





# Smoothed Analysis of Leader Election in Distributed Networks

Anisur Rahaman Molla<sup>1</sup>  and Disha Shur<sup>2</sup> 

<sup>1</sup> Computer and Communication Sciences Division,  
Indian Statistical Institute, Kolkata, Kolkata, India  
molla@isical.ac.in

<sup>2</sup> R. C. Bose Centre for Cryptology and Security,  
Indian Statistical Institute, Kolkata, Kolkata, India  
disha.shur@gmail.com

**Abstract.** We study smoothed analysis of the leader election (LE) problem in distributed networks. Smoothed analysis is a hybrid between worst-case analysis and average-case analysis. It takes a worst-case instance for the algorithm and perturbs the input by adding some random noise and analyzes the algorithm on this perturbed input. We consider smoothed analysis, in which the topology of the input graph  $G$  is randomly perturbed by adding random edges to  $G$ . The complexity of the algorithm is parameterized by a smoothing parameter  $0 \leq \epsilon(n) \leq 1$  which controls the amount of random edges to be added to the input graph  $G$  per round, where  $\epsilon$  is a small function of  $n$ , e.g.,  $n^{-4}$  ( $n$  is the number of nodes in the graph  $G$ ). Informally,  $\epsilon$  is the probability that a random edge can be added to a node per round.

We analyze the time and message complexity of leader election in the above smoothing model. We present the following three results in synchronous *CONGEST* distributed model:

- (i) A simple randomized algorithm that elects a leader with high probability (With high probability (or w.h.p. in short) means with probability  $\geq 1 - 1/n$ .) in  $O((\log n)/\epsilon)$  rounds and uses  $O(\sqrt{n} \log^{2.5} n)$  messages. Note that both the time and the message bounds are optimal (up to a polylog  $n$  factor).
- (ii) A time-improved randomized algorithm that elects a leader with high probability in  $O\left(\frac{\log n}{\sqrt{\epsilon}}\right)$  rounds, but uses  $O(m + n \log n)$  messages, where  $m$  is the number of edges in the input graph  $G$ .
- (iii) A deterministic algorithm (except the randomized smoothing part) which solves leader election in  $O\left(\frac{\log^2 n}{\sqrt{\epsilon}}\right)$  rounds and incurs  $O(m + n\sqrt{\epsilon} \log^2 n)$  messages.

Our work extends the study of smoothed analysis of distributed problems one step further, an open direction raised by [7].

**Keywords:** Distributed algorithms · Smoothed analysis · Random model · Leader election

The work is supported, in part, by a project on IoT Security, funded by Govt. of India at R. C. Bose Centre for Cryptology and Security, Kolkata, India.

## 1 Introduction

Motivated by the work of Dinitz et al. [8], the smoothed analysis of distributed algorithms is first formally modeled by Chatterjee et al. [7] and studied for the minimum spanning tree (MST) problem. In this paper, we extend the study of smoothed analysis of distributed problems, an open direction raised by [7], by considering the smoothed analysis of leader election problem. Leader election is one of the fundamental and well studied problem in the field of distributed computing. It outlines the problem of electing a particular node in a network as the leader. A version of this problem requires only the leader node to be aware of its status. All the nodes other than the leader are simply aware that they are not the leader. They need not be aware the leader's identity. This version is called the *implicit* leader election. The *explicit* version of the leader election problem requires all the nodes in the network to be aware of the identity of the leader. The widespread application of the leader election can be found in many domains, e.g., sensor networks [29], IoT networks [26], grid computing [2], peer-to-peer networks [21, 27] and cloud computing [32]. In IoT networks, a leader node performs crucial tasks such as gathering information, coordinate tasks among the nodes, generating encryption-decryption keys etc. [14, 26].

We consider the same smoothing model as defined in the paper [7] to analyze distributed algorithms. In particular, we consider smoothed analysis, in which the topology of the input graph  $G$  is randomly perturbed by adding random edges to  $G$ . The perturbation is determined by a smoothing parameter  $0 \leq \epsilon(n) \leq 1$  which controls the amount of random edges to be added to the input graph  $G$  per round. Typically  $\epsilon$  is a small function of  $n$ , e.g.,  $n^{-4}$ , where  $n$  is the number of nodes in the graph  $G$ . Smoothed edges are used for communication only. The study of the smoothing analysis investigates how these additional smoothed edges can be exploited to improve the time and message complexity of a distributed algorithm.

Kutten et al. [20] studied the (implicit) leader election problem in both complete networks and general networks. They presented an algorithm that takes in  $O(1)$  time and uses only  $O(\sqrt{n} \log^{3/2} n)$  messages to elect a leader in the complete graph. For the general graphs, they extend this algorithm which takes  $O(\tau)$  time and  $O(\tau \sqrt{n} \log^{3/2} n)$  messages, where  $\tau$  is the mixing time of graph. The mixing time of a graph could be as large as  $O(n^3)$  [23]. This algorithm requires to know the mixing time to be known as input. A major improvement was introduced in [12] where the same problem has been solved without any knowledge of the mixing time of the graph. All these works build on a technique of sampling smaller set of nodes (via random walks) and compute leader in the sampled set. This is a standard technique that is useful for reducing the message complexity. We also use the same idea in our randomized algorithms. The algorithms of [12, 20] analyze the worst case time and message complexities. A smoothed analysis of the same problem is considered in this paper where the objective is to analyze both the time and message complexities of the algorithm. Smoothed analysis can be viewed as a hybrid between worst-case analysis and the average-case analysis. It takes a worst-case instance for the algorithm and

perturbs the input by adding some random noise and analyzes the algorithm on this perturbed input.

The paper [7] analyzed only the time complexity of the MST problem. In this paper, we analyze both the time and the message complexity of the leader election problem. Similar to [7], we assume that the nodes of the input graph are able to distinguish between the smoothed edges and the original graph edges. We present a simple algorithm that solves the problem with high probability in  $O(\frac{\log n}{\epsilon})$  rounds using  $O(\sqrt{n} \log^{5.2} n)$  messages. Then we present an improvement algorithm that takes  $O(\frac{\log n}{\sqrt{\epsilon}})$  rounds and uses  $O(m + n \log n)$  messages. We further present a deterministic algorithm that solves leader election in  $O(\frac{\log^2 n}{\sqrt{\epsilon}})$  rounds and incurs  $O(m + n\sqrt{\epsilon} \log^2 n)$  messages. The algorithm is *deterministic* in the sense that except the randomized smoothing part, all other parts are deterministic. Note that one can directly solve leader election in general graphs in  $O(D \log n)$  rounds and  $O(m \log n)$  messages deterministically using algorithm in [19] or using a randomized algorithm in time  $\tilde{O}(\tau)$  and  $\tilde{O}(\tau\sqrt{n})$  messages [12, 20], but the diameter  $D$  or mixing time  $\tau$  of a graph could be large ( $\tilde{O}$  hides a polylog  $n$  factor).

## 1.1 Model

*Distributed Network Model.* We model the communication network as an undirected, unweighted, connected graph  $G = (V, E)$ , where  $|V| = n$ , and  $|E| = m$ . Every node has limited initial knowledge. We assume an anonymous network, i.e., nodes do not know their neighbors. We assume that nodes are associated with a distinct identity number (e.g., its IP address). If not, then each node can randomly pick a number in the range  $[1, n^4]$  such that the numbers are distinct for all the nodes. The random number can be used as the ID of the nodes. The node may also accept some additional inputs as specified by the problem at hand. The nodes are allowed to communicate through the edges of the graph  $G$ . The communication occurs in synchronous rounds. In one round, nodes can send messages, receive messages (from neighbors) and perform some local computation. Our algorithms use only small-sized messages. In particular, in each round, each node sends a message of size  $O(\log n)$  through its adjacent edges. This is a widely used standard model known as the *CONGEST* model of distributed computing [25], and captures the bandwidth constraints inherent in real-world computer networks.

*Smoothing Model.* There are many smoothing models exists in sequential settings, see [7, 8, 31]. We consider the smoothing model defined by Chatterjee et al. [7]. The smoothing model of [7] is appropriate to analyze distributed network algorithms.

Given an arbitrary graph  $G$ , smoothing allows to introduce some “perturbance” to the input graph. In particular, the smoothing model allows adding some random edges, determined by a smoothing parameter  $\epsilon$ , to the input graph  $G$ . The smoothing parameter  $0 \leq \epsilon = \epsilon(n) \leq 1$  is a function of  $n$ , which controls

the amount of random edges to be added in the graph per round (typically  $\epsilon$  is a small function of  $n$ , e.g.,  $n^{-4}$ ). Thereby the graph structure is altered. This model is called the  $\epsilon$ -smoothing model [7]. More precisely, in every (smoothing) round, every node add an edge with probability  $\epsilon$  to a randomly chosen node from  $V$  in the given graph  $G$ . The added edges are called smoothed edges. In case of multiple edges being present between two nodes, unless specified, only one smoothed edge is used for communication. The added edges persists in the network and can be used for communication in the later rounds. Let the induced graph formed by the random edges be  $R(G) = R(V, S)$ , where  $S$  is the set of only the smoothed edges.  $R(G)$  is called smoothed graph.

As noted in [7], one can view the smoothing model as a generalization of the congested clique model. Suppose the graph  $G$  is embedded in a congested clique. A node, besides using its incident edges in  $E$ , can also choose to use a random edge in the clique (not in  $G$ ) with probability  $\epsilon$  in a round to communicate (once chosen, a random edge can be used subsequently till the end of computation). Thus, the smoothed edges are the clique edges.

Smoothed edges are used for faster communication. The study of the smoothing analysis investigates how these additional smoothed edges can be exploited to improve the time and message complexity of a distributed algorithm. In this paper, we consider only addition of edges to the input graph; while a smoothing model also allows deletion of edges from the graph.

A formal definition of the leader election problem.

**Definition 1 (Leader Election).** *Every node  $u$  maintains a variable  $status_u$  that it can set to a value in  $\{\perp, NON-ELECTED, ELECTED\}$ ; initially  $status_u = \perp$  for all  $u$ . An algorithm  $A$  solves leader election in  $T$  rounds if, from round  $T$  on, exactly one node has its status set to  $ELECTED$  while all other nodes are in state  $NON-ELECTED$ . This is the requirement for standard (implicit) leader election.*

Note that the implicit leader election algorithm can be converted to an *explicit* leader election (where every node knows the ID of the leader) by simply the leader sends its ID to all the nodes.

*Paper Organization.* Section 2 discusses related works. Section 3 contains smoothed analysis of leader election algorithm. We first present a simple optimal time and message complexity leader election algorithm in Sect. 3.1. Then an improved time algorithm in Sect. 3.2 and a deterministic smoothed analysis in Sect. 3.3. Finally, we conclude in Sect. 4.

## 2 Related Work

Leader election is one of the fundamental problem in distributed networks. It has been extensively studied in various models and settings, starting from its introduction by Le Lann [22] in ring network and then by a seminal work of Gallager-Humblet-Spira [11] in general graphs. The problem is well studied in

complete network itself [1, 13, 16–18, 30], and reference there in, and also in general networks [12, 15, 19].

Some closely related leader election works in the classical congest model are [5, 12, 19, 20]. Kutten et al. [20] studies the (implicit) leader election problem in both complete networks and general networks and showed efficient time and message bounds of leader election algorithms. In particular, they presented an algorithm that executes in  $O(1)$  time and uses only  $O(\sqrt{n} \log^{3/2} n)$  messages to elect a leader in a complete graph. They also showed an almost matching lower bound for randomized leader election. The standard technique of sampling smaller set of nodes and compute leader among themselves is further used in general graphs and achieve  $\tilde{O}(\tau)$ -time and  $\tilde{O}(\tau\sqrt{n})$ -message complexity leader election algorithm [12, 20], where  $\tau$  is the mixing time of a graph ( $\tilde{O}$  hides a polylog  $n$  factor). We also use this sampling techniques in the randomized algorithms. Several tight results are shown in general graphs in [19], including a notable deterministic algorithm which solves leader election in  $O(D \log n)$  rounds and  $O(m \log n)$  messages. We adapt this deterministic algorithm in our deterministic smoothing analysis of leader election. The our paper is inspired mainly by the works of [12, 19, 20].

Smoothed algorithms have been explored in [28] as the next step in bridging the gap between the “theoretical predictions and empirical observation” in their performances. Since its introduction in [31], analyses have shown the practical running time of algorithms to be closer to their smoothed complexities than the worst-case ones. Smoothed analysis of some popular graph problems have been explored in [3, 9, 10] and [4]. The first smoothed analysis of distributed algorithms, to the best of our knowledge, has been conducted in [8] where in they study the robustness of their algorithm for dynamic networks. They analyse three problems, namely random walks, flooding and aggregation and their upper and lower bounds on dynamic networks. Their smoothing procedure consists of addition as well as deletion of edges from the evolving graph. Robustness is measured by monitoring the change in bounds with the magnitude of smoothing introduced in the model.

### 3 Leader Election in the Smoothing Model

Given an arbitrary graph  $G = (V, E)$  and CONGEST model of communication, the goal is to compute a leader in the  $\epsilon$ -smoothing model of  $G$ . We first present a simple algorithm that solves the leader election problem in  $O(\log n/\epsilon)$  rounds using  $O(\sqrt{n} \log^{2.5} n)$  messages in the  $\epsilon$ -smoothing model. The message complexity is optimal (up to a polylog  $n$  factor) and the time complexity is also optimal (up to a polylog  $n$  factor) for any  $\epsilon \geq 1/\text{polylog } n$ . Then we present a time improved algorithm which solves leader election in  $O(\log n/\sqrt{\epsilon})$  rounds, but uses  $O(m + n \log n)$  messages. While these two algorithms are randomized, we present a deterministic algorithm that takes  $O(\log^2 n/\sqrt{\epsilon})$  rounds and  $O(m + n\sqrt{\epsilon} \log^2 n)$  messages to elect a leader. The deterministic algorithm builds on the similar idea of the improved algorithm. In all the algorithms, we assume

that the nodes can distinguish between original edges in the given graph and the smoothed edges.

### 3.1 Simple and Efficient Algorithm

We present a simple, yet efficient algorithm for leader election in the  $\epsilon$ -smoothing model. At a high-level, the algorithm consists of two parts: (I) Smoothing, i.e., adding random edges to the given graph in such a way that the smoothed graph becomes an expander, (II) Compute a leader in the smoothed graph. Note that the nodes are fixed; so the elected leader is a valid leader in the given graph. The smoothed edges are used for the communication only.

**(I) Constructing Smoothed Graph (A Random Expander Graph).** In the beginning, the algorithm adds  $O(\log n)$  random edges per node according to the smoothing model. In particular, the algorithm runs the smoothing procedure for  $\Theta(\log n/\epsilon)$  rounds. In every round, each node adds a smoothed edge with probability  $\epsilon$  to one of the nodes selected uniformly at random. Thus, it is easy to show that with high probability a node will add  $\Theta(\log n)$  random edges (smoothed edges). Let us call this graph as the smoothed graph  $R(G) = (V, F)$  which is induced only by the smoothed edges  $F$  after  $\Theta(\log n/\epsilon)$  rounds. It is intuitive that  $R(G)$  is an Erdős-Rényi random graph (expander), which is formally shown in [7] and gives the following lemma.

**Lemma 1 (Lemma 3.1, [7]).** *The smoothed graph  $R(G)$  has a constant conductance and  $O(\log n)$  mixing time.*

**(II) Computing a Leader.** The second part of the algorithm computes leader among the  $n$  nodes. For this, the algorithm uses  $R(G)$  as the communication graph. Our goal is to compute a leader using minimum number of messages and time. For this part, we adapt the leader election approach of [20] for general graphs. If the mixing time  $\tau$  of a graph is known, then an algorithm in [20] computes a leader with high probability in  $O(\tau)$  rounds and uses  $O(\tau\sqrt{n}\log^{1.5}n)$  messages. In essence, we adapt this algorithm in the smoothed graph  $R(G)$  since the mixing time  $O(\log n)$  of  $R(G)$  is known.

Let us discuss an outline of the algorithm. Recall that we consider anonymous network, i.e., nodes do not know each other's ID. We assume that nodes have unique IDs; otherwise each node can randomly pick a number (or rank) in the range  $[1, n^4]$  such that the numbers are distinct for all the nodes. The rank can be used as the ID of the nodes. The idea of the algorithm is to select a smaller committee nodes (called *candidate nodes*) which are responsible for electing a leader node among themselves. In fact, the maximum ID node among the candidate nodes will be elected as the leader and all other nodes put themselves in the NON-ELECTED state. This is a standard technique to reduce the message complexity. A random set of candidate nodes of size  $\Theta(\log n)$  is selected. For this, each node selects itself with probability  $O(\log n/n)$  to become a candidate node. This selection is done locally at each node and hence the candidate

nodes do not know each other initially. All the non-candidate nodes put themselves in the NON-ELECTED state. The candidate nodes communicate among themselves via some other nodes, called *referee nodes*. For this, each candidate node samples  $2\sqrt{n \log n}$  random referee nodes in the network. This referee sampling is done via performing random walks on  $R(G)$  by token forwarding. Each candidate node creates  $2\sqrt{n \log n}$  random walk tokens and each token performs random walk of length  $O(\log n)$  on the smoothed graph  $R(G)$ . Since the mixing time of  $R(G)$  is  $O(\log n)$ , the tokens stop at random nodes after  $O(\log n)$  steps. The ending nodes of the tokens after  $O(\log n)$  steps act as the referee nodes. In this way each candidate node samples  $2\sqrt{n \log n}$  random referee nodes. The reason behind sampling so many referee nodes is to make sure at least one common referee node between any pair of candidate nodes' referees. Each candidate node sends its ID with the random walk tokens. An intermediate node or referee node may receive IDs (or tokens) from multiple candidate nodes. Since our goal is to elect the maximum ID node to be the leader, the referee nodes send back a *winner* message  $\langle \text{"WIN"}, node\_id, count \rangle$  to the maximum ID candidate node only, via back tracking the random walk paths. In the winner message, *node\_id* carries the maximum ID of the candidate node and *count* variable stores the number of tokens having the maximum ID. During the token forwarding process, an intermediate node forwards only the tokens with maximum ID among all the tokens received in a round (and discards all other tokens with smaller ID). The intermediate node also stores this ID and the port numbers, through which it has received the maximum ID token, for back tracking. In subsequent rounds, if an intermediate node receives any token with higher ID than the previous one, then it updates its stored ID and the port number accordingly.

During back tracking, an intermediate node on receiving multiple winner messages  $\langle \text{"WIN"}, node\_id, count \rangle$ , adds up the *count* for the maximum *node\_id* and forwards to the back track node (it discards the winner messages with lower *node\_id*). When back track finishes, each candidate node sums up the *count* in the winner messages it has received. The candidate node which receives  $2\sqrt{n \log n}$  winner messages enters the ELECTED state. All the other candidate nodes enter the NON-ELECTED state. It is easy to see that only the maximum ID candidate node receives  $2\sqrt{n \log n}$  winner messages and elected as the leader.

One difficult part in the algorithm is the congestion over the edges when performing many random walks in parallel as we consider CONGEST model. The congestion is handled by sending only the count of random walk tokens that need to be sent by a particular candidate node, and not the tokens themselves.

To show the correctness of the algorithm, we first discuss the following two results. The first one says that there is at least one candidate node with high probability. Also the size of the candidate nodes is not too large— bounded by  $O(\log n)$ . The second one says that there is at least one common referee node between any pair of candidate nodes with high probability.

**Lemma 2.** *The size of the candidate nodes is  $\Theta(\log n)$  with high probability.*

*Proof.* Each node selects itself with probability  $O(\log n/n)$  to become a candidate node. Thus, in expectation,  $O(\log n)$  candidate nodes are selected. Then

one can show using a standard Chernoff bound that the number of the selected candidate nodes is  $\Theta(\log n)$  with high probability.  $\square$

**Lemma 3 (Theorem 1, [20]).** *With high probability, there is a common referee node between any pair of candidate nodes.*

This implies that the maximum ID candidate node has a common referee node with all other candidate nodes. Thus, those common referees generate the winner message for the maximum ID candidate node only and discards all other random walk tokens. Further, during the token forwarding procedure, the maximum ID candidate node's tokens dominate all other tokens. This means in subsequent rounds when the winner messages reach their respective candidate nodes, no candidate node, other than the one having maximum ID, would have received all  $2\sqrt{n} \log n$  winner messages with high probability. Therefore, only the candidate node with the maximum ID enters the ELECTED state with a high probability.

Thus we get the following result.

**Theorem 1.** *Given an anonymous graph  $G = (V, E)$  in the  $\epsilon$ -smoothing model, there exists a randomized distributed algorithm that computes a leader in  $G$  with high probability in  $O(\frac{\log n}{\epsilon})$  rounds and incurs  $O(\sqrt{n} \log^{2.5} n)$  messages (assuming that the smoothed edges are added without sending any messages).*

*Proof.* We already discussed that the algorithm correctly elects a leader with high probability.

The time complexity of the algorithm is determined by two procedures: (I) Constructing smoothed graph— which takes  $O(\log n/\epsilon)$  rounds. (II) Computing leader in the smoothed graph, which requires to perform random walks of length  $O(\log n)$  and back tracking (in parallel). Further, there is no congestion due to performing multiple random walks in parallel as we are sending the token counts and not the tokens themselves. All other computations are done locally. This procedure takes  $O(\log n)$  rounds. Thus, the time complexity of the algorithm is  $O(\log n/\epsilon + \log n) = O(\log n/\epsilon)$  rounds.

The message complexity of the algorithm is determined by the leader election procedure in the smoothed graph  $R(G)$ . The number of the candidate nodes is  $\Theta(\log n)$  with high probability. Each candidate node performs  $2\sqrt{n} \log n$  random walks for  $O(\log n)$  steps. Thus a total of  $O(\sqrt{n} \log^{5/2} n)$  messages are required for this step. The back tracking of the winner messages may take at most the same number of messages. Therefore message complexity of the algorithm is  $O(\sqrt{n} \log^{2.5} n)$ .  $\square$

*Remark 1.* In the above theorem, we assume that the smoothed edges are added by the system without sending any messages. If we consider one message is used per edge addition, then the message complexity of the smoothing process would be  $O(n \log n)$ , as each node adds  $O(\log n)$  random edges. Then the message complexity of the algorithm would be  $O(n \log n)$ .



### 3.2 An Improved Algorithm

Now we present an improved algorithm which is a variant of the previous algorithm and has a lower time complexity. This algorithm solves the leader election in  $O(\log n/\sqrt{\epsilon})$  rounds. It crucially applies the previous algorithm over a super-graph induced by minimum-spanning-tree (MST) fragments as the super nodes. Broadly, this algorithm consists of three parts: (I) Compute MST fragments. (II) Apply smoothing to add an expander over the super-graph induced by the MST fragments. (III) Compute a leader using the previous random walk based sampling algorithm on the smoothed super-graph.

Let us describe the algorithm and simultaneously its analysis.

**(I) Computing MST Fragments.** We use controlled Gallagher-Humblet-Spira (GHS) algorithm to construct MST fragments, see Sect. 7.4 in [24]. The main difference compared to the standard GHS algorithm is that the growth (size, diameter) of fragments are controlled during merging of fragments.

The controlled GHS algorithm runs in phases [24]. The algorithm starts with each individual node as a fragment and merges fragments in each phase. In each phase, the algorithm maintains the following invariant: Each MST fragment has a leader (which is the root of the tree) and all nodes know their respective parents and children. Initially, each node (a singleton fragment) is a leader node; subsequently each fragment will have one leader (root) node. Each fragment is identified by the identifier of its root (called the fragment ID) and each node in the fragment knows its fragment ID. Each fragment's operation is coordinated by the respective fragment's leader. Each phase consists of two major operations: (1) Finding minimum-weight-outgoing-edge (MOE) of all fragments and (2) Merging fragments via their MOEs. Note that, while the controlled GHS finds an MST in a weighted graph (where edges have weight), it also works on an unweighted graph where one can consider the edge-weights are 1. The details on finding MOE and merging fragments can be found in Sect. 7.3.1 of [24].

The algorithm starts by running a controlled GHS algorithm for  $\log(\frac{1}{\sqrt{\epsilon}})$  phases ( $\epsilon$  is the smoothing parameter). The size and diameter of each MST fragment are  $\Omega(1/\sqrt{\epsilon})$  and  $O(1/\sqrt{\epsilon})$  respectively, and there will be  $O(n\sqrt{\epsilon})$  such fragments ([24], Sect. 7.4). Each fragment will be treated as a super-node. Each node, that is not a root node, maintains two IDs: 1. Its fragment ID, denoted by FID, and 2. Its own ID in the given graph which we denote by GID in this section. For a root node, its FID would be same as its GID.

The following lemma shows that this MST fragments construction takes  $O(\frac{\log^* n}{\sqrt{\epsilon}})$  rounds and uses  $O(m)$  messages.

**Lemma 4.** *The number of MSTs formed after  $\log(\frac{1}{\sqrt{\epsilon}})$  phases is  $O(n\sqrt{\epsilon})$ . The MST fragments construction takes  $O(\frac{\log^* n}{\sqrt{\epsilon}})$  rounds and  $O(m)$  messages, where  $m$  is the number of edges in the given graph.*

*Proof.* It is shown in Corollary 7.1 of [24] that the number of MST fragments at the beginning of phase  $i$  is at most  $\frac{n}{2^i}$ . So after  $\log(\frac{1}{\sqrt{\epsilon}})$  phases, the number of

MSTs is at most  $n/2^{\log(\frac{1}{\sqrt{\epsilon}})} = O(n\sqrt{\epsilon})$ . It also follows from Lemma 7.3 of [24] that the diameter of each MST at the beginning of phase  $i$  is bounded by  $2^i$ . Hence the diameter of the MST fragments after  $\log(\frac{1}{\sqrt{\epsilon}})$  phases is  $O(2^{\log(\frac{1}{\sqrt{\epsilon}})}) = O(1/\sqrt{\epsilon})$ .

The time and message complexities follows from the 3 sub-procedures.

1. Finding MOE from each  $i^{\text{th}}$  node in parallel takes  $O(2^i)$  rounds for the  $i^{\text{th}}$  phase. So for the  $\log(\frac{1}{\sqrt{\epsilon}})$  phases, it takes:  $O\left(\sum_{i=1}^{\log(\frac{1}{\sqrt{\epsilon}})} 2^i\right) = O\left(\frac{2}{\sqrt{\epsilon}}(1 - \sqrt{\epsilon})\right) = O\left(\frac{1}{\sqrt{\epsilon}}\right)$  rounds.  
The number of messages required is:  $O\left(\sum_{v \in V} 2 \cdot d(v) + \sum_{i=1}^{\log(\frac{1}{\sqrt{\epsilon}})} \sum_{v \in V} O(1)\right) = O\left(m + n \log\left(\frac{1}{\sqrt{\epsilon}}\right)\right)$ .
2. Selecting MOE for merging fragments takes  $O(2^i \log^* n)$  rounds and  $O(n \log^* n)$  messages per phase. Thus in total it takes,  $O\left(\sum_{i=1}^{\log(\frac{1}{\sqrt{\epsilon}})} (2^i \log^* n)\right) = O\left(\frac{\log^* n}{\sqrt{\epsilon}}\right)$  rounds and  $O\left(n \log^* n \log\left(\frac{1}{\sqrt{\epsilon}}\right)\right)$  messages.
3. Merging fragments takes  $O(2^i)$  rounds and  $O(n)$  messages per phase. So in total it takes,  $O\left(\frac{1}{\sqrt{\epsilon}}\right)$  rounds and  $O\left(n \log\left(\frac{1}{\sqrt{\epsilon}}\right)\right)$  messages.

Thus, MST fragments construction takes  $O\left(\frac{\log^* n}{\sqrt{\epsilon}}\right)$  rounds and  $O(m)$  messages.  $\square$

**(II) Constructing a Smoothed Graph.** After constructing the MST forest, perform  $O\left(\frac{\log n}{\sqrt{\epsilon}}\right)$  rounds of smoothing. Let  $S$  denotes the set of smoothed edges added in the graph. The probability that a smoothed edge added between two nodes in  $G$  is  $\Theta\left(\frac{\sqrt{\epsilon} \log n}{n}\right)$ . Consider each MST fragment as a super-node. Let the set of super-node be  $V'$ ; then  $|V'| = O(n\sqrt{\epsilon})$  as there are so many MST fragments. Let  $S' \subseteq S$  be the set of inter-super-node smoothed edges. Consider the super-graph  $R'(V', S')$ ; we call it as smoothed super-graph. Then it is easy to show that  $R'(V', S')$  is an Erdős-Rényi random graph [7]. Thus, the mixing time of  $R'(V', S')$  is  $O(\log(n\sqrt{\epsilon})) = O(\log n)$ .

**(III) Computing a Leader.** Now we apply the similar random walk based approach on the smoothed super-graph  $R'(V', S')$  to elect a leader in  $G$ . In particular, we run the ‘‘Computing a leader’’ procedure of the previous algorithm over this super-graph  $R'(V', S')$ , where the root node in each super-node simulates and coordinates the tasks of a node there (recall that each MST fragment has a specified root node). First,  $O(\log n)$  random set of candidate super-nodes are selected. For this, each super-node selects itself with probability  $\Theta\left(\frac{\log n}{n\sqrt{\epsilon}}\right)$ . Note that when we say a ‘‘super-node does something’’, it means the root node inside the super-node does this and communicate to all the nodes in the fragment. To implement this in a super-node, the root node coordinates the tasks with the fragment nodes. Thus, an extra term of  $O(1/\sqrt{\epsilon})$  rounds may be incurred due to the communication within a super-node. This is because, the diameter of each MST

fragment is  $O(1/\sqrt{\epsilon})$ , so the communication within a fragment takes  $O(1/\sqrt{\epsilon})$  rounds. Further recall that each super-node has an ID, the fragment ID which is essentially the ID of the root node in the fragment. Then in the similar way, each candidate super-node samples  $O(\sqrt{n \log n})$  referee nodes (super-nodes)<sup>1</sup> by performing  $O(\sqrt{n \log n})$  random walks of length  $O(\log n)$  (since the mixing time of  $R'$  is  $O(\log n)$ ). Then the referee nodes send back winner message to the maximum ID super-node. Finally, the super-node with maximum ID becomes the leader. Since the super-node carries the ID of its root node, the root node in the maximum ID super-node becomes the leader in  $G$ .

One crucial part remains to discuss is— how the super-nodes perform multiple random walks on  $R'(V', S')$  in parallel. Let  $r$  be the root node in a super-node  $T$ . In the beginning of this procedure – “computing a leader” – every node in  $T$  sends the number of its inter-super-node smoothed edges (i.e., outgoing smoothed edges) to the root  $r$ . The root node computes the total number of outgoing smoothed edges from  $T$  and also stores the ID of the fragment nodes and their outgoing edge number. Suppose a super-node  $T$  has  $k$  random walk tokens to forward in the next round. For each token,  $r$  (locally) selects one outgoing smoothed edge randomly among all the outgoing edges of  $T$ . Then  $r$  sends a count of the number of tokens to the corresponding fragment nodes. A count  $\ell$  of a fragment node  $v$  indicates that  $\ell$  number of random walk tokens are selected to move over the outgoing edges of the node  $v$ . The root sends this random walk count information to all the selected nodes in  $T$  in parallel. Then the fragments nodes forward their tokens to the outgoing neighbors selected uniformly at random (among the outgoing edges). The node forwards the tokens together with the count through its corresponding outgoing smoothed edges to avoid any congestion. When a node receives random walk tokens from a different super-node, it sends the count (of the tokens) to the root node of its super-node. The root node sums up the count to know the total number of received tokens from the same candidate node. Then it forwards the tokens, in the same way as described above, for the next step.

**Theorem 2.** *Given an anonymous graph  $G = (V, E)$  in the  $\epsilon$ -smoothing model, there exists a randomized distributed algorithm that computes a leader in  $G$  with high probability in  $O(\frac{\log n}{\sqrt{\epsilon}})$  rounds and incurs  $O(m + n \log n)$  messages.*

*Proof.* The algorithm correctly elects a leader as the previous algorithm does.

**Time Complexity.** The first procedure (‘computing MST fragments’) takes  $O(\frac{\log^* n}{\sqrt{\epsilon}})$  rounds (follows from Lemma 4). The second procedure (‘constructing smoothed graph’) takes  $O(\frac{\log n}{\sqrt{\epsilon}})$  rounds as we apply smoothing for so many rounds. The third procedure (‘computing a leader’) takes  $O(\frac{\log n}{\sqrt{\epsilon}})$  rounds. This is because, the random walks are performed over the super-nodes in parallel for  $O(\log n)$  rounds and one step of the random walk may take extra  $\frac{1}{\sqrt{\epsilon}}$

<sup>1</sup> Since  $|V'| = O(n\sqrt{\epsilon})$ , it would suffice to sample  $O(\sqrt{n\sqrt{\epsilon} \log n})$  referee nodes to ensure a common referee node between any pair of candidate super-nodes.

rounds for communication inside a super-node. In the calculation, we implicitly assume  $O(\log(n\sqrt{\epsilon})) = O(\log n)$ . Therefore, the time complexity the algorithm is:  $O(\frac{\log^* n}{\sqrt{\epsilon}} + \frac{\log n}{\sqrt{\epsilon}} + \frac{\log n}{\sqrt{\epsilon}}) = O(\frac{\log n}{\sqrt{\epsilon}})$  rounds.

**Message Complexity.** The first procedure uses  $O(m)$  messages (follows from Lemma 4). The second procedure uses no messages if we assume that the smoothing process (addition of random edges) is done by the system without incurring any messages. It may use  $O(n \log n)$  messages if we assume one message cost per one edge addition. The third procedure uses  $O(\sqrt{n} \log^{2.5} n + n \log n)$  messages. The term  $O(\sqrt{n} \log^{2.5} n)$  comes from performing  $O(\sqrt{n \log n})$  random walks for  $O(\log n)$  steps from  $O(\log n)$  candidate nodes. The term  $O(n \log n)$  comes from the communication inside super-nodes or MST fragments via the spanning tree edges for  $O(\log n)$  rounds. Thus, the message complexity of the algorithm is  $O(m + n \log n)$ .  $\square$

### 3.3 Deterministic Algorithm

In this section, we describe a deterministic algorithm for leader election in the smoothing model. Note that the smoothing part uses random bits, which we cannot avoid in this smoothing model. The rest of the algorithm is deterministic. That's why we are calling the algorithm *deterministic*. The algorithm builds on the similar ideas of the previous improved algorithm (cf. Sect. 3.2). Analogously, it has three procedures: (I) Compute MST fragments, (II) Construct smoothed graph, and (III) Compute a leader. The first two procedures (compute MST fragments, construct smoothed graph) follow the same procedures as in the previous algorithm. Note that the controlled GHS algorithm (which is used to compute MST fragments) is deterministic. Now for the third procedure, we use a deterministic approach instead of applying random walks. In fact, we use a deterministic algorithm from [19] which had solved the leader election problem in  $O(D \log n)$  rounds and  $O(m \log n)$  messages.

After running the first two procedures, we obtain the smoothed graph  $R'(V', S')$  where  $V'$  is the set of super-nodes (MST fragments),  $|V'| = O(n\sqrt{\epsilon})$  and  $S'$  is the set of inter-super-node smoothed edges. Since  $R'(V', S')$  is a random expander graph, its diameter is  $O(\log(n\sqrt{\epsilon})) = O(\log n)$ , e.g., see Theorem 8.13 of [6].

Now for the third procedure (computing a leader), we use the deterministic algorithm from [19] and describe how to run it on  $R'$  to elect a leader. This algorithm runs in phases, and each phase consists of 4 stages. First all the nodes become candidate nodes. In each phase  $i$ , every candidate node is required to develop a BFS tree of depth  $2^{i-1}$  (for  $i = 1, 2, \dots, \log(n\sqrt{\epsilon})$ ) and then carry out operations in 4 stages. In the first stage, every candidate node  $v$  (which translates as the root node of the BFS tree) sends a token **ELECT(phase, ID, counter)** on its BFS tree, where phase refers to the phase  $i$  that this candidate node is in, ID is candidate node's ID and counter refers to the depth of BFS tree in any phase  $i$  that this candidate node needs to develop. As the tree develops, this counter is decremented by 1. At the beginning, that is, at the candidate node, it is set

to  $2^{i-1}$ . In the 2nd stage, it receives an ACK token from all its BFS children of the form: **ACK(ID, max, phase\_status)**, where ID is the ID of the candidate node that generated this token, max is the maximum ID encountered by that candidate node, and phase\_status is about whether its phase status is same as or lower than this candidate node's phase. Next, the candidate node  $v$  updates its field  $\mathbf{v.max}$  with the highest ID it has received, that was contained in the max field among the ACK tokens and sends a **CONFIRM(v.max)** token along its BFS tree in the 3rd stage. In the 4th stage,  $v$  receives a **VICTOR(phase, ID)** token from all its neighbors which contains the  $\mathbf{v.max}$  – the highest ID node – encountered by them. If this  $\mathbf{v.max}$  is the same as the ID of the candidate node,  $v$ , then  $v$  continues to the next phase; otherwise assumes a NON-ELECTED state. In every phase, all the above 4 stages are repeated. After  $\log(n\sqrt{\epsilon})$  phases, only the maximum ID node will be ELECTED as a leader and all the other nodes have NON-ELECTED status.

We now explain how to adapt this procedure on the smoothed graph  $R'(V', S')$ . The super-nodes are MST fragments which contain a root node. Since the root node coordinates the tasks inside a super-node, all the root nodes mark themselves as the candidate nodes in the beginning of the algorithm (essentially, all the super-nodes become candidate nodes). Each super-node acts as a single node and implement all the phases on the smoothed graph  $R'(V', S')$  only, e.g., the BFS trees are constructed on  $R'(V', S')$ . Inside a super-node, the root controls and coordinates the tasks and all the communication done through MST edges inside the supernode. Thus, an extra factor of  $O(1/\sqrt{\epsilon})$  rounds may be incurred for each step of the above algorithm due to the communication within a super-node. This is because, the diameter of each MST fragment is  $O(1/\sqrt{\epsilon})$ , so the communication within a fragment takes  $O(1/\sqrt{\epsilon})$  rounds. In the end, one super-node will be elected as the leader. The root node of this super-node will be the leader in the graph.

We now discuss the time and message complexity of the entire deterministic algorithm.

**Theorem 3.** *Given an anonymous graph  $G = (V, E)$  in the  $\epsilon$ -smoothing model, there exists a deterministic distributed algorithm that solves the leader election problem in  $O\left(\frac{\log^2 n}{\sqrt{\epsilon}}\right)$  rounds using  $O(m + n\sqrt{\epsilon}\log^2 n)$  messages.*

*Proof.* It follows from Theorem 2 that the first two procedures takes  $O\left(\frac{\log n}{\sqrt{\epsilon}}\right)$  rounds and  $O\left(m + n\log^* n\log\left(\frac{1}{\sqrt{\epsilon}}\right)\right)$  messages.

It follows Lemma 4.8 in [19] that third procedure can have  $O(\log(n\sqrt{\epsilon}))$  phases. Further, each of the 4 stages can take at most the diameter time of the smoothed graph  $R'$ . Since the diameter of  $R'$  is  $O(\log(n\sqrt{\epsilon}))$ , and communication inside a super-node takes  $O\left(\frac{1}{\sqrt{\epsilon}}\right)$  rounds, the total time for the third procedure is:

$$O\left(\frac{1}{\sqrt{\epsilon}}\right) \cdot \log(n\sqrt{\epsilon}) \cdot \log(n\sqrt{\epsilon}) = O\left(\frac{\log^2(n\sqrt{\epsilon})}{\sqrt{\epsilon}}\right) = O\left(\frac{\log^2 n}{\sqrt{\epsilon}}\right) \text{ rounds.}$$

Thus, the time complexity of the algorithm is  $O\left(\frac{\log^2 n}{\sqrt{\epsilon}}\right)$  rounds.

Let us calculate the message complexity of the third procedure. Consider a phase  $i$  of the algorithm. The message exchanges within a fragment happen over the MST edges. Thus, it uses  $O(n\sqrt{\epsilon}) \times O(\frac{1}{\sqrt{\epsilon}}) = O(n)$  messages inside all the MST fragments (super-nodes). Message exchanges between the MST fragments happen over the inter-super-node smoothed edges, which is  $O(n\sqrt{\epsilon} \log n)$ . Therefore, total number of messages uses in  $O(\log(n\sqrt{\epsilon}))$  phases is:  $O(\log(n\sqrt{\epsilon})) \times O(n + n\sqrt{\epsilon} \log n) = O(n\sqrt{\epsilon} \log^2 n)$ .

Thus, the message complexity of the algorithm is:  $O\left(m + n \log^* n \log\left(\frac{1}{\sqrt{\epsilon}}\right)\right) + O(n\sqrt{\epsilon} \log^2 n) = O(m + n\sqrt{\epsilon} \log^2 n)$ .  $\square$

## 4 Conclusion

We studied smoothed analysis of leader election, one of the fundamental problem in distributed networks. We consider the same smoothing model as introduced by Chatterjee et al. [7] in distributed networks. We present two randomized algorithms and a deterministic algorithm and discuss their smoothed complexity of time and messages. The time and message complexity of our first algorithm are optimal, up to a polylog  $n$  factor. For the second algorithm there is a trade off as it solves the problem in less number of rounds, but incurs more messages. We present a third algorithm which is deterministic but takes slightly more time to solve the leader election problem.

We believe this work extends the study of smoothed analysis of distributed problems. An obvious next step is to investigate how tight these complexities are by analyzing the lower time and message bound of these algorithms. Another line of work could probe the behavior of these algorithms in a different smoothing model and when the nodes can not differentiate between the input edges and the smoothed edges.

## References

1. Afek, Y., Gafni, E.: Time and message bounds for election in synchronous and asynchronous complete networks. *SIAM J. Comput.* **20**(2), 376–394 (1991)
2. Anderson, D.P., Kubiawicz, J.: Introduction to distributed algorithms. *The worldwide computer. Sci. Am.* **286**(3), 28–35 (2002)
3. Angel, O., Bubeck, S., Peres, Y., Wei, F.: Local max-cut in smoothed polynomial time. In: *STOC* (2017)
4. Arthur, D., Manthey, B., Röglin, H.: Smoothed analysis of the k-means method. *J. ACM* **58**(5), 1–31 (2011)
5. Augustine, J., Molla, A.R., Pandurangan, G.: Sublinear message bounds for randomized agreement. In: *PODC*, pp. 315–324. *ACM* (2018)
6. Blum, A., Hopcroft, J., Kannan, R.: *Foundations of Data Science*. Cambridge University Press (2020). <https://doi.org/10.1017/9781108755528>
7. Chatterjee, S., Pandurangan, G., Pham, N.D.: Distributed MST: a smoothed analysis. In: *ICDCN*, pp. 15:1–15:10 (2020)

8. Dinitz, M., Fineman, J., Gilbert, S., Newport, C.: Smoothed analysis of dynamic networks. In: Moses, Y. (ed.) DISC 2015. LNCS, vol. 9363, pp. 513–527. Springer, Heidelberg (2015). [https://doi.org/10.1007/978-3-662-48653-5\\_34](https://doi.org/10.1007/978-3-662-48653-5_34)
9. Elsässer, R., Tscheuschner, T.: Settling the complexity of local max-cut (almost) completely. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) ICALP 2011. LNCS, vol. 6755, pp. 171–182. Springer, Heidelberg (2011). [https://doi.org/10.1007/978-3-642-22006-7\\_15](https://doi.org/10.1007/978-3-642-22006-7_15)
10. Etscheid, M., Röglin, H.: Smoothed analysis of local search for the maximum-cut problem. *ACM Trans. Algorithms* **13**(2), 1–12 (2017)
11. Gallager, R.G., Humblet, P.A., Spira, P.M.: A distributed algorithm for minimum-weight spanning trees. *ACM Trans. Program. Lang. Syst.* **5**(1), 66–77 (1983)
12. Gilbert, S., Robinson, P., Sourav, S.: Leader election in well-connected graphs. In: Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC), pp. 227–236 (2018)
13. Humblet, P.: Electing a leader in a clique in  $o(n \log n)$  messages. Intern. Memo., Laboratory for Information and Decision Systems. M.I.T., Cambridge, MA (1984)
14. Kadjouh, N., et al.: A dominating tree based leader election algorithm for smart cities IoT infrastructure. *Mob. Netw. Appl.*, 1–14 (2020). <https://doi.org/10.1007/s11036-020-01599-z>
15. Khan, M., Kuhn, F., Malkhi, D., Pandurangan, G., Talwar, K.: Efficient distributed approximation algorithms via probabilistic tree embeddings. *Distrib. Comput.* **25**(3), 189–205 (2012). <https://doi.org/10.1007/s00446-012-0157-9>
16. Korach, E., Kutten, S., Moran, S.: A modular technique for the design of efficient distributed leader finding algorithms. *ACM Trans. Program. Lang. Syst.* **12**(1), 84–101 (1990)
17. Korach, E., Moran, S., Zaks, S.: The optimality of distributive constructions of minimum weight and degree restricted spanning trees in a complete network of processors. *SIAM J. Comput.* **16**(2), 231–236 (1987)
18. Korach, E., Moran, S., Zaks, S.: Optimal lower bounds for some distributed algorithms for a complete network of processors. *Theor. Comput. Sci.* **64**(1), 125–132 (1989)
19. Kutten, S., Pandurangan, G., Peleg, D., Robinson, P., Trehan, A.: On the complexity of universal leader election. *J. ACM* **62**(1), 7:1–7:27 (2015)
20. Kutten, S., Pandurangan, G., Peleg, D., Robinson, P., Trehan, A.: Sublinear bounds for randomized leader election. *Theor. Comput. Sci.* **561**, 134–143 (2015)
21. Kutten, S., Zinenko, D.: Low communication self-stabilization through randomization. In: Lynch, N.A., Shvartsman, A.A. (eds.) DISC 2010. LNCS, vol. 6343, pp. 465–479. Springer, Heidelberg (2010). [https://doi.org/10.1007/978-3-642-15763-9\\_45](https://doi.org/10.1007/978-3-642-15763-9_45)
22. Lann, G.L.: Distributed systems - towards a formal approach. In: Information Processing, Proceedings of the 7th IFIP Congress 1977, pp. 155–160 (1977)
23. Levin, D.A., Peres, Y., Wilmer, E.L.: Markov Chains and Mixing Times. American Mathematical Society, Providence (2006)
24. Pandurangan, G.: Distributed network algorithms (2018). <http://sites.google.com/site/gopalpandurangan/dnabook.pdf>
25. Peleg, D.: Distributed Computing: A Locality-Sensitive Approach. SIAM, Philadelphia (2000)
26. Rahman, M.U.: Leader election in the Internet of Things: challenges and opportunities. *CoRR abs/1911.00759* (2019). <http://arxiv.org/abs/1911.00759>
27. Ratnasamy, S., Francis, P., Handley, M., Karp, R.M., Shenker, S.: A scalable content-addressable network. In: SIGCOMM, pp. 161–172. ACM (2001)

28. Roughgarden, T.: Beyond worst-case analysis. *Commun. ACM* **62**(3), 88–96 (2019)
29. Shi, E., Perrig, A.: Designing secure sensor networks. *IEEE Wirel. Commun.* **11**(6), 38–43 (2004)
30. Singh, G.: Efficient distributed algorithms for leader election in complete networks. In: *ICDCS*, pp. 472–479. IEEE Computer Society (1991)
31. Spielman, D.A., Teng, S.H.: Smoothed analysis of algorithms: why the simplex algorithm usually takes polynomial time. *J. ACM* **51**(3), 385–463 (2004)
32. Wright, A.: Contemporary approaches to fault tolerance. *Commun. ACM* **52**(7), 13–15 (2009)