mathcal{R}$ of n agents with unique IDs dispersed on a graph of n nodes with known leader's ID.
Output: MIS configuration.

```
1  Helping-Each-Other (HEO) technique to gather all the neighboring agents at a node as discussed in [26]. A HEO phase
     consists of O(log Δ) rounds, in which agent r_u meets all its neighbors.
2  The leader becomes the MIS agent and converts all its neighbors in non-MIS in log Δ rounds with the help of HEO, passes the
     token via port 1, and becomes the parent of the across-the-port node.
3  while agent r_u's neighbors are unexplored do
4      if agent r_u has the token then
5          if r_u is neither an MIS nor a non-MIS agent then
6              r_u collects all the agents using HEO techniques and figures out the status of all other agents
7              if none of the neighboring agents is an MIS agent then
8                  r_u becomes the MIS agent
9              else
10                 r_u becomes a non-MIS agent
11         if r_u is either MIS or Non-MIS then
12             if any unexplored port exists then
13                 r_u passes the token to the next unexplored port (in increasing order) and becomes the parent of the node
                     across the port
14             else
15                 r_u sends the token to the parent node
```

For case (ii), we prove that the set S computed by the algorithm is a valid Maximal Independent Set by verifying its two defining properties: a) Independence and b) Maximality.

a) Independence: Suppose, for contradiction, that two adjacent agents r_u and r_v both belong to S. This is impossible, due to the statement at Line 11, ensuring that among two adjacent candidates, only one may join S, while the other is excluded by passing the token to the next unexplored port. Hence, no edge exists between two members of S.

b) Maximality: Suppose, for contradiction, that there exists an agent $r_v \notin S$ with no neighbor in S. This is impossible due to the execution of the statement at Line 8. Thus, every non-member of S is adjacent to at least one member.

Since both independence and maximality hold, the set S is an MIS. □

From the above discussion and Theorem 1 of leader election, we have the following result.

Theorem 3 (MIS). *Given any configuration of n agents positioned initially arbitrarily on the nodes of a n-node graph G, there is a deterministic algorithm that finds an MIS of G in $O(n \log^2 n + D\Delta \log n)$ rounds with $O(\log n)$ bits per agent.*

Since every MIS is an MDS, therefore, we have the following results.

Theorem 4 (MDS). *Given any configuration of n agents positioned initially arbitrarily on the nodes of a n-node graph G, there is a deterministic algorithm that finds an MDS of G in $O(n \log^2 n + D\Delta \log n)$ rounds with $O(\log n)$ bits per agent.*

5.3. Gathering

Suppose n agents are initially located arbitrarily on the nodes of an n-node anonymous graph $G = (V, E)$. The goal of the gathering problem is to relocate the agents autonomously to position all of them to a node in G not fixed a priori (n agents at a node of G).

Algorithm 2 can be adapted to gather all the agents at the leader's node. In Algorithm 2 (Line 15), when an agent passes the token to its parent, the parent agent keeps the account of the number of children. Therefore, after $O(n \log \Delta)$ rounds all the agents are traversed, and MIS is formed. The agent without children gathers at its parent node. When the parent finds all its children in its place, the parent moves to its parent with all of its children. This process occurs recursively and eventually all the agents are gathered at the leader's node. In the worst case, an agent can be at most n hop away from its leader. Therefore, it takes at most $O(n)$ rounds to gather all the agents. Notice that an agent only counts its number of children other than the memory used for MIS, therefore, the memory complexity remains unchanged, i.e., $O(\log n)$.

From the above discussion and Leader Election Algorithm 1, we have the following result.

Theorem 5 (Gathering). *Given any configuration of n agents positioned initially arbitrarily on the nodes of a n-node graph G, there is a deterministic algorithm that collects all n agents to a node in G not fixed a priori in $O(n \log^2 n + D\Delta \log n)$ rounds with $O(\log n)$ bits per agent.*

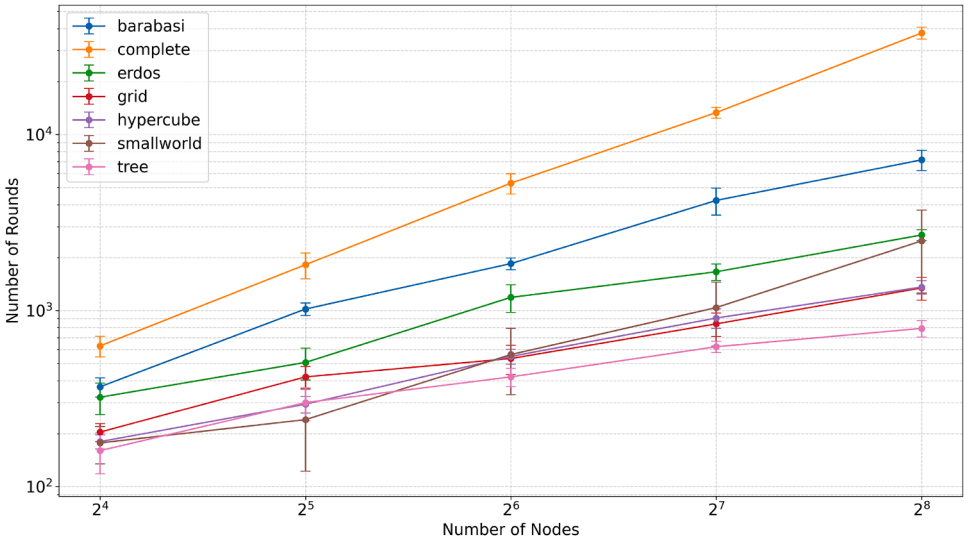


Fig. 1. Number of rounds required for leader election vs. the number of nodes, grouped by graph topology. Both axes use logarithmic scales.

6. Simulation

In this section, we show results of simulations of leader election that we performed on different graphs starting from a dispersed configuration.

6.1. Implementation details

The simulation framework was implemented in Python3 using NetworkX library. We consider the following parameters for our simulation configuration:

(I) Number of nodes are considered to be 2^i for $i = 4, 5, 6, 7, 8$.[3]

(II) Graph topologies: We consider seven types of well-known graphs for the evaluation.
- Erdős-Rényi random graph: a pair of nodes connected with probability $p = \log n/n$, where n is the number of nodes.
- Barabási-Albert preferential attachment graph: an added new node is connected to at least k existing nodes, where k is a parameter chosen to be 3 in our simulation.
- Small world network: a graph with high clustering coefficient and smallest average shortest path length.
- Grid graph: square grid graph with $n = \sqrt{n} \times \sqrt{n}$ vertices. The value of n is approximated to nearest perfect square.
- Hypercube graph: graph formed from the vertices and edges of an n-dimensional hypercube, where degree of each vertex is $\log n$.
- Complete graph.
- Tree.

(III) We randomize the port numbers (8 times) for each graph that we generate and execute the leader election algorithm.

Each simulation run was executed until termination and we tracked the number of rounds and number of edge traversals of each agent.

6.2. Results and analysis

6.2.1. Scaling with network size

Fig. 1 shows the relationship between network size and the number of rounds required for leader election across different graph topologies.

The results reveal that the algorithm's time complexity scales approximately as $O(D\Delta \log n)$ for most graph families, which aligns with the theoretical bounds for leader election in anonymous networks. However, the constant factors vary significantly across topologies.

It is worth noting that the error bars in other graph types show that the leader election depends on the order of edge traversals in each phase as an agent visits its neighbors.

[3] We assume the number of nodes is a power of 2 to be consistent with the Hypercube graph in the plots.

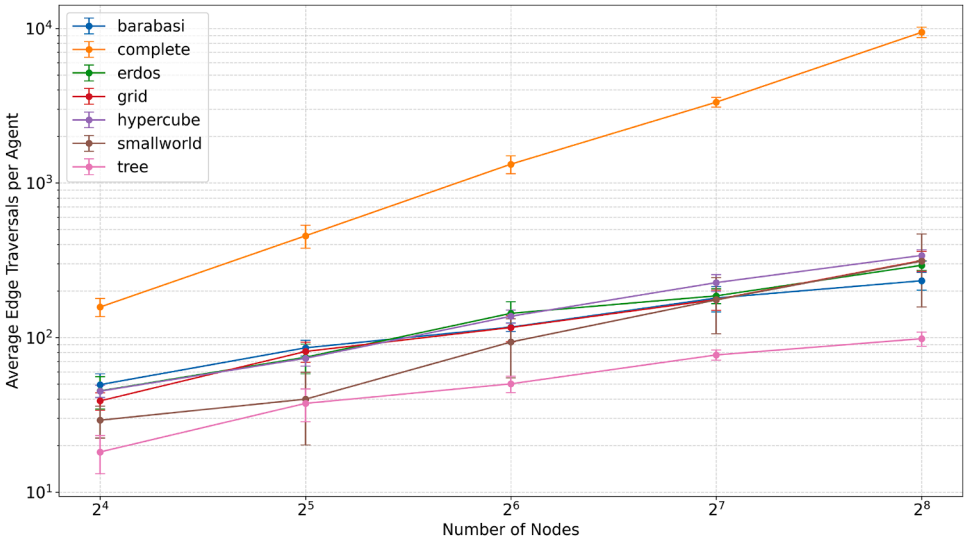


Fig. 2. Average edge traversals per agent as a function of network size. This metric reflects the communication cost of the algorithm.

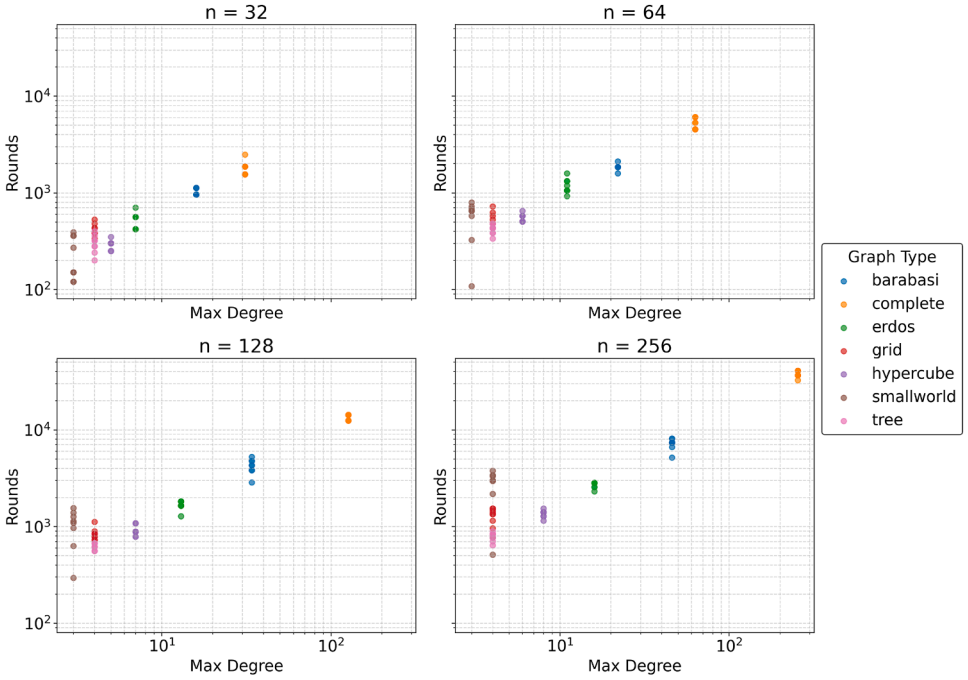


Fig. 3. Relationship between maximum degree and rounds required for leader election.

6.2.2. *Communication overhead*

Fig. 2 presents the average number of edge traversals per agent, which serves as a measure of communication overhead.

Interestingly, the communication cost counted as the number of edge traversals differ from the number of rounds spent to elect a leader since in graphs with fewer high degree nodes, the agents located at lower degree nodes travel in the first few rounds, and remain idle in the rounds until Δ, which is a known parameter. Overall, the edge traversals per agent follow a similar trend as the number of rounds.

6.2.3. *Impact of network diameter vs. maximum degree*

While maximum degree Δ is a known parameter, the network diameter is unknown. As suspected, in Fig. 3 the number of rounds needed grows proportionally with the maximum degree. However, surprisingly, as we see in Fig. 4, the number of rounds is inversely proportional to the diameter. We deduce that, this is due to the fact that for small graphs the communication overhead is small and

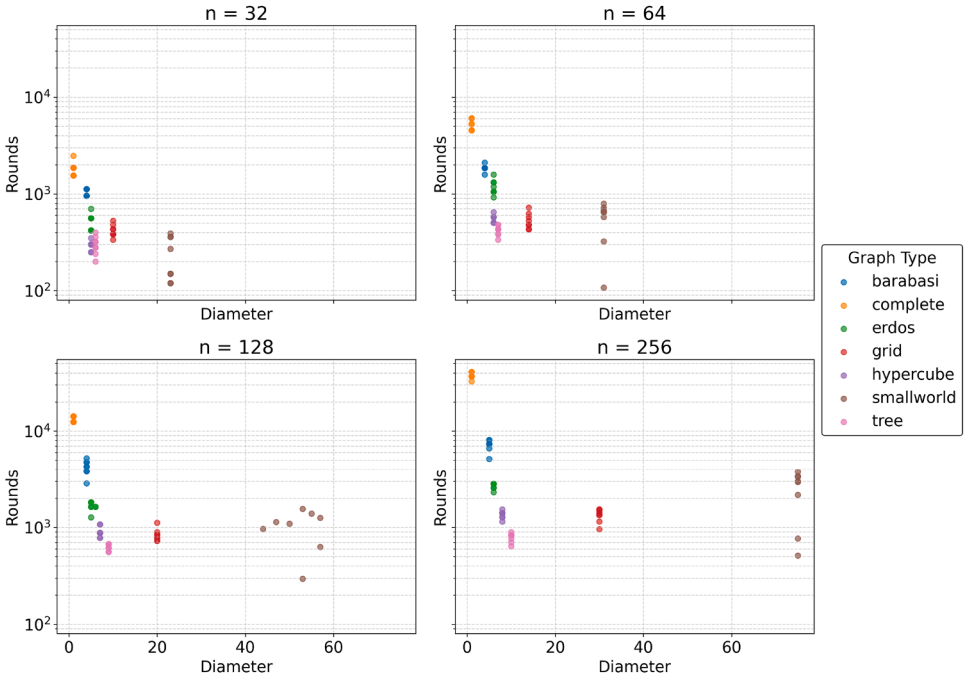


Fig. 4. Relationship between network diameter and rounds required for leader election.

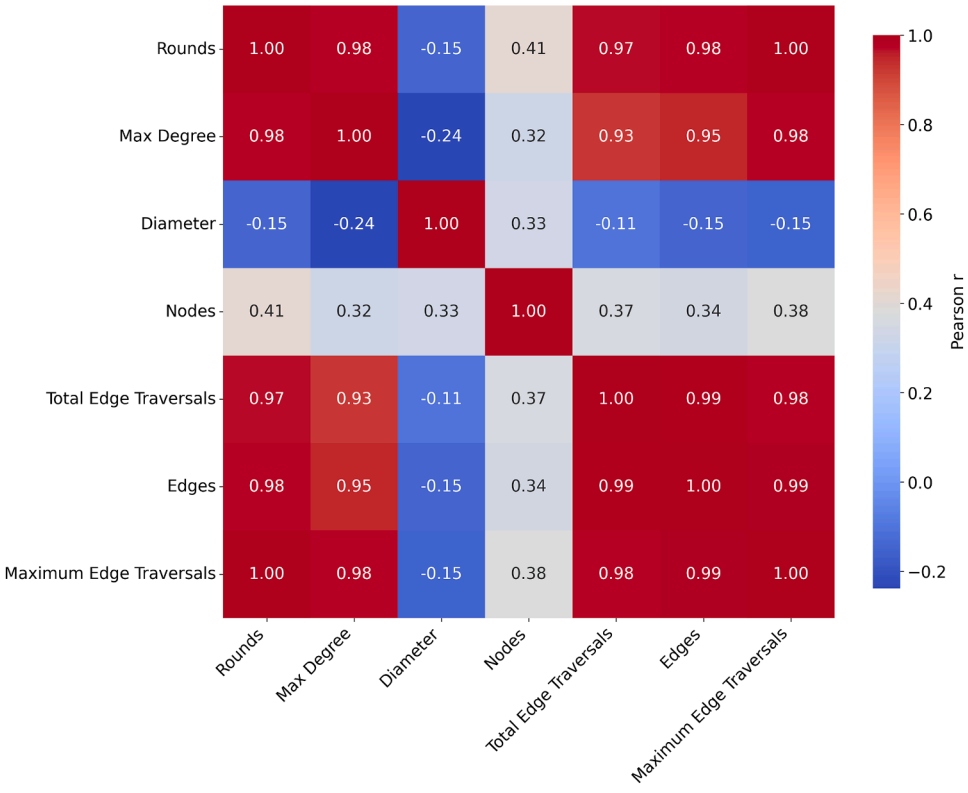


Fig. 5. Correlation among different parameters of interest.

hence the diameter plays little role compared to the maximum degree. Both Figs. 3 and 4 have multiple plots standardized with the same number of node to show the impact of network diameter and maximum degree independent of the network size.

6.3. Discussion

Our simulation results highlight several key insights based on the correlation observed in Fig. 5 among the parameters of interest, such as the size of the network, number of edges, diameter and maximum degree, along with total number of rounds and edge traversals by agents.

1. **Topology dependence**: The algorithm's performance is strongly influenced by the underlying network topology.
2. **Scalability**: The algorithm demonstrates reasonable scalability, with the number of rounds growing sublinearly with network size for most topologies.
3. **Time-communication trade-off**: There exists a clear trade-off between time efficiency (rounds) and communication overhead (edge traversals). Dense graphs does not show a significant reduction due to high degree nodes.
4. **Maximum degree as a predictor**: Maximum degree serves as a strong predictor of performance, with a nearly linear relationship between diameter and required rounds.
5. **Consistency**: Performance is highly consistent for regular structures (complete graphs, hypercubes, grids) but shows higher variance for random topologies, as we suspect from the sensitivity to structural properties, such as degree distribution, in these cases.

These observations provide valuable guidance for deploying distributed leader election in practical settings. For applications where speed is critical, it is best to avoid high degree nodes. Conversely, when communication overhead is a concern, sparser structures like trees or grids may be more appropriate. The inverse correlation with diameter also suggests that topology design should prioritize against minimizing network diameter when optimizing for leader election performance.

7. Conclusions

We solved leader election in the agent-based model in $O(n \log^2 n + D\Delta \log n)$ time with optimal memory of $O(\log n)$ bits per agent for general initial configurations. Furthermore, the memory complexities of the existing results on MST, MIS, MDS, and Gathering on the agent-based model were improved using the leader election result, and we achieved the optimal memory complexity bits per agent for all the aforementioned problems. Our result also solved near-linear time complexity for MIS, MDS, and Gathering problems after leader election in a dispersed setting in $O(n \log \Delta)$ rounds (see Table 1). It would be interesting to improve the time complexity of our leader election, which would directly help to improve the complexity of the other graph-related problems. It would also be interesting to see whether these results can be achieved without any prior knowledge of the parameter.

CRediT authorship contribution statement

Ajay D. Kshemkalyani: Writing – review & editing, Validation, Supervision; **Manish Kumar:** Writing – original draft, Methodology, Investigation, Conceptualization; **Anisur Rahaman Molla:** Validation; **Debasish Pattanayak:** Visualization, Validation, Software; **Gokarna Sharma:** Validation.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] A.D. Kshemkalyani, M. Kumar, A.R. Molla, G. Sharma, Faster leader election and its applications for mobile agents with parameter advice, in: International Conference on Distributed Computing and Intelligent Technology, Springer, 2024, pp. 108–123.
[2] Y. Cong, g. Changjun, T. Zhang, Y. Gao, Underwater robot sensing technology: a survey, Fundam. Res. 1 (2021). https://doi.org/10.1016/j.fmre.2021.03.002
[3] J. Lee, S. Shin, M. Park, C. Kim, Agent-based simulation and its application to analyze combat effectiveness in network-centric warfare considering communication failure environments, Math. Probl. Eng. 2018 (2018) 1–9. https://doi.org/10.1155/2018/2730671
[4] C. Zhuge, C. Shao, B. Wei, An agent-based spatial urban social network generator: a case study of Beijing, China, J. Comput. Sci. 29 (2018). https://doi.org/10.1016/j.jocs.2018.09.005
[5] A. El-Sayed, P. Scarborough, L. Seemann, S. Galea, Social network analysis and agent-based modeling in social epidemiology, Epidemiologic Perspect. Innovations : EP+I 9 (2012) 1. https://doi.org/10.1186/1742-5573-9-1
[6] K. Martinkus, P.A. Papp, B. Schesch, R. Wattenhofer, Agent-based graph neural networks, in: The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023, OpenReview.net, 2023. https://openreview.net/pdf?id=8WTAh0tj2jC.
[7] P.K. Chand, A. Das, A.R. Molla, Agent-based triangle counting and its applications in anonymous graphs, CoRR abs/2402.03653 (2024). https://doi.org/10.48550/ARXIV.2402.03653
[8] A.D. Kshemkalyani, M. Kumar, A.R. Molla, G. Sharma, Brief announcement: agent-based leader election, MST, and beyond, in: DISC 2024, 319 of *LIPIcs*, 2024, pp. 50:1–50:7. https://doi.org/10.4230/LIPICS.DISC.2024.50
[9] J.A. Garay, S. Kutten, D. Peleg, A sub-linear time distributed algorithm for minimum-weight spanning trees (extended abstract), in: FOCS, IEEE Computer Society, 1993, pp. 659–668. https://doi.org/10.1109/SFCS.1993.366821
[10] D. Pattanayak, S. Bhagat, S.G. Chaudhuri, A.R. Molla, Maximal independet set via mobile agents, in: ICDCN, ACM, 2024, pp. 74–83.
[11] P.K. Chand, A.R. Molla, S. Sivasubramaniam, Run for cover: dominating set via mobile agents, in: ALGOWIN, Springer, 2023, pp. 133–150.

[12] A.R. Molla, K. Mondal, W.K. Moses, Jr., Fast deterministic gathering with detection on arbitrary graphs: the power of many robots, in: IEEE International Parallel and Distributed Processing Symposium, IPDPS, IEEE, 2023, pp. 47–57.
[13] A. Ta-Shma, U. Zwick, Deterministic rendezvous, treasure hunts, and strongly universal exploration sequences, ACM Trans. Algorithms 10 (3) (2014) 12:1–12:15. https://doi.org/10.1145/2601068
[14] G.L. Lann, Distributed systems - towards a formal approach, in: B. Gilchrist (Ed.), Information Processing, Proceedings of the 7th IFIP Congress 1977, Toronto, Canada, August 8-12, 1977, North-Holland, 1977, pp. 155–160.
[15] R.G. Gallager, P.A. Humblet, P.M. Spira, A distributed algorithm for minimum-weight spanning trees, ACM Trans. Program. Lang. Syst. 5 (1) (1983) 66-77. https://doi.org/10.1145/357195.357200
[16] B. Awerbuch, Optimal distributed algorithms for minimum weight spanning tree, counting, leader election and related problems (detailed summary), in: A.V. Aho (Ed.), STOC, ACM, 1987, pp. 230–240. https://doi.org/10.1145/28395.28421
[17] D. Peleg, Time-optimal leader election in general networks, J. Parallel Distrib. Comput. 8 (1) (1990) 96-99. https://doi.org/10.1016/0743-7315(90)90074-Y
[18] S. Kutten, G. Pandurangan, D. Peleg, P. Robinson, A. Trehan, On the complexity of universal leader election, J. ACM 62 (1) (2015) 7:1–7:27.
[19] S. Kutten, D. Peleg, Fast distributed construction of small *k*-dominating sets and applications, J. Algorithms 28 (1) (1998) 40–66. https://doi.org/10.1006/jagm.1998.0929
[20] D. Peleg, V. Rubinovich, A near-tight lower bound on the time complexity of distributed minimum-weight spanning tree construction, SIAM J. Comput. 30 (5) (2000) 1427–1442. https://doi.org/10.1137/S0097539700369740
[21] B. Awerbuch, A.V. Goldberg, M. Luby, S.A. Plotkin, Network decomposition and locality in distributed computation, in: FOCS, 30, Citeseer, 1989, pp. 364–369.
[22] A. Panconesi, A. Srinivasan, On the complexity of distributed network decomposition, J. Algorithms 20 (2) (1996) 356–374.
[23] V. Rozhoň, M. Ghaffari, Polylogarithmic-time deterministic network decomposition and distributed derandomization, in: Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, 2020, pp. 350–363.
[24] J. Deurer, F. Kuhn, Y. Maus, Deterministic distributed dominating set approximation in the CONGEST model, in: Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, 2019, pp. 94–103.
[25] Y. Sudo, D. Baba, J. Nakamura, F. Ooshita, H. Kakugawa, T. Masuzawa, A single agent exploration in unknown undirected graphs with whiteboards, Proceedings of the 3rd International ACM Workshop on Reliability, Availability, and Security, WRAS 2010 E98.A (2010). https://doi.org/10.1145/1953563.1953570
[26] Y. Sudo, M. Shibata, J. Nakamura, Y. Kim, T. Masuzawa, Near-linear time dispersion of mobile agents, in: D. Alistarh (Ed.), 38th International Symposium on Distributed Computing, DISC 2024, October 28 to November 1, 2024, Madrid, Spain, 319 of *LIPIcs*, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024, pp. 38:1–38:22. https://doi.org/10.4230/LIPICS.DISC.2024.38