Agentic Distributed Computing ⋆

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

[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, Kent State University, Kent, OH 44242, USA*

Abstract

The most celebrated and extensively studied model of distributed computing is the *message-passing model,* in which each vertex/node of the (distributed network) graph corresponds to a static computational device that communicates with other devices through passing messages. In this paper, we consider the *agentic model* of distributed computing which extends the message-passing model in a new direction. In the agentic model, computational devices are modeled as relocatable or mobile computational devices (called agents in this paper), i.e., each vertex/node of the graph serves as a container for the devices, and hence communicating with another device requires relocating to the same node. We study two fundamental graph level tasks, leader election, and minimum spanning tree, in the agentic model, which will enhance our understanding of distributed computation across paradigms. The objective is to minimize both time and memory complexities. Following the literature, we consider the synchronous setting in which each agent performs its operations synchronously with others, and hence the time complexity can

⋆The results for the case of $k = n$ were appeared as a brief announcement in the Proceedings of DISC 2024 [1].

*Corresponding author.

Email addresses: ajay@uic.edu (Ajay D. Kshemkalyani), manishsky27@gmail.com (Manish Kumar), molla@isical.ac.in (Anisur Rahaman Molla), gsharma2@kent.edu (Gokarna Sharma)

[1]The work of Manish Kumar is supported by CyStar at IIT Madras.

[2]The work of A. R. Molla was supported, in part, by ISI DCSW Project, file no. H5413.

be measured in rounds. In this paper, we present two deterministic algorithms for leader election: one for the case of $k < n$ and another for the case of $k = n$, minimizing both time and memory complexities, where k and n, respectively, are the number of agents and number of nodes of the graph. Using these leader election results, we develop deterministic algorithms for agents to construct a minimum spanning tree of the graph, minimizing both time and memory complexities. To the best of our knowledge, this is the first study of distributed graph level tasks in the agentic model with $k \leq n$. Previous studies only considered the case of $k = n$.

Keywords: Distributed algorithms, message-passing model, agentic model, agents, local communication, leader election, MST, time and memory complexity

---

1. Introduction

The well-studied *message-passing model* 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 $v \in V$ has a distinct $\Theta(\log n)$-bit identifier, $n = |V|$. The structure of G (topology, latency) is assumed to be unknown 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 restriction on storage. In the LOCAL 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 *agentic* model of distributed computing which extends the message-passing model in a new direction. In the agentic model, the computational devices are modeled as *relocatable or mobile computational devices* (called agents in this paper and hence the name 'agentic' for the computing model). 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 agentic model. The agentic model has two major differences with the message-passing model (Table 1 compares the properties).

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

Table 1: Comparison of the message-passing and agentic models.

Difference I. The graph nodes neither have identifiers, computation ability, nor 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.

Difference II is the major problem for the agentic model. To complicate further, while a device relocates to a neighbor, the device at that neighbor might relocate to another neighbor. This was not an issue in the message-passing model since, when a (static) device sends messages, it is always the case that neighboring nodes receive those messages. In this paper, we study the fundamental graph level tasks, such as leader election and minimum spanning tree in the agentic model, which will enhance our understanding of distributed computation across paradigms.

The agentic model is useful, applicable, and preferred to the message-passing model 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. Prominent use of the agentic model for network management is in underwater navigation [2], network-centric warfare in military systems [3], modeling social network [4], studying social epidemiology [5], etc.

First of all, with $k \leq n$ agents working on G, we develop two sets of deterministic algorithms for leader election, one for the case of $k < n$ and another

fo the case of $k = n$, with provable guarantees on two performance metrics that are fundamental to the agentic 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 consideration of storage complexity as the second performance metric is important as this metric was often neglected in the message-passing model with the implicit assumption that the devices have no restriction on the amount of storage needed to run the algorithm successfully. Instead, the focus was given on *message complexity* – the total number of messages sent by all nodes for a solution [6] as the second performance metric in the message-passing model.

Using the proposed leader election algorithms, we give stabilizing and explicit algorithms to construct a minimum spanning tree (MST) of G, another fundamental and well-studied problem in the message-passing model of distributed computing, in the agentic model, and provide both time and memory complexities.

To the best of our knowledge, this is the first time these two fundamental graph level tasks were studied in the agentic model for any $k \leq n$ in a systematic way, which will be more apparent in the following. To the best of our knowledge, no previous studies studied graph level tasks, including leader election and MST, for the case of $k < n$. Leader election and MST were studied for the case of $k = n$ [7, 8, 9], which our work in this paper subsumes as we consider $k \leq n$.

Computing 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 edge incident to it labeled $[1, \ldots, \delta_i]$. For a node w, we denote its neighbor nodes in G by $N(u)$. Consider an edge (u, v) that connects node u and node v (and node v to node u). We note the port at u leading to v by p_{uv} and the port at v leading to u by p_{vu}. We assume that the set $\mathcal{R} = \{r_1, r_2, \ldots, r_k\}$ of $k \leq n$ agents are initially positioned on the nodes of G. The agents have unique IDs in the range $[1, k^{O(1)}]$.

Initially, a node may have zero, one, or multiple agents positioned. An agent at a node can communicate with some (or all) agents present at that node, but not with the agents that are situated at some other nodes (i.e., we consider the *local communication* model [10]).

An agent can move from node v to node u along the edge (v, u). Following the message-passing literature, e.g., [11], we assume that an agent can traverse an edge in a round, irrespective of its (edge) weight (we consider G

weighted only for the MST problem in this paper; leader election does not require G to be weighted). An agent that moves from node v along the port p_{vu} is aware of the port p_{uv} when it arrives at u. Additionally, at any node v, it is aware of the weight $w(e)$ (if G is weighted) of the edge (v, u) that connects v to its neighbor node u. We assume that there is no correlation between the two port numbers of an edge.

Any number of agents are allowed to move along an edge at any time; that is, the agentic model does not put restrictions on how many agents can traverse an edge at a time.

The agents use the synchronous setting as in the standard CONGEST model: In each round, each agent r positioned at a node v can perform some local computation based on the information available in its storage as well as the (unique) port labels at the positioned node and decide to either stay at that node or exit it through a port to reach a neighboring node. Before exiting, the agent might write information on the storage of another agent that is staying at that node. An agent exiting a node always ends up reaching another node (i.e., an agent is never positioned on an edge at the end of a round).

The time complexity is the number of rounds of operations until a solution. The storage (memory) complexity is the number of bits of information stored at each agent throughout the execution.

Notice that at any round, the agent positions on G may satisfy the following:

- *dispersed* – $k(\leq n)$ agents are on k nodes,
- *general* – not *dispersed* (i.e., at least two agents on at least one node).

If the agents are in a dispersed (respectively, general) configuration initially, then we say the agents satisfy a dispersed (respectively, general) initial configuration. We say that an agent is *singleton* if it is alone at a node, otherwise *non-singleton* (or multiplicity).

Contributions. The leader election results are summarized in Table 2. Our results are of two types characterized by whether the elected leader is *overtaken* – the elected leader does not know whether its election is final or if other leaders might overtake it later; nonetheless, the election eventually stabilizes to a unique leader.

- **Stabilizing.** The elected leader may be overtaken.

- **Explicit.** The elected leader is not overtaken.

For the case of $k < n$, we develop two algorithms one stabilizing and another explicit. We design first the stabilizing algorithm and then modify it to make explicit. The stabilizing algorithm works as follows. For the case of $k < n$, only k nodes of G will be occupied even in a dispersed configuration. Let $C_i \subset G$ be a component (subgraph) of G with exactly one agent positioned per node. For any two components C_i, C_j, if $v_i \in C_i$ then any node $v_j \in C_j, C_j \neq C_i$, is at least 2 hops away, i.e., $hop(v_i, v_j) \geq 2$. In other words, if $hop(v_i, v_j) = 1$, then both v_i, v_j belong to the same component. For any $k < n$, there will be $1 \leq \kappa \leq k$ components $C_1, C_2, \ldots, C_\kappa$ such that (i) $C_1 \cup C_2 \cup \ldots \cup C_\kappa = k$ and $C_1 \cap C_2 \cap \ldots \cap C_\kappa = \emptyset$. Our leader election algorithms elect κ agents as leaders of κ components, one per component. For time and memory complexities, we need the following parameters which we define now.

- Let $|C_i|$ be the number of nodes in C_i (i.e., size of C_i)
- Let ℓ_i be the number of non-singleton nodes in C_i
- Let ζ_i be the number of agents that become local leaders (we define local leaders later) in C_i
- Let $|C_{max}| := \max_i |C_i|$, $\ell_{\max} := \max_i \ell_i$, and $\zeta_{max} := \max_i \zeta_i$

The time complexity is $O((|C_{max}| + \log^2 k)\Delta)$ and memory complexity is $O(\max\{\ell_{\max}, \zeta_{\max}, \log(k + \Delta)\} \log(k + \Delta))$ bits per agent. Therefore, our stabilizing algorithm guarantees that if an elected leader is not overtaken by another leader until $O((|C_{max}| + \log^2 k)\Delta)$, then that elected leader remains as a leader, i.e., the system as a whole stabilizes to a fixed leader after $O((|C_{max}| + \log^2 k)\Delta)$ rounds. Notice that the algorithm does not have parameter knowledge.

We now discuss how we make the above stabilizing algorithm for $k < n$ an explicit algorithm. For this we assume the agents to have parameter knowledge (specifically, k and Δ). Given the knowledge of k and Δ, in this algorithm, an agent eligible to become a unique leader in a component C_i waits until round $c_1 \cdot k\Delta$, for some constant c_1, before elevating itself as a leader. We will show that if an eligible agent to become a unique leader does not become aware of another eligible agent until $c_1 \cdot k\Delta$, then it can

Algorithm	Time	Memory/Agent	Known
Stabilizing, $k < n$	$O((\lvert C_{max}\rvert + \log^2 k)\Delta)$	$O(\max\{\ell_{max}, \zeta_{max}, \log(k+\Delta)\}\log(k+\Delta))$	$\times$
Explicit, $k < n$	$O(k\Delta)$	$O(\max\{\ell_{max}, \zeta_{max}\}\log(k+\Delta))$	k, Δ
Explicit, $k = n$	$O(m)$	$O(\max\{\ell, \zeta, \log n\}\log n)$	$\times$

Table 2: Summary of the results for leader election in the agentic model for $k \leq n$. $|C_{max}|$ is the size of the largest component, $\ell_{\max}$ and $\zeta_{\max}$, respectively, are the maximum number of non-singleton nodes and maximum number of local leaders among components, and ℓ and ζ, respectively, are the number of non-singleton nodes and the number of local leaders in G since there is a single component when $k = n$. m is the number of edges in G and Δ is the maximum degree of G.

Algorithm	Time	Memory/Agent	Known
Stabilizing, $k < n$	$O(\lvert C_{max}\rvert(\Delta + \log \lvert C_{max}\rvert) + \Delta \log^2 k)$	$O(\max\{\ell_{max}, \zeta_{max}, \Delta, \log(k+\Delta)\}\log(k+\Delta))$	$\times$
Explicit, $k < n$	$O(k\Delta)$	$O(\max\{\ell_{max}, \zeta_{max}, \Delta\}\log(k+\Delta))$	k, Δ
Explicit, $k = n$	$O(m + n\log n)$	$O(\max\{\ell, \zeta, \Delta, \log n\}\log n)$	$\times$

Table 3: Summary of the results for MST in the agentic model for $k \leq n$.

safely become a leader. We then prove that the runtime becomes $O(k\Delta)$ and memory becomes $O(\max\{\ell_{max}, \zeta_{max}\}\log(k+\Delta))$ bits per agent.

For the case of $k = n$, we develop an explicit deterministic algorithm for leader election without requiring any knowledge (neither exact nor upper bound) on parameters k, n, Δ, and D. Notice that for $k < n$, the explicit algorithm for leader elected needed knowledge of k, Δ. Additionally, we elected a single fixed leader (with no overtaking), i.e., $\kappa = 1$ (a single component C). The time complexity of this algorithm is $O(m)$ rounds and the memory complexity is $O(\max\{\ell, \zeta, \log n\}\log n)$ bits per agent, where ℓ, ζ are the non-singleton nodes in the initial configuration and the number of local leaders in the single component C.

Using these leader election results for both $k < n$ and $k = n$, we develop stabilizing and explicit algorithms for minimum spanning tree (MST) for both $k < n$ and $k = n$. The algorithms are either stabilizing or explicit based on the stabilizing or explicit leader election result used. The MST results are summarized in Table 3.

To the best of our knowledge, the results for both leader election and MST for $k < n$ are established for the first time in this paper. previously, both leader election and MST were studied in the agentic model only for

$k = n$. Since our results are for $k \leq n$, our results subsume those results.

Challenges in the Agentic Model. In the message-passing model, in a single round, a node can send a message to all its neighbors and receive messages from all its neighbors. Consider the problem of leader election. In the message-passing model, since nodes have IDs, in $D - 1$ rounds, all nodes can know the IDs of all other nodes, where D is the diameter of the graph. They then can simply pick the smallest/highest ID node as a leader, solving leader election in $O(D)$ rounds [12]. In contrast, in the agentic model, the messages from an agent, if any, that are to be sent to the 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, it is not clear how a leader could be elected, and if possible to do so, how much would be the time complexity. Additionally, in the agentic model, we have that $k < n$ or $k = n$. For the $k < n$ case, even in a dispersed configuration, there may not be an agent positioned at each node, and the challenge is how to deal with such scenarios. Therefore, computing in the agentic model is challenging compared to the message-passing model.

Simulating Message-passing Model to the Agentic Model. One may suggest solving tasks in the agentic model simulating the rich set of techniques available in the message-passing model. This is indeed possible when $k = n$ and n, Δ are known to agents a priori. When $k = n$ and n, Δ are known, if agents are not initially in a dispersed configuration, they can be dispersed in $O(n)$ rounds [13]. Since n is known, agents wait until $O(n)$ rounds to start solving graph level tasks. We can then show that an agent at a node meets all its neighbors in $O(\Delta \log n)$ rounds, knowing both n, Δ. The meeting helps us to guarantee that an agent can communicate with all its neighbors in $O(\Delta \log n)$ rounds. The argument is as follows. Each agent ID is $c \log n$ bits, for some constant $c \geq 1$. Encode the agent ID in $c \log n$ bits. Starting from the most significant bit until reaching the least significant bit, if the current bit is 1, r_u visits all its neighbors in the increasing order of port numbers, which finishes in $2\delta_u$ rounds. If the current bit is 0, r_u waits at u for 2Δ rounds. Since agent IDs are unique, there must be a round at which when r_u visits a neighbor v, the agent r_v is waiting at v, hence a *meeting*. Therefore, r_u visits all its neighbors in $c \log n \cdot \Delta = O(\Delta \log n)$ rounds. This meeting means that a round in the message-passing model for r_u can be simulated in $O(\Delta \log n)$ rounds in the agentic model. Therefore, any deterministic

algorithm $\mathcal{A}$ that runs for $O(T)$ rounds in the message-passing model can be simulated in the agentic model in $O(T\Delta \log n)$ rounds. Adding the time for dispersion, we have a total time $O(T\Delta \log n + n)$ rounds for simulation in the agentic model.

Although this simulation seems to work nicely for the agentic model, it suffers from two major problems: (i) assumption of $k = n$ and (ii) assumption of n, Δ known to agents a priori. In the agentic model, it may be the case that $k < n$ and n, Δ may not be known to agents a priori. Actually, it is the quest to design solutions in the agentic model that are oblivious to parameter knowledge. Not knowing the parameters, it is not clear how to meet neighbors. Additionally, we would like to solve tasks when $k \leq n$. It is not known whether complexity bounds better than those through simulation could be obtained by designing algorithms directly in the agentic model. This paper sheds light in this direction.

Techniques. We develop a 2-stage technique which can solve leader election in the agentic model, even when the parameters are not known and for any $k \leq n$. In Stage 1, the agents compete to become a 'local' leader. The singleton agents run a *Singleton_Election* procedure to become a local leader. The non-singleton agents run a *Non_Singleton_Election* procedure to become a local leader. The *Singleton_Election* procedure elects an agent at a node u as a local leader based on 1-hop neighborhood information of u. The *Non_Singleton_Election* procedure first disperses all the co-located agents to different nodes (one per node) running a DFS traversal and finally, an agent becomes a local leader. It is guaranteed that starting from any initial configuration, at least one agent becomes a local leader in each component C_i.

In Stage 2, the local leaders in each component C_i compete to become a 'global' leader at C_i. The local leader runs a *Global_Election* procedure to becomes a global leader. The *Global_Election* procedure runs a DFS traversal and checks for the conditions on whether a local leader can elevate itself as a global leader. The challenge is to handle the possibility of multiple local leaders being elected at different times. Care should be taken so that the whole process does not run into a deadlock situation. We do so by giving priority to the agent that becomes the local leader later in time so that such situations are avoided. In the case of known parameters, the agents run Stage 1 for $O(k\Delta)$ rounds and then Stage 2. In case of unknown parameters, the agents run Stage 2 as soon as Stage 1 finishes.

Denote by *home nodes* the graph nodes where agents became local leaders

in Stage 1. *Global_Election* for a local leader starts from its home node and ends at its home node. During *Global_Election*, the home nodes of the local leaders become empty since the agents are traversing G. Therefore, when a *Global_Election* procedure for a local leader reaches an empty node, it needs to confirm whether the empty node is, in fact, *unoccupied* (no agent was ever settled at that node) or *occupied* (a home node of some local leader that is currently away from home running *Global_Election*). This situation also applies for a *Non_Singleton_Election* procedure when it reaches an empty node; it needs to confirm whether the node is unoccupied or occupied. We overcome this difficulty by asking the local leaders to keep their home node information at an agent positioned at a neighbor.

However, this technique needs to handle two situations. There may be the case the there is no neighboring agent. There may also be the case the one neighboring agent may be responsible for storing the home node information about multiple local leaders. We develop an approach that allows the local leader in the first situation to become leader without running *Global_Election*. It becomes *non_candidate* if some other agent running *Global_Election* or *Non_Singleton_Election* visits it. We handle the second situation through the wait and notify approach. Suppose one agent is responsible for storing the home node information for multiple local leaders. We ask it to store the information about only one local leader. All other local leaders wait. After a while that node becomes free (i.e., it does not need to store anymore the home node information of the current local leader since that leader finished *Global_Election*). It then picks the local leader among the ones waiting to start *Global_Election* and keeps its home node information. All other local leaders in the group become *non_candidate*. The node that keeps home node information *oscillates* on the edge connecting it to the home node of the local leader running *Global_Election*. We will show that the wait time is bounded for each local leader and hence our claimed bounds are achieved.

After electing a leader in each component C_i, as an application, we use it to solve another fundamental problem of MST construction.

Related Work. In the message-passing model, 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 and studied heavily in the literature. 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 [12] provided a deterministic algorithm with optimal time complexity $O(D)$ and message complexity $O(mD)$. Recently, an algorithm was given in [17] 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 [17] that the message complexity lower bound of $\Omega(m)$ and time complexity lower bound of $\Omega(D)$ for deterministic leader election in graphs.

In the message-passing model, the minimum spanning tree (MST) problem was first studied by Gallager, Humblet, and Spira [15]. They gave a deterministic algorithm with time complexity $O(n \log n)$ and message complexity $O(m + n \log n)$. The time complexity was improved to $O(n)$ in [16] and to $O(\sqrt{n} \log^* n + D)$ in [11, 18]. Peleg and Rubinovich [19] established a time complexity lower bound of $\Omega(\sqrt{n}/\log n + D)$.

In the agentic model, gathering is the most studied problem which asks for agents initially positioned arbitrarily on the graph to be positioned at a single node not known a priori. The recent results are [20, 21] that solve Gathering under known n. Ta-Shma and Zwick [21] 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 ID among agents. Molla *et al.* [20] provided improved time bounds for large values of k assuming n is known but not k: (i) $O(n^3)$ rounds, if $k \geq \lfloor \frac{n}{2} \rfloor + 1$ (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 $\lfloor \frac{n}{3} \rfloor + 1 > k$. Each agent requires $O(M + m \log n)$ bits, where M is the memory required to implement the universal traversal sequence (UXS) [21].

The opposite of Gathering is the problem of Dispersion which asks the $k \leq n$ agents starting from arbitrary initial configurations in the agentic model to be positioned on k distinct nodes (one per node). Notice that we also solve Dispersion in this paper while electing a leader. In fact, for the case of $k = n$, solving Dispersion while electing a leader helps to make a single component C of $k = n$ nodes and hence only a single unique leader could be elected. For the case of $k < n$, Dispersion helps to come up with the minimum number of components possible, for the given initial configuration. Dispersion has been introduced by Augustine and Moses Jr. [10]. The state-of-the-art is a $O(k)$-round solution with $O(\log(k + \Delta))$ bits at each agent without any parameter knowledge, see [13].

Algorithm 1: Leader election for agent $\psi(u)$ positioned at node u

Input: A set $\mathcal{R}$ of $k < n$ agents with unique IDs positioned initially arbitrarily on the nodes of an n-node, m-edge anonymous graph G of degree Δ.

Ensure: Let C be a component (subgraph) of G such that exactly one agent is positioned on each node of C. An agent in each connected component C of G of agents is elected as a leader with status *leader*.

States: Initially, each agent $\psi(u)$ positioned at node u has $\psi(u).status \leftarrow candidate$, $\psi(u).init\_alone \leftarrow true$ if alone at u, $\psi(u).init\_alone \leftarrow false$ otherwise, and $\psi(u).all\_component\_edges\_visited \leftarrow false$. The *init_alone* variable is never updated for $\psi(u)$ throughout the algorithm, but the *status* variable can take values $\in \{candidate, non\_candidate, local\_leader, leader\}$ and the *all_component_edges_visited* variable can take the value *true*.

```
1 if ψ(u).status = candidate then
2     if ψ(u).init_alone = true then
3         Singleton_Election(ψ(u)) (Algorithm 2)
4     else
5         if ψ(u) is the minimum ID agent at u then
6             Non_Singleton_Election(ψ(u)) (Algorithm 4)
7 if ψ(u).status = local_leader then
8     Oscillation(ψ(u)) (Algorithm 6)
9     Global_Election(ψ(u)) (Algorithm 7)
```

Maximal Independent Set (MST) and Maximal Dominating Set (MDS) problems were also studied recently in the agentic model [22, 23] assuming $k = n$ and parameters n, Δ (additionally m and γ, the number of clusters in the initial configuration, for MDS [23]) are known to agents a priori. Leader election and MST were studied in the agentic model for the case of $k = n$ with known parameters in [9] and without known parameters in [7, 8]. The proposed results in this paper are the first for any graph level task, including leader election and MST, in the agentic model for the case of $k < n$.

Paper Organization. We discuss the stabilizing and explicit deterministic algorithms for leader election for $k < n$ in Section 2. An explicit deterministic algorithm for leader election for $k = n$ is discussed in Section 3. We then discuss stabilizing and explicit algorithms for MST for $k \leq n$ in Section 4. Finally, we conclude in Section 5 with a short discussion.

Algorithm 2: *Singleton_Election*($\psi(u)$)

```
1  N(u) ← neighbors of node u, |N(u)| = δ_u
2  ψ(u) visits all neighbors in N(u) (in the increasing order of port numbers)
3  if ∀v ∈ N(u), δ_u < δ_v then
4      if no node in N(u) is occupied or all occupied nodes in N(u) are singleton
         then
5          ψ(u).status ← local_leader
6      else
7          ψ(u).status ← non_candidate
8  if ∀v ∈ N(u), δ_u is the smallest but ∃v ∈ N(u) with δ_v = δ_u then
9      if (at least one such) v is empty then
10         Neighbor_Exploration_with_Padding(ψ(u)) (Algorithm 3)
11         if v is still empty and all occupied nodes in N(u) are singleton then
12             ψ(u).status ← local_leader
13     if no such v is empty then
14         if all occupied nodes in N(u) are singleton (including ψ(v)), and
             ψ(u).ID > ψ(v).ID then
15             ψ(u).status ← local_leader
16         else
17             ψ(u).status ← non_candidate
```

Algorithm 3: *Neighbor_Exploration_with_Padding*($\psi(u)$)

```
1  b ← number of bits in the ID of ψ(u)
2  b + 2b^2 ← number of bits in the ID of ψ(u) after padding a sequence of '10' bits
     b^2 times after the LSB in the original b-bit ID
3  starting from MSB and ending on LSB, if the bit is '1', ψ(u) visits the nodes in
     N(u) in the increasing order of port numbers, otherwise ψ(u) stays at u for 2δ_u
     rounds
```

Algorithm 4: *Non_Singleton_Election*(r_{min})

```
1  run DFS(r_min) until all R(u) agents initially at u settle at |R(u)| different
     unoccupied nodes of G (one per node). The minimum ID agent r_min which
     settles last becomes local_leader. All others become non_candidate once settled
2  if DFS(r_min) reaches an empty node then
3      run Confirm_Empty() procedure (Algorithm 5) to classify that node as
         occupied or unoccupied
```

Algorithm 5: $Confirm\_Empty(\psi(u))$

```
1 x ← empty node DFS(ψ(u)) is currently visiting
2 wait at x for a round
3 if ψ(u) does not meet any agent then
4     x is unoccupied
5 else
6     x is occupied
```

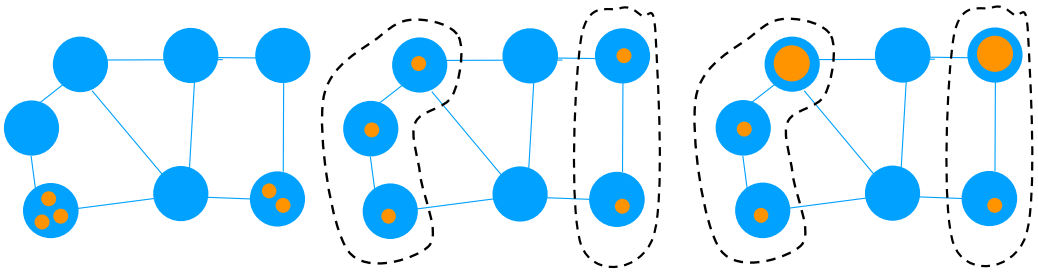

Figure 1: An illustration of leader election. (**left**) A general initial configuration of 5 agents on a 7-node graph G with 2 non-singleton nodes and no singleton node. (**middle**) Two components C_1 and C_2 are formed (shown with dashed lines) after two $Non\_Singleton\_Election$ procedures run by two non-singleton nodes finish; agent are now in a dispersed configuration in each component. (**right**) An agent at each component becomes a global leader (shown as big circles inside blue circles). Each agent in C_1 is (at least) 2 hops apart from an agent in C_2 (and vice-versa).

2. Leader Election, $k < n$

In the message-passing model, the case of $k < n$ does not occur, i.e., each node corresponds to a static computational device. In contrast, in the agentic model, when $k < n$, a node may not necessarily correspond to a computational device even in a dispersed configuration. We present our leader election algorithm which, starting from any initial configuration (dispersed or general) of $k < n$ agents on an n-node graph G, ensures that one agent is elected as a global leader in each component C_i, where C_i represents a subgraph of G on which each node has an agent positioned. As a byproduct, our algorithm transforms the general initial configuration to a dispersed configuration. Notice that there will be $1 \leq \kappa \leq k$ leaders elected if κ components are formed in the dispersed configuration. Fig. 1 provides an illustration of leader election for 5 agents initially positioned at two non-singleton nodes of a 7-node graph G. We develop a stabilizing algorithm (which does not

Algorithm 6: *Oscillation*($\psi(u)$)

```
1  if ψ(u) became local leader through Singleton_Election(ψ(u)) then
2      if ψ(u) found at least a node w ∈ N(u) with a settled agent ψ(w) while
         running Singleton_Election(ψ(u)) then
3          if ψ(w) is currently not oscillating then
4              ψ(w) keeps the information that node u is home(ψ(u))
5              ψ(u) starts Global_Election(ψ(u)) (Algorithm 7) and asks ψ(w) to
                 oscillate on the edge (w, u)
6          else
7              ψ(u) waits until ψ(w) finishes oscillation and notifies ψ(u)
8              if multiple local leaders waiting for ψ(w) to finish oscillation then
9                  if ψ(u) has highest priority among the waiting local leaders then
10                     ψ(u) starts Global_Election(ψ(v)) and asks ψ(w) to
                         oscillate on the edge (w, u)
11                 else
12                     ψ(u) become non_candidate
13     else
14         ψ(u).status ← leader
15         if ψ(u) meets any agent running Global_Election() or
             Non_Singleton_Election() then
16             ψ(u).status ← non_candidate
17 if ψ(u) became local leader through Non_Singleton_Election(ψ(u)) then
18     w ← the parent node of u in DFS tree built by DFS(ψ(u))
19     if w has an agent ψ(w) which is a non_candidate then
20         ψ(u) starts Global_Election(ψ(u)) and asks ψ(w) to oscillate on the
             edge (w, u)
21     if w has an agent ψ(w) which is a local_leader then
22         ψ(u) waits until ψ(w) notifies ψ(u) that it became a non_candidate
23     if w is empty and ψ(w′) is oscillating to w then
24         ψ(u) waits until ψ(w′) notifies ψ(u) that it finished oscillating (i.e., ψ(w)
             returned to w finishing Global_Election)
25         if no local leader agent ψ(w″) is asking ψ(u) to notify then
26             ψ(u) starts Global_Election(ψ(u)) and asks ψ(w) to oscillate on the
                 edge (w, u)
27         else
28             ψ(u) notifies ψ(w″) and becomes non_candidate
```

Algorithm 7: *Global_Election*$(\psi(u))$

```
1  run DFS(roundNo_u, ψ(u)), where roundNo_u is the round ψ(u) became local
     leader, and try to explore component C(ψ(u)).
2  if DFS(roundNo_u, ψ(u)) head reaches an empty node then
3      if Confirm_Empty() returns occupied then
4          continue DFS(roundNo_u, ψ(u)) in the forward phase
5      else
6          backtrack to the node from which DFS(roundNo_u, ψ(u)) entered that
             node
7  if ψ(u).all_component_edges_visited = true then
8      ψ(u).status ← leader; return home(ψ(u))
9  else if head of DFS(roundNo_u, ψ(u)) meets DFS(roundNo_v, ψ(v)) (head or
     nonhead) then
10     if roundNo_u > roundNo_v or
         (roundNo_u = roundNo_v & ψ(u).ID > ψ(v).ID) then
11         DFS(roundNo_u, ψ(u)) continues
12     else
13         DFS(roundNo_u, ψ(u)) stops, ψ(u) becomes non_candidate and returns
             home(ψ(u))
14 else if DFS(roundNo_u, ψ(u)) meets head of DFS(ψ(v)) running
     Non_Singleton_Election(ψ(v)) (Algorithm 4) then
15     DFS(roundNo_u, ψ(u)) stops, ψ(u) becomes non_candidate and returns
         home(ψ(u))
```

need any parameter knowledge, but overtaking of the elected leader in each component may occur for a certain time before stabilizing to a single leader). We will discuss later how to make it explicit (using knowledge of k, Δ but avoiding overtaking of the elected leader in each component).

2.1. Stabilizing Algorithm

The pseudocode of the algorithm is given in Algorithm 1. Initially, a graph node may have zero, one, or multiple agents. An agent at each node is 'candidate' to become a leader. Our algorithm runs in 2 stages, Stage 1 and Stage 2. Stage 2 runs after Stage 1 finishes. In Stage 1, if an agent is initially singleton at a node, it runs *Singleton_Election* to become a 'local leader'. However, if an agent is initially non-singleton, then it runs *Non_Singleton_Election* to become a local leader. As soon as becoming a local leader, Stage 2 starts in which the local leader agent runs *Global_Election* to

become a 'global leader'. To do so, a local leader may need to wait for a while. After the bounded wait, it either directly becomes non_candidate or starts to run *Global_Election*. After *Global_Election* finishes, a local elevates itself to a global leader and returns to its home node (if it is not at home already). The elected global leader may be overtaken by another global leader but we show that the process stabilizes to a global leader after $O((|C_{max}|+\log^2 k)\Delta)$ rounds in each component C_i, where $|C_{max}| := \max_i |C_i|$. The memory needed in $O(\max\{\ell_{max}, \zeta_{\max}, \log(k+\Delta)\}\log(k+\Delta))$ bits per agent. Here $\ell_{\max} := \max_i \ell_i$ and $\zeta_{max} := \max_i \zeta_i$, with ℓ_i, ζ_i, respectively, being the number of non-singleton nodes in C_i and the number of agents that become local leaders in C_i. We discuss Stage 1 and Stage 2 in detail separately below.

2.1.1. Stage 1 – Local Leader Election

Stage 1 concerns with electing local leaders among k agents. Stage 1 differs for singleton and non-singleton agents. Singleton agents run *Singleton_Election* and non-singleton agents run *Non_Singleton_Election*. We describe them separately below.

Singleton Election. The pseudocode of *Singleton_Election* is in Algorithm 2. Let r_u be the agent positioned at node u; we write $\psi(u) = r_u$. For $\psi(u)$ to eligible to become a local leader, either all neighbors in $N(u)$ have to be empty or all the non-empty neighbors in $N(u)$ must have a singleton agent. If this condition is true, then $\psi(u)$ becomes a local leader if one of the following conditions is satisfied:

- **Condition I.** $\forall v \in N(u), \delta_v > \delta_u$. In other words, u has the unique smallest degree among the nodes in $N(u)$.
- **Condition II.** $\forall v \in N(v), \delta_v \geq \delta_u$ with no node $v' \in N(u)$ with $\delta_{v'} = \delta_u$ is empty, and $\psi(u).ID > \psi(v').ID$. In other words, $\psi(u)$ has a higher ID than $\psi(v')$ when u and v' have the same smallest degree.
- **Condition III.** $\forall v \in N(v), \delta_v \geq \delta_u$ with (at least one) node $v' \in N(u)$ with $\delta_{v'} = \delta_u$ is empty, $\psi(u)$ runs $Neighbor\_Exploration\_with\_Padding(\psi(u))$ (Algorithm 3) and v' remains empty even after Algorithm 3 finishes. In other words, $\psi(u)$ finds that the same smallest degree neighbor v' is empty.

If none of the above conditions are satisfied, $\psi(u)$ becomes a 'non-candidate'. $\psi(u)$ checks for **Conditions I** and **II** after visiting all its neighbors in $N(u)$ starting from port 1 and ending at port δ_u. Notice that visiting

a neighbor finishes in 2 rounds, one round to reach to the neighbor and one round to return. If **Condition II** does not satisfy because of empty $v' \in N(u)$ (same smallest degree as u), $\psi(u)$ checks for **Condition III** after running $Neighbor\_Exploration\_with\_Padding(\psi(u))$ (Algorithm 3). We now discuss how $Neighbor\_Exploration\_with\_Padding(\psi(u))$ executes.

Same Smallest Degree Neighbor Exploration. When $\psi(u)$ shares its smallest degree among $N(u)$ with at least a node $v \in N(u)$, i.e., $\delta_u = \delta_v$, **Condition I** cannot be used. We need to decide where **Condition II** or **Condition III** can be used. $\psi(u)$ needs to determine whether v is (i) empty (ii) singleton or (iii) non-singleton. To confirm this, if there is $\psi(v)$, then $\psi(u)$ needs to meet $\psi(v)$ or vice versa. Knowing k, Δ, this meeting can be done in $O(\Delta \log k)$ rounds as discussed under the message-passing model simulation in Section 1. However, not knowing the parameters, this meeting is challenging and we develop the following meeting technique (Algorithm 3). Suppose agent $\psi(u)$ has an ID of b bits; note that $b \leq c \cdot \log k$ for some constant $c > 1$. We pad a sequence of '10' bits b^2 times after the LSB of the $\psi(u)$'s ID, i.e.,

$$b \underbrace{10}_{1} \underbrace{10}_{2} \cdots \underbrace{10}_{b^2}.$$

Now the ID of b bits becomes the ID of $b + 2b^2$ bits. Agent $\psi(u)$ starts from its MSB bit and ends at LSB bit. If the current bit is 1, then $\psi(u)$ visits all neighbors in $N(u)$ in the increasing order of port numbers, which finishes in $2\delta_u$ rounds. However, when the current bit is 0, it remains at u for $2\delta_u$ rounds. (Notice that $\phi(v)$, if any, also runs the same procedure padding appropriately its ID bits.) We will show that using this padding approach $\psi(u)$ meets $\psi(v)$ or vice-versa in $O(\Delta \log^2 n)$ rounds, a crucial component in Algorithm 2 (*Singleton_Election*). After Algorithm 3 finishes, then $\psi(u)$ either becomes a local leader or a non-candidate applying either **Condition II** or **Condition III**.

Non-Singleton Election. The pseudocode of *Non_Singleton_Election* is in Algorithm 4. Suppose a node v has $k' = |\mathcal{R}(v)|$ agents initially. Let r_{min} be the smallest ID agent in $\mathcal{R}(v)$. Agent r_{min} will become a local leader after setting agents in $\mathcal{R}(v) \backslash \{r_{min}\}$ to the nodes of G, one per node. The largest ID agent in $\mathcal{R}(v)$ stays at v and becomes $\psi(v)$. Agent r_{min} runs a DFS algorithm, we denote as $DFS(r_{min})$. As soon as an unoccupied node is visited, the largest ID agent in the group settles and others continue

$DFS(r_{min})$. An empty node visited in confirmed as occupied or unoccupied running Algorithm 5 ($Confirm\_Empty()$). As soon as r_{min} reaches to an unoccupied node u alone, it declares itself as a local leader $\psi(u)$. All the agents in $\mathcal{R}(v)\backslash\{\psi(u)\}$ become *non_candidate*. There are two issues to deal with in this process.

- **Issue 1 – Identifying an empty node visited occupied or unoccupied:** Suppose $DFS(r_{min})$ reaches an empty node x. An empty node can be of three types: (I) unoccupied (II) home of a singleton agent running *Singleton_Election*, (III) home of a local leader running *Global_Election*. *Non_Singleton_Election* that runs $DFS(r_{min})$ can only settle an agent on an unoccupied node. This is done through $Confirm\_Empty(r_{min})$ (Algorithm 5) procedure. At each empty node x visited, $DFS(r_{min})$ waits at x for a round. If an agent at x is doing *Singleton_Election*, it will return to x within a round. If an agent at x was a local leader (either through *Singleton_Election* or *Non_Singleton_Election*) doing *Global_Election*, there is an agent at $N(x)$ that is oscillating to x and that will be at x in the next round. Therefore, if x is empty even after waiting for one round, then x is unoccupied, and $DFS(r_{min})$ can settle an agent at x.

- **Issue 2 – $DFS(r_1)$ meets $DFS(r_2)$:** If the head of $DFS(r_1)$ meets the head of $DFS(r_2)$ at a node w, then, if $r_1.ID > r_2.ID$, then the highest ID agent belonging to $DFS(r_1)$ stays at w (and becomes *non_candidate*), otherwise the highest ID agent belonging to $DFS(r_2)$ stays at w (and becomes *non_candidate*). The winning DFS continues and the losing DFS stops and hands over the unsettled agents to the winning DFS to continue dispersing the agents.

- **Issue 3 – $DFS(r_1)$ meets $DFS(roundNo_v, \psi(v))$:** This case is an example of *Non_Singleton_Election* meeting *Global_Election*. $DFS(r_1)$ continues. If the meeting is at the head of *Global_Election* DFS, the agent doing *Global_Election* DFS becomes *non_candidate* and returns to its home node.

2.1.2. Stage 2 – Global Leader Election

Stage 2 concerns electing global leaders among local leaders. Stage 2 for a local leader agent $\psi(u)$ runs *Oscillation* and *Global_Election* (the pseudocodes are in Algorithms 6 and 7). The goal of running *Oscillation* is

to see whether a local leader can in fact run *Global_Election* or become a *non_candidate* even before running *Global_Election*. For the local leaders that can run *Global_Election*, the goal of running *Global_Election* is to see whether $DFS(roundNo_u, \psi(u))$ can explore $C(\psi(u))$, the component $\psi(u)$ belongs to. Exploration here means visiting all the edges that belong to $C(\psi(u))$. Additionally, $roundNo_u$ denotes the round at which $\psi(u)$ became a local leader which helps in preventing deadlock situations when $DFS(roundNo_u, \psi(u))$ meets $DFS(roundNo_v, \psi(v))$ from another local leader $\psi(v)$. Except the use of $roundNo$, $DFS(roundNo_u, \psi(u))$ is the same as $DFS(\psi(u))$ we use in *Non_Singleton_Election*. If $DFS(roundNo_u, \psi(u))$ can explore $C(\psi(u))$, it elevates itself as a 'global' leader and returns to $home(\psi(u))$ (if it is not already at $home(\psi(u))$). Agent $\psi(u)$ remains as a global leader if no other local leader meets $\psi(u)$ later in time. If another local leader $\psi(v)$ meets $\psi(u)$, then we say that overtaking occurred for $\psi(u)$. $\psi(u)$ (the current global leader) now becomes *non_candidate*. We will show that overtaking does not occur after $roundNo = c_1 \cdot (|C_{max}| + \log^2 k)\Delta$ rounds, i.e., the leader election at each component C_i stabilizes to a single global leader in at most $c_1 \cdot (|C_i| + \log^2 k)\Delta$ rounds. Notice that this approach does not use any parameter knowledge.

There are several challenges to overcome. The first challenge to overcome is, once elected a local leader, whether that local leader can run *Global_Election* and, if so, when. The second challenge to overcome is how to run *Global_Election* provided that there may be multiple *Global_Election* procedures concurrently running from different local leaders. There may also be concurrent situations of a *Global_Election* meeting *Non_Singleton_Election*. We describe our approaches to overcome these challenges separately below.

- **Issue 1 – Can a local leader $\psi(u)$ run *Global_Election* and if so when?** We have two sub-cases. The pseudocode is described in $Oscillation(\psi(u))$ (Algorithm 6).
  - **Issue 1.A – $\psi(u)$ became local leader through *Singleton_Election*:** We further have two sub-cases.
    - **Issue 1.A.i – $\psi(u)$ find no neighbor in $N(u)$ occupied:** This is a special case. $\psi(u)$ elevates itself as a global leader and stays at u. That means, it does not run *Global_Election*. Later, if it meets at u another

agent running *Global_Election* or *Non_Singleton_Election*, then $\psi(u)$ becomes *non_candidate*, since the agent running *Global_Election* or *Non_Singleton_Election* will become leader in later time.

  - **Issue 1.A.ii – $\psi(u)$ finds at least a neighbor in $N(u)$ occupied:** Let w be that neighbor with agent $\psi(w)$ positioned. $\psi(u)$ now checks whether $\psi(w)$ is oscillating. This oscillation helps in the *Confirm_Empty* procedure. If so, there must be another local leader, say $\psi(w')$, $w' \neq w \neq u$, already running *Global_Election* and $\psi(w)$ is oscillating on the edge (w, w'). Therefore, if $\psi(w)$ is not oscillating, $\psi(u)$ starts *Global_Election* and asks $\psi(w)$ to oscillate between w and u. However, if $\psi(w)$ is oscillating, $\psi(u)$ waits asking $\psi(w)$ to notify once $\psi(w)$ stops oscillating. $\psi(w)$ stops oscillating when the local leader $\psi(w')$ running *Global_Election* returns to w'. As soon as $\psi(w)$ notifies $\psi(u)$, $\psi(u)$ starts *Global_Election* and asks $\psi(w)$ to oscillate on the edge (w, u).
    There are two cases that needs further attention: The first case is of multiple local leaders finding $\psi(w)$ non-oscillating in the same round. In such case, symmetry is broken choosing the lexicographically highest priority local leader (we use *roundNo* of when the agent became local leader and its ID together to break symmetry). Others become *non_candidate*. The second case is of multiple local leaders waiting for $\psi(w)$ to stop oscillating. In such case, after $\psi(w)$ stops oscillating, $\psi(u)$ runs *Global_Election* if it is the lexicographically highest priority local leader among those waiting, otherwise it becomes *non_candiadate*.

- **Issue 1.B – $\psi(u)$ became local leader through *Non_Singleton_Election*:** Consider the DFS tree $DFS(\psi(u))$ built while running *Non_Singleton_Election*. Let w be the parent of u in $DFS(\psi(u))$. We have two sub-cases.
  - **Issue 1.B.i – Node w has an agent $\psi(w)$ which is a *non_candidate*:** In this case, $\psi(u)$ starts *Global_Election* and asks $\psi(w)$ to oscillate on the edge (w, u).
  - **Issue 1.B.ii – Node w has a local leader agent $\psi(w)$:** In this case, $\psi(u)$ waits until $\psi(w)$ notifies $\psi(u)$ that $\psi(w)$

became a *non_candidate*. This is the case of local leader chain in which the subsequent agent in the chain became local leader later in time than the previous agent and hence except the last agent in the chain, all others can become *non_candidate*.

  - **Issue 1.B.iii – Node w is empty and an agent $\psi(w')$ is oscillating to w:** There must be the case that $\psi(w)$ was a local leader (i.e., w happened to be $home(\psi(w'))$ of a local leader $\psi(w)$) currently running *Global_Election*. $\psi(u)$ asks $\psi(w')$ to notify once it stops oscillating. $\psi(u)$ waits at u until such notification. $\psi(w')$ stops oscillating once $\psi(w)$ returns to w after finishing *Global_Election*. $\psi(w')$ then notifies $\psi(u)$. $\psi(u)$ now checks if there is a local leader agent waiting (Issue 1.B.ii). If so, $\psi(u)$ informs that local leader and becomes *non_candidate*. Otherwise, $\psi(u)$ starts *Global_Election* and asks $\psi(w)$ to oscillate on the edge (w, u).

- **Issue 2 – How to run *Global_Election* handling meetings:** *Global_Election*$(\psi(u))$ of a local leader $\psi(u)$ runs $DFS(roundNo_u, \psi(u))$ to explore $C(\psi(u))$. We first discuss how $DFS(roundNo_u, \psi(u))$ runs. If a forward move at a node v takes $DFS(roundNo_u, \psi(u))$ to a node w with a settled agent $\psi(w)$, we know that $DFS(roundNo_u, \psi(u))$ is exploring $C(\psi(u))$. Suppose w is empty; i.e., a forward move from v took $DFS(roundNo_u, \psi(u))$ to an empty node. There are two situations: (i) either w is a home node (which is currently empty) or (ii) it is an unoccupied node. $DFS(roundNo_u, \psi(u))$ runs $Confirm\_Empty()$ at w. If $Confirm\_Empty()$ returns w occupied, $DFS(roundNo_u, \psi(u))$ continues making a forward move from w. However, if $Confirm\_Empty()$ returns w unoccupied, $DFS(roundNo_u, \psi(u))$ returns to v. It is easy to see that this approach makes $DFS(roundNo_u, \psi(u))$ to explore only the nodes and edges of component $C(\psi(u))$.

  During the traversal, $DFS(roundNo_u, \psi(u))$ may meet another DFS traversal. Notice that the another DFS traversal may be of *Global_Election* (type $DFS(roundNo_x, \psi(x))$) or of *Non_Singleton_Election* (type $DFS(\psi(x))$). We describe separately below how we handle them.

  - **Issue 2.A – $DFS(roundNo_u, \psi(u))$ meets**

$DFS(roundNo_x, \psi(x))$**:** We give priority to the agent that became local leader later in time, i.e., $DFS(roundNo_x, \psi(x))$ continues if $roundNo_u > roundNo_x$, otherwise $DFS(roundNo_x, \psi(x))$. There may be the case that $roundNo_u = roundNo_x$, which we resolve through agent IDs, i.e., $DFS(roundNo_u, \psi(u))$ continues if $\psi(u).ID > \psi(x).ID$, otherwise $DFS(roundNo_x, \psi(x))$. If $DFS(roundNo_u, \psi(u))$ stops, $\psi(u)$ becomes a non-candidate, and returns $home(\psi(u))$.

– **Issue 2.B** – $DFS(roundNo_u, \psi(u))$ **meets head of** $DFS(r)$ **running** $Non\_Singleton\_Election(r)$ **(Algorithm 7):** $DFS(roundNo_u, \psi(u))$ stops, becomes non-candidate, and returns $home(\psi(u))$. This is because agent r is eligible to become a local leader since it will finish $Non\_Singleton\_Election(r)$ later in time.

2.1.3. Analysis of the Algorithm

We now analyze our stabilizing algorithm (Algorithm 1). We have $k < n$ and agents have no parameter knowledge. We start with Stage 1 procedure *Singleton_Election* (Algorithm 2). Recall that *Singleton_Election* is run by singleton agents.

Lemma 1. *Suppose* $\forall v \in N(u), \delta_v \geq \delta_u$ *and* $\exists v' \in N(u), \delta_{v'} = \delta_u$. *Agent* $\psi(u)$ *meets a singleton agent* $\psi(v')$ *(if any positioned initially at node* v'*) or vice-versa in* $O(\delta_u \log^2 k)$ *rounds, running* $Neighbor\_Exploration\_with\_Padding(\psi(u))$ *(Algorithm 3).*

Proof. Each agent has a unique ID of size $\leq c \cdot \log k$, for some constant c. Therefore, for any two neighboring agents $\psi(u)$ and $\psi(v')$, the number of bits on their IDs are either equal or unequal. Consider the equal case first. If $\psi(u)$ and $\psi(v')$ have an equal number of bits (say, b) then their IDs must differ by at least a bit, i.e., if one has bit '1' at β-th place from MSB, another must have bit '0' at β-th place from MSB. Since an agent explores neighbors in $N(u)$ while the bit is '1' and stays at u when the bit is '0', $\psi(u)$ finds $\psi(v')$ within $2\delta_u \cdot b$ rounds.

Now consider the unequal case. Let $\psi(u)$ and $\psi(v')$, respectively, have b and d bits in their IDs, $b \neq d$. W.l.o.g., suppose $b > d$, i.e., $b = d + c_1$, $d, c_1 \geq 1$. Consider now the padding. The b bit ID becomes $b + 2b^2$ bit ID and the d bit ID becomes $d + 2d^2$ bit ID. We have that

$$b + 2b^2 = (d + c_1) + 2(d + c_1)^2 = 2d^2 + 2c_1^2 + 4 \cdot d \cdot c_1 + d + c_1.$$

The difference in the number of bits of the IDs of $\psi(u)$ and $\psi(v')$ is

$$2c_1^2 + 4 \cdot d \cdot c_1 + c_1.$$

Since $d, c_1 \geq 1$, $2c_1^2+4\cdot d\cdot c_1+c_1$ is at least 7-bit long, and out of the 7 bits, at least 3 bits are '1'. What that means is, if $\psi(v')$ (the smaller ID than $\psi(u)$) stops after $2\delta_v(d+2d^2)$ rounds, then there are at least 3 chances for $\psi(u)$ to meet $\psi(v')$ at v'. Therefore, the round complexity becomes $O(2\delta_u(b+2b^2)) = O(\delta_u \log^2 k)$ rounds, since $b \leq c \cdot \log k$. □

Lemma 2. *If $\psi(u)$ becomes a local leader running Singleton_Election($\psi(u)$), no singleton agent $\psi(v)$, $v \in N(u)$, becomes local leader running Singleton_Election($\psi(v)$). Singleton_Election($\psi(u)$) finishes in $O(\delta_u \log^2 k)$ rounds.*

Proof. Suppose $\forall v \in N(u), \delta_v > \delta_u$. Consider a singleton agent $\psi(v)$ at any node $v \in N(u)$. When $\psi(v)$ runs *Singleton_Election*($\psi(v)$) it finds that $\delta_u < \delta_v$ and $\psi(v)$ becomes a non-candidate. An unoccupied node $v' \in N(u)$ does not run *Singleton_Election*. Therefore, for $\psi(u)$, visiting all neighbors in $N(u)$ finishes in $2\delta_u$ rounds.

Now suppose $\forall v \in N(u), \delta_v \geq \delta_u$ and $\exists v' \in N(u), \delta_{v'} = \delta_u$. We have from Lemma 1 that $\psi(u)$ either finds v' has no singleton agent or meets singleton agent $\psi(v')$ in $O(\delta_u \log^2 k)$ rounds running *Neighbor_Exploration_with_Padding*($\psi(u)$) (Algorithm 3). If there is singleton agent $\psi(v')$ at v', then $\psi(u)$ becomes a local leader if $\psi(u).ID > \psi(v').ID$, otherwise a non-candidate. If v is unoccupied, $\psi(u)$ becomes a local leader and no agent in $N(u)$ becomes a local leader. The time complexity is to first visit all the neighbors in $N(u)$, which finishes in $2\delta_u$ rounds, and after that time to run *Neighbor_Exploration_with_Padding*($\psi(u)$) (Algorithm 3). Therefore, the total time complexity of *Singleton_Election*($\psi(u)$) is $2\delta_u + O(\delta_u \log^2 k) = O(\delta_u \log^2 k)$ rounds. □

Lemma 3. *Consider a dispersed initial configuration. At least one singleton agent $\psi(u)$ at node u becomes a local leader running Singleton_Election($\psi(u)$) (Algorithm 2).*

Proof. Let u be the smallest degree node in G, i.e., $\delta_u = \min_{v \in G} \delta_v$. We have two cases: (I) $\forall v' \in G, v' \neq u, \delta_u < \delta_{v'}$ (II) $\exists v' \in G, \delta_{v'} = \delta_u$. In Case (I), since u is the unique smallest degree node, $\psi(u)$ becomes a local leader in $O(\delta_u \log^2 n)$ rounds. No singleton agent $\psi(w)$ in $N(u)$ becomes

a local leader since $\delta_w > \delta_u$. Now consider Case (II). We have two sub-cases: (II.A) $v' \in N(u)$ (II.B) $v' \notin N(u)$. In Case (II.B), $\psi(u)$ is elected as a local leader as discussed in Case (I). For Case (II.A), we have from Lemma 2 that $\psi(u)$ meets $\psi(v')$ (or vice-versa) in $O(\delta_u \log^2 k)$ rounds. After that, either $\psi(u)$ or $\psi(v')$ becomes a non-candidate depending on their IDs. Suppose $\psi(u).ID > \psi(v').ID$, $\psi(v')$ becomes non-candidate, $\psi(u)$ becomes a local leader, and we are done. Otherwise, if $\psi(u).ID < \psi(v').ID$, $\psi(u)$ becomes a non-candidate. Now suppose even with $\psi(u).ID < \psi(v').ID$, $\psi(v')$ becomes a non-candidate. Then, there must be the case that v' has a neighbor $v'' \in N(v'')$ with $\delta_{v''} = \delta_{v'} = \delta_u$ and $\psi(v'').ID > \psi(v').ID$. This chain stops at the first node $v*$ such that $\psi(v''').ID < \psi(v*).ID > \ldots > \psi(v'').ID > \psi(v').ID > \psi(u).ID$, $v''' \neq v* \neq \ldots \neq v'' \neq v' \neq u$, if v''' has a singleton agent. If v''' is empty, this chain stops at $v*$. In both cases, agent $\psi(v*)$ in this chain becomes a local leader. □

We now analyze Stage 1 procedure *Non_Singleton_Election* (Algorithm 4). Recall that *Non_Singleton_Election* is run by non-singleton agents. The *Non_Singleton_Election* procedure for an agent r_{min} runs $DFS(r_{min})$.

Lemma 4. *Let w be an occupied node that is currently empty. There exists an agent at $w' \in N(w)$ that visits w every 2 rounds.*

Proof. The occupied node w is empty in the following situations.

- The agent $\psi(w)$ at w is doing *Singleton_Election*.
- The agent $\psi(w)$ at w is a local leader doing *Global_Election*.

In the first case, $\psi(w)$ returns w in the next round. In the second case, we consider two sub-cases:

- $\psi(w)$ **became local leader through** *Singleton_Election***:** Suppose all neighbors in $N(w)$ were empty when $\psi(w)$ became the local leader. In this case, $\psi(w)$ never leaves w. If at least one neighbor $w \in N(w)$ is non-empty, Algorithm 6 guarantees that the settled agent $\psi(w')$ oscillates on the edge (w', w), if $\psi(w)$ leaves w.
- $\psi(w)$ **became local leader through** *Non_Singleton_Election***:** Algorithm 6 again guarantees that the settled agent $\psi(w')$ oscillates on the edge (w', w), if $\psi(w)$ leaves w.

□

Lemma 5. *Suppose $DFS(r_{min})$ reaches an empty node w. The $Confirm\_Empty()$ procedure (Algorithm 5) confirms in an additional round whether w is unoccupied or occupied.*

Proof. Throughout the algorithm, an empty node w may be one of the three types:

(I) an unoccupied node,

(II) home of an agent running *Singleton_Election*,

(III) home of a local leader agent running *Global_Election*,

For Case (II), waiting for a round at w is enough since w is the home of the agent running *Singleton_Election* and hence the agent comes home every 2 rounds. For Case (III), Algorithm 6 guarantees that there is a neighboring agent at node $w' \in N(w)$ oscillating between w' and w which finishes in every 2 rounds (Lemma 4). □

Lemma 6. *$Non\_Singleton\_Election(r_{min})$ (Algorithm 4) run by an initially non-singleton agent r_{min} of minimum ID among the $x > 1$ co-located agents finishes dispersing x agents to x different nodes of G in $O(|C(r_{min})|\Delta)$ rounds with $O(\ell_{C(r_{min})} \log(k+\Delta))$ bits memory per agent, where $\ell_{C(r_{min})}$ is the number of non-singleton nodes in component $C(r_{min})$ agent r_{min} belongs to.*

Proof. Let ℓ be the number of non-singleton nodes in the initial configuration. ℓ DFSs will be run by ℓ minimum ID agents, one each from ℓ non-singleton nodes of G. Let u be a non-singleton node of x agents with the minimum ID agent r_{min}. $DFS(r_{min})$ with x initially co-located agents need to visit $x-1$ other empty nodes to settle all its co-located agents. The largest ID agent settles at u. When an empty node is visited by $DFS(r_{min})$ and it is unoccupied (Lemma 5), an agent settles. If an unoccupied node is visited by the heads of two or more DFSs, an agent from one DFS settles. Since $k < n$, $DFS(r_{min})$ may need to traverse $|C(r_{min})|\Delta$ edges to settle its agents. Regarding memory, an agent settled at a node may need to store the information about $\ell_{C(r_{min})}$ DFSs that form the component $C(r_{min})$. Since each DFS needs $O(\log(k+\Delta))$ bits at each node, the total memory needed at an agent at component $C(r_{min})$ is $O(\ell_{C(r_{min})} \log(k+\Delta))$ bits. □

Lemma 7. *If non-singleton nodes in the initial configuration are $\ell \geq 1$, $\ell_i \geq 1$ agents become local leaders in component C_i, where ℓ_i are the non-singleton nodes that belongs to C_i after all agents disperse. The non-singleton local leader election finishes for each agent in $O(|C_{max}|\Delta)$ rounds.*

Proof. Consider first the case of $\ell = 1$. A single $DFS(r_{min})$ runs forming a single component C. When $DFS(r_{min})$ finishes, r_{min} becomes a local leader.

Consider now the case of $\ell \geq 2$. There may be the case that the heads of two DFSs $DFS(r_1)$ and $DFS(r_2)$ meet. In this case, $DFS(r_1)$ continues, if $r_1.ID > r_2.ID$, otherwise $DFS(r_2)$. The DFS that stops will hand over remaining agents to be settled to the winning DFS. Additionally, no agent becomes local leader from the stopped DFS. Therefore, for two DFSs at least an agent from one DFS becomes a local leader. Therefore, if ℓ_i DFSs from ℓ_i non-singleton nodes meet to form a component C_i, there is at least one local leader elected.

We now prove the time complexity. For $\ell = 1$, an agent becomes a local leader in $O(C \cdot \Delta) = O(k\Delta)$ rounds (Lemma 6), as $|C| = k$. For $\ell \geq 2$, the election finishes in $O(C_i \cdot \Delta)$ rounds in each component C_i. Therefore, for each agent, the non-singleton local leader election finishes in $O(|C_{max}|\Delta)$ rounds. □

Lemma 8. *Consider a local leader $\psi(u)$ that cannot become a global leader. $\psi(u)$ can return to $home(\psi(u))$.*

Proof. Consider a local leader $\psi(u)$ that becomes *non_candidate* before running *Global_Election*. That local leader is already at its $home(\psi(u))$. Now consider a local leader $\psi(u)$ that can run *Global_Election*. During *Global_Election* (Algorithm 7), $\psi(u)$ runs $DFS(roundNo_u, \psi(u))$. $DFS(roundNo_u, \psi(u))$ builds a DFS tree T_u with its root $home(\psi(u))$. In T_u, there is a sequence of parent pointers from the head of $DFS(roundNo_u, \psi(u)))$ to $home(\psi(u))$. Since every local leader runs its own $DFS(roundNo_u, \psi(u))$ and maintains T_u, $\psi(u)$ can follow the parent pointers in T_u until reaching $home(\psi(u))$. □

Lemma 9. *Consider an agent $\psi(u)$ that became local leader at $roundNo_u$. Let $\psi(u)$ belongs to component C_i. Using procedure $Oscillation(\psi(u))$ (Algorithm 6), $\psi(u)$ either becomes non_candidate or starts running Global_Election within $O((|C_i| + \log^2 k)\Delta)$ rounds.*

Proof. We look into whether $\psi(u)$ became local leader through *Singleton_Election* or *Non_Singleton_Election*.

We first consider the *Singleton_Election* case. Suppose $\psi(u)$ became local leader at $roundNo_u$ and belongs to C_i. Let $\psi(w)$ be the agent at $w \in N(u)$ that is oscillating. $\psi(w)$ becomes non-oscillating by $roundNo_u + O(|C_i|\Delta)$. Additionally, each agent becoming local leader through *Singleton_Election* will be elected local leaders by $O(\Delta \log^2 k)$ rounds, i.e., $roundNo_u \leq O(\Delta \log^2 k)$ (Lemma 2). Therefore, by $roundNo_u + O(|C_i|\Delta) \leq O(\Delta \log^2 k) + O(|C_i|\Delta)$ rounds, either $\psi(u)$ becomes a non-candidate or starts running *Global_Election*.

We now consider the *Non_Singleton_Election* case. Suppose $\psi(u)$ became local leader at $roundNo_u$ and belongs to C_i. Let $\psi(w)$ be the agent at the parent w of node u in the DFS tree built by $\psi(u)$. Let $\psi(w')$ be the agent at $w' \neq w \neq u$ oscillating to w. $\psi(w')$ becomes non-oscillating by $roundNo_u + O(|C_i|\Delta)$. Additionally, $roundNo_u \leq O(|C_i|\Delta$. Therefore, by $roundNo_u + O(|C_i|\Delta) \leq O(|C_i|\Delta)$ rounds, either $\psi(u)$ becomes a non-candidate or starts running *Global_Election*.

We have the claimed bound, combining the two results. □

Lemma 10. *Global_Election (Algorithm 7) at each component C_i elects a unique global leader at C_i and terminates in $O((|C_i| + \log^2 k)\Delta)$ rounds with $O(\zeta_i \log(k + \Delta))$ bits at each agent.*

Proof. Consider an agent $\psi(u)$ that becomes a local leader at node u of component C_i. It either becomes non-candidate or initiates *Global_Election*$(\psi(u))$ (Algorithm 7) by round $O((|C_i| + \log^2 k)\Delta)$. *Global_Election*$(\psi(u))$ runs $DFS(roundNo_u, \psi(u))$. Suppose $DFS(roundNo_u, \psi(u))$ visits an empty node w. We have from Lemma 5 that it can be confirmed in an additional round whether w is in fact unoccupied or occupied running Algorithm 5. If it is unoccupied, then $DFS(roundNo_u, \psi(u))$ simply backtracks to the previous node since w does not belong to C_i. If $DFS(roundNo_u, \psi(u))$ meets another $DFS(roundNo_v, \psi(v))$ then either $DFS(roundNo_u, \psi(u))$ or $DFS(roundNo_u, \psi(v))$ continues. If $DFS(roundNo_u, \psi(u))$ meets $DFS(r)$ doing *Non_Singleton_Election* then $DFS(roundNo_u, \psi(u))$ stops as agent r is eligible to become a local leader at C_i later in time and which will run *Global_Election* to become a global leader in C_i. Therefore, *Global_Election* elects a unique global leader in C_i. The total time to finish *Global_Election* (Algorithm 7) in C_i is $O(|C_i|\Delta)$ rounds. To store information about

Global_Election for each local leader in C_i, $O(\log(k+\Delta))$ bits at each agent is sufficient. Given ζ_i local leaders in C_i, the total memory requirement is $O(\zeta_i \log(k+\Delta))$ bits at each agent. □

Lemma 11. *After Global_Election finishes, for any two agents $\psi(u), \psi(v)$ such that $u \in C_i$ and $v \in C_j$, $hop(u,v) \geq 2$.*

Proof. We prove this by contradiction. Suppose $hop(u,v) = 1$, meaning that agents $\psi(u)$ and $\psi(v)$ are neighbors but belong to two different components. Let $u \in C_i$ and $v \in C_j$. For this to happen, the *Global_Election* in C_i (C_j) must not visit node v (u). Consider the *Global_Election* DFS on either C_i or C_j that started last in time (suppose that is from C_i). That *Global_Election* DFS visits one-hop neighborhood of all the nodes in C_i. Since $hop(u,v) = 1$, *Global_Election* DFS must visit node v and since v has agent $\psi(v)$, $\psi(v)$ is considered as the agent that belongs to component C_i. Hence, a contradiction. □

Theorem 12 (Stabilizing Algorithm, $k < n$). *There is a stabilizing deterministic algorithm for $k < n$ agents in the agentic model that elects one agent as a leader in each component C_i, without agents knowing any graph parameter a priori. The time complexity of the algorithm is $O((|C_{max}| + \log^2 k)\Delta)$ rounds and the memory complexity is $O(\max\{\ell_{\max}, \zeta_{\max}, \log(k+\Delta)\}\log(k+\Delta))$ bits per agent.*

Proof. Consider the dispersed initial configuration. In Stage 1, only *Singleton_Election* runs. We have from Lemma 3, that at least one agent becomes a local leader. We have from Lemma 2, *Singleton_Election* for each agent finishes in $O(\Delta \log^2 k)$ rounds with memory $O(\log^2(k+\Delta))$ bits per agent. Let ζ_i be the number of local leaders in each C_i. In Stage 2, *Global_Election* for each local leader finishes in $O(|C_{max}|\Delta)$ rounds with $O(\zeta_{max} \log(k+\Delta))$ bits at each agent. Therefore, time complexity becomes $O((|C_{max}| + \log^2 k)\Delta)$ rounds and the memory complexity becomes $O(\max\{\zeta_{\max}, \log(k+\Delta)\}\log(k+\Delta))$ bits per agent.

Consider the general initial configuration. We have two sub-cases. (1) There is no singleton node. (2) There is a mix of singleton and non-singleton nodes. Consider no singleton node case first. ℓ instances of *Non_Singleton_Election* finish in $O(|C_{max}|\Delta)$ rounds in each component. The memory is $O(\ell_{max} \log(k+\Delta))$ bits per agent. *Global_Election* then finishes in next $O(|C_{max}|\Delta)$ rounds in each component with memory

$O(\zeta_{\max} \log(k+\Delta))$ bits per agent (Lemma 10). Consider now the case of ℓ non-singleton nodes and at least a singleton node. *Singleton_Election* finishes in $O(\Delta \log^2 k)$ rounds (Lemma 2). Everything else remains the same.

Consider the local leaders that cannot become global leaders after running *Global_Election*. We have from Lemma 8 that they can return to their home nodes. Notice that returning to home nodes takes $O(|C_{max}|)$ rounds since agents can traverse parent pointers on the DFS tree they build while running *Global_Election*. The memory remains $O(\zeta_{\max} \log(k+\Delta))$ bits per agent. Lemma 11 shows that the agents in two components are at least 2-hop away from each other, satisfying our component definition. Therefore, our leader election is correct that one component has exactly one leader.

Combining all these time and memory bounds, we have time complexity $O((|C_{max}| + \log^2 k)\Delta)$ and memory complexity $O(\max\{\ell_{\max}, \zeta_{\max}, \log(k+\Delta)\} \log(k+\Delta))$ bits per agent. □

2.2. Explicit Algorithm

We now discuss how Algorithm 1 can be made explicit (no overtaking of global leader in each component C_i) with agents knowing k, Δ. Our explicit algorithm follows the 2-stage approach as in the stabilizing algorithm. In Stage 1, singleton agents run *Singleton_Election* and non-singleton agents run *Non_Singleton_Election* to become local leaders. The local leaders then run *Global_Election* in Stage 2. Stage 1 runs for $c_1 \cdot k\Delta$ rounds, i.e., the agents that become local leaders in Stage 1 start Stage 2 at round $c_1 \cdot k\Delta + 1$. Stage 2 also runs for $c_1 \cdot k\Delta$ rounds, i.e., a local leader agent eligible to become a global leader declares itself as a global leader after Stage 2 runs for $c_1 \cdot k\Delta$ rounds. We prove that this approach avoids overtaking, meaning once one local leader agent becomes a global leader, there is no other local leader that becomes global leader later in time.

We now discuss how *Singleton_Election*$(\psi(u))$ executes knowing k, Δ. $\psi(u)$ meets all its neighboring singleton agents in $O(\Delta \log k)$ rounds (see discussion in message-passing model simulation in Section 1). Therefore, *Singleton_Election*$(\psi(u))$ finishes in $O(\Delta \log k)$ rounds. *Non_Singleton_Election*$(\psi(u))$ finishes in $O(k\Delta)$. The eligible agents to become local leader declare themselves as local leaders at round $c_1 \cdot k\Delta$ (not before). Additionally, at $c_1 \cdot k\Delta$, all agents have already dispersed. Let $C_1, C_2, \ldots, C_\kappa$ be κ components formed at round $c_1 \cdot k\Delta$. It is easy to see that components remain fixed after $c_1 \cdot k\Delta$.

The local leaders then run *Oscillation* for $c_1 \cdot k\Delta$ rounds, some of them become *non_candidate* and some of them are eligible to run *Global_Election*. The local leader $\psi(u)$ eligible to run *Global_Election* then runs $DFS(\psi(u))$. We do not need $roundNo$ here since all eligible local leaders run *Global_Election* at the same round and symmetry breaking based on the leader ID is enough. When $DFS(\psi(u)$ meets $DFS(\psi(v))$, $DFS(\psi(v))$ stops if $\psi(u).ID > \psi(v).ID$. If $\psi(u)$ finishes visiting all the nodes in $C(\psi(u))$, then it can declare itself as a global leader of the component $C(\psi(u))$. Notice that if $\psi(u)$ visits all the nodes of $C(\psi(u))$ then it must be the case that it got the highest priority among all other local leaders in $C(\psi(u))$ that ran *Global_Election* and hence $DFS(\psi(u))$ stopped all of them.

Theorem 13 (**Explicit Algorithm, $k < n$**). *There is an explicit deterministic algorithm for $k < n$ agents in the agentic model that elects one agent as a leader in each component C_i with agents knowing k, Δ a priori. The time complexity of the algorithm is $O(k\Delta)$ rounds and the memory complexity is $O(\max\{\ell_{max}, \zeta_{\max}\} \log(k + \Delta))$ bits per agent.*

Proof. The only difference with Theorem 12 for the stabilizing algorithm is that since k, Δ are known, *Singleton_Election* finishes in $O(\Delta \log k)$ rounds with only $O(\log(k + \Delta))$ bits per agent. The total time complexity of Stage 1 and Stage 2 is $O(k\Delta)$. Therefore, the time complexity becomes $O(k\Delta)$ rounds and the memory complexity becomes $O(\max\{\ell_{max}, \zeta_{\max}\} \log(k + \Delta))$ bits per agent. The correctness proof of two agents in two components are at least 2 hops apart follows easily from the proof of Lemma 11. □

3. Leader Election, $k = n$

We discuss here how the stabilizing algorithm for the case of $k < n$ (not knowing k, Δ) can be made explicit for the case of $k = n$ without knowing k, Δ. Additionally, only one global leader will be elected (i.e., the number of components $\kappa = 1$). The time complexity is $O(m)$ and memory complexity is $O(\max\{\ell, \zeta, \log n\} \log n)$ bits per agent with ℓ being the number of non-singleton nodes in the initial configuration and ζ being the number of nodes that become local leaders in Stage 1.

We discuss here where we modify Algorithm 1 for the case of $k = n$. The modified algorithm still has two stages, Stage 1 and Stage 2. In Stage 1, *Non_Singleton_Election* procedure execute as in Algorithm 4. The change is on *Singleton_Election* and *Global_Election*. *Singleton_Election* elects an

agent $\psi(u)$ at node u as a local leader if and only if $\psi(u)$ finds all neighbors in $N(u)$ have a singleton agent positioned. Regarding *Global_Election*, the $DFS(roundNo_u, \psi(u))$ run by local leader agent $\psi(u)$ becomes a global leader if and only if it can visit all the edges of G. That is, if it visits an unoccupied node while running $DFS(roundNo_u, \psi(u))$, then $\psi(u)$ stops $DFS(roundNo_u, \psi(u))$, becomes *non_candidate*, and returns to $home(\psi(u))$. This is because the encounter of an unoccupied node by *Global_Election* means that at least one $Non\_Singleton\_Election(r_v)$ procedure by an initially non-singleton agent has not been finished yet. $Confirm\_Empty()$ (Algorithm 5) can correctly guarantee that whether an empty node visited is occupied or unoccupied since for the occupied node there is an oscillating agent visiting it every two rounds.

Theorem 14 (**Explicit Algorithm,** $k = n$)**.** *There is an explicit deterministic algorithm for $k = n$ agents in the agentic model that elects one agent as a leader in the graph G, without agents knowing any graph parameter a priori. The time complexity of the algorithm is $O(m)$ rounds and the memory complexity is $O(\max\{\ell, \zeta, \log n\} \log n)$ bits per agent.*

Proof. This algorithm follows the stabilizing algorithm for $k < n$ but avoids overtaking. Stage 1 finishes in $O(m)$ rounds as the DFS may need to visit all the edges of the graph before *Non_Singleton_Election* settles all the agents. *Singleton_Election* finishes in $O(\Delta \log^2 n)$ rounds. Stage 2 also finishes in $O(m)$ rounds. A local leader that cannot become a global leader can return its home node in $O(n)$ rounds following the path in the DFS tree built during *Global_Election*. Therefore, the total time complexity is $O(m)$ rounds.

Regarding memory complexity, $Neighbor\_Exploration\_with\_Padding()$ needs $O(\log^2 n)$ bits per agent. *Non_Singleton_Election* needs $O(\max\{\ell, \zeta\} \log n)$ bits per agent, with ζ being the number of local leaders in Stage 1. *Global_Election* also needs $O(\max\{\ell, \zeta\} \log n)$ bits per agent to facilitate local leaders that become *non_candidate* during *Global_Election* to return to their home nodes. Therefore, the total memory complexity is $O(\max\{\ell, \zeta, \log n\} \log n)$ bits per agent. □

4. MST Construction

In this section, we present a stabilizing deterministic algorithm for $k < n$ to construct an MST of each component C_j. We will discuss, towards the

end of this section, an explicit version of this algorithm for $k < n$, knowing k, Δ and explicit version for $k = n$ not knowing any graph parameters.

We start the MST construction process after a leader is elected. Given a leader r_l of Section 3 and its DFS tree T_{r_l} built while running $DFS(roundNo_l, r_l)$, we discuss the algorithm assuming leader r_l has already been stabilized, i.e., it will not be overtaken. We will discuss later how to handle the case of the leader that started MST construction being overtaken later. The agents do not need to know any graph parameters. The MST construction finishes in $O(\Delta \log^2 k + |C_j|\Delta + |C_j| \log |C_j|)$ rounds with $O(\max\{\ell_j, \zeta_j, \Delta, \log(k+\Delta)\} \log(k+\Delta))$ bits at each agent. We consider G to be weighted and hence our MST for each component C_j is a minimum weight MST of C_j. Having κ components with κ leaders, we will have κ MSTs, one per component.

After Algorithm 1, let r_v be a leader of component C_j positioned at node $v \in C_j$. Leader r_v runs $DFS(r_v)$ to visit all other $|C_j| - 1$ agents (at $|C_j| - 1$ nodes) at component C_j and assign them rank based on the order they are visited, i.e., the agent visited i-th in the order receives rank i. This is done by r_v revisiting the DFS tree T_{r_v} built by r_v while running $DFS(roundNo_v, r_v)$ to elect itself as a leader during Algorithm 7 (*Global_Election*). This revisit finishes in $O(|C_j|\Delta)$ rounds.

As soon as an agent receives its rank, it considers itself as a single sub-component. The leader r_v at node v (which has rank-1) starts MST construction. r_v includes the MOE (minimum weight outgoing edge) adjacent to v in its sub-component and passes a token (message) to the rank-2 agent. This process runs iteratively until the rank-$(|C_j| - 1)$ agent passes the token to the rank-$|C_j|$ agent. Consequently, the rank-$|C_j|$ agent includes in its sub-component the MOE available and passes the token to the rank-1 agent (r_v). This whole process of passing the token from rank-1 node to rank-$|C_j|$ node and back to rank-1 node is one phase.

In the next phase, the minimum rank agent would include the MOE available to its sub-component and pass the token to the next minimum rank agent that belongs to another sub-component (within the component). In this way, the token reaches the minimum rank agent from the highest rank agent and this phase is completed. This process is repeated phase-by-phase until there is a single sub-component left, which is component C_j. Eventually, we have an MST of component C_j with $|C_j|$ agents. Let us call this algorithm *MST_Construction*. A complete pseudocode is given in Algorithm 8.

Our algorithm resembles the MST algorithm of Gallager, Humblet, and

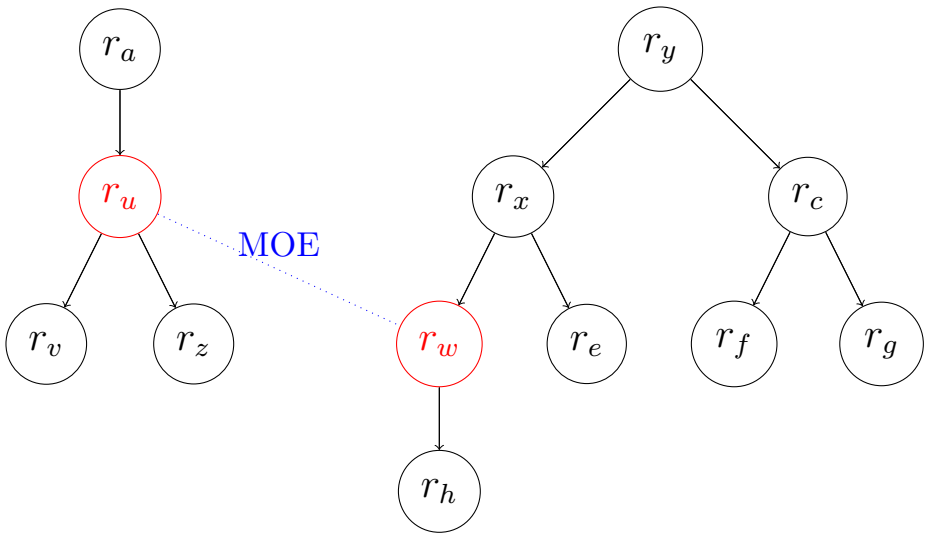

Figure 2: Sub-components C_{r_w} and C_{r_u} before merging.

Spira [15] with the difference that we start MST construction through ranks already provided to agents, whereas in [15] all nodes have the same rank. The merging of two same rank sub-components with rank k in [15] provides a new rank of $k+1$ for the merged sub-component. In ours, there will be no same rank sub-component, and hence the merged sub-component gets the rank of one of the sub-components merged.

Figs. 2 and 3 illustrate these ideas. Fig. 2 shows sub-components C_{r_u} and C_{r_w} before they merge due to the MOE connecting r_u with r_w. Fig. 3 captures the merged sub-component $C^{new}_{r_u}$ such that root of the $C^{new}_{r_u}$ remains unchanged and the pointer changes occurred in the C_{r_w} sub-component during the merging. The directed edge denotes new parent-child relationships.

We now discuss the overtaking issue of a leader that started MST construction in C_j. Suppose leader r_l started MST construction and it is overtaken by another leader r'_l. In the MST construction, we ask leaders to provide nodes with $roundNo$ information at which they became leaders. If an agent r now gets to know that a leader r'_l with $roundNo_{l'} > roundNo_l$ started MST construction, r discards its MST construction for r_l and initiates the process for r'_l. Since leader election stabilizes to a single global leader in each component C_i, eventually, all MST construction processes from overtaken leaders stabilize to a single process from the leader that will not be overtaken.

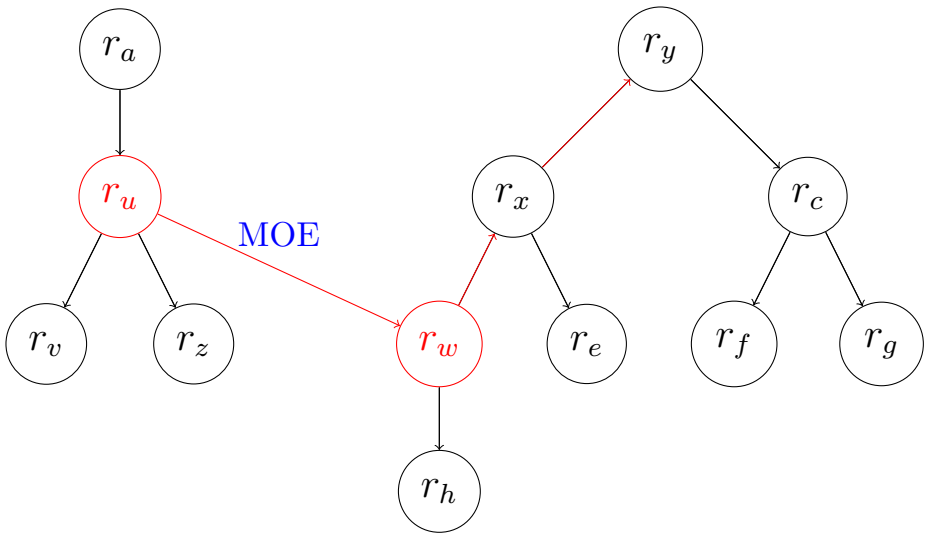

Figure 3: Sub-component $C_{r_u}^{new}$ after merging.

Analysis of the Algorithm. We now analyze Algorithm 8 for its correctness, time, and memory complexities.

Lemma 15. *Suppose a leader r_l running MST construction (Algorithm 8) in component C_j is overtaken by another leader r'_l. Algorithm 8 by r'_l subsumes the MST construction of r_l.*

Proof. The MST construction instance of r'_l meets the MST construction instance of r_l. In every node, r'_l will ask the agent to follow the MST construction instance of r'_l discarding the information regarding r_l. Since agents in C_i do not change their home nodes during MST construction, the subsumption is guaranteed. □

Lemma 16. *Algorithm 8 generates the MST of component C_j.*

Proof. We prove this in three steps: firstly, Algorithm 8 constructs a tree in C_j; secondly, the constructed tree is a spanning tree in C_j; finally, the spanning tree is, indeed, a minimum spanning tree in C_j. Firstly, we prove by contradiction that no cycle is generated during Algorithm 8. Let us suppose, there exists a cycle at any point during the algorithm. Then it implies two sub-components with the same rank merged at some point, which is a contradiction. Secondly, let us suppose there exist at least two sub-components

Algorithm 8: *MST_Construction*

Input: An n-node anonymous network with $k \leq n$ agents with unique IDs placed on k nodes forming κ components with an agent r_l at a node elected as a leader in each component C_j (Algorithm 1) with DFS tree T_{r_l} built while running $DFS(roundNo_l, r_l)$ (Algorithm 7).

Ensure: MST construction of component C_j.

The leader assumes rank 1. It re-traverses the DFS tree T_{r_l} and returns to its home node. While re-traversing T_{r_l}, it provides the distinct rank from the range of $[2, |C_j|]$ to the agents at $|C_j| - 1$ other nodes in order of visit.

Each agent $r_u \in C_j$ considers itself as a sub-component C_{r_u} with $rank(C_{r_u}) \leftarrow rank(r_u)$.

The leader $r_l \in C_j$ generates a token. Let its sub-component be C_{r_l}.

while $|C_{r_l}| < |C_j|$ **do**

  if r_u *has the token* **then**

    if $rank(r_u) \leq rank(C_{r_u})$ **then**

      Agent r_u finds the MOE of the C_{r_u} going to another sub-component C_{r_w} connecting r_u with r_w.

      if $rank(C_{r_u}) < rank(C_{r_w})$ **then**

        r_w becomes the root node of C_{r_w} by reversing the parent-child pointers from r_w up to the root node of C_{r_w} and r_u becomes the parent of r_w (Fig. 2). C_{r_w} then merges with C_{r_u} giving a new sub-component $C^{new}_{r_u}$ with $rank(C^{new}_{r_u}) \leftarrow rank(C_{r_u})$ (Fig. 3).

      if $rank(r_u) < |C_j|$ **then**

        r_u passes the token to agent r_v with $rank(r_v) = rank(r_u) + 1$ using T_{r_l}.

    else if $rank(r_u) > rank(C_{r_u})$ *and* $rank(r_u) < |C_j|$ **then**

      r_u passes the token to agent r_v with $rank(r_v) = rank(r_u) + 1$ using T_{r_l}.

    else if $rank(r_u) = |C_j|$ **then**

      r_u passes the token to the leader r_l (with rank 1) using T_{r_l}. This token passing visits the parents of r_u until the token reaches the root node of T_{r_l}, where the leader is positioned.

at the end of the algorithm. This implies that the leader sub-component (rank-1) did not merge with the other sub-component and terminated the algorithm, which contradicts the fact that the algorithm terminates when the leader is connected to n agents altogether.

Finally, consider that the tree formed for C_j by our algorithm is T and the MST is T^*. Note that in Algorithm 8, each edge added to the MST tree is

selected by the selection of an MOE. If $T = T^*$ then T is minimum spanning tree. If $T \neq T^*$ then there exists an edge $e \in T^*$ of minimum weight such that $e \notin T$. Therefore, there exists a phase in which e was not considered during sub-component merging and an edge with higher weight, say e', was considered. But this is contradictory to Algorithm 8 which merges the sub-components with an MOE. Therefore, Algorithm 8 constructs the MST of C_j. □

Lemma 17. *In Algorithm 8, the leader initiates the merging $O(\log |C_j|)$ times in component C_j.*

Proof. Initially, there are $|C_j|$ single-node sub-components in Algorithm 8 (Line 2) for each component C_j. In each phase (leader initiating a token until the token returns to the leader), each sub-component merges at least once with another sub-component. Therefore, after every phase, the number of sub-components is reduced by at least half. Consequently, after $O(\log n)$ phases, there remains only a single sub-component of $|C_j|$ nodes, which is component C_j itself. □

Theorem 18 (**Stabilizing MST, $k < n$**). *There is a stabilizing deterministic algorithm for constructing MST for each component C_j in the agentic model, without agents knowing any graph parameter a priori. The time complexity of the algorithm is $O(\Delta \log^2 k + |C_{\max}|(\Delta + \log |C_{\max}|))$ rounds and the memory complexity is $O(\max\{\ell_{\max}, \zeta_{\max}, \Delta, \log(k + \Delta)\} \log(k + \Delta))$ bits per agent.*

Proof. Providing ranks to agents finishes in $O(|C_j|\Delta)$ rounds for each component C_j. The while loop (Algorithm 8) performs two operations – token passing and merging. In token passing, the token is passed through the edge of the tree T_{DFS}, and an edge is not traversed more than twice. Therefore, in a phase, to pass the token from rank-1 agent to rank-$|C_j|$ agent takes $O(|C_j|)$ rounds. From rank-n agent the token returns to the leader again in $O(|C_j|)$ rounds. Combining this with Lemma 17, token passing takes $O(|C_j| \log |C_j|)$ rounds.

In the process of merging, an agent r_u visits at most three types of edges: i) MOE edges within its sub-component C_{r_u} ii) edges traversed to find MOE iii) reversing the edges from r_w until the root of C_{r_w} when it merges with another sub-component C_{r_u} at r_w. In the case of i) MOE is the part of the sub-component C_{r_u}, i.e., a tree. Its traversal finishes in $O(|C_{r_u}|)$ rounds. In

a phase, the combined size of all the sub-components is $O(|C_j|)$. In case ii) edges that are traversed to find the MOE were either part of MOE or not. In case they became part of MOE, they were traversed two times. There are at most $(|C_j| - 1)$ such edges throughout the process. On the other hand, if some edges did not become part of the MOE then they were never traversed again. Therefore, there are in total $|C_j|\Delta - (|C_j| - 1)$ such edges. In case iii) reversing an edge from its merging point to the root can not be more than its sub-component size. Therefore, reversing of edge for agent r_u takes $O(|C_{r_u}|)$. Combining the time for the cases i) and iii) per phase with $O(\log |C_j|)$ phases (Lemma 17), we have total runtime $O(|C_j| \log |C_j|)$ rounds and for case ii) we have total $O(|C_j|\Delta)$ rounds throughout the execution. Thus, the overall round complexity becomes $O(|C_j|\Delta + |C_j| \log |C_j|)$ for MST construction. Combining this with the leader election time complexity of $O((|C_j|+\log^2 k)\Delta)$ (Theorem 12), we have the claimed time bound.

For memory, rank numbering takes $O(\log k)$ bits at each agent to retraverse the DFS tree (constructed during Algorithm 7). Furthermore, each agent stores $O(\log k)$ bits to keep the account of the ID/rank and sub-component rank. Also, there might be a case in which all the neighbors are part of the MST. Therefore, in the worst case, the highest degree (Δ) agent (agent placed at the highest degree node) keeps the account of all the MST edges and requires $O(\Delta \log k)$ memory. Hence, the overall memory per agent in Algorithm 8 is $O(\Delta \log k)$ bits. Combining this time/memory with the time/memory needed for leader election (Theorem 12), we have the claimed memory complexity bound. □

Explicit MST, $k < n$. The MST construction starts after a leader is elected through the explicit leader election algorithm, i.e., after $O(k\Delta)$ rounds. Due to explicit leader election, when MST construction starts, there is a unique global leader in each component C_j. The MST construction at C_j then finishes in $O(|C_j|\Delta + |C_j| \log |C_j|)$ rounds with memory $O(\Delta \log k)$ bits. Therefore, combining time/memory bounds for leader election (Theorem 13), the total time complexity becomes $O(k\Delta + |C_j|\Delta + |C_j| \log |C_j|) = O(k\Delta)$ rounds and the memory complexity becomes $O(\max\{\ell_{\max}, \zeta_{\max}, \Delta\} \log(k + \Delta))$ bits per agent.

Theorem 19 (Explicit MST, $k < n$). *There is an explicit deterministic algorithm for constructing MST for each component C_j in the agentic model, with agents knowing k, Δ a priori. The time complexity of the algorithm is*

$O(k\Delta)$ *rounds and the memory complexity is* $O(\max\{\ell_{\max}, \zeta_{\max}, \Delta\} \log(k + \Delta))$ *bits per agent.*

Explicit MST, $k = n$**.** The MST construction starts after a leader is elected, i.e., after $O(m)$ rounds. Due to explicit leader election, when MST construction starts, there is a unique global leader in the single component C which is G. The MST construction at C then finishes in $O(|C_j|\Delta + |C_j| \log |C_j|) \leq O(m + n \log n)$ rounds with memory $O(\Delta \log n)$ bits; when whole graph is a single component, then $|C_j|\Delta$ is an overestimate on time bound since leader election and MST construction both finish after visiting all edges, i.e., $|C_j|\Delta$ can be replaced with m. Therefore, combining time/memory bounds for leader election (Theorem 14), the total time complexity becomes $O(m + n \log n)$ rounds and the memory complexity becomes $O(\max\{\ell, \zeta, \Delta, \log n\} \log n)$ bits per agent.

Theorem 20 (Explicit MST, $k = n$**).** *There is an explicit deterministic algorithm for constructing MST of* G *in the agentic model when* $k = n$*, without agents knowing any graph parameter a priori. The time complexity of the algorithm is* $O(m + n \log n)$ *rounds and the memory complexity is* $O(\max\{\ell, \zeta, \Delta, \log n\} \log n)$ *bits per agent.*

5. Concluding Remarks

We have studied two fundamental distributed graph problems, leader election, and MST, in the agentic model. The agentic model poses unique challenges compared to the well-studied message-passing model. We provided stabilizing as well as explicit deterministic algorithms for leader election when $k < n$. The stabilizing algorithms do not require agents to know graph parameters but overtaking of a leader may occur until a certain time. The explicit algorithms avoid overtaking but agents require knowledge of k, Δ. For the case of $k = n$, we showed that the algorithm for leader election can be made explicit (no overtaking) without knowing k, Δ. We then developed respective algorithms for MST for both $k < n$ and $k = n$. To the best of our knowledge, both problems were studied for $k < n$ for the very first time in this paper.

For future work, it would be interesting to improve the time and/or memory complexities of our leader election and MST algorithms. It would be interesting to see where overtaking can be avoided without parameter knowledge for $k < n$. It would also be interesting to solve other fundamental

distributed graph problems, such as maximal independent set, maximal dominating set, coloring, maximal matching, and minimum cut, in the agentic model for $k \leq n$. Some fundamental problems, such as the maximal independent set (MIS) and maximal dominating set (MDS), have been considered in the agentic model for $k = n$ [22, 23]. The developed algorithms gather the robots to a single node after leader election and then present a technique to solve the problems. For the case of $k < n$, this idea does not extend well since some components may still be in leader election phase and some components may progress toward computing MIS and MDS and this creates a challenge on how to synchronize the agents that are in different phases. Finally, it would also be interesting to consider agent faults (crash or Byzantine) and be able to solve graph level tasks with non-faulty agents.

References

[1] A. D. Kshemkalyani, M. Kumar, A. R. Molla, G. Sharma, Brief announcement: Agent-based leader election, mst, and beyond, in: DISC, Vol. 319 of LIPIcs, 2024, pp. 50:1–50:7. doi:10.4230/LIPICS.DISC.2024.50.

[2] Y. Cong, g. Changjun, T. Zhang, Y. Gao, Underwater robot sensing technology: A survey, Fundamental Research 1 (03 2021). doi: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, Mathematical Problems in Engineering 2018 (2018) 1–9. doi: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, Journal of Computational Science 29 (09 2018). doi: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 perspectives & innovations : EP+I 9 (2012) 1. doi:10.1186/1742-5573-9-1.

[6] G. Pandurangan, P. Robinson, M. Scquizzato, On the distributed complexity of large-scale graph computations, in: C. Scheideler, J. T. Fineman (Eds.), SPAA, ACM, 2018, pp. 405–414. doi:10.1145/3210377.3210409.
URL https://doi.org/10.1145/3210377.3210409

[7] A. D. Kshemkalyani, M. Kumar, A. R. Molla, G. Sharma, Agent-based mst construction, in: arXiv, 2024. arXiv:2403.13716.

[8] A. D. Kshemkalyani, M. Kumar, A. R. Molla, G. Sharma, Near-linear time leader election in multiagent networks, in: AAMAS, AAMAS '25, 2025, p. 1218–1226.

[9] A. D. Kshemkalyani, M. Kumar, A. R. Molla, G. Sharma, Faster leader election and its applications for mobile agents with parameter advice, in: ICDCIT.

[10] J. Augustine, W. K. M. Jr., Dispersion of mobile robots: A study of memory-time trade-offs, in: ICDCN, 2018, pp. 1:1–1:10.

[11] 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. doi:10.1109/SFCS.1993.366821.
URL https://doi.org/10.1109/SFCS.1993.366821

[12] D. Peleg, Time-optimal leader election in general networks, J. Parallel Distrib. Comput. 8 (1) (1990) 96–99. doi:10.1016/0743-7315(90)90074-Y.
URL https://doi.org/10.1016/0743-7315(90)90074-Y

[13] A. D. Kshemkalyani, M. Kumar, A. R. Molla, D. Pattanayak, G. Sharma, Dispersion is (almost) optimal under (a)synchrony, in: SPAA, 2025.

[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. doi:10.1145/357195.357200.
URL 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. doi:10.1145/28395.28421.
URL https://doi.org/10.1145/28395.28421

[17] 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.

[18] S. Kutten, D. Peleg, Fast distributed construction of small k-dominating sets and applications, J. Algorithms 28 (1) (1998) 40–66. doi:10.1006/jagm.1998.0929.
URL https://doi.org/10.1006/jagm.1998.0929

[19] 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. doi:10.1137/S0097539700369740.
URL https://doi.org/10.1137/S0097539700369740

[20] A. R. Molla, K. Mondal, W. K. M. 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.

[21] A. Ta-Shma, U. Zwick, Deterministic rendezvous, treasure hunts, and strongly universal exploration sequences, ACM Trans. Algorithms 10 (3) (2014) 12:1–12:15. doi:10.1145/2601068.
URL https://doi.org/10.1145/2601068

[22] D. Pattanayak, S. Bhagat, S. G. Chaudhuri, A. R. Molla, Maximal independet set via mobile agents, in: ICDCN, ACM, 2024, pp. 74–83.

[23] P. K. Chand, A. R. Molla, S. Sivasubramaniam, Run for cover: Dominating set via mobile agents, in: ALGOWIN, Springer, 2023, pp. 133–150.