Brief Announcement: Agent-Based Leader Election, MST, and Beyond

Ajay D. Kshemkalyani
Department of Computer Science, University of Illinois at Chicago, IL, USA

Manish Kumar
Department of Computer Science & Engineering Indian Institute of Technology, Madras, India

Anisur Rahaman Molla
R. C. Bose Centre for Cryptology and Security, Indian Statistical Institute, Kolkata, India

Gokarna Sharma
Department of Computer Science, Kent State University, OH, USA

Abstract

Leader election is one of the fundamental and well-studied problems in distributed computing. In this paper, we initiate the study of leader election using mobile agents. Suppose n agents are positioned initially arbitrarily on the nodes of an arbitrary, anonymous, n-node, m-edge graph G. The agents relocate themselves autonomously on the nodes of G and elect an agent as a leader such that the leader agent knows it is a leader and the other agents know they are not leaders. The objective is to minimize time and memory requirements. Following the literature, we consider the synchronous setting in which each agent performs its operations synchronously with others and hence the time complexity can be measured in rounds. The quest in this paper is to provide solutions without agents knowing any graph parameter, such as n, a priori. We first establish that, without agents knowing any graph parameter a priori, there exists a deterministic algorithm to elect an agent as a leader in $O(m)$ rounds with $O(n \log n)$ bits at each agent. Using this leader election result, we develop a deterministic algorithm for agents to construct a minimum spanning tree of G in $O(m + n \log n)$ rounds using $O(n \log n)$ bits memory at each agent, without agents knowing any graph parameter a priori. Finally, using the same leader election result, we provide improved time/memory results for other fundamental distributed graph problems, namely, gathering, maximal independent set, and minimal dominating sets, removing the assumptions on agents knowing graph parameters a priori.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases Distributed algorithms, mobile agents, local communication, leader election, MST, MIS, gathering, minimal dominating sets, time and memory complexity, graph parameters

Digital Object Identifier 10.4230/LIPIcs.DISC.2024.50

Related Version *Full Version*: https://doi.org/10.48550/arXiv.2403.13716 [6]

1 Introduction

The well-studied *message-passing* distributed computing model 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, and each node typically knows only its neighboring nodes. The nodes interact with one another by sending messages (hence the name *message-passing*) to achieve a common goal. The computation proceeds according to synchronized *rounds*. In each round, each node v can perform unlimited local computation and may send a distinct message to each of its neighbors. Additionally, each node v is assumed to have no limit on storage. In the LOCAL

Table 1 Comparison of the message-passing and agent-based models.

Model	Devices	Local computation	Device storage	Neighbor communication
Message-passing	Static	Unlimited	No restriction	Messages
Agent-based	Mobile	Unlimited	Limited	Relocation

variant of this model, there is no limit on bandwidth, i.e., a node can send any size message to each of its neighbors. In the CONGEST variant, bandwidth is taken into account, i.e., a node may send only a, possibly distinct, $O(\log n)$-bit message to each of its neighbors.

In this paper, we consider the *agent-based* distributed computing model where the computational devices are modeled as *relocatable or mobile computational devices* (which we call agents). Departing from the notion of vertex/node as a *static* device in the message-passing model, the vertices/nodes serve as *containers* for the devices in the agent-based model. The agent-based model has two major differences with the message-passing model (Table 1 compares the properties of the two models).

Difference I. The graph nodes do not have identifiers, computation ability, and storage, but the devices are assumed to have distinct $O(\log n)$-bit identifiers, computation ability, and (limited) storage.

Difference II. The devices 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 and can exchange information if a device is positioned at the neighbor.

Difference II is the major problem for the agent-based model. To complicate further, while a device relocates to a neighbor, the device at that neighbor might relocate to another neighbor. Therefore, the devices need to coordinate to achieve the common goal.

In this paper, we initiate the study of a graph-level task of leader election in a distributed network under 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. Leader election can also be seen as a form of symmetry breaking, where exactly one special process or node (say a leader) is allowed to make some critical decisions. The problem of leader election in the agent-based model requires a set of agents operating in the distributed network to elect a unique leader among themselves, i.e., exactly one agent must output the decision that it is the leader.

1.1 Motivation

The agent-based model has recently found its use in multiple areas of computing. One prominent example is Martinkus *et al.* [8] which proposes AgentNet – a graph neural network (GNN) architecture, in which a collection of (neural) relocatable devices (called neural agents) *walk* the graph and collectively classify the graph-level tasks, such as triangles, cliques, and cycles. The model allows the neural agents to retrieve information from the node they are occupying, their neighboring nodes (when they visit those nodes), and the co-located devices. They showed that this agent-based model was able to detect cliques and cycles, which was shown to be impossible in the widely-studied GNN architectures based on the message-passing model (i.e., devices are static and communication is via passing messages).

Additionally, a recent study [1] has shown that the fundamental graph-level task of *triangle detection* can be solved in the agent-based model by a deterministic algorithm in $O(\Delta \log n)$ rounds with $O(\Delta \log n)$ bits at each device. In contrast, it is known that in the CONGEST message-passing model it takes $O(n^{1/3} \operatorname{polylog}(n))$ rounds to solve triangle detection by a randomized algorithm [4], which is almost tight since there is the $\Omega(n^{1/3}/\log n)$ lower bound [5, 10], and hence the agent-based model provides a clear advantage when $\Delta < n^{1/3} \operatorname{polylog}(n)$ despite restriction on communication through device relocation.

1.2 Contributions

Table 2 summarizes the problems studied and bounds obtained as well as comparison with the previous results. Specifically, we develop a deterministic algorithm for leader election with provable guarantees on two performance metrics that are fundamental to the agent-based model: *time complexity* of a solution and *storage requirement* per agent. We focus on the *deterministic* algorithms since they may be more suitable for relocatable devices. Our quest is to provide an algorithm that does not ask the agents to rely 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). This is in contrast to the message-passing model which typically assumes that n (exact n or an upper bound N on n) is known to the nodes/devices, and may be additionally Δ and D [3]. This also contrasts research in the agent-based model with known parameters (e.g., [2, 9, 11]). On the one hand, not knowing these parameters has its own merits as the solutions designed are more resilient to network changes and device faults. On the other hand, algorithm design becomes challenging since devices may not know how long to run a procedure to guarantee a solution.

Moreover, the agent-based model treats storage requirement as the first order performance metric in addition to time complexity. This is in contrast to the message-passing model where storage complexity was often neglected with the implicit assumption that the devices have no restriction on the amount of storage needed to successfully run the algorithm; in the message-passing model, the focus was given on *message complexity* (the total number of messages sent by all nodes for a solution [10]) as the first order performance metric in addition to time complexity. The goal is to use storage as small as possible (comparable to the device identifier size of $O(\log n)$ bits per device). The limited storage makes it impossible for the relocatable devices to first traverse the graph to learn the topology and then run graph computation as a second step.

Using the proposed deterministic leader election algorithm with provable guarantees on time and storage, we construct a minimum spanning tree (MST) of G, another fundamental and well-studied problem in distributed computing, for the first time in the agent-based model, without agents knowing any graph parameter a priori. We provide both time and memory complexities. Finally, as an application, using the same leader election result, we provide improved time/memory complexity algorithms for many other fundamental distributed graph problems, namely gathering, maximal independent set (MIS), and minimal dominating sets (MDS), removing the parameter assumptions in the literature.

1.3 Challenges

The message-passing model allows the nodes (processors) to send/receive messages to/from their neighbors, i.e., in a single round, a node can send a message to all its neighbors and receive messages from all its neighbors. In contrast, in the agent-based model, the messages

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. Previous results have parameter assumptions as outlined above. Our results do not have such assumptions. "$-$" means no previous result for the corresponding problem. "$\mathcal{D}$" denotes the dispersed initial configuration.

problem	previous result			our result (no parameter known)	
	time	memory/agent	known	time	memory/agent
leader	$-$	$-$	$-$	$O(m)$	$O(n \log n)$ $O(\log^2 n)$ $(\mathcal{D})$
MST	$-$	$-$	$-$	$O(m+$ $n \log n)$	$O(n \log n)$ $O(\log n \min\{\Delta, \log n\})$ $(\mathcal{D})$
gathering	$O(n^3)$	$O(M + m \log n)$	n [9]	$O(m)$	$O(n \log n)$ $O(\log^2 n)$ $(\mathcal{D})$
MIS	$O(n\Delta \log n)$	$O(\log n)$	n, Δ [11]	$O(n\Delta)$	$O(n \log n)$ $O(\log^2 n)$ $(\mathcal{D})$
MDS	$O(\gamma\Delta \log n+$ $n\gamma + m)$	$O(\log n)$	$n, \Delta,$ m, γ [2]	$O(m)$	$O(n \log n)$ $O(\log^2 n)$ $(\mathcal{D})$

from an agent, if any, that are to be sent to the other agents in the neighboring nodes have to be delivered by the agent visiting those neighbors. Furthermore, it might be the case that when the agent reaches that node, the agent at that node may have already moved to another node. Therefore, any algorithm in the agent-based model needs to guarantee message delivery by synchronizing sender and receiver agents to be co-located at a node.

Additionally, the graph-level tasks (such as MST) demand each node of G to have an agent positioned on it to be able to provide a solution, i.e., if agents are not in a dispersed configuration, then MST constructed may not the MST of whole G but its sub-graph. Additionally, the MST computed may be the MST forest of graph components formed by agent positions. Notice that the initial configuration of n agents in a n-node graph G may not be dispersed.

Suppose initially the agent configuration is dispersed. Surprisingly, even in this initial configuration, the agent positioned at a node does not know this configuration. Therefore, irrespective of whether the nodes have zero, single, or multiple agents initially, it seems highly advantageous to reach a dispersed configuration.

Suppose the agents are in a dispersed configuration and the goal is to construct MST. The question is which agent starts MST construction and when. The leader election problem handles this symmetry breaking issue, since if a leader can be elected, then the authority can be given to the leader agent to initiate MST construction. The remaining agents do not participate in MST construction until the leader grants them authority to do so. Although having a leader seems to make MST construction easier and possibly other problems too, electing a leader itself turned out to be a difficult task.

2 Algorithm Overview

Initially, a graph node may have zero, one, or multiple agents. All these agents are "candidates" to become leader. A candidate needs to first become a "local leader" before becoming a "global leader". Each candidate that cannot become a "local leader" (also each "local leader" that cannot become a "global leader") will become a "non_candidate".

If an agent is initially singleton at a node, then it runs *Singleton_Election* procedure to become a local leader. If an agent is not initially non-singleton then it runs *Multiplicity_Election* procedure to become a local leader. After an agent becomes a local leader, it runs *Global_Election* procedure to become a global leader.

An agent r_u running *Singleton_Election* procedure at a node u will be successful in becoming a local leader if and only if all u's neighbors have initially a singleton agent positioned on them and u has the smallest degree compared to the neighboring nodes. Each initially singleton agent r_u at node u running *Singleton_Election* procedure visits the neighbors of u one by one which finishes in $2\delta_u$ rounds, where δ_u is the degree of u. If not all neighbors have initially singleton agents positioned, the agent gets to know it cannot become a local leader. It then stops executing the *Singleton_Election* procedure and becomes "non_candidate".

An agent r_u initially at node u running *Multiplicity_Election* procedure will be successful in becoming a local leader if and only if it has the smallest ID among the ones positioned with it initially at u. To achieve so, *Multiplicity_Election* procdure executes a Depth First Search (DFS) traversal ans settles the robots on each empty node visited the the traversal until there is only a singleton agent left. As soon as this condition satisfies (the smallest ID agent becomes a singleton at node w), it declares itself as a local leader[1] Except one robot, all the other robots in the *Multiplicity_Election* procedure become "non_candidate".

To make sure that *Multiplicity_Election* procedure meets the *Singleton_Election* procedure (if it is running), *Multiplicity_Election* procedure waits at a node for a round. *Singleton_Election* stops and the agent becomes "non_candidate" when it knows about *Multiplicity_Election*.

After becoming a local leader (irrespective of whether through *Singleton_Election* or *Multiplicity_Election*, the local leader agent runs *Global_Election* procedure to become a global leader. *Global_Election* procedure is again a DFS traversal as in *Multiplicity_Election* but with the goal to visit all the edges of G. To make it easier for other local leaders or *Multiplicity_Election* procedure from another agent to not mistakenly put an agent on the home node (the node where an agent becomes a local leader running *Singleton_Election* or *Multiplicity_Election*) of a local leader the neighbor nodes are asked to store the information about a home node. The agents running *Multiplicity_Election* and *Global_Election* check the neighbors to confirm whether the visited empty node is in fact a home node of a local leader (or a node of an agent that is waiting to possibly become a local leader). This confirmation is obtained running *Confirm_Empty* procedure. If an empty node is a home node (or possible home node of an agent waiting to possibly become a local leader), *Multiplicity_Election* and *Global_Election* continue leaving that node empty as is. Otherwise, *Multiplicity_Election* puts an agent and continues, and *Global_Election* stops as it knows that *Multiplicity_Election* procedure from at least one agent has not finished yet.

There may be the case that while running *Global_Election*, $DFS(roundNo_i, r_i)$ of local leader r_i may *meet* $DFS(roundNo_j, r_j)$ of local leader r_j. In this case, $DFS(roundNo_i, r_i)$ continues if $roundNo_i > roundNo_j$ (if same round number, use agent IDs), otherwise $DFS(roundNo_j, r_j)$. If $DFS(roundNo_j, r_j)$ stops, then r_j becomes "non_candidate" and returns to its home node following parent pointers in $DFS(roundNo_j, r_j)$.

After a leader is elected, as an application, we use it to solve other fundamental problems. One is MST construction which was not considered in the agent-based model before. The

[1] There are cases where the parent node of w in the DFS tree built is empty and it demands the eligible robot to wait at w to decide later whether to become a local leader or a non-candidate.

rest are gathering, MIS, and MDS problems which were considered in the agent-based model before but solved assuming that the agents know one or more graph parameters a priori. We lift those assumptions and additionally provide improved time/memory bounds. This is possible by combining the leader election result with the techniques developed on the previous work under known graph parameters. The results are in Table 2.

For the MST construction, the leader plays a crucial role in synchronizing the agents. The leader ranks the agents and starts constructing an MST. It keeps its rank the highest. The leader, once its job is done, informs that second ranked agent to continue constructing MST. The second informs the third, and so on, until $(n-1)$-ranked agents pass the token to the n-th ranked. The n-th ranked agent passes the token back to the leader and one phase of MST construction finishes. It is guaranteed that at the end of this phase, there will be at least $n/2$ edges of the MST identified. Therefore, repeating this process for $O(\log n)$ phases, we have all $n-1$ edges of MST correctly identified, giving an MST of G.

2.1 Discussion on Memory Requirement

In our leader election algorithm, if n and Δ are known, a dispersed configuration can be achieved starting from any initial configuration in either $O(n \log^2 n)$ rounds using the algorithm of Sudo *et al.* [12] or in $O(m)$ rounds using the algorithm of Kshemkalyani and Sharma [7], with $O(\log n)$ bits per agent. After that, *Singleton_Election* can finish in $O(\Delta \log^2 n)$ rounds with $O(\log n)$ bits per agent. Then finally *Global_Election* procedure finishes electing a unique global leader in $O(m)$ rounds with $O(\log n)$ bits per agent. Therefore, leader election can be done with only $O(\log n)$ bits per agent (n factor improvement compared to our algorithm non-dispersed configurations). For the MST construction, a node may need to remember multiple of its neighboring edges as a part of MST and hence the total memory needed would be $O(\Delta \log n)$ bits per agent. However, notice that this memory improvement assume known n and Δ. The proposed leader election algorithm does not rely on any known graph parameters. Therefore, the proposed leader election algorithms is interesting despite $O(n \log n)$ bits memory requirement as it helped to achieve for the first time results for MST in the agent-based model and also to provide improved time/memory results for gathering, MIS, and MDS in the agent-based model, lifting the assumptions on known graph parameters.

References

1 Prabhat Kumar Chand, Apurba Das, and Anisur Rahaman Molla. Agent-based triangle counting and its applications in anonymous graphs. In *AAMAS*, 2024. doi:10.48550/arXiv.2402.03653.

2 Prabhat Kumar Chand, Anisur Rahaman Molla, and Sumathi Sivasubramaniam. Run for cover: Dominating set via mobile agents. In *ALGOWIN*, pages 133–150. Springer, 2023. doi:10.1007/978-3-031-48882-5_10.

3 Yi-Jun Chang, Seth Pettie, and Hengjie Zhang. Distributed triangle detection via expander decomposition. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 821–840. SIAM, 2019. doi:10.1137/1.9781611975482.51.

4 Yi-Jun Chang and Thatchaphol Saranurak. Improved distributed expander decomposition and nearly optimal triangle enumeration. In Peter Robinson and Faith Ellen, editors, *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019, Toronto, ON, Canada, July 29 - August 2, 2019*, pages 66–73. ACM, 2019. doi:10.1145/3293611.3331618.

5 Taisuke Izumi and François Le Gall. Triangle finding and listing in CONGEST networks. In Elad Michael Schiller and Alexander A. Schwarzmann, editors, *Proceedings of the ACM*

Symposium on Principles of Distributed Computing, PODC 2017, Washington, DC, USA, July 25-27, 2017, pages 381–389. ACM, 2017. doi:10.1145/3087801.3087811.

6 Ajay D. Kshemkalyani, Manish Kumar, Anisur Rahaman Molla, and Gokarna Sharma. Agent-based MST construction. *CoRR*, abs/2403.13716, 2024. doi:10.48550/arXiv.2403.13716.

7 Ajay D. Kshemkalyani and Gokarna Sharma. Near-optimal dispersion on arbitrary anonymous graphs. In *25th International Conference on Principles of Distributed Systems, OPODIS*, pages 8:1–8:19, 2021. doi:10.4230/LIPICS.OPODIS.2021.8.

8 Karolis Martinkus, Pál András Papp, Benedikt Schesch, and Roger 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. URL: https://openreview.net/pdf?id=8WTAh0tj2jC.

9 Anisur Rahaman Molla, Kaushik Mondal, and William K. Moses Jr. Byzantine dispersion on graphs. In *IPDPS*, pages 1–10, 2021.

10 Gopal Pandurangan, Peter Robinson, and Michele Scquizzato. On the distributed complexity of large-scale graph computations. In Christian Scheideler and Jeremy T. Fineman, editors, *Proceedings of the 30th on Symposium on Parallelism in Algorithms and Architectures, SPAA 2018, Vienna, Austria, July 16-18, 2018*, pages 405–414. ACM, 2018. doi:10.1145/3210377.3210409.

11 Debasish Pattanayak, Subhash Bhagat, Sruti Gan Chaudhuri, and Anisur Rahaman Molla. Maximal independet set via mobile agents. In *ICDCN*, pages 74–83. ACM, 2024. doi:10.1145/3631461.3631543.

12 Yuichi Sudo, Masahiro Shibata, Junya Nakamura, Yonghwan Kim, and Toshimitsu Masuzawa. Near-linear time dispersion of mobile agents, 2023. arXiv:2310.04376, doi:10.48550/arXiv.2310.04376.

13 Amnon Ta-Shma and Uri Zwick. Deterministic rendezvous, treasure hunts, and strongly universal exploration sequences. *ACM Trans. Algorithms*, 10(3):12:1–12:15, 2014. doi:10.1145/2601068.