Probabilistic Models and Algorithmic Analysis of Network Problems

by

Nguyen Dinh Pham

A dissertation submitted to the Department of Computer Science,

College of Natural Sciences and Mathematics

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in Computer Science

Chair of Committee: Gopal Pandurangan

Committee Member: Anil Vullikanti

Committee Member: Lennart Johnsson

Committee Member: Aron Laszka

University of Houston

December 2019

Acknowledgements

This work would not have been possible without the financial support from the University of Houston and the National Science Foundation (NSF), and the supervision of Dr. Gopal Pandurangan. I would like to express my sincere gratitude to him. Without his guidance, I would not have been able to follow the daunting journey of doing research. It is not only his expertise, but his example of enthusiasm, hardworking, and consistency that kept me moving.

I am grateful to my collaborators for their help and support, especially Dr. Anil Vullikanti. And since working with a team is always more encouraging, and more fun, I am thankful to have met my labmates, Dr. Soumyottam Chatterjee, Dr. Robert Gmyr, Mr. Emtiaz Ahmed, and Dr. Reza Fathi.

I would like to thank my parents, for their unconditional love and caring, wherever I go. I thank my wife and my little daughter, for their patience and sacrifice during all these years. My special thanks go to my cousin and her husband, who offered me invaluable help.

Abstract

Network-related problems span over many areas in computer science. In this dissertation, we investigate two network problems, one in the domain of distributed computing, and the other in the domain of graph mining. We approach these problems by using probabilistic tools, to model, analyze, and design algorithms.

In the first problem, we aim to improve upon the known complexity bounds of distributed algorithms for fundamental graph problems. We propose the *Smoothed analysis* of distributed algorithms, where the goal is to randomly and slightly perturb the input network and to study how the distributed complexity changes with the amount of perturbation. We present efficient algorithms and lower bounds for the fundamental minimum spanning tree problem and perform smoothed analysis of the time complexity.

In the second problem, we study *influence spreading* in networks, which is a stochastic process. We propose a new optimization problem: minimizing the seed set with *probabilistic guarantees* on the influence. This problem relates to non-submodular target functions, and is hard even for an approximation solution. We design an efficient polynomial algorithm using relaxed multi-criteria approximation and Monte Carlo sampling. We prove theoretical bounds on the performance of our algorithm and validate it with experimental results.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In this dissertation, we study the random phenomenon in network-related problems. In particular, we explore two problems, one in the domain of theory of *distributed computing*, the other in that of *network influence spreading*. Though being in separate fields, both problems are treated from the perspective of probabilistic modeling, which is our main approach.

In the study of the distributed computing system, we follow the graph modeling of a distributed system, i.e., computational *nodes* linked by communication *edges*. We employ the well-known *synchronous* messaging, and look at some fundamental graph algorithms, *minimum spanning tree* (MST) in particular. There are already well-established asymptotic bounds, and improving them is indeed difficult. Our motivation is to use *random perturbation*, which is some random slight modification to the system topology, such that this extra randomness can help to break the current bounds. We named this model *Smoothed Analysis*. Under this model, we derive a new lower bound for the MST problem, which is applicable to other fundamental problems as well. For the MST problem, we also show an almost matching algorithm.

The second study is on network influence spreading. Here, the graph is a model of some interaction or influence, which can be found in social networks, for example. The probability of interaction is given per node or per edge, and the model is studied as a discrete-time stochastic process. This is a highly active area, and the works on *expected influence* is well-established, where the main theme is to design efficient algorithms for the related optimization problems. We approach the subject from another perspective: *Influence Maximizing with Probabilistic Guarantee*, i.e., instead of the expectation value, we are interested in the value with a threshold probability. With this extra guarantee, the optimization problems become harder, as the target function is *non-submodular*. We analyze one such problem: maximizing influence with probabilistic guarantee, within the constraint of the seed set size. We design algorithms with theoretically proven properties and perform empirical analysis. Our resulting algorithm has the same order of complexity as the state-of-the-art algorithms for influence expectation (to the best of our knowledge). Our experiments confirm the runtime complexity and the quality of the solutions.

In the next section, we discuss the mathematical tools and notations in use. We will focus on the materials that are relevant to both of the mentioned works. Then in the chapters dedicated to each problem, we will present more in-depth prior work.

1.1 Preliminaries

1.1.1 Useful Probabilistic Bounds

In our analyses and proofs, we frequently use Chernoff bounds (Chernoff and Rubin) [16, 52, 15]. Though the application is restricted to independent random variables, the bound is very sharp, i.e., of exponential order. There are many versions, each would be handy in a

different situation, thus we introduce only those that relevant to this work.

Let $X_1, X_2, \cdots, X_n$ be a sequence of independent $\{0,1\}$ random variables, where $\Pr[X_i = 1] = p_i$ (i.e., the X_i are Poisson trials). Let $X = \sum_{i=1}^{n} X_i$ and let $\mu = \mathrm{E}[X] = \sum_{i=1}^{n} p_i$. For a given $\delta > 0$, we are interested in the probability that X deviates from the expectation by a factor of $(1 \pm \delta)$.

Theorem 1 (Chernoff bounds [52]). *Let $X_1, X_2, \ldots, X_n$ be a sequence of independent Poisson trials such that $\Pr[X_i = 1] = p_i$. Let $X = \sum_{i=1}^{n} X_i$ and let $\mu = \mathrm{E}[X]$. Then the following Chernoff bounds hold:*

1. $\forall \delta > 0$,

$$\Pr[X \geq (1+\delta)\mu] \leq \left(\frac{e^{\delta}}{(1+\delta)^{(1+\delta)}} \right)^{\mu}; \tag{1.1}$$

2. $\forall \delta \in (0,1]$,

$$\Pr[X \geq (1+\delta)\mu] \leq e^{-\frac{\mu\delta^2}{3}}; \tag{1.2}$$

and

3. $\forall \delta \in (0,1)$,

$$\Pr[X \geq (1-\delta)\mu] \leq \left(\frac{e^{-\delta}}{(1-\delta)^{(1-\delta)}} \right)^{\mu}; \tag{1.3}$$

$$\Pr[X \leq (1-\delta)\mu] \leq e^{-\frac{\mu\delta^2}{2}}. \tag{1.4}$$

The above inequalities deal with upper and lower bounds separately. We can also derive a handy two-sided bound, as follows.

Corollary 2 ([52]). *Let $X_1, X_2, \ldots, X_n$ be a sequence of independent Poisson trials such*

that $\Pr[X_i = 1] = p_i$*. Let* $X = \sum_{i=1}^{n} X_i$ *and let* $\mu = \mathrm{E}[X]$*. For* $\delta > 0$*:*

$$\Pr[|X - \mu| \geq \delta\mu] \leq 2e^{-\frac{\delta^2\mu}{2+\delta}}. \tag{1.5}$$

For the more general setting, where the random variables are not only $\{0, 1\}$ but bounded, we have the Hoeffding inequality [35]. We state the special state, where all random variables are in the same range $[a, b]$.

Theorem 3 (Hoeffding inequality [52])**.** *Let* $X_1, X_2, \cdots, X_n$ *be independent random variables such that for all* $1 \leq i \leq n$*,* $\Pr[a \leq X_i \leq b] = 1$*. Let* $\overline{X} = \frac{1}{n}\sum_{i=1}^{n} X_i$*. For all* $t > 0$

$$\Pr[|\overline{X} - \mathrm{E}[\overline{X}]| \geq t] \leq 2\exp\left(\frac{-2nt^2}{(b-a)^2}\right). \tag{1.6}$$

1.1.2 Monte Carlo Method and Probability of Success

The Monte Carlo method is a class of techniques for estimating values by sampling and simulation. The generic estimation problem is defined as follows.

Definition 1. *An* (ϵ, δ)*-approximation for the value* X *is a value* $\hat{X}$ *such that:*

$$\Pr[|X - \hat{X}| \leq \epsilon X] \geq 1 - \delta. \tag{1.7}$$

The quantity $1 - \delta$ is known as the confidence of the estimation. With probability of at least $1 - \delta$, the estimated value is within the true value by a multiplicative factor. However, there is a chance of δ that the estimation is totally off target, i.e., arbitrarily bad.

We notice that Monte Carlo estimation can be treated as a randomized algorithm; in such a context, $1 - \delta$ is the probability of success. Throughout this work, we use the terms

in their respective contexts. Moreover, for randomized algorithms, we usually desire a high probability of success, which is abbreviated as *w.h.p.* (with high probability).

Definition 2. *Given a randomized algorithm solving a problem of size n, we consider the algorithm successes w.h.p. if such probability is at least $1 - \frac{1}{n^c}$, for some constant $c > 0$.*

1.1.3 Graphs and Some Related Problems

Throughout this work, we use the notation $G(V, E)$ to indicate a graph G with V as the set of vertices and E as the set of edges. As per common convention, we also denote $n = |V|$, $m = |E|$, $d(v)$ the degree of a node v, and D as the diameter of G, unless otherwise specified.

Volume and Conductance

For any nonempty set $S \subseteq V$, we define the following terms.

Definition 3. *The* volume *of S, denoted* $\mathrm{Vol}(S)$ *is defined as the sum of the degrees of nodes in S:* $\mathrm{Vol}(S) \stackrel{\text{def}}{=} \sum_{v \in S} d(v)$.

Definition 4. *For $S \subsetneq V$, the* boundary *of S, denoted by ∂S, is defined as the set of edges crossing S and $\bar{S}$, where $\bar{S} \stackrel{\text{def}}{=} V \setminus S$.*

Definition 5. *For $S \subsetneq V$, the* conductance *of S is defined as* $\varphi(S) \stackrel{\text{def}}{=} \frac{|\partial S|}{\min(\mathrm{Vol}(S), \mathrm{Vol}(\bar{S}))}$.

Definition 6. *The* conductance *of G is defined as*

$$\Phi(G) \stackrel{\text{def}}{=} \min \{\varphi(S) \mid |S \not\subseteq V\}. \tag{1.8}$$

Remark 7. *An alternative, but equivalent, definition of $\Phi(G)$ would be the following:*

$$\Phi(G) \stackrel{\text{def}}{=} \min \left\{ \frac{|\partial S|}{\mathrm{Vol}(S)} | S \not\subseteq V \text{ and } \mathrm{Vol}(S) \leq \frac{\mathrm{Vol}(V)}{2} \right\}. \tag{1.9}$$

Remark 8. *Another equivalent definition of* $\Phi(G)$*, which is more handy in some cases:*

$$\Phi(G) \stackrel{\text{def}}{=} \min\left\{\frac{|\partial S|}{min(\text{Vol}(S), \text{Vol}(\bar{S}))} | S \not\subseteq V \text{ and } |S| \leq \frac{|V|}{2}\right\}. \tag{1.10}$$

1.1.4 Random Walk and Mixing Time

Random Walk. Random walk, in particular, random walk on graphs, is a fundamental technique that has applications in many branches of theoretical computer science (e.g., [1, 2, 5, 48, 17, 62]). On a graph G, a random walk of length k is a *discrete stochastic process* represented by the random variables $X_1, X_2, \cdots, X_k$, where X_i indicates the vertex at step i by the rule: X_{i+1} is chosen at random from the neighbors of X_i. In general, we associate a transition probability with an edge (u, v), for example, uniformly random (equal probability). Denote the matrix M where M_{uv} indicates the probability to move from u to v. Since this process is memoryless and depends on exactly one previous state, it is usually analyzed as *order*-1 *Markov chain* (for details, see, e.g., [47, 8]).

Definition 9 (Probability distribution of vertices)**.** *Consider a random walk on the graph* $G(V, E)$ *with probability matrix* M*. The probability distribution at step* t *is a vector of size* n*:*

$$P^t = (P^t(v_1), P^t(v_2), \cdots, P^t(v_n)); \tag{1.11}$$

where $P^t(v_i)$ *is the probability that the walk is at node* v_i *by step* t*.*

If the random walk starts with some initial distribution P^0, then we have: $P^t = P^0 M^t$.

Definition 10 (Stationary distribution)**.** *A distribution* Π *is stationary if* $\Pi M = \Pi$*.*

We are interested in the stationary distribution of a graph random walk. It is known that for any non-bipartite, connected, undirected graph, the Markov chain is irreducible and

aperiodic and thus has a stationary distribution. To guarantee that the Markov chain is aperiodic, for any general connected, undirected graph, we employ *lazy random walks* (see, e.g., [30]): In every step, the walk remains at the current node with probability $\frac{1}{2}$ and it moves to a uniformly random neighbor otherwise. The *stationary distribution* of such a random walk is proportional to the degree distribution, i.e., when performing enough steps of the random walk, the probability for ending in a node v converges to $\frac{d(v)}{2m}$.

Definition 11 (Mixing time)**.** *For $V = \{v_1, v_2, \ldots, v_n\}$ and a node $v \in V$, let $P_v^t = (P_v^t(v_1), P_v^t(v_2), \ldots, P_v^t(v_n))$ be the probability distribution on the nodes after t steps of a lazy random walk starting at v. Then the* mixing time *(denoted by $\tau_{mix}(G)$) of the graph G is defined as the minimum t such that $\forall\ u, v \in V$,*

$$|P_v^t(u) - \tfrac{d(v)}{2m}| \leq \tfrac{d(v)}{2mn}.$$

We note that — as in [30] — by running a random walk for $O(\tau_{\text{mix}})$ steps, one can improve the deviation from mixing time to $|P_v^t(u) - \frac{d(v)}{2m}| \leq \frac{1}{n^c}$, for any arbitrary (but fixed) positive constant c.

There is a relationship between the conductance of a graph and its (lazy random walk) mixing time, see, e.g., [36, 49].

Lemma 4. *Given a graph $G(V, E)$ with constant conductance, then G has fast mixing time: $\tau_{mix}(G) = O(\log n)$.*

Distributed Computing Model

There are many models for a distributed system. We will be interested in theoretical models, where any detailed implementation issues are not of concern. This approach serves the high abstraction level, and we focus on algorithm feasibility and asymptotic bounds. In general,

we consider a set of isolated *processes*. The computational power of each process varies in different models. For example, the processes in population protocol [3] or programmable matter [70] are relatively weak. We will limit our focus on powerful, i.e., Turing complete, processes. There are further properties that classify the models, discussed next.

Shared Memory vs. Message Passing. The communication among processes can be implicit, via read and write to/from a globally shared memory. Another approach is to require explicit *messages* to be sent and received. In such a message passing system, each process only communicates with its directly connected neighbors. The system is thus conveniently modeled as a graph $G(V, E)$.

Synchronous vs. Asynchronous. This classification is most relevant for message passing systems. In the synchronous model, we assume a global clock that synchronizes all process private clock. Thus the system can progress in lockstep, marked by the completion of sending and receiving messages. In the asynchronous model, there is no synchronizing clock, and messages can arrive in any order. It is known [51, 60] that one can construct synchronizers for any asynchronous network. Thus, any synchronous algorithms can be implemented on an asynchronous network. However, each model still serves different interests. In this work, we focus on synchronous networks, with fundamental problems related to graph theory.

Congest vs. Local. When considering the size of the communication messages, we further classify message passing models as LOCAL and CONGEST. In the LOCAL model, the communication link has unbounded *bandwidth*; processes can send arbitrarily large messages. In the CONGEST model, the bandwidth is bounded by $O(\log n)$ (recall that we imply $n = |V|$, the number of nodes in the system). This is a more realistic assumption. We note that the

bandwidth is enough for essential tasks, such as sending a node ID: $\log n$ bits is necessary to represent a unique ID for each of n nodes.

In this dissertation, the work on distributed computing is limited in the context of the synchronous CONGEST message passing system. We will be interested in the asymptotic bounds of *time complexity* and *message complexity*.

Time Complexity. Since the system is synchronous, time is measured by the number of clock ticks or *rounds*. We take the maximum time over all nodes as the execution time of an algorithm.

Message Complexity. In the CONGEST model, each message has a small size of $O(\log n)$, thus the complexity can be measured in the number of messages. Message complexity is defined as the total number of messages sent by all nodes, during the algorithm.

1.2 Roadmap

We provide an overview on the organization of the rest of this dissertation.

[Chapter 2] Smoothed Analysis and Distributed MST.

[Chapter 3] Influence Maximization with Probabilistic Guarantee.

[Chapter 4] Conclusion.

Since the problems in Chapter 2 and Chapter 3 are somewhat independent, we will take the same approach in the structure of each chapter. Each problem will have an introduction where our contribution is highlighted, followed by a discussion of the related concepts. Next,

we survey the modern literature to give a big picture view of the problem. Then we discuss our approach in detail. We will cover the proofs, the analysis, as well as the intuitive ideas behind. For the relevant problem, implementation and experiment are also reported.

Each problem will have its conclusion where we point out potential research questions. Finally, the last chapter wraps up the contribution of this work and draws some interesting connections.

Chapter 2

Smoothed Analysis and Distributed MST

In this chapter [1], we study smoothed analysis of distributed graph algorithms, focusing on the fundamental minimum spanning tree (MST) problem [29]. With the goal of studying the time complexity of distributed MST as a function of the "perturbation" of the input graph, we posit a *smoothing model* that is parameterized by a smoothing parameter $0 \leq \epsilon(n) \leq 1$ which controls the amount of *random* edges that can be added to an input graph G per round. Informally, $\epsilon(n)$ is the probability (typically a small function of n, e.g., $n^{-\frac{1}{4}}$) that a random edge can be added to a node in one round. The added random edges, once they are added, can be used (only) for communication.

We show upper and lower bounds on the time complexity of distributed MST in the above smoothing model. We present a distributed algorithm that, with high probability, computes an MST and runs in $\tilde{O}(\min\{\frac{1}{\sqrt{\epsilon(n)}}2^{O(\sqrt{\log n})}, D + \sqrt{n}\})$ rounds [2] where ϵ is the smoothing

[1] This is joint work with Soumyottam Chatterjee and Gopal Pandurangan [11]

[2] The notation $\tilde{O}$ hides a polylog(n) factor and $\tilde{\Omega}$ hides a $\frac{1}{\text{polylog}\,(n)}$ factor

parameter. We also show a lower bound of $\tilde{\Omega}(\min\{\frac{1}{\sqrt{\epsilon(n)}}, D+\sqrt{n}\})$. We note that the upper and lower bounds essentially match except for a multiplicative $2^{O(\sqrt{\log n})}\,\mathrm{polylog}(n)$ factor.

To the best of our knowledge, our work can be considered as a first step in understanding the smoothed complexity of distributed graph algorithms.

2.1 Introduction and Motivation

Smoothed analysis of algorithms was introduced in a seminal paper by Spielman and Teng [64] to explain why the well-studied simplex algorithm for linear programming does well in practice, despite having an (worst-case) exponential run time in theory. The high-level idea behind the smoothed analysis of the simplex algorithm is the following:

1. Perturbing the input data with a *small* amount of *random* noise (e.g., Gaussian noise with mean zero, parameterized by the variance of the noise), and then

2. Showing that the perturbed input can be solved efficiently by the simplex algorithm, i.e., in polynomial time. In particular, Spielman and Teng quantify the run time as a function of the perturbation; the more the perturbation (i.e., larger the variance of the noise), the faster the run time.

Smoothed analysis is thus different from the *worst-case* analysis of algorithms. It is also different from the *average-case analysis*, which assumes a probability distribution on the set of all possible inputs. Smoothed analysis, on the other hand, is sort of a hybrid between the above two — it considers the worst-case input and then randomly perturbs it. If even small perturbations (e.g., adding random noise) lead to efficient run time, then this means that the worst-case is quite sensitive to the input parameters. In practice, there will usually be noise and thus the algorithm is likely to avoid the worst-case behavior.

In this chapter, we initiate the study of *smoothed analysis of distributed graph algorithms*. Our work is motivated by the work of Dinitz et al. [19] who initiated the study of smoothed analysis for *dynamic networks* (we refer to Section 2.1.1 for more details). A main contribution of our work is positing smoothing models in the context of *distributed graph algorithms* and performing analyses of the models. While many smoothing models are possible for such algorithms, a key goal is to identify models that lead to non-trivial bounds on the distributed complexity (here we focus on time complexity) of fundamental graph algorithms.

We focus on the distributed minimum spanning tree (MST) problem in synchronous CONGEST networks. The worst-case time (round) complexity of distributed MST has been extensively studied for the last three decades and tight bounds are now well established (see, e.g., [63, 56]). There is an optimal distributed MST algorithm (see, e.g., [55]) that runs in $\tilde{O}(D + \sqrt{n})$ rounds, where D is the graph diameter and n is the number of nodes in the network. Also, there is a (essentially) matching lower bound of $\tilde{\Omega}(D + \sqrt{n})$ rounds that applies even to randomized Monte-Carlo distributed algorithms [61].

The lower bound is shown by presenting a weighted graph (in particular, a family of graphs) and showing that no distributed algorithm can solve MST faster. This raises a motivating question for smoothed analysis: Is the worst-case bound specific to the choice of the weighted graph (family)? Or more precisely, is it specific to the choice of the graph topology or the edge weights or both? If small perturbations do not change the worst-case bound by too much then we can say that the lower bound is *robust* and, if they do we can say that the bounds are *fragile* [19]. Thus smoothed analysis can lead to a better understanding of the complexity of distributed MST by studying perturbations of the worst-case input. This is one of the motivations in studying smoothed analysis of distributed algorithms.

However, to answer the above questions, one has to first come up with a suitable smooth-

ing model. For example, one possible smoothing model, in the spirit of Spielman and Teng's original smoothing model, would be perturbing the edge weights of the input graph by a small amount. It is apparent that if the perturbation is quite small relative to the weights (since the weights can be well-spaced), then this does not affect the lower bound — it remains $\tilde{\Omega}(D + \sqrt{n})$. Another possible model, which we explore in this work, again in the spirit of original model but now applied to perturbing the *topology of the input graph*, is smoothing the *input graph* by adding[3] a *small* number of *random edges*. While there are a few possible ways to accomplish this, we focus on a particular smoothing model described next. We will discuss other smoothing models (which can be considered variants of this model) in Section 2.5. A practical motivation for this kind of smoothing, i.e., adding a small number of random edges to a given graph, is that many real-world networks might be better modeled by graphs with some underlying structure with some amount of randomness. For example, it is well know that real-world graphs have a power-law degree distribution; however they are not arbitrary (worst-case) power-law graphs but can be reasonably modeled by random graphs with power-law degree distribution [26].

We consider a *smoothing model* (see Section 2.2.1) that is parameterized by a smoothing parameter $0 \leq \epsilon = \epsilon(n) \leq 1$ that controls the amount of *random* edges that can be added to an input graph $G = (V, E)$ per round. $\epsilon(n)$ is typically a small function of n, say, $\epsilon(n) = n^{-\frac{1}{4}}$. More precisely, our smoothing model allows any node to add a random edge with probability $\epsilon(n)$ in each round; the added edges can be used for communication in later rounds. (We note that the added edges, otherwise, do not change the underlying solution with respect to G; e.g., for MST, the added edges have weight ∞ and hence don't affect the MST of G.) Besides this additional feature, nodes behave as in the standard model, i.e., can communicate

[3]One can also delete edges, although we do not consider this here, see Section 2.2.1.

using edges of G. We formally define the model in Section 2.2.1. Note that nodes can as well choose *not* to use this additional feature. Depending on $\epsilon(n)$, the number of random edges added per round can be small. (In Section 2.5, we consider a variant of this model, which essentially gives the same bounds as the ones discussed here.)

An alternate way of thinking about our smoothing model is as follows. Assume that the graph G is embedded in a congested clique. The congested clique model has been studied extensively in the distributed computing literature; see, e.g., [9, 6, 57, 32, 37, 42, 58, 59]). A node — besides using its incident edges in E — can also *choose* to use a random edge (not in G, but in the clique) with probability ϵ in a round to communicate (once chosen, a random edge can be used subsequently till end of computation). Note that if ϵ is small, say, for example $\epsilon = O(n^{-\frac{1}{4}})$, then the probability of adding a random edge by a node in a round is small. In particular, if $\epsilon = 0$, then this boils down to the traditional model, i.e., working on the given graph G with no additional random edges, as ϵ increases, the number of random edges increases with it.

We note that the smoothing model is sort of a hybrid between the traditional model where communication is allowed only along the edges of an arbitrary graph G and a model where G is a random graph (e.g., Erdos-Renyi graph model [10, 44]) or an expander (see e.g., [4] and the references therein). In the smoothing model we start with an arbitrary graph G and add random edges (parameterized by ϵ). In Section 2.2.1, we further explore relationships between the smoothing model and other distributed computing models.

Our goal is to study how the distributed complexity of MST varies as a function of $\epsilon(n)$ (among other usual graph parameters such as network size, network diameter, etc.). We show upper and lower bounds on the time complexity of distributed MST in the aforementioned smoothing model. We present a distributed algorithm, which (with high probability)

computes an MST and runs in

$$\tilde{O}(\min\{\frac{1}{\sqrt{\epsilon(n)}}2^{O(\sqrt{\log n})}, D+\sqrt{n}\}) \text{ rounds},$$

where ϵ is the smoothing parameter, D is the network diameter, and n is the network size.

To complement our upper bound, we also show a lower bound of

$$\tilde{\Omega}(\min\{\frac{1}{\sqrt{\epsilon}}, D+\sqrt{n}\}).$$

Our bounds show non-trivial dependence on the smoothing parameter $\epsilon(n)$, and the bounds are essentially matched except for a $2^{O(\sqrt{\log n})}$ factor and a polylogarithmic factor.

2.1.1 Related Work

Smoothed analysis was introduced by Spielman and Teng [64] and has since been applied for various algorithms problems in the sequential setting (see, e.g., [65] for a survey).

The only work that we are aware of in the context of smoothed analysis of distributed algorithms is that of Dinitz et al. [19] who study smoothed analysis of distributed algorithms for *dynamic networks*. Their dynamic network model is a dynamic graph $\mathcal{H} = G_1, G_2, \ldots$ that describes an evolving network topology, where G_i is the graph at round i. It is assumed that all graphs in $\mathcal{H}$ share the same node set, but the edges can change with some restrictions, e.g., each graph should be connected. They define a smoothing model for a dynamic graph that is parameterized with a smoothing factor $k \in \{1, 2, \ldots, \binom{n}{2}\}$. To k-smooth a dynamic graph $\mathcal{H}$ is to replace each static graph G_i in $\mathcal{H}$ with a smoothed graph G_i' sampled uniformly from the space of graphs that are: (1) within *edit distance* k of G, and (2) are allowed by the dynamic network model (e.g., smoothing cannot generate disconnected graph). The edit distance is the number of edge additions/deletions needed to transform one graph to another, assuming they share the same node set.

Our smoothing model can also be thought of in terms of choosing a random graph within a *positive* edit distance (i.e., edges are only *added* to the original input graph) where the number of random edges added is proportional to $n\epsilon(n)$ (per round or in total — see Section 2.5).

Dinitz et al. study three well-known problems that have strong lower bounds in dynamic network models, namely, *flooding*, *random walks*, and *aggregation*. For each problem, they study robustness/fragility of the existing bound by studying how it improves under increasing amounts of smoothing.

2.2 The Model

The model is the distributed synchronous CONGEST system, as presented in Section 1.1.4. Recall that the network is represented as a graph $G(V, E)$, and we define the following extra properties. Each node u runs an instance of a distributed algorithm and has a unique identifier ID_u of $O(\log n)$ bits. Each edge $e \in E$ may have an associated weight $w(e)$, which can be represented using $O(\log n)$ bits. If there is no weight on an edge, then it can be considered to be ∞.

2.2.1 Smoothing Model

Given a (arbitrary) undirected connected graph $G(V, E)$, the smoothing model allows adding some random edges to the input graph G, thereby "perturbing" graph structure. We call this process *smoothing*, where we add a small number of random edges to the original graph. We describe the process of adding edges which is parameterized by a smoothing parameter $0 \leq \epsilon = \epsilon(n) \leq 1$ as follows. The smoothing parameter (which in general is a function of

n, the network size[4] controls the amount of random edges that can be added per round. Henceforth, we call this the *ϵ-smoothing model*.

More precisely, every node, in every round, with probability ϵ (the smoothing parameter) can *add* an edge to a *random* node (chosen uniformly at random from V) in the graph. Let the added random edges form the set R (different from the original edge set E). Note that we allow multi-edges in the random edge choosing process; however, if there is more than one edge between two nodes, then only one edge matters (especially, if it belongs to E). The added edge persists for future rounds and can be used henceforth for communication; its weight is ∞. A distributed algorithm can potentially exploit these additional edges to improve the time complexity.[5]

In this work, we only consider adding edges to the graph; one can also consider deleting edges from the original graph. However, for many problems such as MST, it is arguably more appropriate to (potentially) add edges. In fact, deleting edges can change the graph. In contrast, in the ϵ-smoothing model, since the added edges to the given graph G are purely communicating edges (with weight ∞), the MST with respect to G is unchanged. In fact, the model allows us to study tradeoffs between the amount of random edges added to the efficiency of computing a solution of G.

As mentioned earlier, the ϵ-smoothing model gives a "smooth" tradeoff between the traditional CONGEST model where there no additional random edges ($\epsilon = 0$) in G (the input graph) and a model where there is a random graph embedded in G. In this sense, it is different from studying distributed computing on (purely) random graph models or expander graph models (e.g., [10, 4, 44]). We note the work of Ghaffari et al. [30, 31] that embeds a random graph in a given graph G and uses this embedding to design algorithms that depend

[4]We sometimes just write ϵ, understanding it to be a function of n.

[5]In this work, we focus only on time complexity, but message complexity can also be relevant.

on the mixing time of G. We will use their results in our algorithms.

As mentioned in Section 2.1, we can also relate the well-studied *congested clique* model to the ϵ-smoothing model and also give a way to understand computation tradeoffs between the traditional CONGEST model and the congested clique model. Assuming the input graph G is embedded in a congested clique, the ϵ parameter controls the power to use the non-graph clique edges. If $\epsilon = 0$, then we have the traditional CONGEST model and for any $\epsilon > 0$, if we spend enough rounds, then one can throw a random edge between every pair of nodes which boils down to the congested clique. Of course, this is costly, which illustrates the power of the congested clique model (where the clique edges can be used for "free"). Studying time and message complexity bounds in terms of ϵ can help us understand the power of the clique edges with respect to solving a problem on a given input graph.

2.3 Distributed MST in the Smoothing Model

For the sake of exposition, we first present a distributed MST algorithm that runs in

$$\tilde{O}(\min\{\frac{1}{\epsilon} + 2^{O(\sqrt{\log n})}, D + \sqrt{n}\}) \text{ rounds.}$$

Then we present an improved algorithm that runs in

$$\tilde{O}(\min\{\frac{1}{\sqrt{\epsilon(n)}}2^{O(\sqrt{\log n})}, D + \sqrt{n}\}) \text{ rounds.}$$

The second algorithm is a modification of the first and its time complexity approaches the lower bound of $\tilde{\Omega}(\min\{\frac{1}{\sqrt{\epsilon}}, D + \sqrt{n}\})$ shown in Section 2.4. Thus, up to a multiplicative factor of $2^{O(\sqrt{\log n})}\operatorname{polylog} n$, the bounds are tight.

We give a high-level overview of our approach of our first algorithm before we get into the technical details. We next present an algorithm that computes an MST on a given graph G in the ϵ-smoothing model. The algorithm can be described in two parts which are described in Sections 2.3.1 and 2.3.2, respectively. At the outset we note that if $1/\epsilon$ is larger compared to $\tilde{O}(D+\sqrt{n})$, then we simply run the standard time-optimal MST algorithm ([55]) without doing smoothing.

2.3.1 Part 1: Constructing an Expander

Initially the algorithm exploits the smoothing model to add about $O(\log n)$ random edges per node. This can be accomplished as follows: each node (in parallel) tries to make a random edge selection for the (first) $\Theta(\frac{\log n}{\epsilon})$ rounds, where ϵ is the smoothing parameter. Since the probability of adding a random edge, i.e., a smoothing edge, is ϵ per round, it is easy to show that with high probability a node will add $\Theta(\log n)$ random edges. Via a union bound, this holds for all nodes.

Now consider the graph $R(G)$ induced *only* by the smoothed (random) edges of G after $\Theta(\frac{\log n}{\epsilon})$ rounds. In the following Lemma 6, we show that $R(G)$ is a graph with $O(\log n)$ *mixing time*.

The proof of this result comes from the relation of ϵ-smoothing model to Erdős-Renyi random graph, which we will show next.

Lemma 5. *Consider a graph $G(V, E)$ under ϵ-smoothing. If we invoke smoothing for ℓ rounds (where $\ell\epsilon = o(n)$), then the graph induced by the smoothed edges is an Erdős-Renyi random graph $G(n, p)$ where $p = \Theta\left(\frac{l\epsilon}{n}\right)$.*

Proof. We calculate the probability of a smoothed edge between nodes u and v. Clearly the edge is present if either u or v successfully adds the other end during ℓ steps. Hence,

$p = 1 - \left(1 - \frac{\epsilon}{n}\right)^{2\ell} = \Theta\left(\frac{\ell\epsilon}{n}\right)$. □

Remark 12. *The following lemma (Lemma 6) applies only to $R(G)$ and* not necessarily *to $G \cup R(G)$.*

Lemma 6. *Let $G = (V, E)$ be an arbitrary undirected graph and let $R(G) = (V, F)$ be the random graph induced (only) by the set F of random (smoothed) edges after $\Theta(\frac{\log n}{\epsilon})$ rounds. Then, with high probability, $R(G)$ has mixing time $\tau_{mix}(R) = O(\log n)$.*

Proof. Using Lemma 5, where $\ell = \Theta(\frac{\log n}{\epsilon})$, we have $R(G)$ is a Erdős-Renyi random graph $G(n, p = \Theta(\frac{\log n}{n}))$. It is well-known that with high probability this random graph is an expander (i.e., has constant conductance) and thus has $O(\log n)$ mixing time (see e.g., [30]). (We refer to the Appendix — see Section A.1 — for an alternative, self-contained proof.) □

2.3.2 Part 2: Constructing an MST

In the second part, the algorithm uses $R(G)$ as a "communication backbone" to construct an MST in $O(\log^2 n)2^{O(\sqrt{\log n})}$ rounds.

Our algorithm crucially uses a routing result due to Ghaffari et al. [31, 30] who show, given an arbitrary graph $G = (V, E)$, how to do *permutation* (or more generally, *multi-commodity*) routing fast. We briefly describe the problem and the main result here and refer to [31] for the details. Permutation or multi-commodity routing is defined as follows: given source-destination pairs of nodes $(s_i, t_i) \in V \times V$ and suppose s_i wants to communicate with t_i (t_i does not know s_i beforehand, but s_i knows the ID of t_i). The *width* of the pairs is W if each $v \in V$ appears at most W times as s_i or t_i. The goal is to construct a routing path P_i (not necessarily simple) from s_i to t_i such that the set of routing paths has low *congestion* and low *dilation*. Congestion is the maximum number of times any edge is used

in all the paths. Dilation is simply the maximum length of the paths. The main result of [31] is that routing paths P_i with *low congestion and dilation* can be found *efficiently*. Once such low congestion and dilation routing paths are found, using a standard trick of random delay routing, it is easy to establish that messages can be routed between the source and destination efficiently, i.e., proportional to congestion and dilation.

Theorem 7 (Efficient Routing)**.** *(Theorem 8 from [31]). Suppose we solve a multicommodity routing instance $\{(s_i, t_i)\}_i$ and achieve congestion c and dilation d. Then, in $\tilde{O}(c+d)$ rounds, every node s_i can send one $O(\log n)$-bit message to every node t_i, and vice versa.*

Next, let us formally restate their routing results for ease of discussion. First we state their result on multi-commodity routing on a random graph $G(n, \log n)$, i.e., a random graph where each node has $O(\log n)$ random edges (each endpoint chosen uniformly at random.)

Theorem 8. *(Theorem 1 from [31]) Consider a multicommodity routing instance of width $\tilde{O}(1)$. There is a multicommodity routing algorithm on a random graph $G(n, (\log n))$ that achieves congestion and dilation $2^{O(\sqrt{\log n})}$, and runs in time $2^{O(\sqrt{\log n})}$.*

Then, by the construction of a random-graph-like hierarchy routing over a given graph G, we have the following result.

Lemma 9. *(Lemma 11 from [31]) In any graph G with n nodes and m edges, we can embed a random graph $G(m, d)$ with $d \geq 200 \log n$ into G with congestion $\tilde{O}(\tau_{mix} \cdot d)$ and dilation τ_{mix} in time $\tilde{O}(\tau_{mix} \cdot d)$.*

Using Lemma 9, we have the following trivial corollary for permutation routing.

Corollary 10 (Permutation Routing)**.** *Consider a graph $G = (V, E)$ and a set of n point-to-point routing requests (s_i, t_i), where s_i, t_i are IDs of the corresponding source and destination.*

Each node of G is the source and the destination of exactly one message. Then there is a randomized algorithm that delivers all messages in time $\tau_{mix}(G)2^{O(\sqrt{\log n})}$, w.h.p.

Based on their multicommodity routing algorithm, they showed how to construct an MST of G in $\tau_{mix}(G)2^{O(\sqrt{\log n})}$ rounds (Ghaffari et al. [30]). However, this MST algorithm cannot be directly employed in our setting, i.e., to construct a MST of G, over $G \cup R(G)$.

We briefly summarize the idea from Section 4 in their paper [30], to explain why this algorithm is not applicable directly in a black box manner. There is the main reason for this: while the algorithm of [30] operates on the graph G, ours operates on the graph $G' = G \cup R(G)$. Applying the algorithm directly to $G \cup R(G)$ can (in general) yield an algorithm running in $\tau_{mix}(G')2^{O(\sqrt{\log n})}$ rounds where $\tau_{mix}(G')$ is the mixing time of G'. Note that $\tau_{mix}(G')$ (in general) can be of the same order of $\tau_{mix}(G)$ (even for constant smoothing parameter ϵ) in some graphs[6]. Thus the running time bound does not (in general) depend on ϵ and does not give our desired bound of $\tilde{O}(\log n/\epsilon)$ rounds. We give more details on the approach of [30] and then discuss our algorithm.

The approach of [30] is to modify Boruvka's algorithm [53], where the MST is built by merging tree fragments. In the beginning, each node is a fragment by itself. The fragment size grows by merging. To ensure efficient communication within a fragment, they maintain a virtual balanced tree for each fragment. A virtual tree is defined by virtual edges among nodes, where virtual edges are communication paths constructed by the routing algorithm. It follows that for each iteration, fragment merging may increase the number of virtual edges of some node v by $d_G(v)$, where $d_G(v)$ is the degree of v in G. Since Borukva's method takes $O(\log n)$ iterations, the virtual degree of any node v is at most $d_G(v)O(\log n)$. Thus virtual trees communication is feasible via commodity routing by Theorem 8 which takes

[6]For example consider two cliques of size $n/2$ connected to each other via $O(n)$ edges.

$\tau_{mix}(G)2^{O(\sqrt{\log n})}$ rounds. Applying the above approach directly to our setting yields only an MST algorithm running in $\tau_{mix}(G')2^{O(\sqrt{\log n})}$ rounds as mentioned in the last paragraph.

In our setting, to obtain an algorithm running in $\tilde{O}(\log n/\epsilon)$ rounds, we would like to perform routing (only) on $R(G)$ instead of $G \cup R(G)$. However, if we use the same algorithm of [30] in $R(G)$ then during the computing of the MST in G, some node v may become overloaded by $d_G(v)$ virtual edges which causes too much congestion in $R(G)$ (where each node has degree $\Theta(\log n)$ only) when $d_G(v)$ is large. It follows that the routing is infeasible in $R(G)$ and the algorithm fails. To solve the problem we proposed a modified algorithm that uses *aggregate routing*.

The idea of aggregate routing is to perform permutation routing, where some destinations are the same. In other words, assume a permutation routing problem where there are only $k \leq n$ (distinct) destinations and $t_1, \ldots, t_k$ and there are n sources $s_1, \ldots, s_n$. Let C_i be the set of sources who have the same destination t_i. In aggregate routing, we would like to *aggregate* the set of messages in C_i and deliver it to t_i. The aggregate function can be a separable (decomposable) function such as min, max, or sum. Then we can modify the permutation routing easily as follows. For a node u that is performing the routing, suppose there are multiple messages arriving at u in the same round. Then u computes the aggregate of the messages belong to the same set C_i (for every $1 \leq i \leq k$) and forwards that message according to the routing algorithm. At the destination, the aggregate is computed over any received messages destined for this particular destination. With this intuition, we state the following definition and lemma.

Definition 13 (k-aggregate routing)**.** *Consider a graph $G = (V, E)$, where nodes are divided into k disjoint partitions $C_1, C_2, \ldots, C_k$. For each partition C_i, there is a leader l_i, known to all members of C_i. Each node $u \in V$ has one message to deliver to its leader. Let f*

be a separable aggregate function (such as min, max, or sum). The k-aggregate routing problem is to compute the aggregate f over the nodes in each partition and route it to the corresponding leader of the partition.

We show that k-aggregate routing can be solved in the same time bounds as multi-commodity routing.

Lemma 11. *Consider a graph $G = (V, E)$ with an instance of the k-aggregate routing problem. There is a randomized algorithm that solves the problem in time $\tau_{mix}(G)2^{O(\sqrt{\log n})}$, w.h.p.*

Proof. Let m^u denote the original message at u. We will send the tuple (m^u, l^u) where l^u is the leader of the partition that u belongs to. Let f be the separable aggregate function.

Each node v executes the multi-commodity routing algorithm, with this extra rule: In a round t, suppose v receives multiple messages having the same destination, which is some leader l_i, v computes the aggregate $f_{t,i}$ over those messages, and prepares a tuple $(f_{t,i}, l_i)$. v then forwards the aggregated message to the appropriate next-hop neighbor in the routing path for the next round. v performs the same reduction for messages targeting other leaders.

It is easy to see that this routing schema is not congested, i.e., it is as fast as the multi-commodity/permutation routing. Observing the local invariance of permutation routing: for each destination u, every node, in each round of the algorithm, sends out at most one message routing towards u. This invariance holds in our k-aggregate routing, by the above construction.

At the end of the routing, each leader aggregates over its received messages, which is the aggregate in its partition. □

While k-aggregate routing can be seen as *upcast* [55], the complementary operation of

downcast, i.e., sending a message from a source to several destinations, can also be done efficiently as shown below.

Lemma 12 (k-aggregate routing and downcast)**.** *Consider a graph $G = (V, E)$ with an instance of the k-aggregate routing problem. Furthermore, we require that every member of a partition knows the corresponding aggregate value. There is a randomized algorithm that solve the problem in time $\tau_{mix}(G)2^{O(\sqrt{\log n})}$, w.h.p.*

Proof. Using the algorithm in Lemma 11, each node also records the source of the incoming messages together with the associated leader ID and round number. Then the routing can be reversed. Starting from the leader of each partition, it sends out the aggregate message with its ID, and each node reverses the aggregate message towards the matching sender. Hence the downcast can be accomplished in the same number of rounds as k-aggregate routing. □

We are now ready to implement the MST algorithm.

Theorem 13. *Consider a weighted graph $G = (V, E)$ in the ϵ-smoothing model. There exists a randomized distributed algorithm that finds an MST of G in time $\tilde{O}(\frac{1}{\epsilon} + 2^{O(\sqrt{\log n})})$, w.h.p.*

Proof. As discussed in Part 1 (Section 2.3.1), the algorithm executes $\Theta(\frac{\log n}{\epsilon})$ rounds of random edge selection to construct the random graph $R(G)$ (the graph induced only by the random edges). As shown in Lemma 6, $R(G)$ is an expander with constant conductance and hence has $O(\log n)$ mixing time.

We use the permutation routing result of [31] to construct the routing structure on $R(G)$, which allows permutation routing in $\tau_{mix}(R(G))2^{O(\sqrt{\log n})} = O(\log n)2^{O(\sqrt{\log n})} = 2^{O(\sqrt{\log n})}$ rounds.

Our MST algorithm is based on the standard Gallagher-Humblet-Spira (GHS)/Boruvka algorithm, see e.g., [55] which is also used in [30] and many other MST algorithm see e.g.,

[56, 22]. The main modification compared to the standard GHS algorithm is that the growth (diameter) of fragments is controlled during merging (as in the controlled GHS algorithm [55]).

We summarize the algorithm here and sketch how it is implemented.

Let T be the (unique) MST on G (we will assume that all weights of edges of G are distinct). A *MST fragment* (or simply a *fragment*) F of T is defined as a connected subgraph of T, that is, F is a subtree of T. An *outgoing edge* of a MST fragment is an edge in E where one adjacent node to the edge is in the fragment and the other is not. The *minimum-weight outgoing edge (MOE)* of a fragment F is the edge with *minimum weight* among all outgoing edges of F. As an immediate consequence of the cut property for MST, the MOE of a fragment $F = (V_F, E_F)$ is an edge of the MST.

The GHS algorithm operates in *phases* (see e.g., [55]). In the first phase, the GHS algorithm starts with each individual node as a fragment by itself and continues till there is only one one fragment — the MST. That is, at the beginning, there are $|V|$ fragments, and at the end of the last phase, a single fragment which is the MST. All fragments find their MOE simultaneously in parallel.

In each phase, the algorithm maintains the following invariant: Each MST fragment has a leader and all nodes know their respective parents and children. The root of the tree will be the leader. Initially, each node (a singleton fragment) is a root node; subsequently each fragment will have one root (leader) 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.

We describe one phase of the GHS (whp there will be $O(\log n)$ phases as discussed below). Each fragment's operation is coordinated by the respective fragment's root (leader). Each phase consists of two major operations: (1) Finding the MOE of all fragments and (2)

Merging fragments via their MOEs.

We first describe how to perform the first operation (finding the MOE). Let F be the current set of fragments. Each node in V finds its (local) minimum outgoing edge (if any), i.e., an edge to a neighbor belonging to a different fragment that is of least weight. We then execute a $|F|$-aggregate routing and downcast, using *min* as the aggregate function, with each node being the source and having its fragment leader as its destination. At the end of this step, for each fragment, every member knows the minimum outgoing edge (MOE) of the entire fragment. This MOE edge will be chosen for merging in the second operation (merging fragments). Also, each node keeps the reversed routing paths for further usage.

Once the merging (MST) edges are identified, the second operation — merging — is processed. In order to avoid long chains of fragments, a simple randomized trick is used. Each fragment chooses to be a *head* or *tail* with probability 1/2. Only *tail* fragments will merge if their outgoing edge points to a *head* fragment. It can be shown (e.g., see [30] that this merging (still) leads to a constant factor decrease in the number of fragments (on average) and hence the number of phases will be $O(\log n)$ in expectation and with high probability.

We now describe how a merge can be implemented efficiently, There will be no change in the head fragment, but all the tail ones will update to acknowledge the head leader as the new leader. For a tail fragment T, let $v \in T$ be the node that is making the merge, v knows the ID of the head leader by communicating with its neighbor which is a member of the head fragment). v routes this new leader ID to the current leader of T. This is done in parallel by permutation routing. The current leader of T downcasts the new leader ID to all T members. This is done via the saved reversed routing paths. The merging is now completed, and time for one iteration is the same as that of permutation routing as before,

i.e., $O(\log n)2^{O(\sqrt{\log n})}$.

There are $O(\log n)$ phases and each phase can be implemented in $O(\log n)2^{O(\sqrt{\log n})}$ rounds and hence the total time for Part 2 is $O(\log^2 n)2^{O(\sqrt{\log n})}$.

The total time for MST construction is the number of rounds for Part 1 plus the number of rounds in Part 2:

$\Theta(\frac{\log n}{\epsilon}) + O(\log^2 n)2^{O(\sqrt{\log n})} = \tilde{O}(\frac{1}{\epsilon} + 2^{O(\sqrt{\log n})})$. □

2.3.3 An Improved Algorithm

We now present an algorithm that is a variant of the previous algorithm and improves upon it. The time complexity of the improved algorithm approaches the lower bound (cf. Section 2.4). The idea is to use the *controlled GHS* algorithm [55] to construct MST fragments of suitable size. Then we apply smoothing (with a smaller number of rounds compared to previous algorithm, i.e., $\tilde{O}(\frac{1}{\sqrt{\epsilon}})$ instead of $\tilde{O}(\frac{1}{\epsilon})$) to add an expander over the super-graph induced by the MST fragments where each super-node is one fragment (partition). Then we compute the final MST in a similar fashion to Theorem 13 on the super-graph.

Theorem 14. *Given a weighted graph $G(V, E)$ in the ϵ-smoothing model, there exists a randomized distributed algorithm that finds an MST of G in time $\tilde{O}(\frac{1}{\sqrt{\epsilon}})2^{O(\sqrt{\log n})}$, w.h.p.*

Proof. We give the algorithm along with its analysis, as follows.

Run $O(\frac{\log n}{\sqrt{\epsilon}})$ rounds of smoothing. Denote by S the set of smoothed edges generated. Using Lemma 5, the probability that a smoothed edge occurs between two nodes in G is $p = \Theta(\frac{\sqrt{\epsilon}\log n}{n})$.

Run the *controlled GHS* algorithm [55] for $\log \frac{1}{\sqrt{\epsilon}}$ phases. This takes $O(\frac{\log^* n}{\sqrt{\epsilon}})$ rounds. Every cluster (each of which is an MST fragment) will have size $\Omega(\frac{1}{\sqrt{\epsilon}})$ and diameter $O(\frac{1}{\sqrt{\epsilon}})$, and there will be $O(n\sqrt{\epsilon})$ such clusters [55, Section 7.4]. We call these clusters *base fragments*.

We note that communication within a cluster (i.e., between any node of the cluster and its leader) takes $O(\frac{1}{\sqrt{\epsilon}})$ rounds.

View these clusters as a set of *super-nodes*, denoted by V'. Let $E' \subset E$ be the set of inter-super-node edges, let $S' \subset S$ be the set of inter-super-node smoothed edges. Consider two super-graphs: $G'(V', E')$ and $R'(V', S')$. It is easy to show that due to the probability p of the random edges introduced by the *smoothing* process, the super-graph $R'(V', S')$ is an Erdős–Rényi random graph or a $G(n', p')$-random graph, where

$$n' = O(n\sqrt{\epsilon}) \tag{2.1}$$

and

$$p' \geq \Omega\left(\left(\frac{1}{\sqrt{\epsilon}}\right)^2\right) p = \Omega\left(\frac{\log(n')}{n'}\right) \tag{2.2}$$

Thus, the super-graph $G' \cup R'$ is equivalent to the smoothed graph of our model in Section 3.2.1.

Similar to the previous algorithm, we solve the MST problem using Boruvka's algorithm on the super-graph G', using the routing structure of Ghaffari et al. [30, 31] and the aggregate routing over R'. To implement the algorithm on the super-graph, we pipeline messages within all the super-nodes, i.e., inside the base fragments. There are $O(\log n)$ phases of Boruvka's algorithm and each phase takes

$$O(\tfrac{1}{\sqrt{\epsilon}}) \cdot 2^{O(\sqrt{\log n})} \text{ rounds.}$$

The extra term of $O(\frac{1}{\sqrt{\epsilon}})$ is incurred by communication within a super-node. Thus, in total, the second part takes $O(\frac{1}{\sqrt{\epsilon}} \cdot \log n) \cdot 2^{O(\sqrt{\log n})}$ rounds.

Combining the MST edges over the super-graph G', and the MST edges in each super-node, we have the MST for the original graph. Therefore, the total time complexity of the

algorithm is

$$O(\frac{\log n}{\sqrt{\epsilon}}) + O(\frac{\log n}{\sqrt{\epsilon}}) \cdot 2^{O(\sqrt{\log n})} = \tilde{O}(\frac{1}{\sqrt{\epsilon}}) \cdot 2^{O(\sqrt{\log n})}.$$

□

2.4 Lower Bound

In this section, we show the following lower bound result on the ϵ-smoothing model. We note that $\tilde{\Omega}(D + \sqrt{n})$ is an unconditional lower bound (without smoothing) that holds even for randomized Monte-Carlo approximate MST algorithms [61].

Theorem 15 (Smooth MST Lower Bound)**.** *There exists a family of graphs $\mathcal{G}$, such that, under the ϵ-smoothing model, any distributed MST algorithm must incur a running time of $\tilde{\Omega}(\frac{1}{\sqrt{\epsilon}})$, in expectation.*

We will prove the lower bound theorem by using the technique used in [61]. First, we will briefly recall the lower bound poof of $\tilde{\Omega}(\sqrt{n})$ (we assume $D = O(\log n)$) without smoothing. For purposes of exposition, we simplify and slightly modify the technique in the mentioned paper, to show only the bound for exact distributed MST. Then we extend it to the smoothing model. The procedure is to establish a chain of algorithm reductions which is the same as in [61], such that it relates distributed MST to a problem with a known lower bound. The following is the chain of reductions:

- Set Disjointness (SD) to Distributed Set Disjointness (DSD). We first reduce the set disjointness (SD) verification problem, a standard well-studied problem in two-party communication complexity to the problem of distributed set disjointness (DSD) verification. In the set disjointness problem (SD), we have two parties Alice and Bob, who

each have a k-bit string — $x = (x_1, x_2, \ldots, x_k)$ and $y = (y_1, y_2, \ldots, y_k)$, respectively. The goal is to verify if the set disjointness function is defined to be one if the inner product $\langle x, y \rangle$ is 0 (i.e., there is no i such that $x_i = y_i = 1$) and zero otherwise. The goal is to solve SD by *communicating as few bits* as possible between Alice and Bob.

In the distributed set disjointness (DSD) verification, the goal is to solve SD in a given input graph $G = (V, E)$, where two distinguished nodes $s, t \in V$ have the bit vectors x and y, respectively. In other words, instead of communicating directly as in the two-party problem (between Alice and Bob), the two nodes s and t (standing respectively for Alice and Bob) have to communicate via the edges of G (in the CONGEST model) to solve SD. The goal is to solve the DSD problem using *as few rounds* as possible in the CONGEST model (where only $O(\log n)$ bits per edge per round are allowed).

- Reduction of Distributed set disjointness (DSD) to *connected spanning subgraph (CSS) verification.* In the CSS problem, we want to solve a *graph verification* problem which can be defined as follows. In distributed graph verification, we want to efficiently check whether a given subgraph of a network has a specified property via a distributed algorithm. Formally, given a graph $G = (V, E)$, a subgraph $H = (V, E')$ with $E' \subseteq E$, and a predicate Π, it is required to decide whether H satisfies Π (i.e., when the algorithm terminates, every node knows whether H satisfies Π). The predicate Π may specify statements such as "H is connected" or "H is a spanning tree" or "H contains a cycle". Each vertex in G knows which of its incident edges (if any) belong to H.

  In the connected spanning subgraph (CSS) verification the goal is to verify whether the given subgraph H is connected and spans all nodes of G, i.e., every node in G is incident to some edge in H. The goal is to solve CSS in as few rounds as possible (in the CONGEST model).

- The last reduction is to reduce CSS verification to computing an MST.

The last two reductions above are done exactly as in the paper [61] that shows the time lower bound of $\tilde{\Omega}(D + \sqrt{n})$ rounds; however, the first reduction from SD to DSD is different in the smoothing model, as the input graph used in the DSD can use the (additional) power of the smoothing edges.

We will first briefly discuss the reductions as in [61] and then discuss how to modify the first reduction to work for the smoothing model. Here we first state the bounds that we obtain via these reductions, and refer to [61] for the details. The well-known communication complexity lower bound for the SD problem is $\Omega(k)$ (see e.g.,[45]), where k is the length of the bit vector of Alice and Bob. This lower bound holds even for randomized Monte-Carlo algorithms and even under *shared* public coin.

Due to the graph topology used in the DSD problem (see Figure 2.1 without the smoothing edges) the value of k is to be $\Theta(\sqrt{n})$ (which is the best possible). The reduction from SD to DSD shows that the lower bound for DSD is $\tilde{\Omega}(\sqrt{n})$ *rounds*. (Note that the diameter of the lower bound graph is $O(\log n)$ and hence subsumed.) This reduction uses the *Simulation Theorem* (cf. Theorem 3.1 in [61]) which will be explained later.

The reduction from DSD to CSS shows that the same time lower bound of $\tilde{\Omega}(\sqrt{n})$ rounds holds for the CSS problem. This reduction shows that the given subgraph H in the CSS problem is spanning connected if and only if the input vectors x and y are disjoint.

The reduction from CSS to the MST problem shows that the time lower bound for CSS verification which is $\tilde{\Omega}(\sqrt{n})$ also holds for the MST problem. This reduction takes as input the CSS problem and assigns weight 1 to the edges in the subgraph H and weight n to all other edges in G. It is easy to show that H is spanning connected if and only if the weight of the MST is less than n. Hence the same time lower bound that holds for CSS also holds

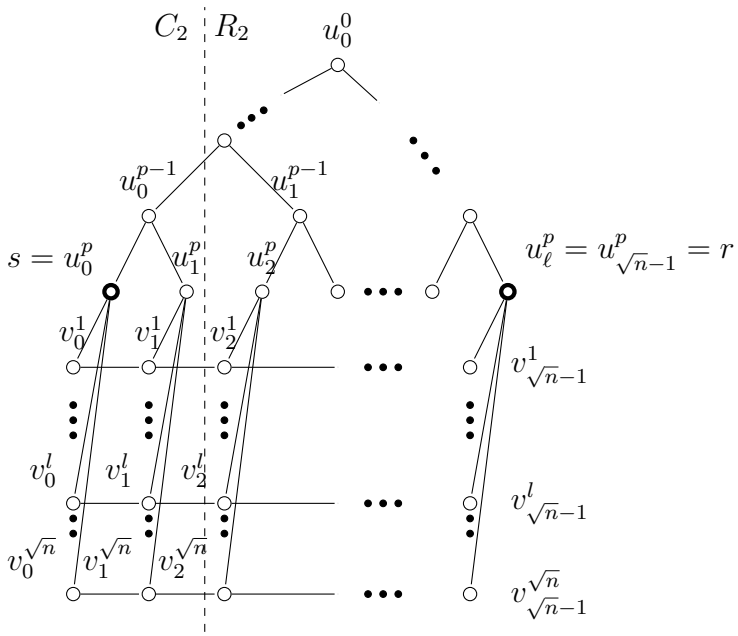

Figure 2.1: The lower bound graph.

for MST.

2.4.1 Reduction of SD to DSD: The Simulation Theorem

In this section, we explain the key reduction from SD to DSD which uses the *Simulation Theorem* as in [61]. The reduction idea is as follows. Assuming that we have an algorithm for DSD that finishes in r rounds, Alice and Bob will simulate this algorithm in the two party model by sending as few bits as possible. The Simulation theorem accomplishes this. The Simulation theorem in [61] that shows the simulation in the standard graph (without smoothing) uses a constant number of bits per round to do the simulation. Thus if the DSD algorithm finishes in r rounds, then Alice and Bob would have solved SD by exchanging $O(r)$ bits. If $r = o(k)$ (where k is the length of the input bit string), then this will contradict the lower bound of set disjointness in the two party model which is $\Omega(k)$ (even for randomized

algorithms).

We note that the reduction is similar to that in [61] as the input graph $G(x, y)$ for the DSD is the same (see Figure 2.1). We will first briefly describe the idea behind the Simulation Theorem as it applicable to G without the smoothing. The lower bound graph $G(x, y)$ used in the Simulation Theorem is described next .

The lower bound graph (family) for MST

$G(x, y)$ is depicted as in Figure 2.1, where $|x| = |y| = \sqrt{n}$. $G(.)$ includes $\sqrt{n}$ paths and a *complete binary tree* (to reduce the diameter to $O(\log n)$). Each *path* has length of $\sqrt{n}$; these are called the *path edges*. The full binary tree has $\sqrt{n}$ leaves, and hence, has the height of $p = \frac{\log n}{2}$. We number the leaves and the path nodes from left to right — $0, 1, \ldots, \ell$. Note that leaf numbered 0 is node s and leaf numbered ℓ is t. Consider each leaf $0 < j < \ell$, let it connect to all the nodes j in all paths; we call these *spoke* edges. Note that the binary tree edges, the path edges, and the spoke edges from leaf nodes other than s and t are present in every graph of the family.

It is straightforward that $G(x, y)$ has $\Theta(n)$ nodes and diameter $D = \Theta(\log n)$.

In the reduction from SD to DSD, Alice and Bob wants to solve SD problem with Alice having input vector $x = (x_1, x_2, \ldots, x_k)$ and Bob having input vector $y = (y_1, y_2, \ldots, y_k)$ where we fix $k = \sqrt{n}$. Depending on x and y, Alice and Bob will fix the spoke edges from nodes s and t. For the SD problem, Alice will add edges from s to the first node in path i if and only if $x_i = 0$. Similarly Bob will add edges from t to the last node in the path i if and only if $y_i = 0$.

Now we are ready to describe the Simulation Theorem which really gives an algorithm for Alice and Bob to simulate any given algorithm for the DSD problem. If the DSD algo-

rithm runs in r rounds, the Simulation Theorem will show how to solve the SD problem by exchanging $O(r)$ bits. We sketch the main idea here, which is quite simple, and omit the full details which can be found in [61].

How will Alice and Bob start the simulation? Note that they want to solve the SD problem on their respective inputs x and y in the two-party model. They will then use their respective inputs to construct $G(x, y)$ as described above. Note that Alice will be able to construct all edges of G except the spoke edges of t (since that depends on Bob's input) and vice versa for Bob. Then, assuming that there is an algorithm for the DSD problem on the same inputs that runs in r rounds, Alice and Bob will simulate the DSD algorithm whose output will also give the output for the SD problem (by definition).

The main idea is for Alice and Bob to keep the simulation going as long as possible. If one *disregards the binary tree edges*, all paths are of length $\ell = \Theta(\sqrt{n})$ and hence it is easy to keep the simulation going for $\ell/2$ rounds (say). This is because, Alice has all the information needed to simulate the DSD algorithm till $\ell/2$ steps (i.e., the middle of the path). Why? Because Alice knows her own input and all other nodes in G do not have any input. She does not know Bob's input, but this does not matter for $\ell/2$ steps since in so many rounds nothing from Bob's part of the graph reaches the "middle" of the path. But of course, the above is not true because of the binary tree edges which has smaller diameter. So to keep the simulation going for Bob, Alice sends the minimum amount of information needed by him. Note that after i rounds the computation from Alice's side (which are the set of nodes numbered u_0^p and $v_0^1, v_0^2, \ldots, v_0^{\sqrt{n}}$ will have reached nodes at distance i on the path.

We define the R_i (intuitively i-Right) set as follows: R_i includes all nodes on the paths with subscript $j \geq i$, all leaf nodes u_j^p where $j \geq i$, and all ancestor of these leaves, see Figure 2.1.

In round i, Bob needs to keep the correct computation for R_i. To achieve that, Alice sends *only the messages sent by the tree nodes in R_{i-1} crossing into the tree nodes in R_i* (see Lemma 3.4 in [61]). (A similar observation applies for Alice to do her simulation). Hence only messages sent by at most $O(\log n)$ nodes in the binary tree are needed. Hence in every round at most $O(\log^2 n)$ bits need to be exchanged by Alice and Bob to keep the simulation going. Thus if the DSD algorithm finishes in $o(\ell/\log^2 n) = o(\sqrt{n}/\log^2 n)$ rounds, then the simulation will also end successfully.

Thus we have shown the following Simulation Theorem.

Theorem 16 (Simulation Theorem)**.** *(Restatement of Theorem 3.1 in [61]) Given the DSD problem with input size $\Theta(\sqrt{n})$, encoded as $G(x, y)$ (i.e., the x and y are bit vectors of length $\Theta(\sqrt{n})$), if there is a distributed algorithm that solves the DSD problem in time at most T rounds, using messages of size $O(\log n)$. Then there is an algorithm in the two-party communication complexity model that decides the SD problem while exchanging at most $O(T \log^2 n)$ bits.*

2.4.2 Lower Bound with Smoothing

Under the ϵ-smoothing model, we will use the same lower bound graph $G(x, y)$. However, the smoothing model gives the algorithm additional power to add random edges in $G(x, y)$ during the course of the simulation. We will show how to modify the Simulation Theorem to apply to the smoothing model. Naturally, since the algorithm has additional power, it can finish faster, and hence the corresponding lower bound will be smaller.

We focus on Bob and show how he can keep his simulation going. (A similar argument applies for Alice.) We consider how Bob maintains correct states for R_i in round i. Besides the required messages as discussed in Theorem 16, Bob needs to know the messages sent

over (potential) *smoothed edges* crossing $V \setminus R_i$ into R_i. Since there are more messages to keep track, Bob cannot keep the simulation longer than that of the non-smoothing case. Let δ be the number of rounds the simulation is valid without exceeding $\Theta(\sqrt{n})$ bits of communication. Since the smoothing process is randomized, we bound δ in expectation.

We will use a pessimistic estimation to estimate the number of smoothed edges that "affect" Bob from Alice's side. Let $C_i = V \setminus R_i$, let S_i be the expected number of smoothed edges crossing C_i and R_i. S_i indicates the number of extra messages Bob needs to know, in round i. Notice that $|C_i| = i\Theta(\sqrt{n})$.

$$S_i = 2\epsilon i\Theta(\sqrt{n})\frac{(n - i\Theta(\sqrt{n}))}{n}. \tag{2.3}$$

Since $\delta < \Theta(\sqrt{n})$, for every round i, $S_i = \Theta(\epsilon\delta\sqrt{n})$. After δ rounds, the expected number of messages over the smoothed edges is thus: $\Theta(\epsilon\delta^2\sqrt{n})$. Also, by Theorem 16, we need to keep track of $(\delta \log n)$ messages. To stay within the budget of $\sqrt{n}$ bits for DSD communication, we require:

$$\epsilon\delta^2\Theta(\sqrt{n}) + \delta\Theta(\log n) \leq \frac{\sqrt{n}}{B}. \tag{2.4}$$

With $B = \Theta(\log n)$ (the message size), we have $\delta \leq \Theta(\frac{1}{\sqrt{\epsilon \log n}})$. Thus, we can keep the simulation going for up to $\Theta(\frac{1}{\sqrt{\epsilon \log n}})$ rounds.

To complete the lower bound argument, the same lower bound applies for MST in the ϵ-smoothing model by the chain of reductions.

2.4.3 Implication of the Lower Bound

Theorem 15 is stated for the MST algorithm, but is in fact applicable for many other fundamental algorithms. Due to the work established by Sarmar et al. [61], the same bound in Theorem 15 trivially holds for other problems: Shallow-light tree problem [60], s-source distance [24], shortest path tree problem [23], Minimum routing cost spanning tree problem [73], Minimum cut problem and Minimum s-t cut problem [25], Shortest s-t path problem [61], Generalized Steiner forest problem [61]. We note that, however, constructing matching algorithms for those problems is non-trivial.

2.5 Other Smoothing Models

In this section, we discuss some of the other plausible smoothing models for the distributed MST problem. The most natural smoothing model that first comes into mind in respect to a numerical-valued computing problem and that is similar in spirit to the original Spielman-Teng smoothing [64] is where one "perturbs" the edge-weights in the given input graph. However, as we have already noted (see Section 2.1) this may not make for anything interesting if the perturbations are too small with respect to the original edge-weights.

One is therefore tempted to make the perturbations *large* relative to the values of the original edge-weights. This, however, (depending on exactly how many edges are thus smoothed) may have the effect of essentially producing an input graph where the edge-weights are *randomly distributed*, i.e., where every edge-weight is chosen uniformly at random from a certain range of values. As has been shown previously [40], an MST can be constructed for such an input graph rather fast (in $\tilde{O}(D)$ rounds, actually, where D is the diameter of the input

graph).[7] Thus, while this model with large perturbations in edge-weights is more interesting than the smoothing model with small perturbations in edge-weights, it has been somewhat — even if not fully — explored in the distributed MST literature.

These considerations motivate us to explore an alternate avenue where the *graph topology* rather than the edge-weights is perturbed. In particular, we consider smoothing models where the perturbation process adds more edges to the original input graphs. Below we describe two such models where additional edges characterize the perturbed graph compared to the original input graph. We call these models the *k-smoothing models*. We distinguish these two models based on whether or not the smoothed edges are known to the algorithm.

Remark 14. *The usual smoothing concept (e.g., as originally put forth by Spielman and Teng [64], and as recently applied in the context of dynamic networks [19]) dictates that the new, "perturbed" graph be chosen according to some distribution (usually the uniform distribution) from a set of graphs that are "close" to the original input graph according to some distance metric. Thus the new, perturbed graph is presented to the algorithm as a whole and the algorithm has no way of knowing which edges are additional and which edges are original.*

Our ϵ-smoothing model, however, focused on the other scenario (where the algorithm does have such knowledge) because (1) it is algorithmically easier to approach and (2) also it may be relevant in somewhat different contexts too (e.g., in the context of congested clique *as discussed in Section 2.1).*

[7] The number of edges whose weights are smoothed can be parameterized, however. For example, one can consider a smoothing model where each edge in the original input graph is smoothed (i.e., its weight is perturbed) with a certain probability ϵ, say, for some small ϵ (e.g., ϵ can be made $\tilde{\Theta}(\frac{1}{\sqrt{n}})$).

2.5.1 The k-smoothing Model with Known Smoothed Edges

The main model that we follow in this chapter, the ϵ-smoothing model (see Section 2.2.1) adds smooth edges by node-local computation during the course of an algorithm. We can look at models where smooth edges are added all at once, by some external process, prior to the commencement of the algorithm.

1. Consider a smoothing model $\mathcal{M}(k, *)$ where the "perturbation" process *adds* k additional edges to the original input graph. In our notation, $*$ denotes the fact that the smoothed (i.e., additional) edges are known to the algorithm.

   These k additional edges are chosen uniformly at random from all the $\binom{n}{2}$ possible edges of the graph.

2. Consider another smoothing model $\mathcal{M}(\delta, *)$ where the "perturbation" process *adds* — independently for each of the $\binom{n}{2}$ possible edges — an edge with probability δ.

Remark 15. *It is not difficult to see that the model $\mathcal{M}(\epsilon, *)$ is essentially equivalent to the model $\mathcal{M}(k, *)$ for the case when $\epsilon = \frac{k}{n}$.*

Remark 16. *By Lemma 5, consider the ϵ-smoothing model with ℓ rounds of smoothing, then $\mathcal{M}(k = 2\ell\epsilon n, *)$ and $\mathcal{M}(\delta = \frac{2\ell\epsilon}{n}, *)$ are the equivalent models.*

2.5.2 The k-smoothing Model with Unknown Smoothed Edges

Essentially, we have the counterparts of $\mathcal{M}(k, *)$ and $\mathcal{M}(\delta, *)$ for the particular case when the smoothed edges are *not* known to the algorithm. We denote these models by $\mathcal{M}(k, \times)$ and $\mathcal{M}(\delta, \times)$. We also note that both $\mathcal{M}(k, \times)$ and $\mathcal{M}(\epsilon, \times)$ are essentially equivalent to the smoothing model proposed by Dinitz et al. [19], where the new, perturbed input graph

is chosen uniformly at random from the set of all possible graphs whose edge-sets are at a (positive) edit-distance k from the original input graph. The only subtle difference is that, in their model [19], the edit-distance can be positive as well as negative. We, however, consider only *positive* edit-distances here.

We note that the algorithms specified in this paper do not (at least directly) work when the smoothing edges are not known. However, note that the lower bound holds (for appropriate choice of k in terms of ϵ).

2.6 Conclusion

In this chapter, we studied smoothed analysis of distributed graph algorithms focusing on the well-studied distributed MST problem. Our work can be considered as a first step in understanding the smoothed complexity of distributed graph algorithms.

We presented a smoothing model, and upper and lower bounds for the time complexity of distributed MST in this model. These bounds quantify the time bounds in terms of the smoothing parameter ϵ. The bounds are within a factor of $2^{O(\sqrt{\log n})}\operatorname{polylog} n$ and a key open problem is whether this gap can be closed.

While we focused on one specific smoothing model, our results also apply to other related smoothing models (discussed in Section 2.5). A commonality among these models, besides adding random edges, is that the added edges are *known* to the nodes. This knowledge of the random edges is crucial to obtaining our upper bounds. Of course, our lower bounds apply *regardless of* this knowledge.

An important open problem is to show non-trivial bounds when the random edges are *unknown* to the nodes; i.e., the input graph consists of the original graph G plus the random edges and the nodes cannot distinguish between edges in G and the added random edges.

It would also be interesting to explore other fundamental distributed graph problems such as leader election, shortest paths, minimum cut, etc., in the smoothing model.

Chapter 3

Influence Maximization with Probabilistic Guarantees

In this chapter [1], we study a different network problem, influence spreading. The well-studied influence maximization problem involves choosing a seed set of a given size, which maximizes the expected influence. However, such solutions might have a significant probability of achieving low influence, which might not be suitable in many applications. We consider a different approach: find a seed set that maximizes the influence set size with a given probability. We show that this objective is not submodular and design a greedy, multi-criteria approximation algorithm for this problem with rigorous approximation guarantees. We also evaluate our algorithm on multiple datasets and show that they have similar or better quality as the ones optimizing the expected influence, but with additional guarantees on the probability.

[1] This is joint work with Maleq Khan, Gopal Pandurangan, Anil Vullikanti and Qin Zhang [41]

3.1 Introduction

A large number of phenomena, e.g., the spread of influence, fads, and ideologies on social networks, can be modeled as a diffusion process on a graph; see, e.g., [21, 39]. From a given seed set S (a subset of vertices), it is assumed that the influence spreads under a well-studied probabilistic diffusion model called the *Independent Cascade (IC)* model (see Sections 3.1.1 and 3.2.1); the size of the final influenced set, denoted $I(S)$, is taken as the *influence* of S. An optimization problem that has been extensively studied is the *influence maximization* problem, stated as follows: find a seed set S of size k, so that the expectation, $E[I(S)]$, is maximized—this is referred to as the MaxExpInf problem. The seminal work by Kempe, Kleinberg and Tardos [38] was the first to give a constant-factor approximation to the MaxExpInf problem. Later, Borgs et al. [7] improved the run time by proposing a nearly-optimal algorithm that had the same approximation guarantee. There has been a lot of work on many variants of influence maximization for several diffusion models; see, e.g., [39, 21] for details.

However, there are instances, in which the expected influence set is large, but the variance is also high, which is not desirable. A common way to understand the variance in a random variable is by examining its quantiles. Motivated by this, we focus on the problem of finding a seed set S of size k such that δ - the quantile value (for a given δ, $0 < \delta \leq 1$) of $I(S)$ is maximized — we denote this by $M_\delta(I(S))= \max\{\alpha | \Pr[I(S) \geq \alpha] \geq \delta\}$. The focus of this chapter is on finding S that maximizes $M_\delta(I(S))$; we refer to this as the MaxProbInf problem.

In this work, we study the structure of the MaxProbInf problem and develop efficient approximation algorithms for it. Our contributions are the following.

1. We formalize a natural notion of influence maximization with probabilistic guarantees

as the quantile value. We also study the structure of solutions of MaxProbInf and what effect the probabilistic guarantees have.

2. We develop a multi-criteria approximation algorithm, MultiCritMdelta, with approximation guarantees. We also design and analyze an efficient sampling method to estimate $M_\delta(I(S))$ for any given seed set S.

   A main technical novelty of our work is the analysis of our multi-criteria approximation algorithm. We use the *Sample Average Approximation* (SAA) technique from stochastic optimization (see, e.g., [66]), which involves constructing samples $G_1, \ldots, G_N$ of the graph, as per the *Independent Cascade* model. Since it is known that $M_\delta(I(S))$ is not submodular [74], we define a different type of submodular function $F_\lambda(S)$, using the *saturation* technique of [43]. We show that finding a minimum cost set S that ensures $F_\lambda(S) \geq \delta\lambda$ is sufficient to give a multi-criteria approximation—this problem is a variant of the standard submodular cover problem. However, the problem does not satisfy an important technical requirement in the submodular cover problem, and we need to modify the analysis of [72] to account for this difference.

3. Finally, we evaluate our methods on several datasets. Not surprisingly, the solutions to $M_\delta(I(S))$ computed using MultiCritMdelta have similar or higher (up to 10% in some cases) quantile values than the MaxExpInf solution. We also observe that the running time of MultiCritMdelta is significantly faster (about 10 times faster) compared to the standard algorithm that implements MaxExpInf (due to Kempe et al. [39]) and comparable to the run time of Borgs et al. [7] which is a nearly-optimal algorithm for influence maximization under the expectation measure. In fact, we also observe that our solutions have similar or better *expected influence value* than the

MaxExpInf solution (though this objective was not being optimized); additionally, we get bounds on the probability. In particular, we find that for many instances with low activation probabilities, the seed set output by MultiCritMdelta yields even better (by up to 10%) expected influence values than the MaxExpInf solution, computed using the approximation algorithm of [38] or [7] which both give a constant-factor approximation to the optimal expected influence.

3.1.1 Related Work

The works [38] and [39] were the first to formulate the problem of influence spreading as a discrete optimization problem. The authors consider two main diffusion models: *Independent Cascade* (which we adopt in this work) and *Linear Threshold*. Works in these models include two main optimization problems: maximizing the expected influence with constraints on the seed set (usually size), and minimizing the seed set (according to some measurement) to achieve a target expected influence. These problems are at least NP-hard [39]. In [14], it was shown that computing the expected influence is #P-hard.

The approach proposed by [38], using a submodular set function and greedy heuristic, gives a $(1 - 1/e - \epsilon)$-approximation to maximizing the expected influence problem. The error ϵ is due to Monte Carlo estimation of the expected influence, which is the main issue in practice, where large real world graphs discourage excessive sampling. Borgs et al. [7] propose an algorithm with a nearly optimal theoretical runtime of $O((m + n)k\epsilon^{-2} \log n)$ while retaining the same approximation guarantee. The technique is to sample *reversed* influence, which is adopted and improved upon in other works, e.g., [68], [67], [54]. While reversed influence sampling has a theoretical guarantee for the expectation problem, it is not extendable to our probabilistic $M_\delta(I(S))$. Another approach, proposed in [50], is to

estimate expected influence as a Riemann sum, using $O(n\epsilon^2\text{polylog}(n))$ samples which can be implemented in parallel by using MapReduce.

Other models are also studied, for example, fixed threshold models [12], [33], time-restricted diffusion model [34], [13], [18], continuous-time diffusion model [20], and diffusion in dynamic network [71].

The work closest to ours is presented by Zhang et al. [74]. They introduce the *Seed minimization with probabilistic coverage guarantee* (SM-PCG) problem: find the smallest seed set S that ensures that $\Pr[I(S) \geq \eta] \geq P$, where η and P are parameters. They also give an additive approximation to the minimum seed set size needed for given η, P, and show that the solutions achieve similar expected influence, but with the additional guarantee on the probability. They show that the size of the output seed set, compared to the optimal one, incurs both a multiplicative error of $(\ln n + O(1))$ and an additive error of $O(\sqrt{n})$, *under the assumption* that the standard deviation of the influence is $O(\sqrt{n})$. In contrast, our goal is different — we want to maximize influence with a probabilistic guarantee, with a constraint on the seed set size. Our multi-criteria approximation does not incur an additive error and does not rely on assumptions about the distribution. There are two limitations of their work. First, the additive approximation is $O(\sqrt{n})$, which can be quite large. In contrast, influence maximization has typically been studied for bounded seed set sizes, which, ideally, should be small. Second, the achievable influence η is not known a priori, and so the algorithm would have to be run for multiple η values.

3.2 Preliminaries

3.2.1 Model and Problem Definition

Consider a graph G with n nodes and m directed edges. This graph models a network of influence, where an edge (u, v) has a weight (probability) $0 \leq p(u, v) \leq 1$, indicating how likely u influences v. The problem of interest is to analyze how influence is propagated in the network. We use the well-studied *Independent Cascades (IC) model* with discrete time to model the spread of influence which we explain below [38].

At any given time t, the nodes have one of three states: *active, newly active, inactive.* At time t, let A_t be the set of active nodes, S_t be the set of newly active nodes, and the rest be inactive. Each node $u \in S_t$ can activate each of its inactive neighbors v with a probability $p(u, v)$. Let U_t be the set of nodes activated in this step. At time $t + 1$, $A_{t+1} = A_t \cup U_t$, and $S_{t+1} = U_t$. Starting from an initial configuration of a *seed set* $S = S_0$, we apply the process until time τ where $U_\tau = \varnothing$, indicating no new nodes can be activated. We denote $I(S) = |A_\tau|$ as the *influence* of the set S. More generally, we have a weight w_v for each node v, and $w(I(S)) = \sum_{v \in A_\tau} w_v$ is the total weight of the influence set $I(S)$. For simplicity, we will focus on the unweighted version of the problem; all our results hold for the weighted version as well, with natural changes in bounds. We note that $I(S)$ is a random variable that depends on S (and the underlying diffusion process).

As mentioned in Section 3.1, the MaxExpInf problem, maximizing $E[I(S)]$, is a coarse optimization. In particular, it does not give probabilistic guarantees on the influence of the chosen seed set. To get a better idea of $I(S)$, we may need to estimate the variance, or even to acquire the distribution of $I(\cdot)$. However, this task is usually quite difficult and costly.

Here, we propose another measure which could be more useful: given a threshold proba-

bility δ, find a seed set S such that the δ-quantile value of $I(S)$ is maximized. This measure gives direct probabilistic guarantees on the random variable $I(S)$.[2] More formally, for some set S, and a threshold δ, define the following measure:

$$M_\delta(I(S)) = max\{a|\Pr[I(S) \geq a] \geq \delta\}. \tag{3.1}$$

The new optimization problem, referred to as MaxProbInf is defined in the following manner: given an instance $(G = (V, E), B, w, \delta)$, find a (seed) set S such that we:

$$\begin{aligned} &\text{maximize} && M_\delta(I(S)) \\ &\text{subject to} && \sum_{i \in S} w_i \leq B. \end{aligned} \tag{3.2}$$

The goal of this work is to study the above optimization problem and design algorithms for it.

3.2.2 Comparison with MaxExpInf

As mentioned, the standard influence maximization problem, MaxExpInf, is defined as follows [39]: given an instance $(G = (V, E), k)$, find a set $S \subseteq V$ of size at most k such that $E[I(S)]$ is maximized.

A natural question is whether one could find a solution to the problem of maximizing $M_\delta(I(S))$, i.e., MaxProbInf by solving MaxExpInf. We show below that, in general, the solutions of these two problems can be quite different.

Lemma 17. *There exist instances (G, k, δ), for which $\frac{E[I(S^*)]}{M_\delta(I(S^*))}$ is arbitrarily large, where S^**

[2]Note that one can obtain a one-sided probability bound from expectation using Markov's inequality; but this is usually quite weak.

is an optimum solution to the MaxExpInf *problem.*

Proof. Consider the following graph $G = (V, E)$: we have r cliques $C_1, \ldots, C_r$. Let $|C_i| = C$ for all i. There is a vertex s which has edge (s, v_i) for a specific vertex $v_i \in C_i$. The edges in the cliques C_i all have probability 1. The edges (s, v_i) all have probability $p = 2/r$.

We consider $k = 1$. Then, $E[I(s)] = prC = 2C$. For a vertex $v \in C_i$ for any i, $E[I(v)] \leq C + p^2 rC \leq C + 2pC < 2C$ if $p < 1/2$, which happens if r is large enough. Therefore, the optimum solution to MaxExpInf$(G, 1)$ is $S^* = \{s\}$ and $E[I(S^*)] = 2C$.

Next, consider the MaxProbInf instance $(G, 1, 0.9)$. For $v \in C_i$, $I(v) \geq C$. However, note that $\Pr[I(s) = 1] = (1 - p)^r = (1 - \frac{2}{r})^r > 0.1$ for $r \geq 10$. Therefore, for $\delta = 0.9$, $M_\delta(I(S^*)) = 1$. This implies for the instance $(G, 1, 0.9)$, the solution computed using MaxExpInf (ignoring δ) is arbitrarily bad, compared to the optimum solution to the MaxProbInf objective. □ □

3.2.3 Hardness

We observe that MaxProbInf is NP-hard to approximate within a factor of $(1 - 1/e)$, which is similar to the hardness of the MaxExpInf problem.

Lemma 18. *It is NP-hard to obtain an approximate solution to the* MaxProbInf *problem, within a factor of* $(1 - 1/e)$.

Proof. The proof is by a reduction from MAXIMUM COVERAGE. An instance of this problem is a ground set $U = \{u_1, \ldots, u_n\}$, a collection of subsets $S_1, \ldots, S_m$ of U, and parameter k. The objective is to select a set of k subsets, whose union has the maximum size – this is NP-hard to approximate within a factor of $(1 - 1/e)$.

We construct the same reduction as in [38]. We construct the graph $G = (V, E)$ in the following form. We have $V = L \cup R$, where L has m nodes, corresponding to the subsets,

and R has n nodes, corresponding to the elements in U. If $u_i \in S_j$, there is a directed edge from the node corresponding to S_j in L to the node corresponding to u_i in R. All edges have diffusion probability 1.

Therefore, for a subset $S \subseteq L$ of seeds, $I(S)$ equals the union of the corresponding subsets. We choose $\delta = 1$. Therefore, there is a set S of size k with $|I(S)| \geq C$ with probability $\delta = 1$ if and only if there is a solution to COVERAGE of size at least C. □

3.3 A Multi-criteria Approximation Algorithm for Computing MaxProbInf

Motivated by the hardness from Lemma 18, and the non-submodularity of $M_\delta(I(S))$ [74], we consider a multi-criteria approximation algorithm, which relaxes the seed set size k, as well as the probability parameter δ. Specifically, we consider the following variant of the MaxProbInf problem: find S such that $M_\delta(I(S)) \geq \gamma M_{\delta'}(I(S^*))$ and $|S| \leq \beta k$, where S^* is an optimal solution, and $\gamma < 1, \beta > 1$ are the relaxation parameters, and $\delta' > \delta$.

3.3.1 Using Submodularity and the Sample Average Approximation Technique

MaxProbInf is a stochastic optimization problem. We reduce this to a deterministic problem using the *sample average approximation* technique: assume we have N samples $G_1, \ldots, G_N$ of the graph $G = (V, E)$; $G_i = (V, E_i)$ is the ith sample, and is obtained by picking each edge $e \in E$ independently with probability $p(e)$. Let $F_i(S)$ denote the total influence due to S in graph G_i – this equals the sum of the sizes of the components containing nodes in S in sample G_i. For any $S \subseteq V$, let $h_\delta(S)$ be the δ-quantile value estimated from

these samples in the following manner: suppose the samples are ordered so that $F_1(S) \geq \ldots \geq F_N(S)$. Then $h_\delta(S) = F_{\lfloor \delta N \rfloor}(S)$. Let S^* be the optimum solution that maximizes $M_\delta(I(S))$.

Lemma 19. *Given $\delta \in (0,1)$, then for any $\epsilon \in (0,1)$, there exists $N = \Omega(n \log n)$ such that $h_\delta(S) \in [M_{\delta(1+\epsilon)}(I(S)), M_{\delta(1-\epsilon)}(I(S))]$, for all $S \subseteq V$, with high probability.*

Proof. Let us fix some set S. Let L be the number of F_i greater than the $(1-\epsilon)\delta$ quantile, and let R be the numberer of F_i greater than the $(1+\epsilon)\delta$ quantile. We have: $\mu_L = E[L] = (1-\epsilon)\delta N$ and $\mu_R = E[R] = (1+\epsilon)\delta N$. $h_\delta(S)$ will fall outside the desired range if either $L > (1+\epsilon)\delta N$ or $R < (1-\epsilon)\delta N$. Let $\mathcal{E}_S$ indicate such a failure event. The probability of failure is:

$$Pr[\mathcal{E}_S] \leq Pr[L > \mu_L(1 + \frac{2\epsilon}{1-\epsilon})] + Pr[R < \mu_R(1 - \frac{2\epsilon}{1+\epsilon})].$$

Applying a Chernoff bound yields

$$Pr[\mathcal{E}_S] \leq \left(\frac{e^{2\epsilon/(1-\epsilon)}}{(\frac{1+\epsilon}{1-\epsilon})^{\frac{1+\epsilon}{1-\epsilon}}} \right)^{(1-\epsilon)\delta N} + \left(\frac{e^{-2\epsilon/(1+\epsilon)}}{(\frac{1-\epsilon}{1+\epsilon})^{\frac{1-\epsilon}{1+\epsilon}}} \right)^{(1+\epsilon)\delta N}.$$

Since the bases of the exponents are in the range $(0,1)$ for $0 < \epsilon < 1$, we have:

$$Pr[\mathcal{E}_S] \leq e^{-cN} \quad \text{for some } c > 0. \tag{3.3}$$

With 2^n possible sets S, let $\mathcal{E}_{2^n}$ indicate the event that at least one set has the wrong $h_\delta(.)$ estimation. By the union bound:

$$Pr[\mathcal{E}_{2^n}] \leq e^{-cN} 2^n = e^{-cN + n \ln 2}. \tag{3.4}$$

Thus, when $N = \Omega(n \log n)$, with probability of $1 - O(n^{-1})$, all $h_\delta(S)$ are correctly estimated.

□

It turns out approximating the function $h_\delta(S)$ directly is hard; in particular, it is not submodular, and thus a greedy strategy might not work. Instead, we will consider a slight variant, using ideas from the result of Krause et al. [43] on the minimum cost submodular cover problem. Let λ be a "guess" of $M_\delta(I(S^*))$, where S^* is an optimal solution. Define the truncated function $F_{i,\lambda}(S) = \min\{F_i(S), \lambda\}$, and

$$F_\lambda(S) = \frac{1}{N} \sum_i F_{i,\lambda}(S).$$

A critical observation is that the truncated functions are submodular, formally:

Lemma 20. *(see [28]) The functions $F_{i,\lambda}(S)$ and $F_\lambda(S)$ are monotone and submodular.*

We show that maximizing $F_\lambda(S)$ is good enough for our purpose, due to the following property.

Lemma 21. *If $h_\delta(S) \geq \lambda$, then $F_\lambda(S) \geq \delta\lambda$. On the other hand, if $F_\lambda(S) \geq \delta\lambda$, then $h_{\delta/2}(S) \geq \delta\lambda/2$.*

Proof. The first part: let $A = \{i : F_{i,\lambda}(S) \geq \lambda\}$. Since $h_\delta(S) \geq \lambda$, it follows that $|A| \geq \delta N$. This implies

$$\begin{aligned} F_\lambda(S) = \frac{1}{N} \sum_i F_{i,\lambda}(S) &\geq \frac{1}{N} \sum_{i \in A} F_{i,\lambda}(S) \\ &\geq \frac{1}{N}(\lambda \delta N) = \lambda\delta. \end{aligned}$$

The second part: let $B = \{i : F_{i,\lambda}(S) \geq \delta\lambda/2\}$. We have:

$$\lambda|B| + \sum_{i \notin B} \delta\lambda/2 \geq \sum_{i \in B} F_{i,\lambda}(S) + \sum_{i \notin B} F_{i,\lambda}(S)$$

$$= F_\lambda(S)N \geq \delta\lambda N.$$

This implies

$$\lambda|B| + (N - |B|)\delta\lambda/2 \geq \delta\lambda N,$$

so that

$$|B|\lambda(1 - \delta/2) \geq N\delta\lambda/2.$$

Therefore,

$$|B| \geq \frac{N\delta/2}{1 - \delta/2} \geq N\delta/2.$$

This implies $h_{\delta/2}(S) \geq \delta\lambda/2$. □ □

Lemma 21 motivates the following strategy in order to maximize $h_\delta(\cdot)$ (which is sufficient for solving MaxProbInf, due to Lemma 19): find S such that $F_\lambda(S) \geq \delta\lambda$ and $\text{cost}(S) = \sum_{j \in S} w_j$ is minimized, where w_j is the weight of node j. This is a variant of the minimum cost submodular cover problem [27], which involves finding a minimum cost set S such that $F(S) = F(V)$ for a submodular function $F(\cdot)$, where V is the universe.

Algorithm 1 describes SimpleGreedy, which guesses λ and greedily computes a set S_λ. This algorithm is the same as that for the standard minimum cost submodular cover problem, except that the stopping condition is $F_\lambda(S_\lambda) \geq \delta\lambda$, instead of $F_\lambda(S_\lambda) = F_\lambda(V)$. We show, however, that the same guarantee can be obtained by appropriately modifying the analysis of the minimum cost submodular cover problem.

Algorithm 1 Algorithm SIMPLEGREEDY

```
1: function SimpleGreedy(G(V, E), w, δ', n, N)
2:     for each guess λ do
3:         S_λ ← ∅
4:         C ← δ'λ
5:         while F_λ(S_λ) < C do
6:             Pick node j that minimizes w_j / (F_λ(S_λ ∪ {j}) − F_λ(S_λ))
7:             S_λ ← S_λ ∪ {j}
       return S_λ = S_{λ,δ'} that maximizes F_λ(S_λ)
```

For a constant $\delta' \in (0,1)$, let $S^*_{\delta'} = \text{argmax}_{S,cost(S)\leq k} h_{\delta'}(S)$, and $\lambda^*_{\delta'} = h_{\delta'}(S^*_{\delta'})$.

Lemma 22. *Let $S^*_{\delta'}$ and $\lambda^*_{\delta'}$ be as defined above. Let set S_λ be the solution returned by algorithm* SIMPLEGREEDY *for some λ, when the probability parameter is set to δ'. Then, we have: (1) $\lambda \geq \lambda^*_{\delta'}$, (2) $F_\lambda(S_\lambda) \geq \lambda^*_{\delta'}\delta'$, and (3) $cost(S) = O(\ln(N\lambda))cost(S^*)$.*

Our proof of Lemma 22 is a variation of the proof by Wolsey [72]. For notational simplicity, we drop the subscript λ in $F_\lambda(S)$ below. Let $\rho_j(S) = F(S \cup \{j\}) - F(S)$. First observe that the following IP is feasible: (IP) $\min \sum_j w_j x_j$ such that for all $S \subseteq V$, with $x_j \in \{0,1\}$ we have $\sum_{j \notin S} \rho_j(S) x_j \geq C - F(S)$.

Lemma 23. *The program (IP) is valid, i.e., the optimum solution S_{IP} satisfies $F(S_{IP}) \geq C$ and S_{IP} has the minimum cost.*

The dual program of the linear relaxation of (IP) is the following

$$\begin{aligned} \text{(D) } \max \quad & \sum_S (C - F(S)) y_S \quad \text{such that} \\ \sum_{S: j \notin S} \rho_j(S) y_S \quad & \leq \quad w_j \text{ for all } j \\ y_S \quad & \geq \quad 0. \end{aligned}$$

Observe that the algorithm actually picks element j which minimizes $\frac{w_j}{\rho_j(S)}$. We construct an approximate dual solution. Suppose the greedy algorithm picks elements $\{j_1, \ldots, j_\ell\}$. Let

$S_i = \{j_1, \ldots, j_i\}$. Define

$$\theta_i = \begin{cases} \frac{w_{j_i}}{F(S_i)-F(S_{i-1})} = \frac{w_{j_i}}{\rho_{j_i}(S_{i-1})}, \text{ for } i < \ell, \\ \frac{w_{j_\ell}}{C-F(S_{i-1})}, \text{ for } i = \ell. \end{cases}$$

Define

$$y_S = \begin{cases} \theta_1, \text{ if } S = S_0 = \emptyset, \\ \theta_{i+1} - \theta_i, \text{ if } S = S_i \text{ for } 0 < i < \ell, \\ 0, \text{ otherwise.} \end{cases}$$

Observe that the dual solution y_{S_i} covers the cost of the solution S_ℓ:

$$\begin{aligned} \sum_S (C - F(S)) y_S &= \sum_{i=0}^{\ell-1} (C - F(S_i)) y_{S_i} \\ &= \sum_{i=0}^{\ell-1} (C - F(S_i))(\theta_{i+1} - \theta_i) \\ &= \sum_{i=1}^{\ell-1} \theta_i (F(S_i) - F(S_{i-1})) + \theta_\ell (C - F(S_{\ell-1})) \\ &= \sum_{i=1}^{\ell-1} w_{j_i} + w_{j_\ell} = \text{cost}(S_\ell). \end{aligned}$$

Next, we observe that the dual constraints are approximately feasible. First, consider any

$j \in V - S_\ell$. We have

$$\begin{aligned}
\sum_{S: j \notin S} \rho_j(S) y_S &= \sum_{i=0}^{\ell} \rho_j(S_i) y_{S_i} \\
&= \rho_j(S_0)\theta_1 + \sum_{i=1}^{\ell-1} \rho_j(S_i)(\theta_{i+1} - \theta_i) \\
&= \sum_{i=1}^{\ell-1} \theta_i(\rho_j(S_{i-1})) + \theta_\ell \rho_j(S_{\ell-1}) \\
&\leq \sum_{i=1}^{\ell-1} w_j \frac{\rho_j(S_{i-1}) - \rho_j(S_i)}{\rho_j(S_{i-1})} + w_j,
\end{aligned}$$

because $\theta_i \leq \frac{w_j}{\rho_j(S_{i-1})}$ by construction, for each $i \leq \ell$.

For $x \in (0, \rho_j(\emptyset))$, define $h(x) = \frac{1}{\rho_j(S_{i-1})}$, if $\rho_j(S_i) < x \leq \rho_j(S_{i-1})$. Let

$$\delta = \min_{S: \rho_j(S) > 0} \rho_j(S) = \min_{S: \rho_j(S) > 0} F(S + j) - F(S) \geq \frac{1}{N}.$$

Therefore,

$$\begin{aligned}
&\sum_{i=1}^{\ell-1} \frac{\rho_j(S_{i-1}) - \rho_j(S_i)}{\rho_j(S_{i-1})} = \int_0^{\rho_j(\emptyset)} h(x)dx \\
&\leq \int_0^{\delta} \frac{1}{\delta} dx + \int_{\delta}^{\rho_j(\emptyset)} \frac{1}{x} dx = 1 + \ln \frac{\rho_j(\emptyset)}{\delta} \leq 1 + \ln(N\lambda),
\end{aligned}$$

since $\rho_j(\emptyset) \leq F(j) \leq \lambda$. This implies

$$\sum_{S: j \notin S} \rho_j(S) y_S \leq (2 + \ln(N\lambda)) w_j.$$

Next, suppose $j \in S_r$ for some r. Then, $\rho_j(S_r) = F(S_{r+1}) - F(S_r) = 0$. Let $r' < r$ be the

largest index such that $\rho_j(S_{r'}) > 0$. In that case, the dual constraint for j can be written as

$$\begin{aligned}
&\sum_{S:j\notin S} \rho_j(S) y_S = \sum_{i=0}^{r'} \rho_j(S_i) y_{S_i} \\
&= \sum_{i=1}^{r'+1} \theta_i(\rho_j(S_{i-1}) - \rho_j(S_i)) \\
&\leq \sum_{i=1}^{r'+1} w_j \frac{\rho_j(S_{i-1})\rho_j(S_i)}{\rho_j(S_{i-1})} \leq 1 + \ln(N\lambda),
\end{aligned}$$

as before. Therefore, $\frac{1}{\alpha} y$ is a feasible solution, for $\alpha = 2 + \ln N\lambda$, which completes the proof for Lemma 22. □

Finally, we put everything together to obtain an approximation to the MaxProbInf problem.

Theorem 24. *Let $\delta \in (0,1)$ be a constant, and let k be a parameter. Let S^{opt}_δ denote an optimum solution to the* MaxProbInf *problem for parameter δ, i.e., $M_\delta(I(S^{opt})) \geq M_\delta(I(S'))$, for all $|S'| \leq k$. For any $\epsilon \in (0,1)$, there exists $N = \Omega(n \log n)$ such that if set S is the solution computed by algorithm* SimpleGreedy *with parameter $\delta' = 2\delta/(1-\epsilon)$, we have $M_\delta(I(S)) \geq \frac{\delta}{1-\epsilon} M_{2\delta(1+\epsilon)/(1-\epsilon)}(I(S^{opt}_{2\delta/(1-\epsilon)}))$, and $|S| = O(k \log n)$ The running time of* SimpleGreedy *is $O(n^2 mk \log^2 n)$.*

Proof. Using Lemma 22, for the unweighted case, $cost(S) = |S|$. Let $S^*_{\delta'} = \text{argmax}_{S,|S|\leq k} h_{\delta'}(S)$, and $\lambda^*_{\delta'} = h_{\delta'}(S^*_{\delta'})$ defined similarly. We have $\lambda \geq \lambda^*_{\delta'}$, and $F_\lambda(S') \geq \lambda\delta'$. By Lemma 21, this implies that $h_{\delta'/2}(S) \geq \lambda\delta'/2 \geq \lambda^*_\delta \delta'/2 = \frac{\delta'}{2} h_{\delta'}(S^*_{\delta'})$.

By Lemma 19, we have $M_{\delta'(1-\epsilon)/2}(I(S)) \geq h_{\delta'/2}(S)$. Next, $h_{\delta'}(S^*_{\delta'}) \geq h_{\delta'}(S^{opt}_{\delta'})$, by definition of $S^*_{\delta'}$. Again, by Lemma 19, we have $h_{\delta'}(S^{opt}_{\delta'}) \geq M_{\delta'(1+\epsilon)}(I(S^{opt}_{\delta'}))$. Combining all these, we get

$$M_{\delta'(1-\epsilon)/2}(I(S)) \geq \frac{\delta'}{2} M_{\delta'(1+\epsilon)}(I(S^{opt}_{\delta'})).$$

Setting $\delta' = 2\delta/(1-\epsilon)$, we get the approximation guarantee

$$M_\delta(I(S)) \geq \frac{\delta}{1-\epsilon} M_{2\delta(1+\epsilon)/(1-\epsilon)}(I(S^{opt}_{2\delta/(1-\epsilon)})).$$

The algorithm may need to check all n possible values of λ, each check constructs a set of size $O(k \log n)$. The number of samples $N = \Omega(n \log n)$ is sufficient for high probability of success, by adjusting the union bound from equation 3.4:

$$Pr[Failure] \leq e^{-cN} n 2^n k \log n = k e^{-cN + \ln n + n \ln 2 + \ln \log n} = 1 - O(n^{-1}).$$

The complexity for one sample is $O(m)$, and the total time complexity is: $O(Nnmk \log n) = O(n^2 mk \log^2 n)$.

□

3.3.2 Fast Implementation of Algorithm MultiCritMdelta.

The pseudocode is shown in Algorithm 2. To improve the running time, we use binary search as follows. With a given λ, an iteration decides if it is feasible to construct S such that $F_\lambda(S) = C = \delta\lambda$, with a log factor constraint on the size of S. If it is the case, λ is a feasible value, and S is a feasible solution for $maxF_\lambda(\cdot)$ problem. Starting with $\lambda = n/2$, the binary search increases or decreases λ as per the feasibility. The output is a feasible value and the respective set. Lemma 22 shows the approximation guarantee of this algorithm.

Algorithm 2 k-Influence set by MULTICRITMDELTA

function MULTICRITMDELTA($G(V,E), k, \delta, n, N$)
  $l \leftarrow 1$
  $h \leftarrow n$
  for $step \leftarrow [1 \dots \log n]$ **do**
    $S \leftarrow \emptyset$
    $\lambda \leftarrow (l+h)/2$
    $C \leftarrow \delta\lambda$, and $F(S) \leftarrow F_\lambda(S)$
    while $F(S) < C$ **do**
      Pick node j that minimizes $\frac{w_j}{F(S\cup\{j\})-F(S)}$
      $S \leftarrow S \cup \{j\}$
    feasible if $|S| \leq k\ln(N\lambda)$: $l \leftarrow \lambda$
    infeasible if $|S| > k\ln(N\lambda)$: $h \leftarrow \lambda$
  return S

3.4 Computing $M_\delta(I(S))$ for a Fixed Set S

In this section, we show how to compute an approximation of $M_\delta(I(S))$ for a *fixed* set S, efficiently. It is known that computing $M_\delta(I(S))$ exactly even for a fixed set S is #P-Hard [74]. However, we show that we can estimate $M_\delta(I(S))$ by using Monte-Carlo sampling.

3.4.1 Monte-Carlo Approximation of $M_\delta(.)$

Firstly, it will be useful to consider the following general problem of estimating $M_\delta(.)$ for a random variable that takes values between 0 and 1 (both included).

$$\begin{array}{ll} 0 \leq X \leq 1 & \text{a random variable,} \\ 0 < \delta \leq 1 & \text{a probability threshold,} \\ \text{find} & M_\delta(X) = sup\{a|\Pr[X \geq a] \geq \delta\}. \end{array} \tag{3.5}$$

We define $M_\delta(X)$ in a way that is valid for both discrete and continuous distributions. Indeed, let F_X be the cumulative distribution of X; if F_X is continuous, we can define: $M_\delta(X) = sup\{a|\Pr[X \geq a] = \delta\}$, and the solution is given by: $M_\delta(X) = F_X^{-1}(1-\delta)$.

In general, we cannot afford finding the distribution function, thus, we apply a Monte Carlo sampling to give an (ν, ϵ, ζ)-approximation of $M_\delta(X)$.

Definition 17. *$\widetilde{M}_\delta(X, \nu, \epsilon, \zeta)$ is an (ν, ϵ, ζ)-approximation of $M_\delta(X)$ if the following inequality holds, with (confidence) probability $1 - \zeta$,*

$$M_{\delta+\eta}(X) - \nu \leq \widetilde{M}_\delta(X, \nu, \eta, \zeta) \leq M_{\delta-\eta}(X) + \nu. \tag{3.6}$$

Notice that both the approximation errors η and ν are additive, i.e., absolute values. In the general setting, an additive error is less desirable than a multiplicative error, since we do not know the order of the estimated parameter in advance. We need the absolute errors for the design and analysis of our efficient sampling method, and we observe that this is not an issue for the specific influence problem. F irst, δ is a given constant, and we can convert between multiplicative and additive errors, $\epsilon = \frac{\eta}{\delta}$, or $\eta = \delta\epsilon$. Second, for the influence problem, X is a discrete random variable in $[0, n]$, thus we can set $\nu = \frac{1}{n}$ to have a tighter bound.

We now describe the estimation by sampling. The idea is to perform binary search to find the best value of a, the approximation. Initially, we guess $a = 1/2$, then verify if $\Pr(X \geq a) \geq \delta$. If the test is confirmed, we make the next guess as $a = 3/4$, otherwise, we guess $a = 1/4$. Continue until the step of the guess is smaller than ν.

Consider one iteration, with some fixed a; we need an estimation for the unknown probability $p = \Pr(X \geq a)$. We proceed similarly to the proof of Lemma 19, with a different form of Chernoff bound, which is more convenient to bound the number of samples. Take N samples $X_1, X_2, \cdots, X_N$, where N will be specified later. Define N indicator random variables Z_i, where $Z_i = 1$ if the sample $X_i \geq a$, 0 otherwise. Let $Z = \sum_{i=1}^{N} Z_i$. We have

$\mu_Z = \mathrm{E}[Z] = Np$. Using a Chernoff bound, we bound the empirical probability $\bar{p} = \frac{Z}{N}$, within some multiplicative error κ:

$$\begin{aligned} &\Pr(|Z - Np| \geq \kappa Np) \leq 2exp\left(-\frac{\kappa^2}{2+\kappa}Np\right) \\ \iff &\Pr\left(|\bar{p} - p| \geq \kappa p\right) \leq 2exp\left(-\frac{\kappa^2}{2+\kappa}Np\right). \end{aligned}$$

With the desired error $\eta = \kappa p \iff \kappa = \frac{\eta}{p}$, the tail bound becomes:

$$\Pr\left(|\bar{p} - p| \geq \eta\right) \leq 2exp\left(-\frac{\eta^2}{2p+\eta}N\right) \leq 2exp\left(-\frac{\eta^2 N}{3}\right). \tag{3.7}$$

Given $|\bar{p} - p| \geq \eta$, clearly we have: $\bar{p} - \eta \geq \delta \implies p \geq \delta$, and $\bar{p} + \eta < \delta \implies p < \delta$. This method leaves an undecided region when $\bar{p} - \eta < \delta \leq \bar{p} + \eta$. As we allow an η error around δ, we can simplify the decision rule:

$$\begin{aligned} &\text{If } \bar{p} \geq \delta \implies \text{increase } a; \\ &\text{If } \bar{p} < \delta \implies \text{decrease } a. \end{aligned} \tag{3.8}$$

Algorithm 3 shows the pseudo code for the approximation. The following lemma states the guarantee bound as per Equation (3.6).

Lemma 25. *Assuming Algorithm 3 is successful, it returns an (ν, η, ζ)-approximation of $M_\delta(X)$.*

Proof. The decision rule in Equation (3.8) ensures this invariance: if the guess increases then at least $p \geq \delta - \eta$, if the guess decreases then at least $p \leq \delta + \eta$. This invariance holds for every iteration. At the final iteration, the bounds are offset by $\pm\nu$ by the precision of the binary search, taking the conservative case, we have the approximation (3.6). □

The next lemma states the number of samples required for succeeding with probability at least $1 - \zeta$.

Lemma 26. *Algorithm 3 is successful with probability of at least* $(1 - \zeta)$ *using*

$$N \geq \frac{3}{\eta^2} \log \nu^{-1} (\ln(\log \nu^{-1}) + \ln(2\zeta^{-1})) \quad \textit{samples.}$$

Proof. Let N_{iter} be the number of samples in one iteration. From the bound in Equation (3.7), using a union bound for $\log \frac{1}{\nu}$ iterations, we require:

$$\begin{aligned} & 2exp\left(-\frac{\eta^2 N_{iter}}{3}\right) \log \frac{1}{\nu} \leq \zeta \\ \iff & \frac{\eta^2 N_{iter}}{3} \geq \ln \log \frac{1}{\nu} + \ln \frac{2}{\zeta}. \end{aligned}$$

Thus the total number of samples is:

$$N = N_{iter} \log \frac{1}{\nu} \geq \frac{3}{\eta^2} \log \frac{1}{\nu} \left(\ln \log \frac{1}{\nu} + \ln \frac{2}{\zeta} \right).$$

□

3.4.2 Approximation of $M_\delta(I(.))$

We now apply the binary search to the $M_\delta(I(.))$ approximation. Since influence is a discrete random variable bounded by the size n of the graph, we set $\nu = 1/n$, to obtain the following corollary.

Corollary 27. *Given a graph with* n *nodes, and seed set* S*, for given parameters* δ*,* η*, and* ζ*. Using* $N \geq \frac{3}{\eta^2} \log n (\ln \log n + \ln(2\zeta^{-1}))$ *samples, Algorithm 3 finds an approximation of*

Algorithm 3 Approximate $M_\delta(X)$ with confidence $(1-\zeta)$

```
1: function MDelta(X[0,1], δ, ν, η = εδ, ζ)
2:     N ← (3/η²) log ν⁻¹ (ln(log ν⁻¹) + ln(2ζ⁻¹))
3:     l ← ν
4:     h ← 1
5:     while h − l ≥ ν do
6:         mid = (h + l)/2
7:         X_1, ··· , X_N ← samples of X
8:         p̄ = (1/N)(#X_i, X_i ≥ mid)
9:         if p̄ ≥ δ then l ← mid
10:        else h ← mid
       return l
```

$M_\delta(I(S))$ with success probability $1-\zeta$, such that: $M_{\delta+\eta}(I(S)) \leq \widetilde{M}_\delta(I(S)) \leq M_{\delta-\eta}(I(S))$.

Proof. Direct application of Lemma 25 with $\nu = 1/n$. □

As an example, for the approximation to succeed with high probability, we set $\zeta = 1/n$, and the number of samples required will be $N = O(\log^2 n)$.

Combining the binary search method with the multicriteria approximation, we have the MultiCritMdelta as listed in Algorithm 2, with the following approximation guarantee and runtime complexity.

Theorem 28. *Let $\delta \in (0,1)$ be a constant, and let k be a parameter. Let S^{opt}_δ denote an optimum solution to the* MaxProbInf *problem for parameter δ, i.e., $M_\delta(I(S^{opt})) \geq M_\delta(I(S'))$, for all $|S'| \leq k$. For any $\epsilon \in (0,1)$, there exists $N = O(\log^2 n)$ such that, if set S is the solution computed by algorithm* MultiCritMdelta *with parameter $\delta' = 2\delta/(1-\epsilon)$, we have, with high probability, $M_\delta(I(S)) \geq \frac{\delta}{1-\epsilon} M_{2\delta(1+\epsilon)/(1-\epsilon)}(I(S^{opt}_{2\delta/(1-\epsilon)}))$, and $|S| = O(k \log n)$. The running time of* MultiCritMdelta *is $O(mk \log n(\ln n + \ln\ln n))$.*

Proof. Let $\mathcal{A}_{S,\lambda}$ denote the event that the probability $\Pr[I(S) \geq \lambda]$ can be estimated with absolute error $\eta = \delta\epsilon$. The correctness of the algorithm depends on the validation of the

feasible λ, which requires all events $\mathcal{A}_{S,\lambda}$ to succeed for all S, λ during the course of the algorithm.

Assume $M = o(n)$, the number of samples taken in the for-loop of Algorithm 2. By the greedy construction of the feasible S, the algorithm needs to check $O(n)$ candidate sets. The number of iterations is the relaxed size of S, $|S| = O(k \ln(M\lambda)) = O(k \ln(n^2))$. Thus the total number of events is: $O(2nk \ln n \log n)$. From Equation (3.7), we take a union bound for the failure of all events:

$$2exp(-\frac{\eta^2 M}{3})O(2nk \ln n \log n) \leq \zeta.$$

For success w.h.p., we set: $\zeta = 1/n$

$$-\frac{\eta^2 M}{3} \geq \ln n + O(\ln n + \ln \log n)$$
$$\implies M = O(\frac{3}{\eta^2}(\ln n + \ln \log n)).$$

We verify that $M = o(n)$ as previously assumed. Thus the total number of samples is: $N = O(\frac{3}{\eta^2} \log n(\ln n + \ln \log n)) = O(\frac{3}{\epsilon^2 \delta^2} \log n(\ln n + \ln \log n))$. The running time is dominated by the complexity of the sampling, which is $Nm = O(mk \log n(\ln n + \ln \log n))$ where m is the number of edges. □

3.5 Empirical Evaluation

We examine the empirical performance of MultiCritMdelta on real world social networks. For a base line method, we also implement ProbInf-Heu, which simply constructs the seed set greedily, using Algorithm 3 to maximize $M_\delta(I(.))$ in every step. Also, we compare the

quality of the result seed sets with the solutions of MaxExpInf, maximizing influence expectation. Beside the naive direct sampling method of MaxExpInf [39], referred to as NaiveInf, we use DSSA [54], one of the state-of-the-art algorithms, based on reverse sampling [7]. There are other efficient algorithms, see [7, 14, 68, 67] for example. However, we reemphasize that our problem is different from maximizing the influence expectation, and thus the running time comparison is for reference only.

We implemented MultiCritMdelta, ProbInf-Heu, and NaiveInf in C++ with OpenMP. The source code is available at `https://github.com/ngpham/probinf`. For DSSA, we used the implementation made public from [54]. The execution is carried out on a machine with 24 cores and $16GB$ of memory.

We study the following questions. (1) How does $M_\delta(I(S))$ depend on $|S|$ and δ? (2) How are the actual execution times compared to one another and to the theoretical bounds? (3) How does the solution to our algorithms compare with that to the MaxExpInf problem, in terms of both the ExpInf and $M_\delta(I(.))$ objectives?

3.5.1 Experimental Setup

Datasets

We use four social network datasets, which were crawled from public sources, as provided by [46]. The basic statistics of the datasets can be found in Table 3.1.

Parameters of the Algorithms

In NaiveInf, we used 10^4 samples, following the claim in [39] that these many samples suffice for datasets of this scale, although the worst case bound is $O(n^2)$ samples. In DSSA, we set the desired multiplicative error to 0.1 (notice that for large input, in scale of millions

Table 3.1: Datasets.

Dataset	Nodes	Edges	Diameter
Facebook	4039	88234	8
Twitter	81306	1768149	7
Slashdot	77360	905468	10
Pokec	1632803	30622564	11

and over, it is recommended to set this factor to 0.5 [54]).

For our algorithms, ProbInf-Heu and MultiCritMdelta, we take $\eta = 0.1$, which is the additive error for the probability guarantee δ (as discussed in Section 3.4). Following Theorem 27, we choose the number of samples to be $100 \times \log(n)$, where n is the number of nodes of the input graph. Finally, recall that MultiCritMdelta can relax the size of the seed set by a multiplicative factor of $\ln(n)$. In order to make the comparison reasonable, we run it with a smaller budget, so that the final seed set size (after the relaxation) is within the bound of k; this follows the approach in other works which obtain bicriteria approximation results, e.g., [43].

Model parameters.

We use the Independent Cascade (IC) model, with the standard setting widely used in the literature, where edge activation probability is set to the reverse of its incident node degree. We also experiment with activation probabilities of $0.01, 0.05, 0.1$, and two random settings, where the activation probability of each edge is picked uniformly randomly in the range $[0.001, 0.05]$ and $[0.001, 0.01]$. This setup is to understand the effect of *sparsity* of the activation.

For ProbInf-Heu and MultiCritMdelta, we conduct two sets of experiments. The first one is configured with probability guarantee δ in $\{0.2, 0.5, 0.7, 0.9\}$. We observe that

the resulting seed set qualities do not vary much with δ (consistent with the observation in [74]), and, therefore, report most of our results for $\delta = 0.7$ because of limited space. In the second set of experiments, we try ProbInf-Heu and MultiCritMdelta with extreme values of δ, which are very close to 0 or 1.

For the constraint on size of the seed set, k, we experiment with all values in $[0, 40]$. It is known from the literature (e.g., [54]) that the influence is quickly saturated when one increases k, such that increasing the seed set does not provide significant gain.

3.5.2 Results and Evaluation

We compare the algorithms by asserting the quality of the output seed sets on the original graph. Recall from Section 3.4.1 that knowing the distribution function would give us complete understanding of $I(.)$, and $M_\delta(I(.))$ can be inferred. Thus, with (fixed) result seed sets, we compute and compare the expected influences, and the empirical cumulative distribution functions (ECDF) [69].

Comparison with MaxExpInf Solution.

The experimental results, with edge activation randomly in the range $[0.001, 0.05]$, are compared according to expected influence (shown in Figure 3.1 and Figure 3.2), and probabilistic guarantee (shown in Figure 3.3 and Figure 3.4). Since DSSA has the same estimation guarantee as NaiveInf, we will refer to these solutions as MaxExpInf. The seed sets are computed by their respective algorithms, where both ProbInf-Heu and MultiCritMdelta has δ set to 0.7.

In Figures 3.1 and 3.2, we observe that MultiCritMdelta gives the best seed sets, while ProbInf-Heu gives comparable or better sets, versus MaxExpInf solutions. The

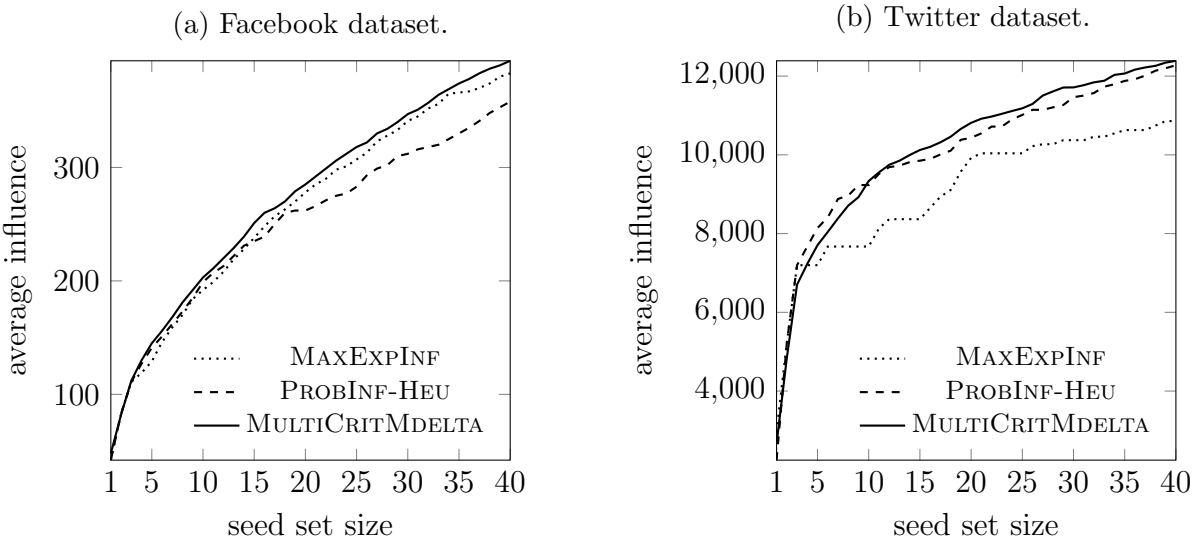

Figure 3.1: Comparison of average influence computed by three algorithms, with edge activation randomly in 0.001 to 0.05. ProbInf-Heu and MultiCritMdelta are invoked with the guarantee probability $\delta = 0.7$. Results are for the Facebook and Twitter datasets.

exception is the result for the Pokec dataset, Figure 3.2(b), where all algorithms give seed sets with similar expected influence.

The observation for the Pokec dataset that MaxProbInf solutions do not have significant advantage over MaxExpInf solutions, yields some interesting though. As per the existential proof of Lemma 17, we infer that there could be graphs such that MaxProbInf and MaxExpInf solutions are close to each other. The experimental result on the Pokec dataset implies that this is one such graph.

Probabilistic Guarantees

We study the probabilistic guarantees of the different algorithms through empirical cumulative distribution functions (ECDF) [69]. We will only report the results for the case where edge activation is picked randomly in the range $[0.001, 0.05]$, with $\delta = 0.7$. We take the

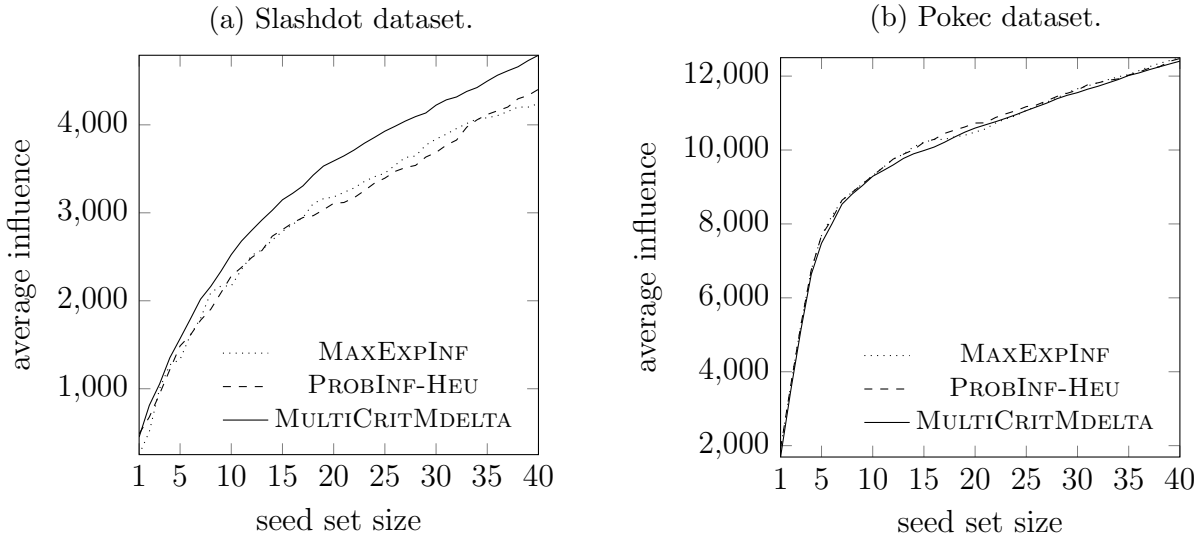

Figure 3.2: Comparison of average influence computed by three algorithms, with edge activation randomly in 0.001 to 0.05. ProbInf-Heu and MultiCritMdelta are invoked with the guarantee probability $\delta = 0.7$. Results are for the Slashdot and Pokec datasets.

solutions for $k = 40$ and fit their ECDF's by kernel density estimation with a Gaussian kernel. These are plotted in Figures 3.3 and 3.4. The measurement $M_\delta(.)$ can be asserted by the ECDF at probability $(1 - \delta)$. For example, with $\delta = 0.7$, in Figure 3.3(b), MultiCritMdelta seed set and ProbInf-Heu seed set is guaranteed to have an influence higher than 12100 for 70% of times, which is better than the MaxExpInf seed set corresponding value of 10800. Generally, the further the ECDF to the right, around probability $(1 - \delta)$, the higher the quality of the seed set as per $M_\delta(.)$.

Again, we observe that the Pokec dataset does not benefit much from probabilistic guarantee solution. In this dataset, MaxExpInf solution already has similar probabilistic guarantee, compared to that of MultiCritMdelta and ProbInf-Heu.

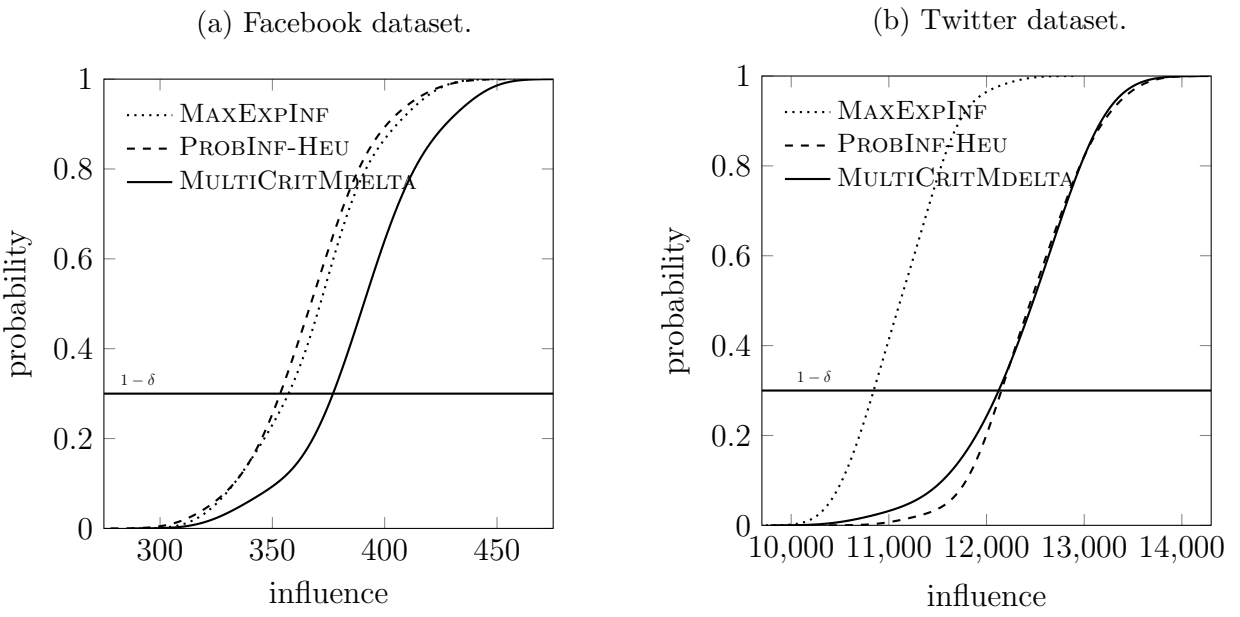

Figure 3.3: Empirical cumulative distribution function (ECDF) of influence of seed sets of size 40. For ProbInf-Heu and MultiCritMdelta algorithms, the seed sets are optimized with $\delta = 0.7$. Edge activations are random values in the range $[0.001, 0.05]$. The further to the right around probability of $1 - \delta = 0.3$, the better the guarantee influence. Results are for the Facebook and Twitter datasets.

Effects of Varying δ.

This is only applied to ProbInf-Heu and MultiCritMdelta. Figures 3.5 and 3.6 show the influence guarantee computed by the MultiCritMdelta and ProbInf-Heu algorithms for $\delta \in \{0.005, 0.01, 0.05, 0.1, 0.95, 0.995\}$, with $k \in \{15, 30, 40\}$. We observe that, for a fixed seed set S, the $M_\delta(I(S))$ value from ProbInf-Heu decreases steadily with δ, with a sharp decrease when δ is closer to 1. In contrast, the $M_\delta(I(S))$ value from MultiCritMdelta is quite insensitive to δ. Further, ProbInf-Heu has a higher $M_\delta(I(S))$ than MultiCritMdelta for small values of δ.

In particular, MultiCritMdelta gives a good solution in most of the scenarios, except when δ is close to 0 or 1, due to the 1/2 factor in approximating δ.

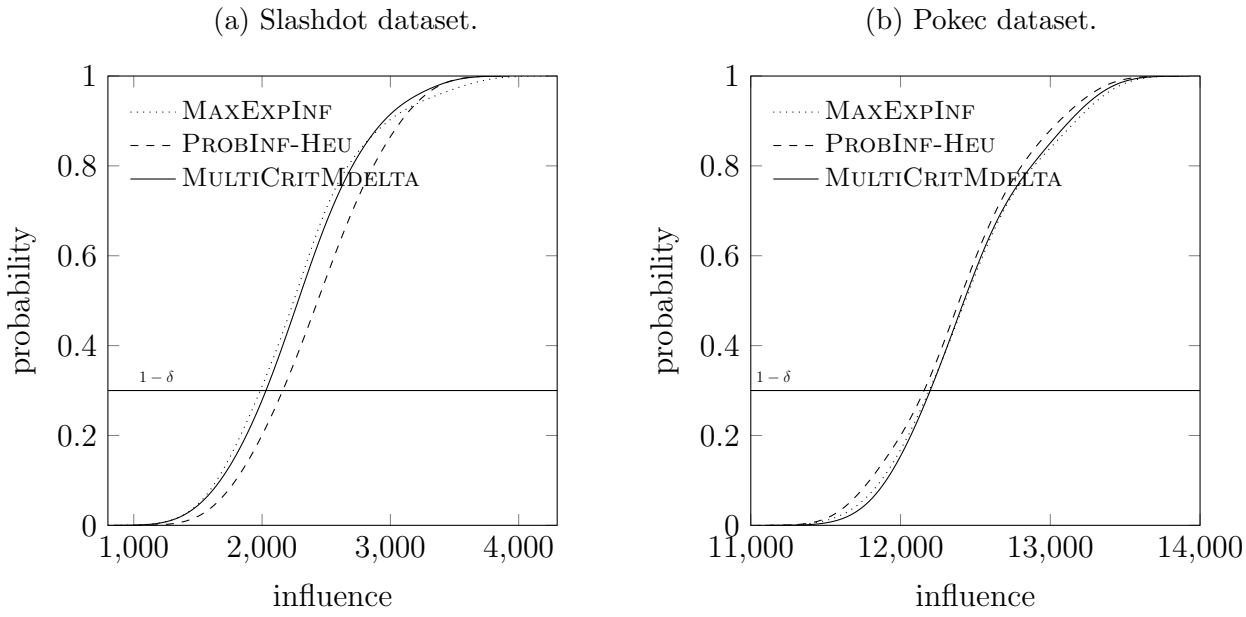

Figure 3.4: Empirical cumulative distribution function (ECDF) of influence of seed sets of size 40. For ProbInf-Heu and MultiCritMdelta algorithms, the seed sets are optimized with $\delta = 0.7$. Edge activations are random values in the range $[0.001, 0.05]$. The further to the right around probability of $1 - \delta = 0.3$, the better the guarantee influence. Results are for the Slashdot and Pokec datasets.

Execution Time.

One of the main advantages of the MultiCritMdelta and ProbInf-Heu algorithms is that their execution times are much smaller compared to NaiveInf, while the quality of the solutions obtained is similar or better. In fact, the MultiCritMdelta algorithm is also faster than ProbInf-Heu. The execution time depends on the number of samples. Each sample complexity depends on the density of the graph and the edge activation probability. We note that both MultiCritMdelta and ProbInf-Heu algorithms need significantly fewer samples compared to that of the algorithm of [39]. We also observe that the MultiCritMdelta asymptotic running time is close to that of DSSA, especially when the activation is *dense*. DSSA has the advantage that it can stop early, by verifying the current

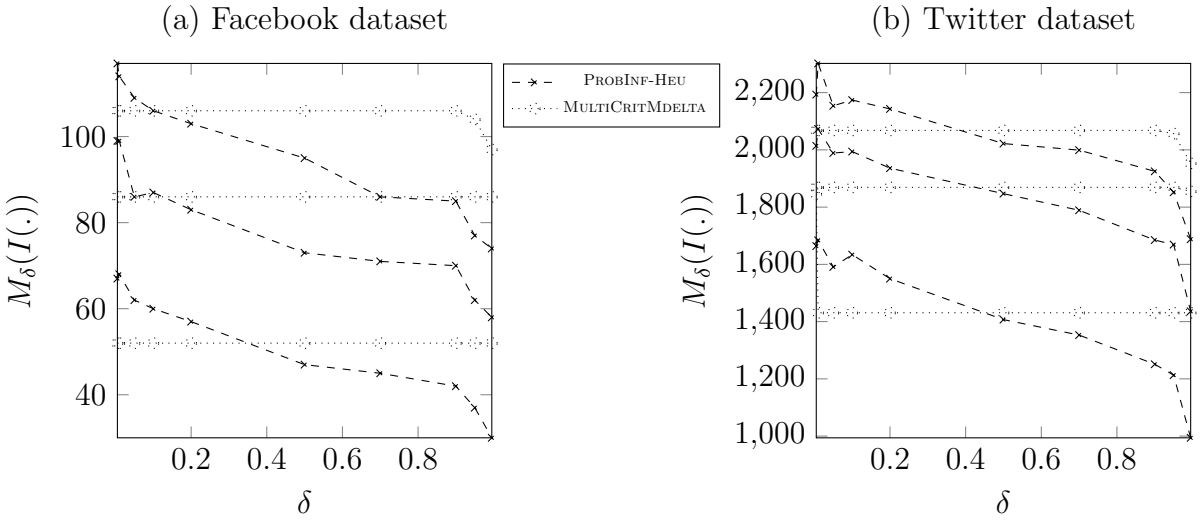

Figure 3.5: Varying δ, with random edge activation in the range $[0.001, 0.01]$. For each algorithm, we report the guarantee influence of the output seed sets of size 15, 30, and 40. Results are for the Facebook and Twitter datasets.

estimation error, thus, for networks with low activation, it achieves the best runtime. MultiCritMdelta, on the other hand, has to ensure the probabilistic guarantee and does not check for early termination.

We notice that the variation of δ does not have much affect on the running time, and only report results for $\delta = 0.7$, as shown in Table 3.2. Also, DSSA runs out of memory on the Pokec dataset, with the most dense activation setting (0.1). We believe this is due to our hardware limitation of 16GB memory.

3.6 Conclusion

Our results on the MaxProbInf problem suggest that the quality of solutions to influence maximization can be improved by adding probabilistic requirements. We developed the first rigorous approximation algorithms with guarantee bound and efficient running time.

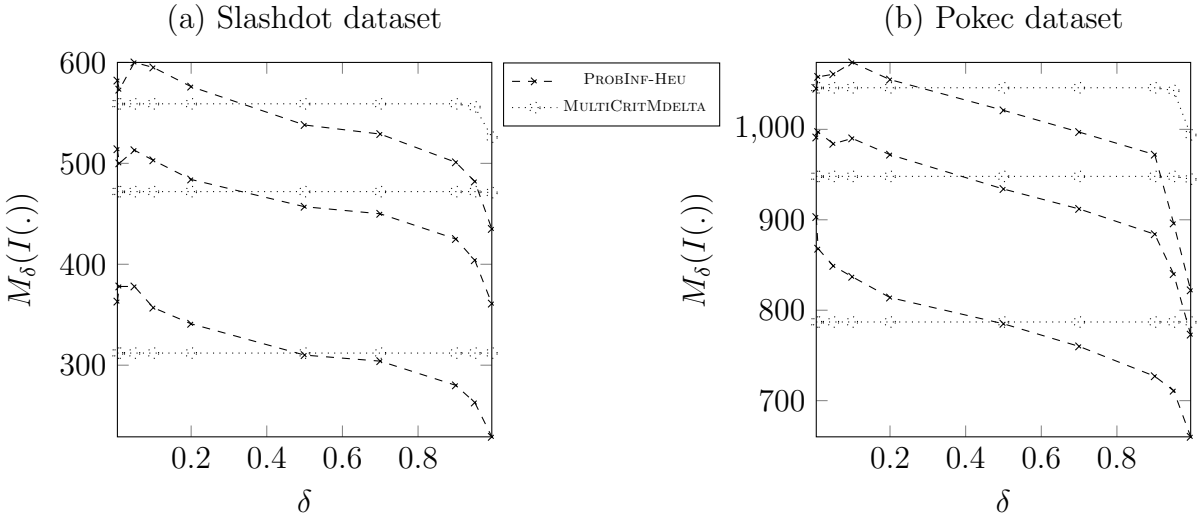

Figure 3.6: Varying δ, with random edge activation in the range $[0.001, 0.01]$. For each algorithm, we report the guarantee influence of the output seed sets of size 15, 30, and 40. Results are for the Slashdot and Pokec datasets.

Our experimental results show that our algorithms outperform the naive greedy algorithm for the MaxExpInf problem by an order of magnitude, while at the same time there is improvement in the expected influence as well. The results suggest that optimizing based on the MaxProbInf criterion using the MultiCritMdelta algorithm can be a better way to solve the influence maximization problem.

One possible improvement is to incorporate the reverse sampling method [7] and its variation, for example, the early stop detection [54]. It would be good to quantify which theoretical bounds and guarantees can be achieved for $M_\delta(I(.))$ following that direction.

Table 3.2: Execution Time, for $k = 40$, $\delta = 0.7$ (applied for ProbInf-Heu and MultiCrit-Mdelta), and various edge activation probabilities. The random probability is uniformly in the range $[0.001, 0.05]$. Execution time is measured in seconds.

Dataset	Algorithm	Activation				
		$1/deg$	rand	0.01	0.05	0.1
Facebook	NaiveInf	144	150	138	162	182
	ProbInf-Heu	19	24	22	24	26
	DSSA	3	3	5	5	7
	MultiCritMdelta	8	9	9	9	10
Twitter	NaiveInf	3319	3483	3014	4038	4216
	ProbInf-Heu	662	690	634	799	879
	DSSA	193	216	221	223	230
	MultiCritMdelta	225	242	249	252	260
Slashdot	NaiveInf	1913	1906	1825	2130	2517
	ProbInf-Heu	411	402	383	438	506
	DSSA	83	80	84	92	96
	MultiCritMdelta	227	225	223	232	242
Pokec	NaiveInf	58685	59445	53703	68646	75530
	ProbInf-Heu	19921	20067	16951	19723	22600
	DSSA	5118	5580	5972	6233	—
	MultiCritMdelta	6436	6553	6537	6850	6670

Chapter 4

Conclusion and Future Directions

In this dissertation, we conducted two studies in network-related problems, under the theme of randomized algorithm and probabilistic analysis. The problems seem far apart, but the underlying principles are connected. Both are captured by the graph model, though with different semantics.

The first problem, *Smoothed Analysis*, focuses on the theoretical aspect of distributed algorithms. We showed improved asymptotic bounds by adding some randomness to the topology of a distributed system. For the MST algorithm, we designed a randomized distributed algorithm that almost matches the lower bound. This still leaves a gap that is interesting to reduce. Since the smoothed model is newly proposed (to the best of our knowledge), there would have many open research problems, from studying other fundamental algorithms to considering different flavors of smoothing.

The second problem, *Influence Maximizing with Probabilistic Guarantee*, is more balanced on both theoretical analysis and empirical study. We proposed the probabilistic measurement for the network influence problem, being unaware of the previous work by Zhang et al. [74] at first. However, our work is still significantly different and novel. Our proposed optimization

problem is more natural, and our assumption is more relaxed (does not require constraint on the variance of the unknown influence). Compared to state-of-the-art algorithms, which do not have probabilistic guarantees, our algorithm has the same order of execution time. Since there are many other models for the network influence problem, there are multiple open questions on applying the probabilistic guarantee.

Another direction is to consider the network influence in the distributed setting, for example, using distributed random walks to estimate the influence (and the probabilistic guarantee). To gain more performance, one possibility is to use the k-machine model [55].

Bibliography

[1] David Aldous and James Fill. *Reversible Markov Chains and Random Walks on Graphs*. Book. Berkeley, California, USA: University of California, Berkeley, 1995.

[2] Noga Alon, Chen Avin, Michal Koucky, Gady Kozma, Zvi Lotker, and Mark R Tuttle. "Many random walks are faster than one". In: *Combinatorics, Probability and Computing*. Vol. 20. 4. Cambridge, United Kingdom: Cambridge University Press, 2011, pp. 481–502.

[3] James Aspnes and Eric Ruppert. "An introduction to population protocols". In: *Middleware for Network Eccentric and Mobile Applications*. Berlin, Heidelberg, Germany: Springer, 2009, pp. 97–120.

[4] John Augustine, Gopal Pandurangan, Peter Robinson, and Eli Upfal. "Towards robust and efficient computation in dynamic peer-to-peer networks". In: *Proceedings of the Twenty-third Annual ACM-SIAM Symposium on Discrete Algorithms*. Society for Industrial and Applied Mathematics (SIAM). Kyoto, Japan, 2012, pp. 551–569.

[5] Chen Avin, Michal Koucky, and Zvi Lotker. "How to explore a fast-changing world (cover time of a simple random walk on evolving graphs)". In: *ICALP 2008 35th International Colloquium on Automata, Languages, and Programming*. European Associa-

tion for Theoretical Computer Science (EATCS). Reykjavik, Iceland, 2008, pp. 121–132.

[6] Leonid Barenboim and Victor Khazanov. "Distributed symmetry-breaking algorithms for congested cliques". In: *International Computer Science Symposium in Russia*. Ed. by Fedor V. Fomin and Vladimir V. Podolskii. Springer. Moscow, Russia, 2018, pp. 41–52. ISBN: 978-3-319-90530-3.

[7] Christian Borgs, Michael Brautbar, Jennifer Chayes, and Brendan Lucier. "Maximizing social influence in nearly optimal time". In: *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics (SIAM). Portland, Oregon, USA, Jan. 2014, pp. 946–957.

[8] Pierre Brémaud. *Markov Chains: Gibbs Fields, Monte Carlo Simulation, and Queues*. Vol. 31. Book. Berlin, Heidelberg, Germany: Springer, 2013.

[9] Keren Censor-Hillel, Michal Dory, Janne H Korhonen, and Dean Leitersdorf. "Fast approximate shortest paths in the congested clique". In: *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*. PODC 2019. Association for Computing Machinery (ACM). Toronto, Ontario, Canada, 2019, pp. 74–83. ISBN: 978-1-4503-6217-7. DOI: 10.1145/3293611.3331633. URL: http://doi.acm.org/10.1145/3293611.3331633.

[10] Soumyottam Chatterjee, Reza Fathi, Gopal Pandurangan, and Nguyen Dinh Pham. "Fast and efficient distributed computation of Hamiltonian cycles in random graphs". In: *38th IEEE International Conference on Distributed Computing Systems (ICDCS 2018)*. Institute of Electrical and Electronics Engineers (IEEE). Vienna, Austria, 2018, pp. 764–774.

[11] Soumyottam Chatterjee, Gopal Pandurangan, and Nguyen Dinh Pham. "Distributed MST: A smoothed analysis". In: *21th International Conference on Distributed Computing and Networking (ICDCN 2020)*. To appear. Jadavpur University, Kolkata, India, 2020.

[12] Ning Chen. "On the approximability of influence in social networks". In: *SIAM Journal on Discrete Mathematics*. Vol. 23. 3. Philadelphia, Pennsylvania, USA: Society for Industrial and Applied Mathematics (SIAM), 2009, pp. 1400–1415.

[13] Wei Chen, Wei Lu, and Ning Zhang. "Time-critical influence maximization in social networks with time-delayed diffusion process". In: *26th AAAI Conference on Artificial Intelligence*. Association for the Advancement of Artificial Intelligence (AAAI). Toronto, Ontario, Canada, 2012.

[14] Wei Chen, Chi Wang, and Yajun Wang. "Scalable influence maximization for prevalent viral marketing in large-scale social networks". In: *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*. Association for Computing Machinery (ACM). Washington DC, DC, USA, July 2010, pp. 1029–1038.

[15] Herman Chernoff. "A career in statistics". In: *Past, Present, and Future of Statistical Science*. Vol. 29. Boca Raton, Florida, USA: Chapman and Hall/CRC, 2014.

[16] Herman Chernoff et al. "A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations". In: *The Annals of Mathematical Statistics*. Vol. 23. 4. Institute of Mathematical Statistics (IMS), 1952, pp. 493–507. URL: `https://imstat.org/`.

[17] Atish Das Sarma, Danupon Nanongkai, Gopal Pandurangan, and Prasad Tetali. "Distributed random walks". In: *Journal of the ACM*. Vol. 60. 1. New York City, New York, USA: Association for Computing Machinery (ACM), Feb. 2013, 2:1–2:31. DOI: 10.1145/2432622.2432624. URL: http://doi.acm.org/10.1145/2432622.2432624.

[18] Thang N. Dinh, Huiyuan Zhang, Dzung T. Nguyen, and My T. Thai. "Cost-effective viral marketing for time-critical campaigns in large-scale social networks". In: *IEEE/ACM Transactions on Networking (ToN)*. Vol. 22. 6. New York City, New York, USA: Institute of Electrical and Electronics Engineers (IEEE), 2014, pp. 2001–2011.

[19] Michael Dinitz, Jeremy T. Fineman, Seth Gilbert, and Calvin Newport. "Smoothed analysis of dynamic networks". In: *Distributed Computing*. Vol. 31. 4. Berlin, Heidelberg, Germany: Springer, Aug. 2018, pp. 273–287. DOI: 10.1007/s00446-017-0300-8. URL: https://doi.org/10.1007/s00446-017-0300-8.

[20] Nan Du, Le Song, Manuel Gomez Rodriguez, and Hongyuan Zha. "Scalable influence estimation in continuous-time diffusion networks". In: *Advances in Neural Information Processing Systems (NIPS 2013)*. Neural Information Processing Systems (NeurIPS). South Lake Tahoe, California, USA, 2013, pp. 3147–3155.

[21] David Easley, Jon Kleinberg, et al. *Networks, crowds, and markets*. Vol. 8. Cambridge, United Kingdom: Cambridge University Press, 2010.

[22] Michael Elkin. "A simple deterministic distributed MST algorithm, with near-optimal time and message complexities". In: *Proceedings of the ACM Symposium on Principles of Distributed Computing*. PODC 2017. Association for Computing Machinery (ACM). Washington DC, DC, USA, 2017, pp. 157–163. ISBN: 978-1-4503-4992-5. DOI: 10.1145/3087801.3087823. URL: http://doi.acm.org/10.1145/3087801.3087823.

[23] Michael Elkin. "An unconditional lower bound on the time-approximation trade-off for the distributed minimum spanning tree problem". In: *SIAM Journal on Computing*. Vol. 36. 2. Philadelphia, Pennsylvania, USA: Society for Industrial and Applied Mathematics (SIAM), 2006, pp. 433–456.

[24] Michael Elkin. "Computing almost shortest paths". In: *ACM Transactions on Algorithms (TALG)*. Vol. 1. 2. New York City, New York, USA: Association for Computing Machinery (ACM), 2005, pp. 283–323.

[25] Michael Elkin. "Distributed approximation: A survey". In: *ACM SIGACT News*. Vol. 35. 4. New York City, New York, USA: Association for Computing Machinery (ACM), 2004, pp. 40–57.

[26] Alessandro Ferrante, Gopal Pandurangan, and Kihong Park. "On the hardness of optimization in power-law graphs". In: *Theoretical Computer Science*. Vol. 393. 1-3. Amsterdam, Netherlands: Elsevier, 2008, pp. 220–230.

[27] Satoru Fujishige. *Submodular Functions and Optimization*. Vol. 58. Book. Amsterdam, Netherlands: Elsevier, 2005.

[28] Toshihiro Fujito. "Approximation algorithms for submodular set cover with applications". In: *IEICE Trans. on Information and Systems*. Vol. 83. 3. Tokyo, Japan: The Institute of Electronics, Information and Communication Engineers, 2000, pp. 480–487.

[29] Robert G. Gallager, Pierre A. Humblet, and Philip M. Spira. "A distributed algorithm for minimum-weight spanning trees". In: *ACM Transactions on Programming Languages and Systems (TOPLAS)*. Vol. 5. 1. New York City, New York, USA: Association for Computing Machinery (ACM), 1983, pp. 66–77.

[30] Mohsen Ghaffari, Fabian Kuhn, and Hsin-Hao Su. "Distributed MST and routing in almost mixing time". In: *Proceedings of the ACM Symposium on Principles of Distributed Computing*. PODC 2017. Association for Computing Machinery (ACM). Washington DC, DC, USA, 2017, pp. 131–140. ISBN: 978-1-4503-4992-5. DOI: 10.1145/3087801.3087827. URL: http://doi.acm.org/10.1145/3087801.3087827.

[31] Mohsen Ghaffari and Jason Li. "New distributed algorithms in almost mixing time via transformations from parallel algorithms". In: *32nd International Symposium on Distributed Computing (DISC 2018)*. Ed. by Ulrich Schmid and Josef Widder. Vol. 121. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018, 31:1–31:16. ISBN: 978-3-95977-092-7. DOI: 10.4230/LIPIcs.DISC.2018.31. URL: http://drops.dagstuhl.de/opus/volltexte/2018/9820.

[32] Mohsen Ghaffari and Krzysztof Nowicki. "Congested clique algorithms for the minimum cut problem". In: *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing*. PODC 2018. Association for Computing Machinery (ACM). Egham, United Kingdom, 2018, pp. 357–366. ISBN: 978-1-4503-5795-1. DOI: 10.1145/3212734.3212750. URL: http://doi.acm.org/10.1145/3212734.3212750.

[33] Sharon Goldberg and Zhenming Liu. "The diffusion of networking technologies". In: *Proceedings of the twenty-fourth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics (SIAM). New Orleans, Louisiana, USA, Jan. 2013, pp. 1577–1594.

[34] Amit Goyal, Francesco Bonchi, Laks V.S. Lakshmanan, and Suresh Venkatasubramanian. "On minimizing budget and time in influence propagation over social net-

works". In: *Social Network Analysis and Mining*. Vol. 3. 2. Berlin, Heidelberg, Germany: Springer, 2013, pp. 179–192.

[35] Wassily Hoeffding. "Probability inequalities for sums of bounded random variables". In: *The Collected Works of Wassily Hoeffding*. Berlin, Heidelberg, Germany: Springer, 1994, pp. 409–426.

[36] Mark Jerrum and Alistair Sinclair. "Conductance and the rapid mixing property for Markov chains: The approximation of permanent resolved". In: *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*. STOC 1988. Association for Computing Machinery (ACM). Chicago, Illinois, USA, 1988, pp. 235–244. ISBN: 0-89791-264-0. DOI: 10.1145/62212.62234. URL: http://doi.acm.org/10.1145/62212.62234.

[37] Tomasz Jurdziński and Krzysztof Nowicki. "MST in $O(1)$ rounds of congested clique". In: *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*. SODA 2018. Society for Industrial and Applied Mathematics (SIAM). New Orleans, Louisiana, 2018, pp. 2620–2632. ISBN: 978-1-6119-7503-1. URL: http://dl.acm.org/citation.cfm?id=3174304.3175472.

[38] David Kemp, Jon Kleinberg, and Éva Tardos. "Maximizing the spread of influence through a social network". In: *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. Association for Computing Machinery (ACM). Washington DC, DC, USA, 2003, pp. 137–146.

[39] David Kempe, Jon Kleinberg, and Éva Tardos. "Maximizing the spread of influence through a social network". In: *Theory of Computing*. Vol. 11. 4. 2015, pp. 105–147.

DOI: 10.4086/toc.2015.v011a004. URL: http://www.theoryofcomputing.org/articles/v011a004.

[40] Maleq Khan and Gopal Pandurangan. "A fast distributed approximation algorithm for minimum spanning trees". In: *Distributed Computing*. Vol. 20. 6. Berlin, Heidelberg, Germany: Springer, Apr. 2008, pp. 391–402. DOI: 10.1007/s00446-007-0047-8. URL: https://doi.org/10.1007/s00446-007-0047-8.

[41] Maleq Khan, Gopal Pandurangan, Nguyen Dinh Pham, Anil Vullikanti, and Qin Zhang. "A multi-criteria approximation algorithm for influence maximizationwith probabilistic guarantees". In: *SIAM Symposium on Algorithm Engineering and Experiments (ALENEX20)*. To appear. Salt Lake City, Utah, USA, 2020.

[42] Janne H Korhonen and Jukka Suomela. "Towards a complexity theory for the congested clique". In: *Proceedings of the 30th on Symposium on Parallelism in Algorithms and Architectures*. SPAA 2018. Association for Computing Machinery (ACM). Vienna, Austria, 2018, pp. 163–172. ISBN: 978-1-4503-5799-9. DOI: 10.1145/3210377.3210391. URL: http://doi.acm.org/10.1145/3210377.3210391.

[43] Andreas Krause, H Brendan McMahan, Carlos Guestrin, and Anupam Gupta. "Robust submodular observation selection". In: *Journal of Machine Learning Research*. JMLR, 2008, pp. 2761–2801. URL: http://www.jmlr.org/papers/volume9/krause08b/krause08b.pdf.

[44] K. Krzywdziński and K. Rybarczyk. "Distributed algorithms for random graphs". In: *Theoretical Computer Science*. Vol. 605. C. Amsterdam, Netherlands: Elsevier, 2015, pp. 95–105.

[45] Eyal Kushilevitz and Noam Nisan. *Communication Complexity*. Book. Cambridge, United Kingdom: Cambridge University Press, 1997. ISBN: 978-0-521-56067-2.

[46] Jure Leskovec and Andrej Krevl. *SNAP Datasets: Stanford large network dataset collection*. `http://snap.stanford.edu/data`. June 2014.

[47] David A Levin and Yuval Peres. *Markov Chains and Mixing Times*. Vol. 107. Book. Providence, Rhode Island, USA: American Mathematical Society (AMS), 2017.

[48] László Lovász et al. "Random walks on graphs: A survey". In: *Combinatorics, Paul Erdős is eighty*. Vol. 2. 1. Budapest, Hungary: Janos Bolyai Mathematical Society, 1993, pp. 1–46.

[49] László Lovász and Ravi Kannan. "Faster mixing via average conductance". In: *Proceedings of the Thirty-first Annual ACM Symposium on Theory of Computing*. STOC 1999. Association for Computing Machinery (ACM). Atlanta, Georgia, USA, 1999, pp. 282–287. ISBN: 1-58113-067-8. DOI: `10.1145/301250.301317`. URL: `http://doi.acm.org/10.1145/301250.301317`.

[50] Brendan Lucier, Joel Oren, and Yaron Singer. "Influence at scale: Distributed computation of complex contagion in networks". In: *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. Association for Computing Machinery (ACM). Sydney, NSW, Australia, 2015, pp. 735–744. ISBN: 978-1-4503-3664-2. DOI: `10.1145/2783258.2783334`. URL: `http://doi.acm.org/10.1145/2783258.2783334`.

[51] Nancy A. Lynch. *Distributed Algorithms*. Book. Amsterdam, Netherlands: Elsevier, 1996.

[52] Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomization and Probabilistic Techniques in Algorithms and Data Analysis*. Book. Cambridge, United Kingdom: Cambridge University Press, 2017.

[53] Jaroslav Nešetřil, Eva Milková, and Helena Nešetřilová. "Otakar Borůvka on minimum spanning tree problem Translation of both the 1926 papers, comments, history". In: *Discrete Mathematics* 233.1 (2001). Czech and Slovak 2, pp. 3–36. ISSN: 0012-365X. DOI: https://doi.org/10.1016/S0012-365X(00)00224-7. URL: http://www.sciencedirect.com/science/article/pii/S0012365X00002247.

[54] Hung T. Nguyen, My T. Thai, and Thang N. Dinh. "Stop-and-stare: Optimal sampling algorithms for viral marketing in billion-scale networks". In: *Proceedings of the 2016 ACM SIGMOD International Conference on Management of Data*. Association for Computing Machinery (ACM). San Francisco, California, USA, June 2016, pp. 695–710.

[55] Gopal Pandurangan. *Distributed Network Algorithms*. Book. URL: https://sites.google.com/site/gopalpandurangan/dnabook.pdf (visited on 10/01/2019).

[56] Gopal Pandurangan, Peter Robinson, and Michele Scquizzato. "A time-and message-optimal distributed algorithm for minimum spanning trees". In: *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*. STOC 2017. Association for Computing Machinery (ACM). Montreal, Canada, 2017, pp. 743–756. ISBN: 978-1-4503-4528-6. DOI: 10.1145/3055399.3055449. URL: http://doi.acm.org/10.1145/3055399.3055449.

[57] Merav Parter. "$(\Delta+1)$-coloring in the congested clique model". In: *45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*. Ed. by Ioannis

Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella. Vol. 107. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018, 160:1–160:14. ISBN: 978-3-95977-076-7. DOI: 10.4230/LIPIcs.ICALP.2018.160. URL: http://drops.dagstuhl.de/opus/volltexte/2018/9164.

[58] Merav Parter and Hsin-Hao Su. "Randomized $(\Delta+1)$-coloring in $O(\log^* \Delta)$ congested clique rounds". In: *32nd International Symposium on Distributed Computing (DISC 2018)*. Ed. by Ulrich Schmid and Josef Widder. Vol. 121. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018, 39:1–39:18. ISBN: 978-3-95977-092-7. DOI: 10.4230/LIPIcs.DISC.2018.39. URL: http://drops.dagstuhl.de/opus/volltexte/2018/9828.

[59] Merav Parter and Eylon Yogev. "Congested clique algorithms for graph spanners". In: *32nd International Symposium on Distributed Computing (DISC 2018)*. Ed. by Ulrich Schmid and Josef Widder. Vol. 121. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018, 40:1–40:18. ISBN: 978-3-95977-092-7. DOI: 10.4230/LIPIcs.DISC.2018.40. URL: http://drops.dagstuhl.de/opus/volltexte/2018/9829.

[60] David Peleg. *Distributed computing: A locality-sensitive approach*. Book. Society for Industrial and Applied Mathematics (SIAM), 2000. eprint: https://epubs.siam.org/doi/pdf/10.1137/1.9780898719772.

[61] Atish Das Sarma, Stephan Holzer, Liah Kor, Amos Korman, Danupon Nanongkai, Gopal Pandurangan, David Peleg, and Roger Wattenhofer. "Distributed verification and hardness of distributed approximation". In: *SIAM Journal on Computing*. Vol. 41. 5. Philadelphia, Pennsylvania, USA: Society for Industrial and Applied Mathematics

(SIAM), 2012, pp. 1235–1265. DOI: 10.1137/11085178X. URL: https://doi.org/10.1137/11085178X.

[62] Atish Das Sarma, Anisur Rahaman Molla, and Gopal Pandurangan. "Distributed computation in dynamic networks via random walks". In: *Theoretical Computer Science*. Vol. 581. Amsterdam, Netherlands: Elsevier, 2015, pp. 45–66. DOI: https://doi.org/10.1016/j.tcs.2015.02.044. URL: http://www.sciencedirect.com/science/article/pii/S0304397515001930.

[63] Stefan Schmid, Gopal Pandurangan, Peter Robinson, and Michele Scquizzato. "The distributed minimum spanning tree problem". In: *Bulletin of the EATCS*. Vol. 125. European Association for Theoretical Computer Science (EATCS), 2018. URL: http://eatcs.org/beatcs/index.php/beatcs/article/view/538.

[64] Daniel A. Spielman and Shang-Hua Teng. "Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time". In: *Journal of the ACM*. Vol. 51. 3. New York City, New York, USA: Association for Computing Machinery (ACM), May 2004, pp. 385–463. DOI: 10.1145/990308.990310. URL: http://doi.acm.org/10.1145/990308.990310.

[65] Daniel A. Spielman and Shang-Hua Teng. "Smoothed analysis: An attempt to explain the behavior of algorithms in practice". In: *Communications of the ACM* 52.10 (Oct. 2009), pp. 76–84. ISSN: 0001-0782. DOI: 10.1145/1562764.1562785. URL: http://doi.acm.org/10.1145/1562764.1562785.

[66] Chaitanya Swamy and David B Shmoys. "Approximation algorithms for 2-stage stochastic optimization problems". In: *ACM SIGACT News*. Vol. 37. 1. New York City, New York, USA: Association for Computing Machinery (ACM), 2006, pp. 33–46.

[67] Youze Tang, Yanchen Shi, and Xiaokui Xiao. "Influence maximization in near-linear time: A Martingale approach". In: *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. Association for Computing Machinery (ACM). Melbourne, Victoria, Australia, 2015, pp. 1539–1554.

[68] Youze Tang, Xiaokui Xiao, and Yanchen Shi. "Influence maximization: Near-optimal time complexity meets practical efficiency". In: *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*. Association for Computing Machinery (ACM). Snowbird, Utah, USA, 2014, pp. 75–86.

[69] George R. Terrell and David W. Scott. "Variable kernel density estimation". In: *The Annals of Statistics* (1992), pp. 1236–1265. URL: `https://imstat.org/`.

[70] Tommaso Toffoli and Norman Margolus. "Programmable matter: concepts and realization". In: *Physica. D, Nonlinear Phenomena*. Vol. 47. 1-2. Amsterdam, Netherlands: Elsevier, 1991, pp. 263–272.

[71] Guangmo Tong, Weili Wu, Shaojie Tang, and Ding-Zhu Du. "Adaptive influence maximization in dynamic social networks". In: *IEEE/ACM Transactions on Networking (TON)*. Vol. 25. 1. New York City, New York, USA: Institute of Electrical and Electronics Engineers (IEEE), 2017, pp. 112–125.

[72] Laurence A. Wolsey. "An analysis of the greedy algorithm for the submodular set covering problem". In: *Combinatorica*. Vol. 2. 4. Berlin, Heidelberg, Germany: Springer, 1982, pp. 385–393.

[73] Bang Ye Wu, Giuseppe Lancia, Vineet Bafna, Kun-Mao Chao, Ramamurthy Ravi, and Chuan Yi Tang. "A polynomial-time approximation scheme for minimum routing cost spanning trees". In: *SIAM Journal on Computing*. Vol. 29. 3. Philadelphia, Pennsyl-

vania, USA: Society for Industrial and Applied Mathematics (SIAM), 2000, pp. 761–778.

[74] Peng Zhang, Wei Chen, Xiaoming Sun, Yajun Wang, and Jialin Zhang. "Minimizing seed set selection with probabilistic coverage guarantee in a social network". In: *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. KDD 2014. Association for Computing Machinery (ACM). New York City, New York, USA, 2014, pp. 1306–1315. ISBN: 978-1-4503-2956-9.

Appendix A

Proofs

A.1 Alternative Proof for Lemma 6

Proof. We will show that, there exist constants $c > 0$ and $0 < \alpha, \beta < 1$, such that: $\Phi(R(G)) \geq \frac{c(1-\beta)(1-\alpha)}{4(1+\alpha)}$.

Let the selection run for $c\frac{\log n}{\epsilon}$ rounds. For a node u, let $r(u)$ indicate the number of added edges by u. We have $\mathrm{E}[r(u)] = c \log n$. We will show that, with high probability:

$$\forall u \in V, (1-\alpha)c \log n \leq r(u) \leq (1+\alpha)c \log n \tag{A.1}$$

Let E_1 be the bad event, that at least one node added too few or too many edges outside the above range. Using a standard Chernoff bound to bound the probability that, for one particular node u, $r(u)$ falls outside the above range and then union-bounding over n nodes,

we have:

$$\begin{aligned}\Pr[E_1] &\leq n\left(\Pr[r(u) \leq (1-\alpha)c\log n] + \Pr[r(u) \geq (1+\alpha)c\log n]\right) \\ &\leq n\left(e^{-\frac{\alpha^2 c\log n}{3}} + e^{-\frac{\alpha^2 c\log n}{2}}\right) \\ &\leq exp\left(\ln n + \ln 2 - \frac{\alpha^2 c\log n}{3}\right) = o(n^{-1}),\end{aligned} \tag{A.2}$$

for suitable constants α and c.

We note that $R(G)$ is essentially equivalent to the Erdős-Renyi random graph $G(n,p)$ where $p = \Theta(\log n/n)$ is the probability of having an edge between any pair of nodes. Note that, however, there are slight differences — $R(G)$ is a multi-graph unlike $G(n,p)$. Since $G(n,p)$ with $p = \Theta(\log n/n)$ is known to be an expander and hence has constant conductance and $O(\log n)$ mixing time, here we give a self-contained proof for the conductance

Now we bound the conductance $\Phi(R)$ of the graph $R(G)$, using the definition in equation 1.10:

$$\Phi(R(G)) = \min_{S\subset V, |S|\leq \frac{n}{2}} \frac{|\partial S|}{\min(\mathrm{Vol}(S), \mathrm{Vol}(\bar{S}))} \tag{A.3}$$

Fix a subset S of V where $|S| = k \leq \frac{n}{2}$, and bound the conductance $\varphi(S)$:

$$\varphi(S) = \frac{|\partial S|}{\min(\mathrm{Vol}(S), \mathrm{Vol}(\bar{S}))} \tag{A.4}$$

We consider the numerator and the denominator of $\varphi(S)$ separately. For the denominator, we have: $\mathrm{Vol}(S) \leq 2k(1+\alpha)c\log n$ and $\mathrm{Vol}(\bar{S}) \leq 2(n-k)(1+\alpha)c\log n$. Thus:

$$\min(\mathrm{Vol}(S), \mathrm{Vol}(\bar{S})) \leq 2k(1+\alpha)c\log n \tag{A.5}$$

Consider the numerator, and recall that each node u has $r(u)$ edges where the destination of each edge is chosen uniformly at random. Using the bound in equation A.1, we have:

$$\begin{aligned} E[|\partial S|] &\geq \min\left(k(1-\alpha)c\log n \cdot (\frac{n-k}{n}), (n-k)(1-\alpha)c\log n \cdot (\frac{k}{n})\right) \\ &= \frac{n-k}{n}k(1-\alpha)c\log n \end{aligned} \tag{A.6}$$

Since $k \leq \frac{n}{2}$, we have:

$$E[|\partial S|] \geq \frac{k}{2}(1-\alpha)c\log n \tag{A.7}$$

Let $\mu_S = E[|\partial S|]$. Next, similar to the steps for equation A.1, we will bound $|\partial S|$. For some constant $0 < \beta < 1$, we need to show that, with high probability:

$$|\partial S| \geq (1-\beta)\mu_S \tag{A.8}$$

The Chernoff bound gives:

$$\Pr[|\partial S| < (1-\beta)\mu_S] \leq e^{-\frac{\beta^2\mu_S}{2}} \tag{A.9}$$

Let $\mathcal{E}_2^k$ be the bad event that at least one of $\binom{n}{k}$ set has less than $(1-\beta)\mu_S$ boundary edges. Using this fact (and $k \geq 1$):

$$\left(\frac{n}{k}\right)^k \leq \binom{n}{k} \leq \left(\frac{ne}{k}\right)^k \leq (e^{k\ln n}),$$

and the union bound gives:

$$\Pr[\mathcal{E}_2^k] \leq exp(-\frac{\beta^2\mu_S}{2} + k\ln n) \leq \exp(-\frac{\beta^2 k}{4}(1-\alpha)c\log n + k\ln n) \tag{A.10}$$

We can choose the constants c, α, β such that:

$$\Pr[\mathcal{E}_2^k] \leq e^{-2\ln n} \tag{A.11}$$

Let $\mathcal{E}_3$ be the bad event that at least one bad event $\mathcal{E}_2^k$ happens, for all $1 \leq k \leq \frac{n}{2}$. Using the union bound:

$$\Pr[\mathcal{E}_3] \leq \sum_{k=1}^{n/2} \Pr[\mathcal{E}_2^k] \leq e^{\ln n - \ln 2} e^{-2\ln n} = o(n^{-1}) \tag{A.12}$$

Thus, from equations A.5 and A.8, for all S where $|S| \leq \frac{n}{2}$, we have, with high probability (for suitably chosen constants c, α, β):

$$\varphi(S) \geq \frac{(1-\beta)\frac{k}{2}(1-\alpha)c\log n}{2k(1+\alpha)c\log n} = \frac{c(1-\beta)(1-\alpha)}{4(1+\alpha)} \tag{A.13}$$

It follows that $\Phi(R(G))$ is constant.

Since $R(G)$ has constant conductance, it follows that (see, e.g., [36, 49]) $R(G)$ has fast mixing time: $\tau_{mix}(R) = O(\log n)$. □