Scalable Algorithms for Mining Graphs and Social Networks via Sampling

by
Ahmad Mahmoody
B.Sc., Sharif University of Technology, 2009
M.Sc., Simon Fraser University, 2011
M.Sc., Brown University, 2013

A dissertation submitted in partial fulfillment of the
requirements for the Degree of Doctor of Philosophy
in the Department of Computer Science at Brown University

Providence, Rhode Island
April 2017

This dissertation by Ahmad Mahmoody is accepted in its present form by the Department of Computer Science as satisfying the dissertation requirement for the degree of Doctor of Philosophy.

Date _____ _____

Eli Upfal, Advisor

Recommended to the Graduate Council

Date _____ _____

Philip N. Klein, Reader
Brown University

Date _____ _____

Alessandro Epasto, Reader
Google Research

Approved by the Graduate Council

Date _____ _____

Andrew G. Campbell
Dean of the Graduate School

Acknowledgement

My life at brown has been full of unforgettable experiences and wonderful people who made all those experiences enjoyable and memorable. I cannot possibly thank my advisor Eli Upfal enough, for his guidance and care. I learnt a lot from him, and I owe him a debt of gratitude.

The first two years of my PhD program in Ben Raphael's lab was a grand opportunity for me and I am very thankful for all his help.

I would also like to thank Philip Klein, Pedro Felzenszwalb, and Alessandro Epasto for graciously accepting to be on my comps and thesis committee.

I spent two summers at Google and had wonderful experiences with my wonderful hosts Silvio Lattanzi, Aditya Bhaskara, Qian Yu, and Yeqing Li, to whom I am very thankful.

I must also thank our amazing t-staff and a-staff (Lauren Clarke in particular) for answering my numerous questions and following up on my problems for the past six years.

It is my greatest pleasure to thank my awesome friends with whom we spent numerous hours of thinking on problems, playing foosball, getting coffee, or talking about politics :) Thank you Mehdi Aghagolzadeh, Kavosh Asadi, Lorenzo De Stefani, Thomas Dickerson, Max Leiserson, Layla Oesper, Sobhan Naderi, Hassan Ghassemi, Alexandra Papoutsaki, Matteo Riondato, Anna Ritz, Eric Sodomka, Hsin-Ta Wu, Erfan Zamanian for being such cool friends.

Lastly, I am greatly indebted to Zainab, Mohammad, Mahmood, Teeba, Yahya, Eyman, and our parents for their love and support. Without them, this would not have been possible.

Abstract of "Scalable Algorithms for Mining Graphs and Social Networks via Sampling":
Sampling based and randomized algorithms are powerful tools in studying big data problems. In this thesis we provide sampling based algorithms for mining graphs and social network in three different contexts:

Detecting Valuable Information. Detecting new information and events in a dynamic system by probing individual nodes has many practical applications: discovering new webpages, analyzing influence properties in network, and detecting failure propagation in electronic circuits or infections in public drinkable water systems. In practice, it is infeasible for anyone but the owner of the network (if existent) to monitor all nodes at all times. We study the constrained setting when the observer can only probe a small set of nodes at each time step to check whether new pieces of information (items) have reached those nodes.

Centrality Maximization. *Betweenness centrality* (BWC) is a fundamental centrality measure in social network analysis. Given a large-scale network, how can we find the most central nodes? This question is of great importance to many key applications that rely on BWC, including community detection and understanding graph vulnerability. Despite the large amount of work on scalable approximation algorithm design for BWC, estimating BWC on large-scale networks remains a computational challenge. In this paper, we study the *Centrality Maximization* problem (CMP): given a graph $G = (V, E)$ and a positive integer k, find a set $S^* \subseteq V$ that maximizes BWC subject to the cardinality constraint $|S^*| \leq k$. We present an efficient randomized algorithm that provides a $(1 - 1/e - \epsilon)$-approximation with high probability, where $\epsilon > 0$. Our results improve the current state-of-the-art result [127].

Influence Estimation. Social networks are important communication and information media. Individuals in a social network share information and influence each other through their social connections. Understanding social influence and information diffusion is a fundamental research endeavor and it has important applications in online social advertising, viral marketing, and trending topic prediction.

We first study the *Targeted-Influence* problem (TIP): Given a network $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and a model of influence, we want to be able to estimate in *real-time* (say in a few seconds per query) the influence of an arbitrary subset of users S over an arbitrary

subset of users T, for any possible query $(S;T)$, $S,T \subseteq \mathcal{V}$. To do so, we allow an efficient preprocessing. Finally, we conclude the thesis by studying the problem of *Conditional Influence Estimation* (CIE) whose goal is to estimate the influence of a node given that a cascade started from that node has already reached a group of nodes (target nodes).

Contents

Chapter 1

Introduction

In the era of *Big Data*, having access to scalable algorithms is greatly important. Sampling based methods and randomized algorithms are strong tools for analytical tasks in Data and Graph mining. In this thesis we study sampling based algorithms in three different contexts of Big Data problems, and more specifically, in analyzing graphs and social networks. The first context we study is the detection of valuable information in dynamic networks when we have limited resources. Next, in the context of Social Network Analysis (SNA), we study the topics of *betweenness centrality maximization*, *real-time targeted-influence queries over large graphs*, and finally *conditional influence estimation*. In following we give an introduction to these topics divided into three parts.

Part 1: Detecting Valuable Information

Many applications require the detection of events in a network as soon as they happen or shortly thereafter, either because the events are critical (e.g. anomalies, malfunctions) and late detection can be very costly, or because the value of the information obtained by detecting the events decays rapidly as time passes (e.g., news or trending topics).

Monitoring for new events or information is a fundamental search and detection problem in a distributed data setting. We call each possible event of interest an *item*, and assume that items are generated at individual nodes. To study our search and

detection problem we need to answer the following questions, in order to specify the setting of our model:

- *Can items **propagate** (being copied) in the network?*
- *Do items lose their **novelty** (or relevance/freshness) over time?*

Depending on the application, we may have different answers to the above questions. For example, when the goal is to detect *anomaly*, *malfunctionality*, or the information that are *specific to their location*, we assume the items do not propagate, and items found in different parts of the network are considered different. Whereas in the context of *news aggregation* or *rumor detection*, the same item (e.g. news, rumor) can be found in different parts of the network. Also, note that in the former case, the items do not lose their novelty/relevance as they are issues that need to be detected and fixed, but in the latter case, their novelty decays over time. Also note that, it is always sufficient to detect a copy of each individual item, as there is no benefit in seeing multiple copies of the same news item or rumor. In this proposal, we study our detection problem in these two settings:

- **Schedule Optimization Problem:** Items do not propagate and do not lose their novelty. We may probe up to c nodes at any time, and a probing schedule can be deterministic or randomized. See Chapter 2.
- (θ, c)**-Optimal Probing Schedule Problem:** Items can propagate and lose their novelty with decay rate θ. We may probe up to c nodes at any time, and probing schedules are randomized. Chapter 3.

We outline several important applications of this schedule optimization problem:

News and Feed Aggregators. To provide up-to-date summary of the news, news aggregator sites need to constantly browse the Web, and often also the blogosphere and social networks, for new items. Scanning a site for new items requires significant communication and computation resources, thus the news aggregator can scan only a few sites simultaneously. The frequency of visiting a site has to depend on the likelihood of finding new items in that site. [17, 60, 114]

Algorithmic Trading on Data. An emerging trend in algorithmic stock trading is the use of automatic search through the Web, the blogosphere, and social networks

for relevant information that can be used in fast trading, before it appears in the more popular news sites [34, 51, 59, 79, 86, 90, 91]. The critical issue in this application is the speed of discovering new events, but again there is a resource limit on the number of sites that the search algorithm can scan simultaneously.

Detecting Anomaly and Machine Malfunction. In large server farm or any other large collection of semi-autonomous machines, a central controller needs to identify and contain anomalies and malfunctions as soon as possible, before they spread in the system. To minimize interference with the system's operation the controller must probe only a small number of machines in each step.

In these examples, as well as in many other applications, the main cost is in probing a remote server or machine. There are efficient techniques for identifying new items with minimum communication [18, 21, 31, 123], and the cost of transmitting the (relatively rare) new items, or a pointer to that information is also marginal. We therefore focus here on optimal allocations of probes, ignoring other related costs.

Part 2: Centrality Maximization

Betweenness centrality (BWC) is a fundamental measure in network analysis, measuring the effectiveness of a vertex in connecting pairs of vertices via shortest paths [42]. Numerous graph mining applications rely on betweenness centrality, such as detecting communities in social and biological networks [47] and understanding the capabilities of an adversary with respect to attacking a network's connectivity [62]. However, in many applications, e.g. [47, 62], we are interested in centrality of sets of nodes. For this reason, the notion of BWC has been extended to sets of nodes [61, 127]. In Chapter 4 we study the betweenness centrality maximization problem which asks for a subset of fixed size whose betweenness centrality is maximized.

Finding the most central nodes in a network is a computationally challenging problem that we are able to handle accurately and efficiently, and improve upon the state-of-the-art algorithm in [127].

Part 3: Influence Estimation

Online social networks allow their users to connect, share information, and interact with each other. These interactions between connected users can cause the members of the networks to be influenced by each other. For instance, a rumor can spread through Facebook as a result of being re-shared, a piece of news can reach a large audience by being re-tweeted multiple times, a user can adopt a new product after seeing it reviewed by his friends, etc. In these scenarios the members of the network are influenced (i.e. they adopted the product or they received the news) via a *cascade* of pairwise interactions which might extend far from the source of the influence. Understanding the phenomena of social influence and information propagation in networks is a fundamental research endeavor and as such it has been subject to many studies including works on identifying the influential members of a network [19, 36, 48, 49, 88, 104, 108]. Understanding social influence has also important applications in the context of online advertisement and viral marketing.

We first (Chapter 5) focus on the problem of estimating the influence exerted by a group of users over another target group of users. This is an important primitive in the context of online social advertisement. Social networks' advertisement campaigns, in fact, are often targeted: The advertiser has a specific group of users in mind that we refer to as the *target set*, such as users in a certain geographic area, in a social group, of a specific occupation, or at an income level. However, direct communication from the advertiser to the entire target set may not be possible and may not be as effective as a third person recommendation. For instance, the advertiser might only directly contact the users that already liked her page (or signed up to a mailing list) and may want to influence a larger population. Therefore, in viral marketing, the goal of the advertiser is to spend her budget, e.g. by free samples or sending coupons, to some influential users of the network that may recommend her product to their friends/followers, with the hope that the product is adopted by a large number of users in her target set through the cascade of recommendations. We refer to the set of users over which the advertiser makes the initial spending as the *seed set*. The advertiser, however, can not spend on all nodes, and she can be restricted to a smaller group of users. Therefore, she needs to estimate the influence of potential seed sets over her target set, in order to decide which seed set to pick. Another potential

application of this method is in the realm of online shopping. Here, one wants to estimate the likelihood that a customer visiting a set of products to end up visiting another set of products by following the similar item recommendations links. This issue as well can be modeled as an influence-propagation instance.

Despite extensive research in modeling, computing, and optimizing the influence in social networks, the focus of most works has been on *global* influence, where the target set is the whole network and the advertiser can access any seed set. This is a significant limitation in the context of target advertisement where only a potentially small fraction of such users are of interest. Note that efficiency of computation in this context is key as a great number of advertisers operate in on-line advertisement systems [57] and each requires obtaining estimates on multiple query sets in real-time to tune her campaign. For this reason, we design efficient preprocessing algorithms that help us serve these advertisers in reasonable time.

We next (Chapter 6) study the problem of estimating the influence with partial observation. Many of the phenomena in real social networks can be modeled as a *spread of influence*. For instance, spread of influence, also known as *cascade*, can model spread of a post/meme on Facebook/Twitter through re-sharing, diffusion of an infectious virus in a network of connected machines, or adaption of a new product when a user received recommendation from her friends. In all of these cases, members in the social network are influenced via pairwise interactions that might go beyond the source of the influence [37, 84, 117].

Estimating the final size of a cascade, before it completes its progression, is of great importance, such as detecting (potential) trending topics or catching popular rumor/news on a social media. There have been some studies on estimating the cascade size that assume the access to the full history of the cascade progression, i.e. when and where in the social network is affected by the cascade [44, 77, 113, 129]. Since we might become interested in a cascade and its final size at a time after its initiation, when there are signs of that being/becoming viral or trending, it is not a realistic assumption that we have access or record of the full history of the cascade. For instance, a website might publish a story and after a while we see the story is mentioned in some other websites that we follow, or a tweet may get our attention when it is retweeted by some people that we follow. Thus, in these cases, we may

only know the source and only some part of the cascade, and therefore, we need to estimate the cascade size via *partial observation* of the cascade.

Part I

Detecting Valuable Information

Chapter 2

Schedule Optimization Problem: *Isolated Items*

We consider an infinite, discrete-time process in which n nodes generate new items according to a stochastic process which is governed by a generating vector π (see Section 2.1 for details). An algorithm can probe up to c nodes per step to discover all new items in these nodes. The goal is to minimize the *cost* of the algorithm (or the probing schedule), which we define as the long term (steady state) expected number of undiscovered items in the system.

We first show that the obvious approach of probing at each step the nodes with maximum expected number of undiscovered items at that step, is not optimal. In fact, the cost of such a schedule can be arbitrary far from the optimal.

Our first result toward the study of efficient schedules is a lower bound on the cost of any deterministic or random schedule as a function of the generating vector π, and the number of simultaneous probes c.

Next we assume that the generating vector π is known and study explicit constructions of deterministic and random schedules. We construct a deterministic schedule whose cost is within a factor of $\left(2 + \frac{2c-1}{2c}\right)$ of the optimal cost, and a very simple, memoryless random schedule with cost that is within a factor of $(2 + (c-1)/c)$ from optimal, where c is the maximum number of probes at each step. We also address the more realistic scenario in which the generating vector, π, is not known to the algorithm and may change in time. We construct an adaptive scheduling algorithm

that learns from probing the nodes and converges to the optimal memoryless random schedule.

We then provide the proof for existence of an ℓ-cyclic optimal c-schedule for **some** positive integer ℓ: A schedule is ℓ-cyclic if after a time t_0, it repeats itself every period of length ℓ steps. We also show that for a **given** fixed ℓ, there exists a deterministic ℓ-cyclic c-schedule whose cost is within a factor of $\frac{2}{1-g(\ell)}$ from the optimal, for $g = o(1)$. Note that this is a better approximation ratio compared to the $\left(2 + \frac{2c-1}{2c}\right)$ ratio, for large enough ℓ's. Finally, at the cost of more memory space, we **construct** a randomized $\left(\frac{2}{1-g(\ell)} + o(1)\right)$-approximation c-schedule. This algorithm can also extend to an adaptive schedule.

2.1 Model and Problem Definition

We study an infinite, discrete-time process in which a set of n *nodes*, indexed by $1, \ldots, n$, generate new *item*s according to a random generating process. The generating process at a given time step is characterized by a *generating vector* $\pi = (\pi_1, \ldots, \pi_n)$, where π_i is the expected number of new items generated at node i at that step (by either a Bernoulli or a Poisson process). The generation processes in different nodes are independent.

We focus first on a *static generating process* in which the generating vector does not change over time. We then extend our results to adapt to generating vectors that change over time.

Our goal is to detect new events as fast as possible by probing in each step a small number of nodes. In particular, we consider probing schedules that can probe up to c nodes per step.

Definition 1 (Schedule)**.** *A* c-schedule *is a function* $\mathcal{S} : \mathbb{N} \to \{1, \ldots, n\}^c$ *specifying a set of* c *nodes to be probed at any time* $t \in \mathbb{N}$*. A deterministic function* $\mathcal{S}$ *defines a* deterministic *schedule, otherwise the schedule is* random*.*

Definition 2 (Memoryless Schedule)**.** *A random schedule is* memoryless *if it is defined by a vector* $p = (p_1, \ldots, p_n)$*, such that at any step the schedule probes a set* C *of* c *items with probability* $\prod_{j \in C} p_i$ *independent of any other event. In that case we use the notation* $\mathcal{S} = p$*.*

Note that for a memoryless schedule $\mathcal{S} = (p_1, \ldots, p_n)$, each p_i is nonnegative and $\sum_{i=1}^{n} p_i = 1$.

Definition 3 (Cyclic Schedule)**.** *A schedule, $\mathcal{S}$, is* ℓ-cyclic *if there is a finite time t_0 such that from time t_0 on, the schedule repeats itself every period of ℓ steps. A schedule is cyclic if it is ℓ-cyclic for some positive integer ℓ. Obviously, every cyclic schedule is deterministic.*

The quality of a probing schedule is measured by the speed in which it discovers new items in the system. When a schedule probes a node i at a time t, all items that were generated at that node by time $t-1$ are discovered (thus, each item is not discovered in at least one step). We define the *cost* of a probing schedule as the long term expected number of undiscovered items in the system.

Definition 4 (Cost)**.** *The* cost *of schedule $\mathcal{S}$ in a system of n nodes with generating vector π is*

$$cost\,(\mathcal{S}, \pi) = \lim_{t\to\infty} \frac{1}{t} \sum_{t'=1}^{t} \mathbb{E}\left[Q^{\mathcal{S}}(t')\right] = \lim_{t\to\infty} \frac{1}{t} \sum_{t'=1}^{t} \sum_{i=1}^{n} \mathbb{E}\left[Q_i^{\mathcal{S}}(t')\right],$$

where $Q_i^{\mathcal{S}}(t')$ is the number of undiscovered items at node i and at time t', and $Q^{\mathcal{S}}(t') = \sum_{i=1}^{n} Q_i^{\mathcal{S}}(t')$. The expectation is taken over the distribution of the generating system and the probing schedule.

While the cost can be unbounded for some schedules, the cost of the optimal schedule is always bounded (we show the existence of an optimal schedule in Section 2.2.6). To see that, consider a round-robin c-schedule, $\mathcal{S}$, that probes each node every $\lceil n/c \rceil$ steps. Clearly no item is undiscovered in this schedule for more than $\lceil n/c \rceil$ steps, and the expected number of items generated in an interval of $\lceil n/c \rceil$ steps is $\lceil n/c \rceil \sum_{i=1}^{n} \pi_i$. Thus, $Q^{\mathcal{S}}(t) \leq \lceil n/c \rceil \sum_{i=1}^{n} \pi_i$, which implies $\text{cost}\,(\mathcal{S}, \pi) \leq \lceil n/c \rceil \sum_{i=1}^{n} \pi_i$. Therefore, without loss of generality we can restrict our discussion to bounded cost schedules. Also, note that when the sequence $\left\{\mathbb{E}\left[Q^{\mathcal{S}}(t)\right]\right\}_{t\in\mathbb{N}}$ converges we have

$$\text{cost}\,(\mathcal{S}, \pi) = \lim_{t\to\infty} \mathbb{E}\left[Q^{\mathcal{S}}(t)\right]$$

by Cesaro Means [54, Sect. 5.4].

One can equivalently define the cost of a schedule in terms of the expected time that an item is in the system until it is discovered.

Lemma 1. *Let ω_i^S be the expected waiting time of an item generated at node i until node i is probed by schedule $\mathcal{S}$. Then*

$$cost\,(\mathcal{S}, \pi) = \sum_{i=1}^{n} \pi_i \omega_i^{\mathcal{S}}.$$

Proof. Following the definition of the cost function we have

$$\begin{aligned}\text{cost}\,(\mathcal{S}, \pi) &= \lim_{t\to\infty} \frac{1}{t} \sum_{t'=1}^{t} \sum_{i=1}^{n} \mathbb{E}\left[Q_i^{\mathcal{S}}(t')\right] = \sum_{i=1}^{n} \left[\lim_{t\to\infty} \frac{\sum_{t'=1}^{t} \mathbb{E}\left[Q_i^{\mathcal{S}}(t')\right]}{t}\right] \\ &= \sum_{i=1}^{n} \pi_i \omega_i^{\mathcal{S}},\end{aligned}$$

where the last eqaulity is obtained by applying Little's Law [81]. □

Corollary 1. *A schedule that minimizes the expected number of undiscovered items in the system simultaneously minimizes the expected time that an item is undiscovered.*

Corollary 2. *For any schedule $\mathcal{S}$, $cost\,(\mathcal{S}, \pi) \geq \sum_{i=1}^{n} \pi_i$.*

Proof. As mentioned above, when we probe a node i at time t we discover only the items that have been generated by time $t-1$. Therefore, $\omega_i^{\mathcal{S}} \geq 1$, and by Lemma 1 the proof is complete. □

Now, our main problem is defined as the following:

Definition 5 (Schedule Optimization)**.** *Given a generating vector π and a positive integer c, find a c-schedule with minimum cost.*

When the generating vector is not known a priori to the algorithm the goal is to design a schedule that *converges* to an optimal one. For that we need the following definition:

Definition 6 (Convergence)**.** *We say schedule $\mathcal{S}$* converges *to schedule $\mathcal{S}'$, if for any generating vector π, $\lim_{t\to\infty} \left|\mathbb{E}\left[Q^{\mathcal{S}}(t)\right] - \mathbb{E}\left[Q^{\mathcal{S}'}(t)\right]\right| = 0$.*

2.2 Results

We start this section by, first, showing that the obvious approach of maximizing the expected number of detections at each step is far from optimal. Next, we prove a lower bound on the cost of any schedule, and provide deterministic and memoryless c-schedules that are within a factor of $\left(2+\frac{2c-1}{2c}\right)$ and $(2+(c-1)/c)$, respectively, from the optimal. Also, introduce an algorithm, `Adaptive`, which outputs a schedule $\mathcal{A}$ that converges to the optimal *memoryless* 1-schedule when the generating vector π is not known in advance. We also show that `Adaptive` can be used to obtain a c-schedule $\mathcal{A}^c$ whose cost is within $(2+(c-1)/c)$ factor of any optimal c-schedule. We then, study cyclic c-schedules and provide the proof for existence of an ℓ-cyclic optimal c-schedule for **some** positive integer ℓ. We also show that for a **given** fixed ℓ, there exists an ℓ-cyclic c-schedule whose cost is within a factor of $\frac{2}{1-g(\ell)}$ from the optimal, for $g=o(1)$. Note that this is a better approximation ratio compared to the $\left(2+\frac{2c-1}{2c}\right)$ ratio, for large enough ℓ's. Finally, at the cost of more memory space, we *construct* a randomized $\left(\frac{2}{1-g(\ell)}+o(1)\right)$-approximation c-schedule. This algorithm can also extend to an adaptive schedule.

Throughout this section, by $\tau_i^{\mathcal{S}}(t)$ we mean the number of steps from the last time that node i was probed until time t, while executing schedule $\mathcal{S}$; if i has not been probed so far, we let $\tau_i^{\mathcal{S}}(t)=t$. Using the definition, it is easy to see that

$$\mathbb{E}\left[Q_i^{\mathcal{S}}(t)\right]=\pi_i\mathbb{E}\left[\tau_i^{\mathcal{S}}(t)\right], \tag{2.1}$$

when the expectations are over the randomness of *both* $\mathcal{S}$ and π. Therefore, if the expectation is over *only* the randomness of π we have

$$\mathbb{E}\left[Q_i^{\mathcal{S}}(t)\right]=\pi_i\tau_i^{\mathcal{S}}(t). \tag{2.2}$$

2.2.1 On Maximizing Immediate Gain

Let $\mathcal{S}$ be a 1-schedule that at each step, probes the node with the maximum expected number of undetected items. By (2.2), the expected number of undetected items at node i and at time t is $\pi_i\tau_i^{\mathcal{S}}(t)$, and thus, $\mathcal{S}(t)=\arg\max_i \pi_i\tau_i^{\mathcal{S}}(t)$.

Now, suppose $\pi_i=2^{-i}$, for $1\le i\le n$. Since the probability that node 1 has an undetected item in each step is at least $1/2$, node i is probed no more than once in

each 2^{i-1} steps. Thus, the expected number of time steps that an item at node i will stay undetected is at least $\frac{1}{2^{i-1}}(1+\ldots+2^{i-1}) = \frac{2^{i-1}+1}{2} > 2^{i-2}$. Using Lemma 1, the cost of this schedule is at least $\sum_{i=1}^{n} \pi_i\omega_i > \sum_{i=1}^{n} 2^{-i}2^{i-2} = \Omega(n)$. Now, consider an alternative schedule that probes node i in each step with probability $2^{-i/2}/Z$, where $Z = \sum_{j=1}^{n} 2^{-j/2}$. The expected number of steps between two probes of i is $Z/2^{-i/2}$, and the cost of this schedule is

$$\sum_{i=1}^{n} 2^{-i}\left(\frac{2^{-i/2}}{\sum_{j=1}^{n} 2^{-j/2}}\right)^{-1} = \left(\sum_{j=1}^{n} 2^{-j/2}\right)^{2} = O(1).$$

Thus, optimizing the immediate gain is not optimal in this example, and its cost can be arbitrarily far from optimal.

2.2.2 Lower Bound on Optimal Cost

In this section we provide a lower bound on the optimal cost, i.e., the cost of an optimal schedule. Later in Section 2.2.6 we show that there always exists an (ℓ-cyclic) optimal schedule.

Theorem 1. *For any optimal c-schedule $\mathcal{O}$ we have*

$$\mathit{cost}(\mathcal{O},\pi) \geq \max\left\{\sum_{i=1}^{n}\pi_i, \frac{1}{2c}\left(\sum_{i=1}^{n}\sqrt{\pi_i}\right)^{2}\right\}.$$

Proof. By Corollary 2, $\mathrm{cost}(\mathcal{O},\pi) \geq \sum_{i=1}^{n}\pi_i$. It remains to show that $\mathrm{cost}(\mathcal{O},\pi) \geq \frac{1}{2c}\left(\sum_{i=1}^{n}\sqrt{\pi_i}\right)^{2}$. Fix a positive integer $t > 0$, and suppose that during the time interval $[0,t]$, $\mathcal{O}$ probes node i at steps $t_1, t_2, \ldots, t_{n_i}$. Let $t_0 = 0$ and $t_{n_i+1} = t$. The sequence $t_0, \ldots, t_{n_i+1}$ partition the interval $[0,t]$ into n_i+1 intervals $I_i(j) = [t_j+1, t_{j+1}]$, for $0 \leq j \leq n_i$. Denote the length of interval $I_i(j)$ by $\ell_i(j) = t_{j+1} - t_j$. Applying the Cauchy-Schwartz inequality we have:

$$\sum_{j=0}^{n_i}\ell_i(j)^2\sum_{j=0}^{n_i}1 \geq \left(\sum_{j=0}^{n_i}\ell_i(j)\right)^{2}$$

$$\Longrightarrow \sum_{j=0}^{n_i}\ell_i(j)^2 \geq \frac{1}{n_i+1}\left(\sum_{j=0}^{n_i}\ell_i(j)\right)^{2} = \frac{t^2}{n_i+1} = \frac{t^2}{n_i}\left(1-\frac{1}{n_i+1}\right). \qquad (2.3)$$

Since new items are generated in node i by a process with expectation π_i per step, for any $t' \in I_i(j)$, the number of undiscovered items at node i at time t', $Q_i^{\mathcal{O}}(t')$, is equal

to the number of items generated in that node since time t_j, which has expectation $\pi_i(t' - t_j)$. Therefore,

$$\begin{aligned}\sum_{t=1}^{t} \mathbb{E}\left[Q_i^{\mathcal{O}}(t')\right] &= \sum_{j=0}^{n_i} \sum_{t' \in I_i(j)} \mathbb{E}\left[Q_i^{\mathcal{O}}(t')\right] = \sum_{j=0}^{n_i} \sum_{t' \in I_i(j)} \pi_i(t' - t_j) \\ &= \sum_{j=0}^{n_i} \pi_i(1 + \ldots + \ell_i(j)) = \pi_i \sum_{j=0}^{n_i} \frac{\ell_i(j)(\ell_i(j)+1)}{2} \\ &\geq \frac{\pi_i}{2} \sum_{j=0}^{n_i} \ell_i(j)^2 \geq \frac{\pi_i}{2} \frac{t^2}{n_i} \left(1 - \frac{1}{n_i + 1}\right). \end{aligned} \tag{2.4}$$

In the first equality above we partitioned the sum to the different intervals, in the second equality we used the fact that for $t' \in I_i(j)$ we have $\mathbb{E}\left[Q_i^{\mathcal{O}}(t')\right] = \pi(t' - t_j)$, the third equation uses the definition of $\ell_i(j)$, and in the last inequality we apply Equation (2.3).

By summing over all nodes and averaging over t, we have

$$\begin{aligned}\sum_{i=1}^{n} \sum_{t'=1}^{t} \frac{1}{t} \mathbb{E}\left[Q_i^{\mathcal{O}}(t')\right] &\geq \sum_{i=1}^{n} \frac{1}{t} \frac{\pi_i}{2} \frac{t^2}{n_i} \left(1 - \frac{1}{n_i + 1}\right) \\ &= \sum_{i=1}^{n} \frac{\pi_i}{2} \frac{t}{n_i} \left(1 - \frac{1}{n_i + 1}\right) &(2.5) \\ &\geq \frac{1}{c} \left(\sum_{i=1}^{n} \frac{n_i}{t}\right) \left(\sum_{i=1}^{n} \frac{\pi_i}{2} \frac{t}{n_i} \left(1 - \frac{1}{n_i + 1}\right)\right) &(2.6) \\ &\geq \frac{1}{2c} \left(\sum_{i=1}^{n} \sqrt{\pi_i} \sqrt{\left(1 - \frac{1}{n_i + 1}\right)}\right)^2, &(2.7) \end{aligned}$$

where in the second line we use the fact that if the schedule executed c probes in each step then $\sum_{i=1}^{n} \frac{n_i}{t} \leq c$, and the third line is obtained by applying the Cauchy-Schwartz inequality.

It remains to show that for an optimal schedule $\mathcal{O}$, and for any i such that $\pi_i > 0$, $\lim_{t \to \infty} n_i = \infty$. For sake of contradiction assume that there is a time s such that the node i is never probed by $\mathcal{O}$ at time $t > s$. So, $\mathbb{E}\left[Q_i^{\mathcal{O}}(t)\right] = \pi(t - s)$ and we have

$$\text{cost}\left(\mathcal{O}, \pi\right) \geq \lim_{t \to \infty} \frac{1}{t} \sum_{t'=s}^{t} \mathbb{E}\left[Q_i^{\mathcal{O}}(t)\right] = \lim_{t \to \infty} \frac{\pi_i}{t} \frac{(t-s)(t-s+1)}{2} = \infty,$$

which is a contradiction. Hence, for all i, $\lim_{t \to \infty} n_i = \infty$, and using (2.4) we obtain

$$\text{cost}\left(\mathcal{O}, \pi\right) \geq \lim_{t \to \infty} \frac{1}{2c} \left(\sum_{i=1}^{n} \sqrt{\pi_i} \sqrt{\left(1 - \frac{1}{n_i + 1}\right)}\right)^2 = \frac{1}{2c} \left(\sum_{i=1}^{n} \sqrt{\pi_i}\right)^2,$$

which completes the proof. □

2.2.3 Deterministic $\left(2+\frac{2c-1}{2c}\right)$-Approximation Schedule

Here, we first construct a deterministic 1-schedule in which each node i is probed approximately every $n_i = \frac{\sum_{j=1}^{n}\sqrt{\pi_j}}{\sqrt{\pi_i}}$ steps, and using that, present our $\left(2+\frac{2c-1}{2c}\right)$-approximation c-schedule. For each i let r_i be a nonnegative integer such that $2^{r_i} \geq n_i > 2^{r_i-1}$, and let $\rho = \max_i r_i$.

Lemma 2. *There is a 2^ρ-cyclic 1-schedule $\mathcal{D}$ such that node i is probed exactly every 2^{r_i} steps.*

Proof. Without loss of generality assume $\sum_{i=1}^{n} 2^{-r_i} = 1$, otherwise we can add auxiliary nodes to complete the sum to 1, with the powers (r_i's) associated with the auxiliary nodes all bounded by ρ.

We prove the lemma by induction on ρ. If $\rho = 0$, then there is only one node, and the schedule is 1-cyclic. Now, assume the statement holds for all $\rho' < \rho$. Since the smallest frequency is $2^{-\rho}$, and the sum of the frequencies is 1, there must be two nodes, v and u, with same frequency $2^{-\rho}$. Join the two nodes to a new node w with frequency $2^{-\rho+1}$. Repeat this process for all nodes with frequency $2^{-\rho}$. We are left with a collection of nodes all with frequencies $> 2^{-\rho}$. By the inductive hypothesis there is a $(2^{\rho-1})$-cyclic schedule $\mathcal{D}'$ such that each node i is probed exactly each 2^{r_i} steps. In particular a node w that replaced u and v is probed exactly each $2^{-\rho+1}$ steps.

Now, we create an 2^ρ-schedule, $\mathcal{D}$, whose cycle is obtained by repeating the cycle of $\mathcal{D}'$ two times. For each probe to w that replaced a pair u, v, in the first cycle we probe u and in the second cycle we probe v. Thus, u and v are probed exactly every 2^ρ steps, and the new schedule does not change the frequency of probing nodes with frequency larger than $2^{-\rho}$. □

Theorem 2. *The cost of the deterministic 1-schedule $\mathcal{D}$ is no more than* 2.5 *times of the optimal cost.*

Proof. By Lemma 2 each node i is probed exactly every 2^{r_i} steps. Using $2^{r_i-1} <$

$\frac{\sum_{j=1}^{n}\sqrt{\pi_j}}{\sqrt{\pi_i}}$ we have $2^{r_i}+1 \le \frac{2\cdot\sum_{j=1}^{n}\sqrt{\pi_j}}{\sqrt{\pi_i}}+1$, and therefore

$$\begin{aligned}\lim_{t\to\infty}\frac{1}{t}\sum_{t'=1}^{t}\mathbb{E}\left[Q_i^{\mathcal{D}}(t')\right] &= \lim_{t\to\infty}\frac{1}{t}\left(\frac{t}{2^{r_i}}\sum_{t'=1}^{2^{r_i}}\mathbb{E}\left[Q_i^{\mathcal{D}}(t')\right]\right) = \frac{1}{2^{r_i}}\sum_{t'=1}^{2^{r_i}}\pi_i t' \\ &= \frac{\pi_i}{2^{r_i}}\frac{2^{r_i}(2^{r_i}+1)}{2} \le \frac{\pi_i}{2}\left(\frac{2\sum_{j=1}^{n}\sqrt{\pi_j}}{\sqrt{\pi_i}}+1\right) \\ &= \sqrt{\pi_i}\cdot\sum_{j=1}^{n}\sqrt{\pi_j}+\frac{\pi_i}{2}.\end{aligned}$$

Thus by Theorem 1, we have

$$\begin{aligned}\mathrm{cost}\left(\mathcal{D},\pi\right) &= \lim_{t\to\infty}\frac{1}{t}\sum_{i=1}^{n}\sum_{t'=1}^{t}\mathbb{E}\left[Q_i^{\mathcal{D}}(t')\right] \le \sum_{i=1}^{n}\left(\sqrt{\pi_i}\cdot\sum_{j=1}^{n}\sqrt{\pi_j}\right)+\frac{1}{2}\sum_{i=1}^{n}\pi_i \\ &= \left(\sum_{j=1}^{n}\sqrt{\pi_j}\right)^2+\frac{1}{2}\sum_{j=1}^{n}\pi_j \le 2.5\cdot\mathrm{cost}\left(\mathcal{O},\pi\right),\end{aligned}$$

where $\mathrm{cost}\left(\mathcal{O},\pi\right)$ is the optimal cost. □

Using the previous deterministic 1-schedule, the following corollary provides a c-schedule whose cost is within $\left(2+\frac{2c-1}{2c}\right)$ factor of the optimal cost.

Corollary 3. *There is a deterministic c-schedule $\mathcal{D}^c$ whose cost is at most $\left(2+\frac{2c-1}{2c}\right)$ times of the optimal cost.*

Proof. Consider the execution of the deterministic 1-schedule $\mathcal{D}$ constructed in Theorem 2 on generating vector $\frac{1}{c}\pi$. Let $\mathcal{D}^c$ be a deterministic c-schedule obtained by grouping c consecutive probes of $\mathcal{D}$ into one step. Suppose $\mathcal{O}$ is an optimal c-schedule. Applying Lemma 1,

$$\begin{aligned}\mathrm{cost}\left(\mathcal{D}^c,\pi\right) &= \sum_{i=1}^{n}\pi_i\omega_i^{\mathcal{D}^c} = \sum_{i=1}^{n}\frac{\pi_i}{c}c\omega_i^{\mathcal{D}^c} \le \sum_{i=1}^{n}\frac{\pi_i}{c}(\omega_i^{\mathcal{D}}+c-1) \\ &= \mathrm{cost}\left(\mathcal{D},\frac{1}{c}\pi\right)+\sum_{i=1}^{n}\frac{(c-1)\pi_i}{c} \\ \text{(from the proof of Theorem 2)} &\le \left(\sum_{i=1}^{n}\sqrt{\frac{\pi_i}{c}}\right)^2+\frac{1}{2}\sum_{i=1}^{n}\frac{\pi_i}{c}+\sum_{i=1}^{n}\frac{(c-1)\pi_i}{c} \\ &= \frac{1}{c}\left(\sum_{i=1}^{n}\sqrt{\pi_i}\right)^2+\frac{2c-1}{2c}\sum_{i=1}^{n}\pi_i \\ \text{(by Theorem 1)} &\le \left(2+\frac{2c-1}{2c}\right)\mathrm{cost}\left(\mathcal{O},\pi\right),\end{aligned}$$

where the first inequality holds because some items could be detected in less than c steps in the 1-schedule but are counted in one full step of the c-schedule. □

2.2.4 On Optimal Memoryless Schedule

Here, we consider memoryless schedules, and show that the memoryless 1-schedule with minimum cost can be easily computed. We call a memoryless schedule with minimum cost among memoryless schedules, an optimal memoryless schedule. We also provide an upper bound on the minimum cost of a memoryless c-schedule.

Theorem 3. *Let $\mathcal{R} = (p_1, \ldots, p_n)$ be a memoryless 1-schedule. Then we have cost$(\mathcal{R}, \pi) \geq \left(\sum_{i=1}^{n} \sqrt{\pi_i}\right)^2$, and the equality holds if and only if $p_i = \frac{\sqrt{\pi_i}}{\sum_{j=1}^{n} \sqrt{\pi_j}}$, for all i.*

Proof. Since probing each node i is a geometric distribution with parameter p_i, the expected time until an item generated at node i is discovered, is $\omega_i^{\mathcal{R}} = 1/p_i$. Therefore, by Lemma 1, we have $\text{cost}(\mathcal{R}, \pi) = \sum_{i=1}^{n} \frac{\pi_i}{p_i}$. We find $p^* = \arg\min_{\mathcal{S}=p} \text{cost}(\mathcal{R}, \pi)$, using the Lagrange multipliers:

$$\frac{\partial}{\partial p_j}\left(\sum_{i=1}^{n} \frac{\pi_i}{p_i} + \lambda \sum_{i=1}^{n} p_i\right) = 0 \Longrightarrow p_j \propto \sqrt{\pi_j}.$$

Therefore, $\text{cost}(\mathcal{R}, \pi)$ is minimized if $p_i = \frac{\sqrt{\pi_i}}{\sum_{j=1}^{n} \sqrt{\pi_j}}$, and in this case the (minimized) cost will be

$$\text{cost}(\mathcal{R}, \pi) = \sum_{i=1}^{n} \left(\sqrt{\pi_i} \cdot \sum_{j=1}^{n} \sqrt{\pi_j}\right) = \left(\sum_{i=1}^{n} \sqrt{\pi_i}\right)^2.$$

□

Corollary 4. *The cost of the optimal memoryless 1-schedule is within a factor of 2 of the cost of any optimal 1-schedule.*

Proof. The cost of the schedule $\mathcal{R}$ in Theorem 3 is $\left(\sum_{i=1}^{n} \sqrt{\pi_i}\right)^2$, which is bounded by $2 \cdot \text{cost}(\mathcal{O}, \pi)$ for an optimal 1-schedule $\mathcal{O}$ using Theorem 1. □

Corollary 5. *There is memoryless c-schedule, $\mathcal{R}^c$, whose cost is within a factor of $(2 + (c-1)/c)$ of any optimal c-schedule.*

Proof. Suppose $\mathcal{R}^c$ is a memoryless c-schedule obtained by choosing c probes in each step, each chosen according to the optimal memoryless 1-schedule, $\mathcal{R}$, computed in Theorem 3. Using the same argument as in the proof of Corollary 3 we have

$$\operatorname{cost}(\mathcal{R}^c, \pi) \leq \frac{1}{c}\left(\sum_{i=1}^{n}\sqrt{\pi_i}\right)^2 + \frac{c-1}{c}\sum_{i=1}^{n}\pi_i \leq \left(2 + \frac{c-1}{c}\right)\operatorname{cost}(\mathcal{O}, \pi),$$

for an optimal c-schedule $\mathcal{O}$. □

2.2.5 On Adaptive Algorithm for Memoryless Schedules

Assume now that the scheduling algorithm starts with no information on the generating vector π (or that the vector has changed). We design and analyze an adaptive algorithm, `Adaptive`, that outputs a schedule $\mathcal{A}$ convergent to the optimal memoryless algorithm $\mathcal{R}$ (see Section 2.2.4) by gradually learning the vector π by observing the system. To simplify the presentation we present and analyze a 1-schedule algorithm. The results easily scale up to any integer $c > 1$, where the adaptive algorithm outputs a c-schedule convergent to $\mathcal{R}^c$ (as in Section 2.2.4).

Each iteration of the algorithm `Adaptive` starts with $\tilde{\pi} = (\tilde{\pi}_1, \ldots, \tilde{\pi}_n)$ as an estimate of the unknown generating vector $\pi = (\pi_1, \ldots, \pi_n)$. Based on this estimate the algorithm chooses to probe node i with probability $p_i(t) = \frac{\sqrt{\tilde{\pi}_i}}{\sum_{j=1}^{n}\sqrt{\tilde{\pi}_j}}$ (which is the optimal memoryless schedule if $\tilde{\pi}$ was the correct estimate). If node i_0 is probed at time t, the estimate of π_{i_0} is updated to $\tilde{\pi}_{i_0} \leftarrow \frac{\max(1, c_{i_0})}{t}$, where c_{i_0} is the total number of new items discovered in that node since time 0.

We denote the output of `Adaptive` schedule by $\mathcal{A}$ and the optimal memoryless 1-schedule by $\mathcal{R} = p^* = (p_1^*, \ldots, p_n^*)$; see Section 2.2.4. Our main result of this section is the following theorem.

Theorem 4. *The schedule $\mathcal{A}$ converges to $\mathcal{R}$, and thus,*

$$\mathit{cost}(\mathcal{A}, \pi) = \mathit{cost}(\mathcal{R}, \pi).$$

To prove Theorem 4 we need the following lemmas.

Lemma 3. *For any time t and $i \in [n]$ we have $p_i(t) \geq \frac{1}{n\sqrt{t}}$.*

Proof. It is easy to see that $p_i(t)$ will reach its lowest value at time t only if for $j \neq i$ we have $\tilde{\pi}_j = 1$ and $\tilde{\pi}_i = \frac{1}{t-1}$ (which requires i to be probed at time $t-1$). Therefore, $p_i(t) \geq \frac{1/\sqrt{t-1}}{1/\sqrt{t-1}+n-1} = \frac{1}{1+(n-1)\sqrt{t-1}} \geq \frac{1}{n\sqrt{t}}$. □

Algorithm 1: `Adaptive`

Outputs: $\mathcal{A}^1(t)$, for $t = 1, 2, \ldots$;
begin
  $(c_1, \ldots, c_n) \leftarrow (0, \ldots, 0)$;
  $(\tilde{\pi}_1, \ldots, \tilde{\pi}_n) \leftarrow (1, \ldots, 1)$;
  for $t = 1, 2, \ldots$ **do**
    for $i \in \{1, \ldots, n\}$ **do**
      $p_i(t) \leftarrow \frac{\sqrt{\tilde{\pi}_i}}{\sum_{j=1}^{n} \sqrt{\tilde{\pi}_j}}$;
    end
    $\mathcal{A}^1(t) \sim p(t)$;
    output $\mathcal{A}^1(t)$;
    $c' \leftarrow$ number of new items caught at $i_0 = \mathcal{A}^1(t)$;
    $c_{i_0} \leftarrow c_{i_0} + c'$;
    $\tilde{\pi}_{i_0} \leftarrow \frac{\max(1, c_{i_0})}{t}$;
  end
end

Define $\delta(t) = 4n \exp\left(-\frac{\pi_* t^{1/3}}{6}\right)$, where $\pi_* = \min\{\pi_1, \ldots, \pi_n\}$, and let N_0 be the smallest integer t such that $\exp\left(-\frac{\sqrt{t}}{2n}\right) \leq 2\exp\left(-\frac{\pi_* t^{1/3}}{6}\right)$. Note that one can choose $\delta(t) = 4ne^{-\frac{\pi_* t^{1/2-\epsilon}}{6}}$ for any $\epsilon \in (0, 1/2)$, and for convenience we chose $\epsilon = 1/6$.

Lemma 4. *For any time $t \geq N_0$, with probability $\geq 1 - \delta(t)/2$, all the nodes are probed during the time interval $[t/2, t)$.*

Proof. By Lemma 3, the probability of not probing i during the time interval $[t/2, t)$ is at most

$$\prod_{t'=t/2}^{t-1} (1 - p_i(t')) \leq \left(1 - \frac{1}{n\sqrt{t}}\right)^{t/2} \leq e^{-\frac{t}{2n\sqrt{t}}} \leq 2\exp\left(-\frac{\pi_* t^{1/3}}{6}\right) = \frac{\delta(t)}{2n},$$

where we used the fact that for $t \geq N_0$ we have $\exp\left(-\frac{\sqrt{t}}{2n}\right) \leq 2\exp\left(-\frac{\pi_* t^{1/3}}{6}\right)$. Finally, a union bound over all the nodes completes the proof. □

Lemma 5. *Suppose node i is probed at a time $t' \geq t/2$. Then,*

$$Pr\left[|\tilde{\pi}_i(t') - \pi_i| > t^{-\frac{1}{3}}\pi_i\right] < \frac{\delta(t)}{2n}.$$

Proof. We estimate π from $t' \geq t/2$ steps, each with π_i expected number of new items. Applying a Chernoff bound [92, Ch. 4] for the sum of t' independent random variables with either Bernulli or Poisson distribution we have

$$\Pr\left[|\tilde{\pi}_i(t') - \pi_i| > t^{-\frac{1}{3}}\pi_i\right] < 2\exp\left(-\frac{t^{-\frac{2}{3}}\pi_i t'}{3}\right) \leq 2\exp\left(-\frac{t^{-\frac{2}{3}}\pi_* t}{6}\right) = \frac{\delta(t)}{2n}.$$

□

Note that by union bound, Lemma 5 holds, with probability at least $1 - \delta(t)/2$, for all the nodes that are probed after $t/2$.

Lemma 6. *Suppose $t \geq N_0$. With probability at least $1 - \delta(t)$ we have for all $i \in [n]$,*

$$\left(1 - \frac{1}{t^{1/3}+1}\right)p_i^* \leq \sqrt{\frac{1-t^{-1/3}}{1+t^{-1/3}}}p_i^* \leq p_i(t) \leq \sqrt{\frac{1+t^{-1/3}}{1-t^{-1/3}}}p_i^* \leq \left(1 + \frac{1}{t^{1/3}-1}\right)p_i^*$$

Proof. Applying Lemma 4, Lemma 5 and a union bound, with probability $1-\delta(t)$ all the nodes are probed during the time $[t/2, t)$ and $|\tilde{\pi}_i(t) - \pi_i| \leq t^{-1/3}\pi_i$ for all $i \in [n]$. Since $p_i(t) = \frac{\sqrt{\tilde{\pi}_i(t)}}{\sum_j \sqrt{\tilde{\pi}_j(t)}}$, we obtain

$$p_i(t) \geq \frac{\sqrt{(1-t^{-1/3})\pi_i}}{\sum_j \sqrt{(1+t^{-1/3})\pi_j}} = \sqrt{\frac{1-t^{-1/3}}{1+t^{-1/3}}}\frac{\sqrt{\pi_i}}{\sum_j \sqrt{\pi_j}} = \sqrt{\frac{1-t^{-1/3}}{1+t^{-1/3}}}p_i^* \geq \left(1 - \frac{1}{t^{1/3}+1}\right)p_i^*$$

where the last inequality uses the Taylor series of $\sqrt{1+x}$. The upper bound is obtained by a similar argument. □

Corollary 6. *The variation distance between the distribution used by algorithm* `Adaptive` *at time $t \geq N_0$, $p(t) = (p_1(t), \ldots, p_n(t))$, and the distribution $p^* = (p_i^*, \ldots, p_n^*)$ used by the optimal memoryless algorithm satisfy*

$$\| p(t) - p^* \| = \frac{1}{2}\sum_{i=1}^{n} |p_i(t) - p_i^*| \leq \frac{n}{t^{1/3}-1} + \delta(t) \overset{t\to\infty}{\longrightarrow} 0.$$

Finally, we present our proof for Theorem 4.

Proof of Theorem 4. Recall that we defined $\tau_i^{\mathcal{S}}(t)$ as the number of steps from the last time that node i was probed until time t in an execution of an schedule $\mathcal{S}$, and $\mathbb{E}\left[Q_i^{\mathcal{S}}\right] = \pi_i \mathbb{E}\left[\tau_i^{\mathcal{S}}(t)\right]$ (see Equation (2.2)).

Let $F(t)$ indicate the event that the inequalities in Lemma 6 are held for $\forall t' \in [t/2, t)$. Therefore, $\Pr[F(t)] \geq 1 - \frac{\delta(t/2)\cdot t}{2}$ by applying union bound over all $t' \in [t/2, t)$, and using the fact that $\delta(t') \leq \delta(t/2)$. Therefore,

$$\begin{aligned}
|\mathbb{E}\left[Q^{\mathcal{A}}(t)\right] - \mathbb{E}\left[Q^{\mathcal{R}}(t)\right]| &= \left|\sum_{i=1}^{n} \pi_i \mathbb{E}\left[\tau_i^{\mathcal{A}}(t)\right] - \sum_{i=1}^{n} \pi_i \mathbb{E}\left[\tau_i^{\mathcal{R}}(t)\right]\right| \\
&\leq \sum_{i=1}^{n} \pi_i \left|\mathbb{E}\left[\tau_i^{\mathcal{A}}(t)\right] - \mathbb{E}\left[\tau_i^{\mathcal{R}}(t)\right]\right| \\
&\leq \sum_{i=1}^{n} \pi_i \left|\mathbb{E}\left[\tau_i^{\mathcal{A}}(t)\right] - \mathbb{E}\left[\tau_i^{\mathcal{A}}(t) \mid F(t)\right]\right| \\
&+ \sum_{i=1}^{n} \pi_i \left|\mathbb{E}\left[\tau_i^{\mathcal{A}}(t) \mid F(t)\right] - \mathbb{E}\left[\tau_i^{\mathcal{R}}(t)\right]\right|,
\end{aligned}$$

where we used the triangle inequality for both inequalities. So, it suffices to show that for every i,

$$\lim_{t\to\infty} \left|\mathbb{E}\left[\tau_i^{\mathcal{A}}(t)\right] - \mathbb{E}\left[\tau_i^{\mathcal{A}}(t) \mid F(t)\right]\right| = \lim_{t\to\infty} \left|\mathbb{E}\left[\tau_i^{\mathcal{A}}(t) \mid F(t)\right] - \mathbb{E}\left[\tau_i^{\mathcal{R}}(t)\right]\right| = 0.$$

Obviously, $\tau_i^{\mathcal{A}}(t) \leq t$. Now by letting $t \geq 2N_0$ we have,

$$\begin{aligned}
\mathbb{E}\left[\tau_i^{\mathcal{A}}(t)\right] &= \Pr[F(t)]\, \mathbb{E}\left[\tau_i^{\mathcal{A}}(t) \mid F(t)\right] + \Pr[\neg F(t)]\, \mathbb{E}\left[\tau_i^{\mathcal{A}}(t) \mid \neg F(t)\right] \\
&\leq \mathbb{E}\left[\tau_i^{\mathcal{A}}(t) \mid F(t)\right] + \frac{\delta(t/2)t}{2} t = \mathbb{E}\left[\tau_i^{\mathcal{A}}(t) \mid F(t)\right] + \frac{\delta(t/2)t^2}{2}. \qquad (2.8)
\end{aligned}$$

We also get

$$\begin{aligned}
\mathbb{E}\left[\tau_i^{\mathcal{A}}(t)\right] &\geq \left(1 - \frac{\delta(t/2)t}{2}\right) \mathbb{E}\left[\tau_i^{\mathcal{A}}(t) \mid F(t)\right] \\
&= \mathbb{E}\left[\tau_i^{\mathcal{A}}(t) \mid F(t)\right] - \frac{\delta(t/2)t}{2} \mathbb{E}\left[\tau_i^{\mathcal{A}}(t) \mid F(t)\right] \\
&\geq \mathbb{E}\left[\tau_i^{\mathcal{A}}(t) \mid X\right] - \frac{\delta(t)t^2}{2}. \qquad (2.9)
\end{aligned}$$

Note that $\lim_{t\to\infty} \frac{\delta(t/2)t^2}{2} = \lim_{t\to\infty} 4ne^{-\frac{\pi_* t^{1/3}}{6\sqrt[3]{2}}} t^2 = 0$, and thus by (2.8) and (2.9) we have

$$\lim_{t\to\infty} \mathbb{E}\left[\tau_i^{\mathcal{A}}(t)\right] - \mathbb{E}\left[\tau_i^{\mathcal{A}}(t) \mid F(t)\right] = 0 \Rightarrow \lim_{t\to\infty} \left|\mathbb{E}\left[\tau_i^{\mathcal{A}}(t)\right] - \mathbb{E}\left[\tau_i^{\mathcal{A}}(i) \mid F(t)\right]\right| = 0. \qquad (2.10)$$

Now, we show that $\lim_{t\to\infty} \left|\mathbb{E}\left[\tau_i^{\mathcal{A}}(t) \mid F(t)\right] - \mathbb{E}\left[\tau_i^{\mathcal{R}}(t)\right]\right| = 0$. So here, we assume $F(t)$ holds. So for every $i \in [n]$, node i is probed in $[t/2, t)$, and for all $t' \in [t/2, t)$ we have

(i) $p_i(t') \geq \left(1 - \frac{1}{t'^{1/3}+1}\right) p_i^* \geq \left(1 - \frac{1}{(t/2)^{1/3}+1}\right) p_i^*$. So,

$$\mathbb{E}\left[\tau_i^{\mathcal{A}}(t) \mid F(t)\right] \leq \left(1 - \frac{1}{(t/2)^{1/3}+1}\right)^{-1} \frac{1}{p_i^*} = \left(1 + (t/2)^{-1/3}\right) \frac{1}{p_i^*}.$$

(ii) $p_i(t') \leq \left(1 + \frac{1}{t'^{1/3}-1}\right) p_i^* \leq \left(1 + \frac{1}{(t/2)^{1/3}-1}\right) p_i^*$. Hence,

$$\mathbb{E}\left[\tau_i^{\mathcal{A}}(t) \mid F(t)\right] \geq \left(1 + \frac{1}{(t/2)^{1/3}-1}\right)^{-1} \frac{1}{p_i^*} = \left(1 - (t/2)^{-1/3}\right) \frac{1}{p_i^*}.$$

Obviously $\mathbb{E}\left[\tau_i^{\mathcal{R}}(t)\right] = \frac{1}{p_i^*}$, since probing node i by $\mathcal{R}$ can be viewed as a geometric distribution with parameter p_i^*, and since $\delta(t) \to 0$ as $t \to \infty$ we have

$$\begin{aligned}\mathbb{E}\left[\tau_i^{\mathcal{R}}(t)\right] = \frac{1}{p_i^*} = \lim_{t\to\infty}\left(1 - (t/2)^{-1/3}\right)\frac{1}{p_i^*} \leq \lim_{t\to\infty}\mathbb{E}\left[\tau_i^{\mathcal{A}}(t) \mid F(t)\right] \\ \leq \lim_{t\to\infty}\left(1 + (t/2)^{-1/3}\right)\frac{1}{p_i^*} = \frac{1}{p_i^*} = \mathbb{E}\left[\tau_i^{\mathcal{R}}(t)\right].\end{aligned}$$

Therefore,

$$\lim_{t\to\infty} |\mathbb{E}\left[\tau_i^{\mathcal{A}}(t) \mid F(t)\right] - \mathbb{E}\left[\tau_i^{\mathcal{R}}(t)\right]| = 0. \tag{2.11}$$

Thus, by (2.10) and (2.11) we have $\lim_{t\to\infty} |\mathbb{E}\left[Q^{\mathcal{A}}(t)\right] - \mathbb{E}\left[Q^{\mathcal{R}}(t)\right]| = 0$, and $\mathcal{A}$ converges to $\mathcal{S}$, and since $\lim_{t\to\infty}\mathbb{E}\left[Q^{\mathcal{S}}(t)\right] = \left(\sum_{i=1}^n \sqrt{\pi_i}\right)^2$, it implies that

$$\lim_{t\to\infty}\mathbb{E}\left[Q^{\mathcal{A}}(t)\right] = \left(\sum_{i=1}^n \sqrt{\pi_i}\right)^2 = \text{cost}\left(\mathcal{A}, \pi\right)$$

by Cesaro Mean [54, Sect. 5.4]. □

Note that one can obtain an adaptive schedule $\mathcal{A}^c$ by choosing c probes in each step, at each round of `Adaptive`, and using similar argument as in Section 2.2.4 (and similar to Corollary 3), it is easy to see that $\mathcal{A}^c$ converges to $\mathcal{R}^c$.

Finally, if π changes, the `Adaptive` algorithm converges to the new optimal memoryless algorithm, as the change in the rate of generating new items is observed by `Adaptive`.

2.2.6 On Cyclic Schedules

In this section we study cyclic schedules (which are also deterministic). In particular, we first show that for **some** ℓ, there exists an ℓ-cyclic optimal schedule. Next, we show that for a **given** ℓ there exists a $\left(\frac{2}{1-g(\ell)}\right)$-approximation c-schedule, where g is a decreasing function that converges to 0.

2.2.6.1 Existence of Cyclic Optimal Schedule

Our main result in this section is the following theorem.

Theorem 5. *For any positive integer c, there always exists a cyclic optimal c-schedule.*

In order to prove Theorem 5 we need the following definitions and lemmas.

Definition 7 (Configuration State)**.** *Suppose $\mathcal{S}$ is a c-schedule. The **configuration state** (or simply **state**) of $\mathcal{S}$ at time t is the vector $\tau^{\mathcal{S}}(t) = (\tau_1^{\mathcal{S}}(t), \ldots, \tau_n^{\mathcal{S}}(t))$, where $\tau_i^{\mathcal{S}}(t)$ is the number of steps from the last time that node i was probed until time t.*

Definition 8 (Configuration Graph)**.** *The (π, c)-**configuration graph** is a directed node-weighted infinite sized graph $\mathcal{G}_{\pi,c} = (\mathcal{V}, \mathcal{E}, z)$, where $\mathcal{V}$ is the set of all possible configuration states of **any** schedule $\mathcal{S}$ at **any** time t. For two nodes $\eta_1, \eta_2 \in \mathcal{V}$ we have $\eta_1\eta_2 \in \mathcal{E}$ if there exist a time t and a c-schedule $\mathcal{S}$ such that $\eta_1 = \tau^{\mathcal{S}}(t)$ and $\eta_2 = \tau^{\mathcal{S}}(t+1)$. The weight of every node $\eta = (\tau_1, \ldots, \tau_n)$ is defined as $z(\eta) = \sum_{i=1}^{n} \pi_i \cdot \tau_i$.*

Note that as we mentioned before in Equation (2.2), if $\eta = \tau^{\mathcal{S}}(t)$ we have

$$\mathbb{E}\left[Q^{\mathcal{S}}(t)\right] = \sum_{i=1}^{n} \pi_i \tau_i^{\mathcal{S}}(t) = z(\eta). \tag{2.12}$$

For any finite subset X of states in $\mathcal{G}_{\pi,c}$, define $z(X) = \sum_{\eta \in X} z(\eta)$. We have to following important lemma about the cycles[1] in $\mathcal{G}_{\pi,c}$:

Lemma 7. *There exists a cycle C in $\mathcal{G}_{\pi,c}$ such that*

$$\frac{z(C)}{|C|} = \min\left\{ \frac{z(C')}{|C'|} \mid C' \text{ is cycle in } \mathcal{G}_{\pi,c} \right\}.$$

[1] A (directed) cycle does not pass a state twice.

Proof. Let C_R be the cycle of length n obtained by performing a round-robin schedule:

$$C_R = (1, \ldots, n-1, n) \to (2, 3, \ldots, n, 1) \to (3, 4, \ldots, n, 1, 2) \to \ldots$$

Also, let $0 < B = z(C_R)/|C_R|$ and $\mathcal{F} = \{\eta \in \mathcal{G}_{\pi,c} \mid z(\eta) \leq 2B\}$. Note that $\mathcal{F}$ is finite: if $\eta = (\tau_1, \ldots, \tau_n) \in \mathcal{F}$ then $\tau_i \leq \left\lfloor \frac{2B}{\pi_i} \right\rfloor$, $1 \leq i \leq N$, since otherwise $z(\eta) > 2B$. So,

$$\mathcal{F} \subseteq \left\{ \eta \in \mathcal{G}_{\pi,c} \mid 1 \leq \tau_i \leq \left\lfloor \frac{2B}{\pi_i} \right\rfloor \text{ and } \tau_i \text{ is positive integer, for } 1 \leq i \leq n \right\},$$

and thus $|\mathcal{F}| \leq \prod_{i=1}^{N} \left\lfloor \frac{2B}{\pi_i} \right\rfloor$.

Claim. *If $|C| \geq 2|\mathcal{F}|$ for a cycle C, then $z(C)/|C| > B$.*

This is because

$$\begin{aligned}
\frac{z(C)}{|C|} &\geq \frac{1}{|C|} \left(\sum_{\eta \in \mathcal{F}} z(\eta) + \sum_{\eta \in C \setminus \mathcal{F}} z(\eta) \right) \\
&> \frac{1}{|C|} \left(\sum_{\eta \in \mathcal{F}} 0 + \sum_{\eta \in C \setminus \mathcal{F}} 2B \right) \geq \frac{2B}{|C|} \left(|C| - |\mathcal{F}| \right) \\
&= 2B - \frac{|\mathcal{F}|}{|C|} 2B \geq 2B - B = B.
\end{aligned}$$

Now, suppose $\mathcal{F}' \subseteq \mathcal{F}$ is the set of states than can be reached from a state in $\mathcal{F}$ via a path of length at most $2|\mathcal{F}|$. Therefore, by the above claim, if $z(C)/|C| \leq B$ for a cycle C, then $C \subseteq \mathcal{F}'$. Obviously, $\mathcal{F}'$ is finite, and so is the set

$$\{C \mid C \text{ is a cycle and } C \subseteq \mathcal{F}'\},$$

and therefore, there is cycle C that minimizes $z(C)/|C|$. □

From now on, we denote C^* to be the cycle in $\mathcal{G}_{\pi,c}$ that minimizes $z(C)/|C|$.

Corollary 7. *Suppose W is a closed walk[2] in $\mathcal{G}_{\pi,c}$. We have $\frac{z(W)}{|W|} \geq \frac{z(C^*)}{|C^*|}$.*

Proof. We prove by induction on the length of W. If $|W| = 2$, W is a cycle (since there is no self-loop in $G_{\pi,c}$ for $n > 1$), and by Lemma 7 the statement holds. Now, suppose the statement holds for all the closed walks of length smaller than $|W|$. If W is a cycle, again by Lemma 7 the statement holds. So, assume that W is not a cycle.

[2] A walk in a graph is a sequence of nodes obtained by traversing the edges.

In this case note that we can decompose W as $W = W_1 C W_2$, where W_1, W_2 are two walks and C is a cycle, such that $|W_1| + |W_2| > 0$ (since W is not a cycle) and $W_1 W_2$ is a closed walk. Therefore,

$$\frac{z(W)}{|W|} \geq \frac{z(W_1W_2) + z(C)}{|W_1W_2| + |C|} \geq \min\left\{\frac{z(W_1W_2)}{|W_1W_2|}, \frac{z(C)}{|C|}\right\},$$

where we used the fact that $\frac{\alpha+\beta}{\gamma+\delta} \geq \min\left\{\frac{\alpha}{\gamma}, \frac{\beta}{\delta}\right\}$, for positive integers $\alpha, \beta, \gamma, \delta$. Now, by induction hypothesis we know that $\frac{z(W_1W_2)}{|W_1W_2|} \geq \frac{z(C^*)}{|C^*|}$ (since W_1W_2 is a closed walk), and by Lemma 7 we have $\frac{z(C)}{|C|} \geq \frac{z(C^*)}{|C^*|}$. Therefore, $\frac{z(W)}{|W|} \geq \frac{z(C^*)}{|C^*|}$. □

Definition 9 (Walk)**.** *The **walk** of a c-schedule $\mathcal{S}$, denoted by $W(\mathcal{S})$, is the infinite walk in graph $G_{\pi,c}$ that $\mathcal{S}$ traverse, i.e.,*

$$W(\mathcal{S}) = \tau^{\mathcal{S}}(1), \tau^{\mathcal{S}}(2), \tau^{\mathcal{S}}(3), \ldots$$

Now, by Equation (2.12), the cost of a c-schedule $\mathcal{S}$ is the average of weights of all nodes in $W(\mathcal{S})$, i.e.,

$$\text{cost}\,(\mathcal{S}, \pi) = \lim_{t\to\infty} \frac{1}{t} \sum_{t'=1}^{t} z(\tau^{\mathcal{S}}(t')). \tag{2.13}$$

Remark 1. *Note that an infinite walk W, that starts from the state $(1, \ldots, 1)$, uniquely defines a schedule $\mathcal{S}$ such that $W(\mathcal{S}) = W$: each directed edge in $G_{\pi,c}$ corresponds to probing a set of at most c nodes. So, $\mathcal{S}$ is a schedule that at time step t probes the nodes corresponding to the i-th edge traversed by W.*

Theorem 6. *For any schedule $\mathcal{S}$, we have cost$\,(\mathcal{S}, \pi) \geq \frac{z(C^*)}{|C^*|}$.*

Proof. If $\text{cost}\,(\mathcal{S}, \pi) = \infty$ there is nothing to prove. So, assume $\text{cost}\,(\mathcal{S}, p) < \infty$. Define

$$\mathcal{F}_{\mathcal{S}} = \{\eta \in \mathcal{G}_{\pi,c} \mid z(\eta) \leq \text{cost}\,(\mathcal{S}, p) + 1\}.$$

We claim that the walk of $\mathcal{S}$ visits $\mathcal{F}_{\mathcal{S}}$ infinite times, i.e., there is an infinite sequence of time steps $t_1 < t_2 < \ldots$ such that $\tau^{\mathcal{S}}(t_i) \in \mathcal{F}_{\mathcal{S}}$: otherwise, there is a time t_0 such that for $t \geq t_0$ we have $\tau^{\mathcal{S}}(t) \notin \mathcal{F}_{\mathcal{S}}$, and thus, $\tau^{\mathcal{S}}(t) > \text{cost}\,(\mathcal{S}, \pi) + 1$. Now by

Equation (2.13), we have

$$\begin{aligned}\operatorname{cost}(\mathcal{S},\pi) &= \lim_{t\to\infty}\frac{1}{t}\left(\sum_{t'=1}^{t_0}\tau^{\mathcal{S}}(t')+\sum_{t'=t_0+1}^{t}\tau^{\mathcal{S}}(t')\right)=\lim_{t\to\infty}\frac{1}{t}\left(\sum_{t'=t_0+1}^{t}\tau^{\mathcal{S}}(t')\right)\\ &\geq \lim_{t\to\infty}\frac{1}{t}\left(\sum_{t'=t_0+1}^{t}\operatorname{cost}(\mathcal{S},\pi)+1\right)=\lim_{t\to\infty}\frac{t-t_0}{t}\left(\operatorname{cost}(\mathcal{S},\pi)+1\right)\\ &= \operatorname{cost}(\mathcal{S},\pi)+1>\operatorname{cost}(\mathcal{S},\pi),\end{aligned}$$

which is a contradiction. Therefore, $W(\mathcal{S})$ visits $\mathcal{F}_{\mathcal{S}}$ infinite times, and since $\mathcal{F}_{\mathcal{S}}$ is a finite set (same reason for finiteness of $\mathcal{F}$ defined above), there exists a state $\eta\in\mathcal{F}_{\mathcal{S}}$ that is visited by $W(\mathcal{S})$ infinite times. So, $W(S)$ can be written as $W(S)=W_0,W_1,W_2\ldots$ where each W_i, for $i\geq 1$, is a closed walk that ends in η. So, by Corollary 7 and using Equation (2.13) we have

$$\begin{aligned}\operatorname{cost}(\mathcal{S},\pi) &= \lim_{i\to\infty}\frac{z(W_0)+\ldots+z(W_i)}{|W_0|+\ldots+|W_i|}\geq\lim_{i\to\infty}\frac{z(W_1)+\ldots+z(W_i)}{|W_1|+\ldots+|W_i|}\\ &\geq \lim_{i\to\infty}\min\left\{\frac{z(W_1)}{|W_1|},\ldots,\frac{z(W_i)}{|W_i|}\right\}\geq\lim_{i\to\infty}\frac{z(C^*)}{|C^*|}=\frac{z(C^*)}{|C^*|},\end{aligned}$$

which completes the proof. □

We are finally ready to give the proof for the main result of this section.

Proof of Theorem 5. By Theorem 6, for every schedule $\mathcal{S}$ we have $\operatorname{cost}(\mathcal{S},\pi)\geq\frac{z(c^*)}{|C^*|}$. To complete the proof, we show that there exists a cyclic schedule whose cost is $\frac{z(c^*)}{|C^*|}$.

As mentioned in Remark 1, each edge $e=\eta_1\eta_2$ in $\mathcal{G}_{\pi,c}$ corresponds to a set u_e^c of at most c nodes, such that by probing the nodes in u_e^c the configuration state changes from η_1 to η_2. Now, assume $C^*=\eta_1,\ldots,\eta_\ell$, and $u_1^c,\ldots,u_\ell^c$ are the corresponding sets of nodes to the edges of C^*. It is easy to see that from any state we can reach η_ℓ, after exactly ℓ time steps, by probing the nodes by following $u_1^c,\ldots,u_\ell^c$. Let $\mathcal{O}$ be a schedule that periodically probes the nodes by following $u_1^c,\ldots,u_\ell^c$. Obviously $\mathcal{O}$ is a cyclic schedule, and we can write $W(\mathcal{O})=W_0,C^*,C^*,\ldots$ for a walk W_0. Again, by Equation (2.13) we have

$$\operatorname{cost}(\mathcal{S},\pi)=\lim_{i\to\infty}\frac{z(W_0)+c\cdot z(C^*)}{|W_0|+i\cdot|C^*|}=\frac{z(C^*)}{|C^*|},$$

and the proof is complete. □

Note that we proved, by finding the cycle C^* one can obtain a cyclic optimal schedule. However, note that finding C^* can be an intractable task, as the size of the $\mathcal{F}'$, and the number its cycles can be exponentially large.

2.2.6.2 On (≈ 2)-Approximation Cyclic Schedules

In the previous section, we showed that there is always an ℓ-cyclic optimal algorithm, for some ℓ. However, we did not specify the length of the cycle, ℓ. Here, we answer a related important question: given a fixed ℓ, how well an ℓ-cyclic schedule can approximate the optimal one? To answer this question, we prove the following theorem in this section:

Theorem 7. *For a given positive integer ℓ, there exists an ℓ-cyclic $\left(\frac{2}{1-g(\ell)}\right)$-approximation schedule, where g is a decreasing function, and $\lim_{\ell\to\infty} g(\ell) = 0$.*

We use a probabilistic method to prove this theorem. But first, we need the following lemmas.

Lemma 8. *Let ℓ, f be two positive integers, where $f \leq \ell$, and let F be a random variable that takes values in $\{f, \ldots, \ell\}$. We have*

$$\mathbb{E}\left[\min\left\{X_1, \ldots, X_F\right\}\right] \leq \frac{\ell+1}{f+1},$$

if $\{X_1, \ldots, X_F\} \subseteq \{1, 2, \ldots, \ell\}$ is a subset of size F chosen uniformly at random.

Proof. Denote $X_{\min} = \{X_1, \ldots, X_F\}$. For $i \in \{1, \ldots, \ell - F + 1\}$ we have

$$\Pr\left[X_{\min} \geq i \mid F\right] = \Pr\left[X_1 \geq i, \ldots, X_F \geq i\right] = \frac{\binom{\ell-i+1}{F}}{\binom{\ell}{F}},$$

and thus

$$\begin{aligned}
\mathbb{E}\left[X_{\min} \mid F\right] &= \sum_{i=1}^{\ell-F+1} \Pr\left[X_{\min} \geq i \mid F\right] = \frac{1}{\binom{\ell}{F}} \sum_{i=1}^{\ell-F+1} \binom{\ell-i+1}{F} \\
&= \frac{1}{\binom{\ell}{F}} \left(\binom{F}{F} + \binom{F+1}{F} + \ldots \binom{\ell}{F}\right) \\
&= \frac{1}{\binom{\ell}{F}} \left(\binom{F+1}{F+1} + \binom{F+1}{F} + \ldots \binom{\ell}{F}\right) \\
&= \frac{\binom{\ell+1}{F+1}}{\binom{\ell}{F}} = \frac{\ell+1}{F+1}.
\end{aligned}$$

For the first equality we used the equality $\mathbb{E}[X] = \sum_{i\geq 1} \Pr[X \geq i]$ for random variable X with positive integer values. We also used the fact that $\binom{n}{r}+\binom{n}{r+1} = \binom{n+1}{r+1}$. Finally

$$\begin{aligned}\mathbb{E}[X_{\min}] &= \sum_{F=f}^{\ell} \Pr[F]\,\mathbb{E}[X_{\min} \mid F] = \sum_{F}^{\ell} \Pr[F=f]\,\frac{\ell+1}{F+1} \\ &\leq \sum_{F=f}^{\ell} \Pr[F]\,\frac{\ell+1}{f+1} = \frac{\ell+1}{f+1},\end{aligned}$$

which completes the proof. □

Lemma 9. *Suppose $\ell, f_1, \ldots, f_n$ are positive integers such that $\sum_{i=1}^{n} f_i \leq \ell$. There exists an ℓ-cyclic 1-schedule whose cost is at most $(\ell+1)\sum_{i=1}^{n} \frac{\pi_i}{f_i+1}$.*

Proof. Define $\mathcal{U} = \mathcal{U}(\ell, \{f_1, \ldots, f_n\})$ is the space of all sequences $U \in \{1, \ldots, n\}^{\ell}$ of length ℓ such $i \in \{1, \ldots, n\}$ appears in U at least f_i times. By $U \sim \mathcal{U}$ we mean U is a sequence chosen uniformly at random from $\mathcal{U}$. For a $U \in \mathcal{U}$ let $\mathcal{S}_U$ be an ℓ-cyclic 1-schedule that probes the nodes periodically following the sequence U.

Claim. *The expected waiting time an item at node i following the schedule $\mathcal{S}_U$, denoted by $\omega_i^{\mathcal{S}_U}$, is at most $\frac{\ell+1}{f_i+1}$.*

To show this, suppose an item is generated at time t and at node i. Let T be the time interval $[t+1, t+2, \ldots, t+\ell]$. Since $\mathcal{S}_U$ is an ℓ-cyclic schedule that repeats the sequence U, the number of times $\mathcal{S}_U$ probes the node i during T is a least f_i, and if $U \sim \mathcal{U}$, these probes of i during the interval T are chosen uniformly at random. Therefore, by Lemma 8, we have

$$\omega_i^{\mathcal{S}_U} = \mathbb{E}[\min\{X_1, \ldots, X_{F_i}\}] \leq \frac{\ell+1}{f_i+1},$$

where F_i is a random variable that indicates the frequency of i in U, and thus, $f_i \leq F_i \leq \ell$. Also, $\{X_1, \ldots, X_{F_i}\}$ is a subset of $\{1, \ldots, \ell\}$ of size F_i and chosen uniformly at random, that indicates the probes of the node i during T. Therefore,

$$\mathbb{E}_{U\sim\mathcal{U}}(\mathrm{cost}(\mathcal{S}_U, \pi)) \leq \sum_{i=1}^{n} \frac{\pi_i(\ell+1)}{f_i+1}.$$

Therefore, there exists a sequence U_0, and thus a schedule S_{U_0}, such that $\mathrm{cost}(\mathcal{S}_{U_0}) \leq \mathbb{E}_{U\sim\mathcal{U}}(\mathrm{cost}(\mathcal{S}_U, \pi)) \leq \sum_{i=1}^{n} \frac{\pi_i(\ell+1)}{f_i+1}$ □

Using a similar argument we have the following extension to c-schedules.

Lemma 10. *Suppose $\ell, f_1, \ldots, f_n$ are positive integers such that $\sum_{i=1}^{n} f_i \leq c \cdot \ell$. There exists an ℓ-cyclic c-schedule whose cost is at most $(\ell+1)\sum_{i=1}^{n} \frac{\pi_i}{f_i+1}$.*

Proof. Define $\mathcal{U}_c = \mathcal{U}(\ell, \{f_1, \ldots, f_n\}, c)$ is the space of all sequences $U = u_1^c, \ldots, u_\ell^c$ such that each u_j^c is c-subset[3] of $\{1, \ldots, n\}$ and each i appears at least f_i times in *multiset* $\cup_{j=1}^{\ell} u_j^c$, i.e., at least f_i of u_j^c's contain i. Also, define $\mathcal{S}_U$ be the ℓ-cyclic c-schedule that at anytime t probes the c nodes in u_j^c where $j = t \mod \ell$.

Now, similar to the proof of Lemma 9, the expected waiting time of any item at any node i is at most $\frac{\ell+1}{f_i+1}$, and thus, there exists an ℓ-cyclic c-schedule whose cost is at most $(\ell+1)\sum_{i=1}^{n} \frac{\pi_i}{f_i+1}$. □

Proof of Theorem 7. Let $p_i^* = \frac{\sqrt{\pi_i}}{\sum_{j=1}^{n}\sqrt{\pi_j}}$, and $f_i = \lfloor c \cdot \ell \cdot p_i^* \rfloor$. Therefore,

$$\frac{f_i}{c \cdot \ell \cdot p_i^*} \geq \frac{c \cdot \ell \cdot p_i^* - 1}{c \cdot \ell \cdot p_i^*} = 1 - \frac{1}{c \cdot \ell \cdot p_i^*}.$$

Now, if $p_{\min}^* = \min\{p_1^*, \ldots, p_n^*\}$ define $g(\ell) = \frac{1}{c \cdot \ell \cdot p_{\min}^*}$. Therefore, $f_i \geq (1 - g(\ell)) c \cdot \ell \cdot p_i^*$. Also g is a decreasing function and converges to 0. Also, note that

$$f_1 + \ldots + f_n \leq \sum_{i=1}^{n} c \cdot \ell \cdot p_i^* = c \cdot \ell.$$

Therefore, by Lemma 10, there exists an ℓ-cyclic c-schedule $\mathcal{S}$ such that

$$\begin{aligned}
\operatorname{cost}(\mathcal{S}, \pi) &\leq \sum_{i=1}^{n} \frac{(\ell+1)\pi_i}{f_i+1} < \sum_{i=1}^{n} \frac{\ell \pi_i}{f_i} \leq \sum_{i=1}^{n} \frac{\ell \pi_i}{(1-g(\ell)) c \cdot \ell \cdot p_i^*} \\
&= \frac{1}{c(1-g(\ell))} \sum_{i=1}^{n} \frac{\pi_i}{p_i^*} = \frac{1}{c(1-g(\ell))} \left(\sum_{i=1}^{n} \sqrt{\pi_i} \right)^2 \leq \frac{2}{1-g(\ell)} \operatorname{cost}(\mathcal{O}, \pi),
\end{aligned}$$

where we used the lower bound for the cost of optimal c-schedule $\mathcal{O}$ in Theorem 1. □

2.2.7 On (≈ 2)-Approximation Random Schedule

In the previous section we talked about existence of deterministic schedules, whether being optimal, or approximation. In this section, we **compute** a random c-schedule, $\mathcal{R}_+$, that is $\left(\frac{2}{1-g(\ell)} + o(1)\right)$-approximation (the same function g defined in the previous

[3] An r-subset is a subset of size r.

section), at the cost of using more space. Note that this improves the approximation ratio $\left(2+\frac{c-1}{c}\right)$ in Corollary 4, for large enough ℓ, memory space, and $c \geq 2$.

Suppose $\mathcal{U}_c = \mathcal{U}(\ell, \{f_1, \ldots, f_n\}, c)$ is the space of all random sequences U as defined in the proof of Lemma 10, and by $U \sim \mathcal{U}_c$ we mean U is sampled from $\mathcal{U}$ uniformly at random. The c-schedule $\mathcal{R}_+$ is given in Algorithm 2. The idea is to partition the timeline into time intervals $T_1, T_2, \ldots$, where the length of all of them is exactly $h\ell$, for a positive integer h. Next, for each T_i, we sample $U_i \sim \mathcal{U}_c$ and let $U = U_i U_i \ldots U_i$ be the h times concatenation of U_i with itself. Then, during the time interval T_i, we probe the nodes by following the sequence U.

Algorithm 2: Computing the schedule $\mathcal{R}_+$

Inputs: Positive integers ℓ, h, c, and generating vector π.
Outputs: $\mathcal{R}_+(t)$, for $t = 1, 2, \ldots$.
begin
  $p^*_{\min} \leftarrow \min_{1 \leq i \leq n} \left\{ \frac{\sqrt{\pi_i}}{\sum_{j=1}^{n} \sqrt{\pi_j}} \right\}$;
  for $i \in \{1, \ldots, n\}$ **do**
    $f_i \leftarrow \lfloor c \cdot \ell \cdot p^*_{\min} \rfloor$;
  end
  $r \leftarrow 0$;
  for $t = 1, 2, 3, \ldots$ **do**
    if $r = 0$ **then**
      $U_i \sim \mathcal{U}(\ell, \{f_1, \ldots, f_n\}, c)$;
      $U \leftarrow U_i$;
      for $j = 1, \ldots, h-1$ **do**
        $U \leftarrow U U_i$;
      end
    end
    $r \leftarrow r + 1 \mod h\ell$;
    $\mathcal{R}_+(t) \leftarrow r$-th element of U;
  end
end

We have the following theorem:

Theorem 8. *The cost of $\mathcal{R}_+$ (given in Algorith 2) is a most $\left(\frac{2}{1-g(\ell)}+\frac{\ell}{h}\right)$ times the optimal cost.*

Proof. Recall that $\omega_i^{\mathcal{R}_+}$ is the expected waiting of an item generated at node i. Let $\omega_i^{\mathcal{R}_+}(t)$ be the expected waiting time an item generated at node i and at time t. Therefore,

$$\omega_i^{\mathcal{R}_+} = \lim_{t\to\infty} \frac{1}{t} \sum_{t'=1}^{t} \omega_i^{\mathcal{R}_+}(t). \tag{2.14}$$

Each of these time intervals can be written as $T_j = A_j B_j$ where B_j is the last ℓ time steps in T_j, i.e., the lengths of A_j and B_j are $(h-1)\ell$ and ℓ, respectively.

Note that by Lemma 10, if $t \in A_j$, for some j, we have

$$\omega_i^{\mathcal{R}_+}(t) \le \frac{\ell+1}{f_i+1} \le \frac{\ell}{f_i}.$$

Also, note that if $t \in B_j$, for some j, then $t+\ell \in A_{j+1}$ and

$$\omega_i^{\mathcal{R}_+}(t) \le \ell + \omega_i^{\mathcal{R}_+}(t+\ell) \le \ell + \frac{\ell}{f_i}.$$

Therefore, by Equation (2.14) we get

$$\begin{aligned}
\omega_i^{\mathcal{R}_+} &= \lim_{j\to\infty} \frac{1}{|T_1|+\ldots+|T_j|} \sum_{r=1}^{j} \sum_{t\in T_r} \omega_i^{\mathcal{R}_+}(t) \\
&= \lim_{j\to\infty} \frac{1}{jh\ell} \sum_{r=1}^{j} \left(\sum_{t\in A_r} \omega_i^{\mathcal{R}_+}(t) + \sum_{t\in B_r} \omega_i^{\mathcal{R}_+}(t) \right) \\
&\le \lim_{j\to\infty} \frac{1}{jh\ell} \sum_{r=1}^{j} \left(|A_r|\frac{\ell}{f_i} + |B_r| \left(\frac{\ell}{f_i} + \ell \right) \right) \\
&= \lim_{j\to\infty} \frac{1}{jh\ell} \sum_{r=1}^{j} \left(|T_r|\frac{\ell}{f_i} + \ell^2 \right) = \lim_{j\to\infty} \frac{1}{jh\ell} \left(\frac{jh\ell\cdot\ell}{f_i} + j\ell^2 \right) \\
&= \frac{\ell}{f_i} + \frac{\ell}{h}.
\end{aligned}$$

Now, similar to the proof of Theorem 7, we have

$$\begin{aligned}
\operatorname{cost}(\mathcal{S},\pi) &\le \sum_{i=1}^{n}\pi_i\left(\frac{\ell}{f_i}+\frac{\ell}{h}\right) \le \sum_{i=1}^{n}\frac{\ell\pi_i}{(1-g(\ell))c\cdot\ell\cdot p_i^*}+\frac{\ell}{h}\left(\sum_{i=1}^{n}\pi_i\right)\\
&= \frac{1}{c(1-g(\ell))}\sum_{i=1}^{n}\frac{\pi_i}{p_i^*}+\frac{\ell}{h}\left(\sum_{i=1}^{n}\pi_i\right)\\
&= \frac{1}{c(1-g(\ell))}\left(\sum_{i=1}^{n}\sqrt{\pi_i}\right)^2+\frac{\ell}{h}\left(\sum_{i=1}^{n}\pi_i\right)\\
&\le \left(\frac{2}{1-g(\ell)}+\frac{\ell}{h}\right)\operatorname{cost}(\mathcal{O},\pi),
\end{aligned}$$

for an optimal schedule $\mathcal{O}$, and since $\lim_{h\to\infty}\frac{\ell}{h}=0$, for large enough h compared to ℓ, $\frac{\ell}{h}=o(1)$ and the proof is complete. □

Extension to Adaptive Algorithm. The assumption in Algorithm 2 is that the generating vector π is given. However, similar to Algorithm 1, we can compute an adaptive schedule $\mathcal{A}_+$, that starts with an estimate of the generating vector $\tilde{\pi}$ (as in Algorithm 1), that updates the estimates of each π_i after each probe, but update the p_i^*'s after each time interval T_i.

Chapter 3

Optimal Probing Schedule Problem: *Propagating Items*

In this chapter, we formalize a generic process that describes the creation and distribution of information in a network, and define the computational task of learning this process by probing the nodes in the network according to a schedule. The process and task are parametrized by the resource limitations of the observer and the decay rate of the novelty of items. We introduce a *cost* measure to compare different schedules: the cost of a schedule is the limit of the average expected novelty of uncaught items at each time step. On the basis of these concepts, we formally define the *Optimal Probing Schedule Problem*, which requires finding the schedule with minimum cost.

We conduct a theoretical study of the cost of a schedule, showing that it can *be computed explicitly* and that it is a *convex function* over the space of schedules. We then introduce WIGGINS,[1] an algorithm to compute the optimal schedule by solving a constrained convex optimization problem through the use of an iterative method based on Lagrange multipliers.

We discuss variants of WIGGINS for the realistic situation where the parameters of the process needs to be learned or can change over time. We show how to compute a schedule which is (probabilistically) guaranteed to have a cost very close to the optimal by only observing the generating process for a limited amount of time. We

[1] In the Sherlock Holmes novel *A study in scarlet* by A. Conan Doyle, Wiggins is the leader of the "Baker Street Irregulars", a band of street urchins employed by Holmes as intelligence agents.

also present a MapReduce adaptation of WIGGINS to handle very large networks.

Finally, we conduct an extensive experimental evaluation of WIGGINS and its variants, comparing the performances of the schedules it computes with natural baselines, and showing how it performs extremely well in practice on real social networks when using well-established models for generating new items (e.g., the independence cascade model [68]).

3.1 Problem Definition

In this section we formally introduce the problem and define our goal.

Let $G = (V, E)$ be a graph with $|V| = n$ nodes. W.l.o.g. we let $V = [n]$. Let $\mathcal{F} \subseteq 2^V$ be a collection of subsets of V, i.e., a collection of sets of nodes. Let π be a function from $\mathcal{F}$ to $[0, 1]$ (not necessarily a probability distribution). We model the generation and diffusion of information in the network by defining a *generating process* $\Gamma = (\mathcal{F}, \pi)$. Γ is a infinite discrete-time process which, at each time step t, generates a collection of sets $\mathcal{I}_t \subseteq \mathcal{F}$ such that each set $S \in \mathcal{F}$ is included in $\mathcal{I}_t$ with probability $\pi(S)$, independent of t and of other sets generated at time $t' \leq t$. For any t and any $S \in \mathcal{I}_t$, the ordered pair (t, S) represents an *item* - a piece of information that was generated at time t and reached *instantaneously* the nodes in S. We choose to model the diffusion process as instantaneous because this abstraction accurately models the view of an outside *resource-limited observer* that does not have the resources to monitor simultaneously all the nodes in the network at the fine time granularity needed to observe the different stages of the diffusion process.

Probing and schedule. The observer can only monitor the network by *probing* nodes. Formally, by *probing a node $v \in V$ at time t*, we mean obtaining the set $I(t, v)$ of items (t', S) such that $t' \leq t$ and $v \in S$:[2]

$$I(t, v) \coloneqq \{(t', S) \ : \ t' \leq t, S \in \mathcal{I}_{t'}, v \in S\} \ .$$

Let U_t be the union of the sets $\mathcal{I}_{t'}$ generated by Γ at any time $t' \leq t$, and so $I(t, v) \subseteq U_t$.

We model the *resource limitedness of the observer* through a constant, user-specified, parameter $c \in \mathbb{N}$, representing the maximum number of nodes that can

[2]The set S appears in the notation for an item only for clarity of presentation: we are *not* assuming that when we probe a node and find an item (t, S) we obtain information about S.

be probed at any time, where probing a node v returns the value $I(t, v)$.

The observer chooses the c nodes to probe by following a schedule. In this work we focus on *memoryless* schedules, i.e., the choice of nodes to probe at time t is independent from the choice of nodes probed at any time $t' < t$. More precisely, a *probing c-schedule* p is a probability distribution on V. At each time t, the observer chooses a set P_t of c nodes to probe, such that P_t is obtained through *random sampling of V without replacement according to* p, independently from $P_{t'}$ from $t' < t$. Memoryless schedules are simple, easy to store, and fast to implement.

Caught items, uncaught items, and novelty. We say that an item (t', S) is *caught at time $t \geq t'$* iff

1. a node $v \in S$ is probed at time t; and
2. no node in S was probed in the interval $[t', t-1]$.

Let C_t be the set of items caught by the observer at any time $t' \leq t$. We have $C_t \subseteq U_t$. Let $N_t = U_t \setminus C_t$ be the *set of uncaught items at time t*, i.e., items that were generated at any time $t' \leq t$ and have not been caught yet at time t. For any item $(t', S) \in N_t$, we define the *θ-novelty of (t', S) at time t* as

$$\mathsf{f}_\theta(t, t', S) \coloneqq \theta^{t-t'},$$

where $\theta \in (0, 1)$ is a user-specified parameter modeling how fast the value of an item decreases with time if uncaught. Intuitively, pieces of information (e.g., rumors) have high value if caught almost as soon as they have appeared in the network, but their value decreases quickly (i.e., exponentially) as more time passes before being caught, to the point of having no value in the limit.

Load of the system and cost of a schedule. The set N_t of uncaught items at time t imposes a *θ-load, $L_\theta(t)$, on the graph at time t*, defined as the sum of the θ-novelty at time t of the items in N_t:

$$L_\theta(t) \coloneqq \sum_{(t', S) \in N_t} \mathsf{f}_\theta(t, t', S) \ .$$

The quantity $L_\theta(t)$ is a random variable, depending both on Γ and on the probing schedule p, and as such it has an expectation $\mathbb{E}[L_\theta(t)]$ w.r.t. all the randomness in the

system. The θ-*cost of a schedule* p is defined as the limit, for $t \to \infty$, of the average expected load of the system:

$$\begin{aligned}\mathsf{cost}_\theta(\mathsf{p}) &\coloneqq \lim_{t\to\infty} \frac{1}{t} \sum_{t' \le t} \mathbb{E}[L_\theta(t)] \\ &= \lim_{t\to\infty} \frac{1}{t} \sum_{t' \le t} \mathbb{E}\left[\sum_{(t'',S)\in N_{t'}} \mathsf{f}_\theta(t', t'', S) \right] \ .\end{aligned}$$

Intuitively, the load at each time indicates the amount of novelty we did not catch at that time, and the cost function measures the average of such loss over time. The limit above always exists (Lemma 11).

We now have all the necessary ingredients to formally define the problem of interest in this work.

Problem definition. Let $G = (V, E)$ be a graph and $\Gamma = (\mathcal{F}, \pi)$ be a generating process on G. Let $c \in \mathbb{N}$ and $\theta \in (0, 1)$. The (θ, c)-*Optimal Probing Schedule Problem* ((θ, c)-OPSP) requires finding the *optimal c-schedule* p^*, i.e., the schedule with *minimum* θ-cost over the set S_c of c-schedules:

$$\mathsf{p}^* = \arg\min_{\mathsf{p}} \{\mathsf{cost}_\theta(\mathsf{p}), \mathsf{p} \in \mathsf{S}_c\} \ .$$

Thus, the goal is to design a c-schedule that discovers the maximum number of items weighted by their novelty value (which correspond to those generated most recently). The parameter θ controls how fast the novelty of an item decays, and influences the choices of a schedule. When θ is closed to 0, items are relevant only for a few steps and the schedule must focus on the most recently generated items, catching them as soon as they are generated (or at most shortly thereafter). At the other extreme ($\theta \approx 1$), an optimal schedule must maximizes the total number of discovered items, as their novelty decays very slowly.

Viewing the items as "information" disseminated in the network, an ideal schedule assigns higher probing probability to nodes that act as *information hubs*, i.e., nodes that receive a large number of items. Thus, an optimal schedule p^*, identifies information hubs among the nodes. This task (finding information hubs) can be seen as the complement of the *influence maximization problem* [68, 69]. In the influence maximization problem we look for a set of nodes that *generate* information that reach most nodes. In the information hubs problem, we are interested in a set of nodes that *receive* the most of information, thus the most informative nodes for an observer.

In the following sections, we may drop the specification of the parameters from θ-novelty, θ-cost, θ-load, and c-schedule, and from their respective notation, as the parameters will be clear from the context.

3.2 The WIGGINS Algorithm

In this section we present the algorithm WIGGINS (and its variants) for solving the Optimal Probing Schedule Problem (θ, c)-OPSP for generating process $\Gamma = (\mathcal{F}, \pi)$ on a graph $G = (V, E)$.

We start by assuming that we have complete knowledge of Γ, i.e., we know $\mathcal{F}$ and π. This strong assumption allows us to study the theoretical properties of the cost function and motivates the design of our algorithm, WIGGINS, to compute an optimal schedule. We then remove the assumption and show how we can extend WIGGINS to only use a collection of observations from Γ. Then we discuss how to recalibrate our algorithms when the parameters of the process (e.g., π or $\mathcal{F}$) change over time. Finally, we show an algorithm for the MapReduce framework that allows us to scale to large networks.

3.2.1 Computing the Optimal Schedule

We first conduct a theoretical analysis of the cost function cost_θ, and then use the results to develop WIGGINS, our algorithm to compute the optimal c-schedule (i.e., solve the (θ, c)-OPSP).

3.2.1.1 Analysis of the cost function

Assume for now that we know Γ, i.e., we have complete knowledge of $\mathcal{F}$ and π. Under this assumption, we can exactly compute the θ-cost of a c-schedule.

Lemma 11. *Let* $\mathsf{p} = (\mathsf{p}_1, \ldots, \mathsf{p}_n)$ *be a c-schedule. Then*

$$\mathsf{cost}_\theta(\mathsf{p}) \coloneqq \lim_{t\to\infty} \frac{1}{t} \sum_{t'=0}^{t} \mathbb{E}[L_\theta(t')] = \sum_{S\in\mathcal{F}} \frac{\pi(S)}{1-\theta(1-\mathsf{p}(S))^c}, \tag{3.1}$$

where $\mathsf{p}(S) = \sum_{v\in S} \mathsf{p}_v$.

Proof. Let t be a time step, and consider the quantity $\mathbb{E}[L_\theta(t)]$. By definition we have

$$\mathbb{E}[L_\theta(t)] = \mathbb{E}\left[\sum_{(t',S)\in N_t} \mathsf{f}_\theta(t,t',S)\right] = \mathbb{E}\left[\sum_{(t',S)\in N_t} \theta^{t-t'}\right],$$

where N_t is the set of uncaught items at time t. Let now, for any $t' \leq t$, $N_{t,t'} \subseteq N_t$ be the set of uncaught items in the form (t', S). Then we can write

$$\mathbb{E}[L_\theta(t)] = \mathbb{E}\left[\sum_{t'=0}^{t} \sum_{(t',S)\in N_{t,t'}} \theta^{t-t'}\right] \quad .$$

Define now, for each $S \in \mathcal{F}$, the random variable $X_{S,t',t}$ which takes the value $\theta^{t-t'}$ if $(t', S) \in N_{t,t'}$, and 0 otherwise. Using the linearity of expectation, we can write:

$$\begin{aligned}\mathbb{E}[L_\theta(t)] &= \sum_{S\in\mathcal{F}} \sum_{t'=0}^{t} \mathbb{E}[X_S, t', t] \\ &= \sum_{S\in\mathcal{F}} \sum_{t'=0}^{t} \theta^{t-t'} \Pr(X_{S,t,t'} = \theta^{t-t'}) \quad . \end{aligned} \tag{3.2}$$

The r.v. $X_{S,t,t'}$ takes value $\theta^{t-t'}$ if and only if the following two events E_1 and E_2 both take place:

- E_1: the set $S \in F$ belongs to $\mathcal{I}_{t'}$, i.e., is generated by Γ at time t';
- E_2: the item (t', S) is uncaught at time t. This is equivalent to say that no node $v \in S$ was probed in the time interval $[t', t]$.

We have $\Pr(E_1) = \pi(S)$, and

$$\Pr(E_2) = (1 - \mathsf{p}(S))^{c(t-t')} \quad .$$

The events E_1 and E_2 are independent, as the process of probing the nodes is independent from the process of generating items, therefore, we have

$$\Pr(X_{S,t,t'} = \theta^{t-t'}) = \Pr(E_1)\Pr(E_2) = \pi(S)(1 - \mathsf{p}(S))^{c(t-t')} \quad .$$

We can plug this quantity in the rightmost term of (3.2) and write

$$\begin{aligned}\lim_{t\to\infty} \mathbb{E}[L_\theta(t)] &= \lim_{t\to\infty} \sum_{S\in\mathcal{F}} \sum_{t'=0}^{t} \theta^{t-t'} \pi(S)(1 - \mathsf{p}(S))^{c(t-t')} \\ &= \lim_{t\to\infty} \sum_{S\in\mathcal{F}} \pi(S) \sum_{t'=0}^{t} (\theta(1 - \mathsf{p}(S))^c)^t \\ &= \sum_{S\in\mathcal{F}} \frac{\pi(S)}{1 - \theta(1 - \mathsf{p}(S))^c}, \end{aligned} \tag{3.3}$$

where we used the fact that $\theta(1-\mathsf{p}(S))^c < 1$. We just showed that the sequence $(\mathbb{E}[L_\theta(t)])_{t\in\mathbb{N}}$ converges as $t\to\infty$. Therefore, its Cesàro mean, i.e., $\lim_{t\to\infty}\frac{1}{t}\sum_{t'=0}^{t}\mathbb{E}[L_\theta(t)]$, equals to its limit [55, Sect. 5.4] and we have

$$\begin{aligned}\mathsf{cost}_\theta(\mathsf{p}) &= \lim_{t\to\infty}\frac{1}{t}\sum_{t'=0}^{t}\mathbb{E}[L_\theta(t)] = \lim_{t\to\infty}\mathbb{E}[L_\theta(t)]\\ &= \sum_{S\in\mathcal{F}}\frac{\pi(S)}{1-\theta(1-\mathsf{p}(S))^c}\ .\end{aligned}$$

□

We now show that $\mathsf{cost}_\theta(\mathsf{p})$, as expressed by the r.h.s. of (3.1) is a convex function over its domain S_c, the set of all possible c-schedules. We then use this result to show how to compute an optimal schedule.

Theorem 9. *The cost function* $\mathsf{cost}_\theta(\mathsf{p})$ *is a convex function over* S_c.

Proof. For any $S\in\mathcal{F}$, let

$$f_S(\mathsf{p}) = \frac{1}{1-\theta(1-\mathsf{p}(S))^c}\ .$$

The function $\mathsf{cost}_\theta(\mathsf{p})$ is a linear combination of $f_S(\mathsf{p})$'s with positive coefficients. Hence to show that $\mathsf{cost}_\theta(\mathsf{p})$ is convex it is sufficient to show that, for any $S\in\mathcal{F}$, $f_S(\mathsf{p})$ is convex.

We start by showing that $g_S(\mathsf{p}) = \theta(1-\mathsf{p}(S))^c$ is convex. This is due to the fact that its Hessian matrix is positive semidefinite [14]:

$$\frac{\partial}{\partial\mathsf{p}_i\partial\mathsf{p}_j}g_S(\mathsf{p}) = \begin{cases}\theta c(c-1)(1-\mathsf{p}(S))^{c-2} & i,j\in S\\ 0 & \text{otherwise}\end{cases}$$

Let $\mathbf{v}_S$ be a $n\times 1$ vector in $\mathbb{R}^n$ such that its i-th coordinate is $[c(c-1)(1-\mathsf{p}(S))^{c-2}]^{1/2}$ if $i\in S$, and 0 otherwise. We can write the Hessian matrix of g_S as

$$\nabla^2 g_S = V_S * V_S^{\mathsf{T}},$$

and thus, $\nabla^2 g_S$ is positive semidefinite matrix and g is convex. From here, we have that $1-g_S$ is a *concave* function. Since $f_S(\mathsf{p}) = \frac{1}{1-g_S(\mathsf{p})}$ and the function $h(x)=\frac{1}{x}$ is convex and non-increasing, then f_S is a convex function. □

If for every $v \in V$, $S = \{v\}$ belongs to $\mathcal{F}$, then the function g_S in the above proof is *strictly* convex, and so is f_S.

We then have the following corollary of Thm. 9.

Corollary 8. *Any schedule* p *with locally minimum cost is an optimal schedule (i.e., it has global minimum cost). Furthermore, if for every* $v \in V$, $\{v\}$ *belongs to* $\mathcal{F}$, *the optimal schedule is* unique.

3.2.1.2 The Algorithm

Corollary 8 implies that one can compute an optimal $c-$schedule p^* (i.e., solve the (θ, c)-OPSP) by solving the unconstrained minimization of cost_θ over the set S_c of all c-schedules, or equivalently by solving the following constrained minimization problem on $\mathbb{R}^n$:

$$\boxed{\begin{array}{lrcll} \min\limits_{\mathsf{p}\in\mathbb{R}^n} & \mathsf{cost}_\theta(\mathsf{p}) & & & \\ & \sum\limits_{i=1}^{n} \mathsf{p}_i & = & 1 & \\ & \mathsf{p}_i & \geq & 0 & \forall i \in \{1, \ldots, n\} \end{array}} \tag{3.4}$$

Since the function cost_θ is convex and the constraints are linear, the optimal solution can, theoretically, be found efficiently [14]. In practice though, available convex optimization problem solvers can not scale well with the number n of variables, especially when n is in the millions as is the case for modern graphs like online social networks or the Web. Hence we developed WIGGINS, an iterative method based on *Lagrange multipliers* [14, Sect. 5.1], which can scale efficiently and can be adapted to the MapReduce framework of computation [33], as we show in Sect. 3.2.4. While we can not prove that this iterative method always converges, we can prove (Thm. 10) that (i) if at any iteration the algorithm examines an optimal schedule, then it will reach convergence at the next iteration, and (ii) if it converges to a schedule, that schedule is optimal. In Sect. 4.3 we show our experimental results illustrating the convergence of WIGGINS in different cases.

WIGGINS takes as inputs the collection $\mathcal{F}$, the function π, and the parameters c and θ, and outputs a schedule p which, if convergence (defined in the following) has been reached, is the optimal schedule. It starts from a uniform schedule $\mathsf{p}^{(0)}$, i.e., $\mathsf{p}_i^{(0)} = 1/n$ for all $1 \leq i \leq n$, and iteratively refines it until convergence (or until

a user-specified maximum number of iterations have been performed). At iteration $j \geq 1$, we compute, for each value i, $1 \leq i \leq n$, the function

$$W_i(\mathsf{p}^{(j-1)}) \coloneqq \sum_{\substack{S \in \mathcal{F} \\ \text{s.t. } i \in S}} \frac{\theta c \pi(S)(1 - \mathsf{p}^{(j-1)}(S))^{c-1}}{(1 - \theta(1 - \mathsf{p}^{(j-1)}(S))^c)^2} \tag{3.5}$$

and then set

$$\mathsf{p}_i^{(j)} = \frac{\mathsf{p}_i^{(j-1)} W_i(\mathsf{p}^{(j-1)})}{\sum_{z=1}^{n} \mathsf{p}_z^{(j-1)} W_z(\mathsf{p}^{(j-1)})} \ .$$

The algorithm then checks whether $\mathsf{p}^{(j)} = \mathsf{p}^{(j-1)}$. If so, then we reached convergence and we can return $\mathsf{p}^{(j)}$ in output, otherwise we perform iteration $j+1$. The pseudocode for WIGGINS is in Algorithm 3. The following theorem shows the correctness of the algorithm in case of convergence.

Theorem 10. *We have that:*

1. *if at any iteration j the schedule $\mathsf{p}^{(j)}$ is optimal, then* WIGGINS *reaches convergence at iteration $j+1$; and*
2. *if* WIGGINS *reaches convergence, then the returned schedule p is optimal.*

Proof. From the method of the Lagrange multipliers [14, Sect. 5.1], we have that, if a schedule p is optimal, then there exists a value $\lambda \in \mathbb{R}$ such that p and λ form a solution to the following system of $n+1$ equations in $n+1$ unknowns:

$$\nabla[\mathsf{cost}_\theta(\mathsf{p}) + \lambda(\mathsf{p}_1 + \ldots + \mathsf{p}_n - 1)] = 0, \tag{3.6}$$

where the gradient on the l.h.s. is taken w.r.t. (the components of) p and to λ (i.e., has $n+1$ components).

For $1 \leq i \leq n$, the i-th equation induced by (3.6) is

$$\frac{\partial}{\partial \mathsf{p}_i} \mathsf{cost}_\theta(\mathsf{p}) + \lambda = 0,$$

or, equivalently,

$$\sum_{\substack{S \in \mathcal{F} \\ \text{s.t.} i \in S}} \frac{\theta c \pi(S)(1 - \mathsf{p}(S))^{c-1}}{(1 - \theta(1 - \mathsf{p}(S))^c)^2} = \lambda \ . \tag{3.7}$$

The term on the l.h.s. is exactly $W_i(\mathsf{p})$. The $(n+1)$-th equation of the system (3.6) (i.e., the one involving the partial derivative w.r.t. λ) is

$$\sum_{z=1}^{n} \mathsf{p}_z = 1 \ . \tag{3.8}$$

Consider now the first claim of the theorem, and assume that we are at iteration j such that j is the minimum iteration index for which the schedule $\mathsf{p}^{(j)}$ computed at the end of iteration j is optimal. Then, for any i, $1 \leq i \leq n$, we have

$$W_i(\mathsf{p}^{(j)}) = \lambda$$

because $\mathsf{p}^{(j)}$ is optimal and hence all identities in the form of (3.7) must be true. For the same reason, (3.8) must also hold for $\mathsf{p}^{(j)}$. Hence, for any $1 \leq i \leq n$, we can write the value $\mathsf{p}_i^{(j+1)}$ computed at the end of iteration $j+1$ as

$$\mathsf{p}_i^{(j+1)} = \frac{\mathsf{p}_i^{(j)} W_i(\mathsf{p}^{(j)})}{\sum_{z=1}^{n} \mathsf{p}_z^{(j)} W_z(\mathsf{p}^{(j)})} = \frac{\mathsf{p}_i^{(j)} \lambda}{1 \cdot \lambda} = \mathsf{p}_i^{(j)},$$

which means that we reached convergence and WIGGINS will return $\mathsf{p}^{(j+1)}$, which is optimal.

Consider the second claim of the theorem, and let j be the first iteration for which $\mathsf{p}^{(j)} = \mathsf{p}^{(j-1)}$. Then we have, for any $1 \leq i \leq n$,

$$\mathsf{p}_i^{(j)} = \frac{\mathsf{p}_i^{(j-1)} W_i(\mathsf{p}^{(j-1)})}{\sum_{z=1}^{n} \mathsf{p}_z^{(j-1)} W_z(\mathsf{p}^{(j-1)})} = \mathsf{p}_i^{(j-1)} \quad .$$

This implies

$$W_i(\mathsf{p}^{(j-1)}) = \sum_{z=1}^{n} \mathsf{p}_z^{(j-1)} W_z(\mathsf{p}^{(j-1)}) \tag{3.9}$$

and the r.h.s. does not depend on i, and so neither does $W_i(\mathsf{p}^{(j-1)})$. Hence we have $W_1(\mathsf{p}^{(j-1)}) = \cdots = W_n(\mathsf{p}^{(j-1)})$ and can rewrite (3.9) as

$$W_i(\mathsf{p}^{(j-1)}) = \sum_{z=1}^{n} \mathsf{p}_z^{(j-1)} W_i(\mathsf{p}^{(j-1)}),$$

which implies that the identity (3.8) holds for $\mathsf{p}^{(j-1)}$. Moreover, if we set

$$\lambda = W_1(\mathsf{p}^{(j-1)})$$

we have that all the identities in the form of (3.7) hold. Then, $\mathsf{p}^{(j-1)}$ and λ form a solution to the system (3.6), which implies that $\mathsf{p}^{(j-1)}$ is optimal and so must be $\mathsf{p}^{(j)}$, the returned schedule, as it is equal to $\mathsf{p}^{(j-1)}$ because WIGGINS reached convergence.

□

Algorithm 3: WIGGINS

input : $\mathcal{F}$, π, c, θ, and maximum number T of iterations
output: A c-schedule p (with globally minimum θ-cost, in case of convergence)

```
for i ← 1 to n do
    p_i ← 1/n
end
for j ← 1 to T do
    for i ← 1 to n do
        W_i ← 0
    end
    for S ∈ F do
        for i ∈ S do
            W_i ← W_i + θcπ(S)(1−p(S))^{c−1} / (1−θ(1−p(S))^c)^2
        end
    end
    p^old ← p
    for i ← 1 to n do
        p_i ← p_i^old · W_i / Σ_i p_i^old · W_i
    end
    if p^old = p then // test for convergence
        break
    end
end
return p
```

3.2.2 Approximation through Sampling

We now remove the assumption, not realistic in practice, of knowing the generating process Γ exactly through $\mathcal{F}$ and π. Instead, we observe the process using, for a limited time interval, a schedule that iterates over all nodes (or a schedule that selects each node with uniform probability), until we have observed, for each time step t in a limited time interval $[a, b]$, the set $\mathcal{I}_t$ generated by Γ, and therefore we have access to a collection

$$\mathcal{I} = \{\mathcal{I}_a, \mathcal{I}_{a+1}, \ldots, \mathcal{I}_b\}. \tag{3.10}$$

We refer to $\mathcal{I}$ as a *sample gathered in the time interval* $[a, b]$. We show that a schedule computed with respect to a sample $\mathcal{I}$ taken during an interval of $\ell(\mathcal{I}) = b - a = O(\varepsilon^{-2} \log n)$ steps has cost which is within a multiplicative factor $\varepsilon \in [0, 1]$ of the

optimal schedule. We then adapt WIGGINS to optimize with respect to such sample.

We start by defining the cost of a schedule w.r.t. to a sample $\mathcal{I}$.

Definition 10. *Suppose* p *is a c-schedule and* $\mathcal{I}$ *is as in Equation* (3.10)*, with* $\ell(\mathcal{I}) = b - a$*. The* θ*-cost of* p *w.r.t. to* $\mathcal{I}$ *denoted by* $\mathsf{cost}_\theta(\mathsf{p}, \mathcal{I})$ *is defined as*

$$\mathsf{cost}_\theta(\mathsf{p}, \mathcal{I}) \coloneqq \frac{1}{\ell(\mathcal{I})} \sum_{S \in \mathcal{I}} \frac{1}{1 - \theta(1 - \mathsf{p}(S))^c} \ .$$

For $1 \leq i \leq n$, define now the functions

$$W_i(\mathsf{p}, \mathcal{I}) = \frac{1}{\ell(\mathcal{I})} \sum_{S \in \mathcal{I} : i \in S} \frac{\theta c (1 - \mathsf{p}(S))^{c-1}}{(1 - \theta(1 - \mathsf{p}(S))^c)^2} \ .$$

We can then define a variant of WIGGINS, which we call WIGGINS-APX. The differences from WIGGINS are:

1. the loop on line 8 in Alg. 3 is only over the sets that appear in at least one $\mathcal{I}_j \in \mathcal{I}$.
2. WIGGINS-APX uses the values $W_i(\mathsf{p}, \mathcal{I})$ (defined above) instead of $W_i(\mathsf{p})$ (line 10 in Alg. 3);

If WIGGINS-APX reaches convergence, it returns a schedule with the minimum cost w.r.t. the sample $\mathcal{I}$. More formally, by following the same steps as in the proof of Thm. 10, we can prove the following result about WIGGINS-APX.

Lemma 12. *We have that:*

1. *if at any iteration* j *the schedule* $\mathsf{p}^{(j)}$ *has minimum cost w.r.t.* $\mathcal{I}$*, then* WIGGINS-APX *reaches convergence at iteration* $j + 1$*; and*
2. *if* WIGGINS-APX *reaches convergence, then the returned schedule* p *has minimum cost w.r.t.* $\mathcal{I}$*.*

Let $\ell(\mathcal{I})$ denote the length of the time interval during which $\mathcal{I}$ was collected. For a c-schedule p, $\mathsf{cost}_\theta(\mathsf{p}, \mathcal{I})$ is an approximation of $\mathsf{cost}_\theta(\mathsf{p})$, and intuitively the larger $\ell(\mathcal{I})$, the better the approximation.

We now show that, if $\ell(\mathcal{I})$ is large enough, then, with high probability (i.e., with probability at least $1 - 1/n^r$ for some constant r), the schedule p returned by WIGGINS-APX in case of convergence has a cost $\mathsf{cost}_\theta(\mathsf{p})$ that is close to the cost $\mathsf{cost}_\theta(\mathsf{p}^*)$ of an optimal schedule p^*.

Theorem 11. *Let r be a positive integer, and let $\mathcal{I}$ be a sample gathered during a time interval of length*

$$\ell(\mathcal{I}) \geq \frac{3(r\ln(n)+\ln(4))}{\varepsilon^2(1-\theta)} \ . \tag{3.11}$$

Let p^* be an optimal schedule, i.e., a schedule with minimum cost. If WIGGINS-APX *converges, then the returned schedule p is such that*

$$\mathsf{cost}_\theta(\mathsf{p}^*) \leq \mathsf{cost}_\theta(\mathsf{p}) \leq \frac{1+\varepsilon}{1-\varepsilon}\mathsf{cost}_\theta(\mathsf{p}^*) \ .$$

To prove Thm. 11, we need the following technical lemma.

Lemma 13. *Let p be a c-schedule and $\mathcal{I}$ be a sample gathered during a time interval of length*

$$\ell(\mathcal{I}) \geq \frac{3(r\ln(n)+\ln(2))}{\varepsilon^2(1-\theta)}, \tag{3.12}$$

where r is any natural number. Then, for every schedule p we have

$$\Pr(|\mathsf{cost}_\theta(\mathsf{p},\mathcal{I}) - \mathsf{cost}_\theta(\mathsf{p})| \geq \varepsilon \cdot \mathsf{cost}_\theta(\mathsf{p})) < \frac{1}{n^r} \ .$$

Proof. For any $S \in \mathcal{F}$, let X_S be a random variable which is $\frac{1}{1-\theta(1-\mathsf{p}(S))^c}$ with probability $\pi(S)$, and zero otherwise. Since $\mathsf{p}(S) \in [0,1]$, we have

$$1 \leq X_S \leq \frac{1}{1-\theta} \ .$$

If we let $X = \sum_{S\in\mathcal{F}} X_S$, then

$$\mathsf{cost}_\theta(\mathsf{p}) = \mathbb{E}[X] = \sum_{S\in\mathcal{F}} \mathbb{E}[X_S] \geq \sum_{S\in\mathcal{F}} \pi(S) \ . \tag{3.13}$$

Let $Z = \sum_{S\in\mathcal{F}} \pi(S)$. Then we have

$$Z \leq X \leq \frac{Z}{1-\theta} \ .$$

Let X_S^i be the i-th draw of X_S, during the time interval $\mathcal{I}$ it was sampled from, and define $X^i = \sum_{S\in\mathcal{F}} X_S^i$. We have

$$\mathsf{cost}_\theta(\mathsf{p},\mathcal{I}) = \frac{1}{\ell(\mathcal{I})}\sum_i X^i \ .$$

Let now

$$\mu = \frac{\ell(\mathcal{I})(1-\theta)}{Z}\mathsf{cost}_\theta(\mathsf{p}) \ .$$

By using the Chernoff bound for Binomial random variables [93, Corol. 4.6], we have

$$\begin{aligned}
&\Pr\left(|\mathsf{cost}_\theta(\mathsf{p},\mathcal{I}) - \mathsf{cost}_\theta(\mathsf{p})| \geq \varepsilon\mathsf{cost}_\theta(\mathsf{p})\right) \\
&= \Pr\left(\left|\sum_i X^i - \ell(\mathcal{I})\mathsf{cost}_\theta(\mathsf{p})\right| \geq \varepsilon\ell(\mathcal{I})\mathsf{cost}_\theta(\mathsf{p})\right) \\
&= \Pr\left(\left|\frac{1-\theta}{Z}\sum_i X^i - \mu\right| \geq \varepsilon\mu\right) \leq 2\exp\left(-\frac{\varepsilon^2\mu}{3}\right) \\
&\leq 2\exp\left(-\frac{\varepsilon^2\ell(\mathcal{I})(1-\theta)\mathsf{cost}_\theta(\mathsf{p})}{3Z}\right) \leq 2\exp\left(-\frac{\varepsilon^2\ell(\mathcal{I})(1-\theta)}{3}\right),
\end{aligned}$$

where the last inequality follows from the rightmost inequality in (3.13). The thesis follows from our choice of $\ell(\mathcal{I})$. □

We can now prove Thm. 11.

of Thm. 11. The leftmost inequality is immediate, so we focus on the one on the right. For our choice of $\ell(\mathcal{I})$ we have, through the union bound, that, with probability at least $1 - 1/n^r$, at the same time:

$$\begin{aligned}
(1-\varepsilon)\mathsf{cost}_\theta(\mathsf{p}) \leq \mathsf{cost}_\theta(\mathsf{p},\mathcal{I}) &\leq (1+\varepsilon)\mathsf{cost}_\theta(\mathsf{p}), \text{ and} \\
(1-\varepsilon)\mathsf{cost}_\theta(\mathsf{p}^*) \leq \mathsf{cost}_\theta(\mathsf{p}^*,\mathcal{I}) &\leq (1+\varepsilon)\mathsf{cost}_\theta(\mathsf{p}^*)
\end{aligned} \tag{3.14}$$

Since we assumed that WIGGINS-APX reached convergence when computing p, then Thm. 11 holds, and p is a schedule with minimum cost w.r.t. $\mathcal{I}$. In particular, it must be

$$\mathsf{cost}_\theta(\mathsf{p},\mathcal{I}) \leq \mathsf{cost}_\theta(\mathsf{p}^*,\mathcal{I}) \ .$$

From this and (3.14), We then have

$$(1-\varepsilon)\mathsf{cost}_\theta(\mathsf{p}) \leq \mathsf{cost}_\theta(\mathsf{p},\mathcal{I}) \leq \mathsf{cost}_\theta(\mathsf{p}^*,\mathcal{I}) \leq (1+\varepsilon)\mathsf{cost}_\theta(\mathsf{p}^*)$$

and by comparing the leftmost and the rightmost terms we get the thesis. □

3.2.3 Dynamic Settings

In this section we discuss how to handle changes in the parameters $\mathcal{F}$ and π as the (unknown) generating process Γ evolves over time. The idea is to maintain an estimation $\tilde{\pi}(S)$ of $\pi(S)$ for each set $S \in \mathcal{F}$ that we discover in the probing process, together with the last time t such that an item (t, S) has been generated (and caught at a time $t' > t$). If we have not caught an item in the form (t'', S) in an interval significantly longer than $1/\tilde{\pi}(S)$, then we assume that the parameters of Γ changed. Hence, we trigger the collection of a new sample and compute a new schedule as described in Sect. 3.2.2.

Note that when we adapt our schedule to the new environment (using the most recent sample) the system converges to its stable setting exponentially (in θ) fast. Suppose L items have been generated since we detected the change in the parameters until we adapt the new schedule. These items, if not caught, lose their novelty exponentially fast, since after t steps their novelty is at most $L\theta^t$ and decreases exponentially. In our experiments (Sect. 4.3) we provide different examples that illustrate how the load of the generating process becomes stable after the algorithm adapts itself to the changes of parameters.

3.2.4 Scaling up with MapReduce

In this section, we discuss how to adapt WIGGINS-APX to the MapReduce framework [33]. We denote the resulting algorithm as WIGGINS-MR.

In MapReduce, algorithms work in *rounds*. At each round, first a function map is executed independently (and therefore potentially massively in parallel) on each element of the input, and a number of (or zero) key-value pairs of the form (k, v) are emitted. Then, in the second part of the round, the emitted pairs are partitioned by key and elements with the same key are sent to the same machine (called the *reducer* for that key), where a function reduce is applied to the whole set of received pairs, to emit the final output.

Each iteration of WIGGINS-APX is spread over two rounds of WIGGINS-MR. At each round, we assume that the current schedule p is available to all machines (this is done in practice through a distributed cache). In the first round, we compute the values $p_i W_i$, $1 \leq i \leq n$, in the second round these values are summed to get the

normalization factor, and in the third round the schedule p is updated. The input in the first round are the sets $S \in \mathcal{I}$. The function $\mathsf{map}_1(S)$ outputs, a key-value pair (i, v_S) for each $i \in S$, with

$$v_S = \frac{\theta c(1 - \mathsf{p}(S))^{c-1}}{\ell(\mathcal{I})(1 - \theta(1 - \mathsf{p}(S))^c)^2} \ .$$

The reducer for the key i receives the pairs (i, v_S) for each $S \in \mathcal{I}$ such that $i \in S$, and aggregates them to output the pair (i, g_i), with

$$g_i = \mathsf{p}_i \sum v_S = \mathsf{p}_i W_i \ .$$

The set of pairs (i, g_i), $1 \leq i \leq n$ constitutes the input to the next round. Each input pair is sent to the same reducer,[3] which computes the value

$$g = \sum_{i=1}^{n} g_i = \sum_{i=1}^{n} \mathsf{p}_i W_i$$

and uses it to obtain the new values $\mathsf{p}_i = g_i/g$, for $1 \leq i \leq n$. The reducer then outputs (i, p_i). At this point, the new schedule is distributed to all machines again and a new iteration can start.

The same results we had for the quality of the final schedule computed by WIGGINS-APX in case of convergence carry over to WIGGINS-MR.

3.3 Experimental Results

In this section we present the results of our experimental evaluation of WIGGINS-APX.

Goals. First, we show that for a given sample $\mathcal{I}$, WIGGINS-APX converges quickly to a schedule p^* that minimizes $\mathsf{cost}_\theta(\mathsf{p}, \mathcal{I})$ (see Thm. 10). In particular, our experiments illustrate that the sequence $\mathsf{cost}_\theta(\mathsf{p}^{(1)}, \mathcal{I}), \mathsf{cost}_\theta(\mathsf{p}^{(2)}, \mathcal{I}), \ldots$ is descending and converges after few iterations. Next, we compare the output schedule of WIGGINS-APX to four other schedules: (i) uniform schedules, (ii) proportional to out-degrees, (iii) proportional to in-degrees, and (iv) proportional to *undirected* degrees, i.e., the number of incident edges. Specifically, we compute the costs of these schedules according to a sample $\mathcal{I}$ that satisfies the condition in Lemma 13 and compare them.

[3]This step can be made more scalable through *combiners*, an advanced MapReduce feature.

Then, we consider a specific example for which we know the *unique* optimal schedule, and show that for larger samples WIGGINS-APX outputs a schedule closer to the optimal. Finally, we demonstrate how our method can adapt itself to the changes in the network parameters.

Datasets	#nodes	#edges	$(\lvert V_{1K}\rvert, \lvert V_{500}\rvert, \lvert V_{100}\rvert)$	gen. rate
Enron-Email	36692	367662	(9,23,517)	7.22
Brightkite	58228	428156	(2,7,399)	4.54
web-Notredame	325729	1497134	(43,80,1619)	24.49
web-Google	875713	5105039	(134,180,3546)	57.86

Table 3.1: The datasets, corresponding statistics, and the rate of generating new items at each step.

Environment and Datasets. We implemented WIGGINS-APX in C++. The implementation of WIGGINS-APX never loads the entire sample to the main memory, which makes it very practical when using large samples. The experiments were run on a Opteron 6282 SE CPU (2.6 GHz) with 12GB of RAM. We tested our method on graphs from the SNAP repository[4] (see Table 5.1 for details). We always consider the graphs to be directed, replacing undirected edges with two directed ones.

Generating process. The generating process $\Gamma = (\mathcal{F}, \pi)$ we use in our experiments (except those in Sect. 3.3.1.1) simulates an *Independent-Cascade (IC) model* [68]. Since explicitly computing $\pi(S)$ in this case does not seem possible, we simulate the creation of items according to this model as follows. At each time t, items are generated in two phases: a "creation" phase and a "diffusion" phase. In the creation phase, we simulate the creation of "rumors" at the nodes: we flip a biased coin for each node in the graph, where the bias depends on the out-degree of the node. We assume a partition of the nodes into classes based on their out-degrees, and, we assign the same head probability for the biased coins of nodes in the same class, as shown in Table 3.2. In Table 5.1, for each dataset we report the size of the classes and the expected number of flipped coins with outcome head at each time (rightmost column). Let now v be a node whose coin had outcome head in the most recent flip. In the "diffusion" phase we simulate the spreading of the "rumor" originating at v through network according to the IC model, as follows. For each directed edge $e = u \to w$ we fix a probability p_e that a rumor that reached u is propagated through this edge

[4] `http://snap.stanford.edu`

Class	Nodes in class	Bias
V_{1K}	$\{i \in V \ : \ \mathsf{deg}^+(i) \geq 1000\}$	0.1
V_{500}	$\{i \in V \ : \ 500 \leq \mathsf{deg}^+(i) < 1000\}$	0.05
V_{100}	$\{i \in V \ : \ 100 \leq \mathsf{deg}^+(i) < 500\}$	0.01
V_0	$\{i \in V \ : \ \mathsf{deg}^+(i) < 100\}$	0.0

Table 3.2: Classes and bias for the generating process.

to node w (as in IC model), and events for different rumors and different edges are independent. Following the literature [22, 23, 66, 68, 121], we use $p_{u\to w} = \frac{1}{\mathsf{deg}^-(w)}$. If we denote with S the final set of nodes that the rumor created at v reached during the (simulated) diffusion process (which always terminates), we have that through this process we generated an item (t, S), without the need to explicitly define $\pi(S)$.

each directed edge $e = v \to w$ has a probability

3.3.1 Efficiency and Accuracy

In Sect. 3.2.1 we showed that when a run of WIGGINS-APX converges (according to a sample $\mathcal{I}$) the computed c-schedule is optimal with respect to the sample $\mathcal{I}$ (Lemma 12). In our first experiment, we measure the rate of convergence and the execution time of WIGGINS-APX. We fix $\epsilon = 0.1$, $\theta = 0.75$, and consider $c \in \{1, 3, 5\}$. For each dataset, we use a sample $\mathcal{I}$ that satisfies (3.12), and run WIGGINS-APX for 30 iterations. Denote the schedule computed at round i by p^i. As shown in Figure 3.1, the sequence of cost values of the schedules p^i's, $\mathsf{cost}_\theta(\mathsf{p}^i, \mathcal{I})$, converges extremely fast after few iterations.

Datasets	$\lvert\mathcal{I}\rvert$	avg. item size	avg. iter. time (sec)
Enron-Email	97309	12941.33	204.59
Brightkite	63652	17491.08	144.35
web-Notredame	393348	183.75	10.24
web-Google	998038	704.74	121.88

Table 3.3: Sample size, average size of items in the sample, and the running time of each iteration in WIGGINS-APX (for $c = 1$).

For each graph, the size of the sample $\mathcal{I}$, the average size of sets in $\mathcal{I}$, and the average time of each iteration is given in Table 3.3. Note that the running time of

each iteration is a function of both sample size and sizes of the sets (informed-sets) inside the sample.

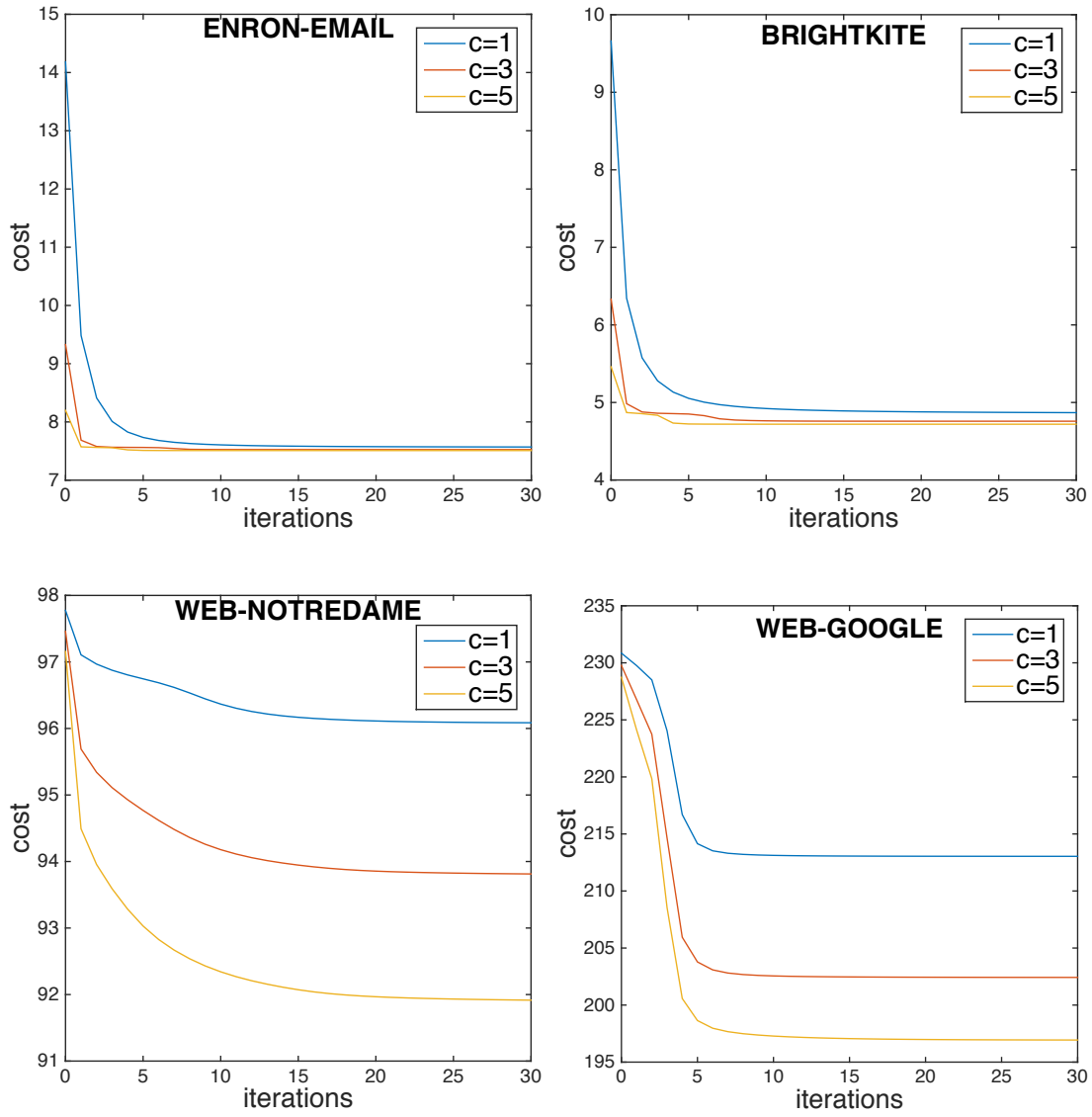


Figure 3.1: The cost of intermediate c-schedules at iterations of WIGGINS-APX according to $\mathcal{I}$.

Next, we extract the 1-schedules output by WIGGINS-APX, and compare its cost to four other natural schedules: `unif`, `outdeg`, `indeg`, and `totdeg` that probe each node, respectively, uniformly, proportional to its out-degree, proportional to its in-degree, and proportional to the number of incident edges. Note that for undirected graphs `outdeg`, `indeg`, and `totdeg` are essentially the same schedule.

To have a fair comparison among the costs of these schedules and WIGGINS-APX, we calculate their costs according to 10 independent samples, $\mathcal{I}_1, \ldots, \mathcal{I}_{10}$ that satisfy (3.12), and compute the average. The results are shown in Table 3.4, and show that WIGGINS-APX outperforms the other four schedules.

Dataset	WIGGINS-APX	`uniform`	`outdeg`	`indeg`	`totdeg`
Enron-Email	7.55	14.16	9.21	9.21	9.21
Brightkite	4.85	9.64	6.14	6.14	6.14
web-Notredame	96.10	97.78	97.37	97.43	97.40
web-Google	213.15	230.88	230.48	230.47	230.47

Table 3.4: Comparing the costs of 5 different 1-schedules.

3.3.1.1 A Test on Convergence to Optimal Schedule

Here, we further investigate the convergence of WIGGINS-APX, using an example graph and process for which we know the *unique* optimal schedule. We study how close the WIGGINS-APX output is to the optimal schedule when (i) we start from different initial schedules, p^0, or (ii) we use samples $\mathcal{I}$'s obtained during time intervals of different lengths.

Suppose $G = (V, E)$ is the complete graph where $V = [n]$. Let $\Gamma = (\mathcal{F}, \pi)$ for $\mathcal{F} = \{S \in 2^{[n]} \mid 1 \leq |S| \leq 2\}$, and $\pi(S) = \frac{1}{|\mathcal{F}|}$. It is easy to see that $\mathsf{cost}_\theta(\mathsf{p})$ is a symmetric function, and thus, the uniform schedule is optimal. Moreover, by Corollary 8 the uniform schedule is the only optimal schedule, since $\{v\} \in \mathcal{F}$ for every $v \in V$. Furthermore, we let $\theta = 0.99$ to increase the sample complexity (as in Lemma 13) and make it harder to learn the uniform/optimal schedule.

In our experiments we run the WIGGINS-APX algorithm, using (i) different *random* initial schedules, and (ii) samples $\mathcal{I}$ obtained from time intervals of different lengths. For each sample, we run WIGGINS-APX 10 times with 10 different random initial schedules, and compute the **exact** cost of each schedule, and its variation distance to the uniform schedule. Our results are plotted in Figure 3.2, and as shown, by increasing the sample size (using longer time intervals of sampling) the output schedules gets very close to the uniform schedule (the variance gets smaller and smaller).

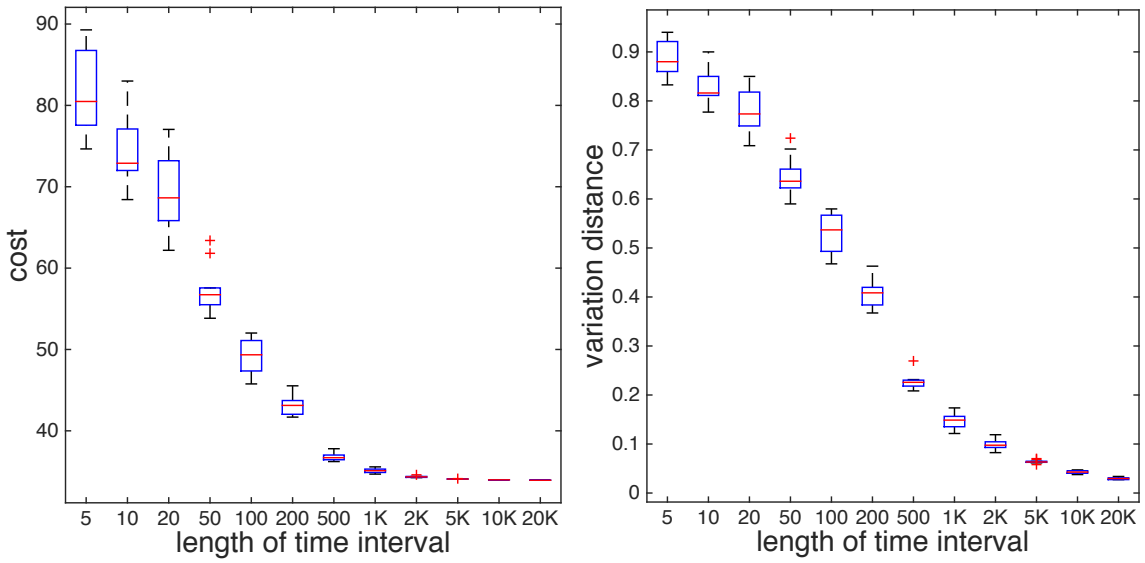


Figure 3.2: The cost of WIGGINS-APX outputs and their variation distance to the optimal schedule: The top and bottom edge of each box are the 25^{th} and 75^{th} percentiles respectively , and the median (50^{th} percentile) is shown by a red line segment. The $+$ symbols denote outliers, i.e., points larger than $q_3 + 1.5(q_3 - q_1)$ or smaller than $q_1 - 1.5(q_3 - q_1)$, where q_1 and q_3 are the 25^{th} and 75^{th} percentiles, respectively. The whiskers extend to the most extreme data points that are not outliers.

3.3.2 Dynamic Settings

In this section, we present experimental results that show how our algorithm can adapt itself to the new situation. The experiment is illustrated in Fig. 3.3. For each graph, we start by following an optimal 1-schedule in the graph. At the beginning of each "gray" time interval, the labels of the nodes are permuted randomly, to impose great disruptions in the system. Following that, at the beginning of each "green" time interval our algorithm starts gathering samples of Γ. Then, WIGGINS-APX computes the schedule for the new sample, using 50 rounds of iterations, and starts probing. The length of each colored time interval is $R = \frac{3(\log(n)+\log(2))}{\epsilon^2(1-\theta)}$, for $\epsilon = 0.5$ and $\theta = 0.75$, motivated by Theorem 11.

Since the cost function is defined asymptotically (and explains the asymptotic behavior of the system in response to a schedule), in Figure 3.3 we plot the load of the system $L_\theta(t)$ over the time (blue), and the *average* load in the normal and perturbed time intervals (red). Based on this experiment, and as shown in Figure 3.3, after adapting to the new schedule, the effect of the disruption caused by the perturbation disappears immediately. Note that when the difference between the optimal cost

and any other schedule is small (like web-Notredame), the jump in the load will be small (e.g., as shown in Figure 3.1 and Table 3.4, the cost of the initial schedule for web-Notredame is very close the optimal cost, obtained after 30 iteration).

3.4 Related Work

The novel problem we focus on in this work generalizes and complements a number of problems studied in the literature.

The "Battle of Water Sensor Network" challenge [100] motivated a number of works on *outbreak detection*: the goal is to optimally place static or moving sensors in water networks to detect contamination [56, 73, 83]. The optimization can be done w.r.t. a number of objectives, such as maximizing the probability of detection, minimizing the detection time, or minimizing the size of the subnetwork affected by the phenomena [83]. A related work [4] considered sensors that are sent along fixed paths in the network with the goal of gathering sufficient information to locate possible contaminations. Early detection of contagious outbreaks by monitoring the neighborhood (friends) of a randomly chosen node (individual) was studied by [27]. [76] present efficient schedules for minimizing energy consumption in battery operated sensors, while other works analyzed distributed solutions with limited communication capacities and costs [50, 74, 75]. In contrast, our work is geared to detection in huge but virtual networks such as the Web or social networks embedded in the Internet, where it is possible to "sense" or probe (almost) any node at approximately the same cost. Still only a restricted number of nodes can be probed at each steps but the optimization of the probing sequence is over a much larger domain, and the goal is to identify the outbreaks (items) regardless of their size and solely by considering their interest value.

Our methods complement the work on *Emerging Topic Detection* where the goal is to identify emerging topics in a social network, assuming full access to the stream of all postings. Providers, such as Twitter or Facebook, have an immediate access to all tweets or postings as they are submitted to their servers [20, 89]. Outside observers need an efficient mechanism to monitor changes, such as the methods developed in this work.

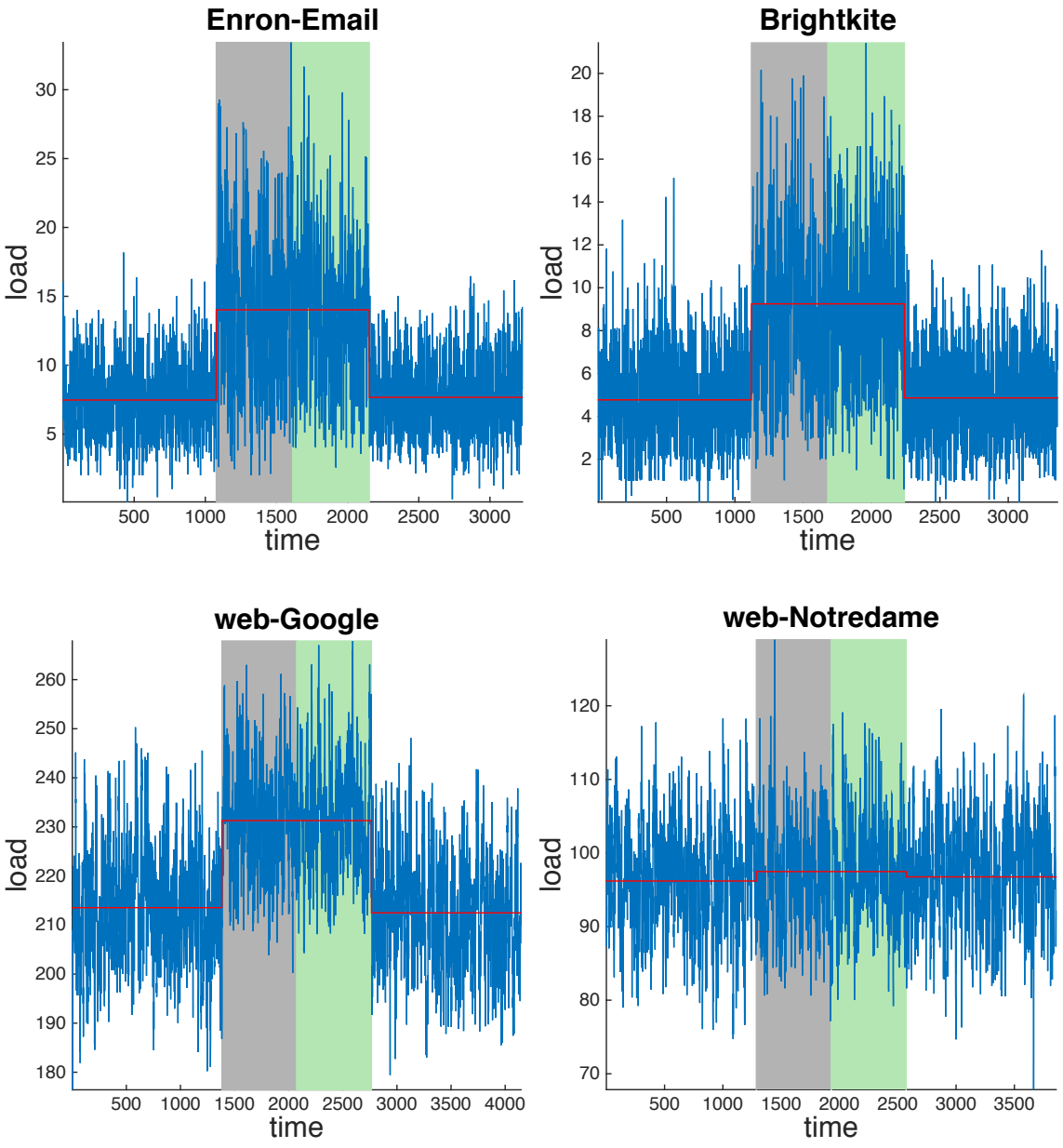


Figure 3.3: Perturbation, Sampling, and Adapting (For details see Section 3.3.2).

Web-crawling is another research area that study how to obtain the most recent snapshots of the web. However, it differs from our model in two key points: our model allows items to propagate their copies, and they will be caught if any of their copies is discovered (where snapshots of a webpage belong to that page only), and all the generated items should be discovered (not just the recent ones) [32, 125].

The goal of the News and Feed Aggregation problem is to capture updates in news websites (e.g. by RSS feeds) [17, 60, 99, 114]. Our model differs from that setting in that we consider copies of the same news in different web sites as equivalent and therefore are only interested in discovering one of the copies.

Part II

Centrality Maximization

Chapter 4

Scalable Betweenness Centrality Maximization via Sampling

In this chapter we study the betweenness centrality maximization problem. The betweenness centrality of a node u is defined as

$$B(u) = \sum_{s,t} \frac{\sigma_{s,t}(u)}{\sigma_{s,t}},$$

where $\sigma_{s,t}$ is the number of s-t shortest paths, and $\sigma_{s,t}(u)$ is the number of s-t shortest paths that have u as their internal node. However, in many applications, e.g. [47, 62], we are interested in centrality of sets of nodes. For this reason, the notion of BWC has been extended to sets of nodes [61, 127]. For a set of nodes $S \subseteq V$, we define the betweenness centrality of S as

$$B(S) = \sum_{s,t \in V} \frac{\sigma_{s,t}(S)}{\sigma_{s,t}},$$

where $\sigma_{s,t}(S)$ is the number of s-t shortest paths that have an internal node in S. Note that we cannot obtain $B(S)$ from the values $\{B(v), v \in S\}$. In this work, we study the *Centrality Maximization problem* (CMP) defined formally as follows:

Definition 11 (CMP)**.** *Given a network $G = (V, E)$ and a positive integer k, find a subset $S^* \subseteq V$ such that*

$$S^* \in \arg\max_{S \subseteq V : |S| \leq k} B(S).$$

We also denote the maximum centrality of a set of k nodes by $\texttt{OPT}_k$, i.e., $\texttt{OPT}_k = \max_{S \subseteq V : |S| \leq k} B(S)$.

It is known that CMP is APX-complete [41]. The best deterministic algorithms for CMP rely on the fact that BWC is monotone-submodular and provide a $(1-1/e)$-approximation [35, 41]. However, the running time of these algorithms is at least quadratic in the input size, and do not scale well to large-scale networks.

In this chapter we focus on scalability of CMP, and graph mining applications. Our main contributions are summarized as follows.

Efficient algorithm. We provide a randomized approximation algorithm, HEDGE, based on sampling shortest paths, for accurately estimating the BWC and solving CMP. Our algorithm is simple, scales gracefully as the size of the graph grows, and improves the previous result [127], by (i) providing a $(1-1/e-\epsilon)$-approximation, and (ii) smaller sized samples. Specifically, in Yoshida's algorithm [127], a sample contains all the nodes on "any" shortest path between a pair, whereas in our algorithm, each sample is just a set of nodes from a single shortest path between the pair.

The $\texttt{OPT}_k = \Theta(n^2)$ assumption. Prior work on BWC estimation strongly relies on the assumption that $\texttt{OPT}_k = \Theta(n^2)$ for a constant integer k [127]. As we show in Section 4.2.3, this assumption is not true in general. Only empirical evidence so far supports this strong assumption.

We show that two broad families of networks satisfy this assumption: bounded treewidth networks and a popular family of stochastic networks that provably generate scale-free, small-world graphs with high probability. Note that the classical Barabási-Albert scale-free random tree model generates bounded treewidth networks [10, 94]. Our results imply that the $\texttt{OPT}_k = \Theta(n^2)$ assumption holds even for $k = 1$, for these families of networks. To our knowledge, this is the first theoretical evidence for the validity of this *crucial* assumption on real-world networks.

General analytical framework. To analyze our algorithm, HEDGE, we provide a general analytical framework based on Chernoff bound and submodular optimization, and show that it can be applied to any other centrality measure if it (i) is monotone-submodular, and (ii) admits a hyper-edge sampler (defined in Sect. 4.1). Two examples of such centralities are the *coverage* [127] and the *κ-path* centralities [5].

Experimental evaluation. We provide an experimental evaluation of our algorithm

that shows that it scales gracefully as the graph size increases and that it provides accurate estimates. We also provide a comparison between the method in [127] and our sampling method.

Applications. Our scalable algorithm enables us to study some interesting characteristics of the central nodes. In particular, if S is a set of nodes with high BWC, we focus on answering the following questions.

(1) How does the centrality of the most central set of nodes change in time-evolving networks? We study the DBLP and the AUTONOMOUS systems graphs. We mine interesting growth patterns, and we compare our results to stochastic Kronecker graphs, a popular random graph model that mimics certain aspects of real-world networks. We observe that the Kronecker graphs behave similarly to real-world networks.

(2) Influence maximization has received a lot of attention since the seminal work of Kempe *et al.* [68]. Using our scalable algorithm we can compute a set of central nodes that can be used as seeds for influence maximization. We find that betweenness centrality performs relatively well compared to a state-of-the-art influence maximization algorithm.

(3) We study four strategies for attacking a network using four centrality measures: betweenness [42], coverage, κ-path, and triangle centrality. Interestingly, we find that the κ-path and triangle centralities can be more effective at destroying the connectivity of a graph.

4.1 Algorithm

In this section we provide our algorithm, `HEDGE` (Hyper-EDge GrEedy), and a general framework for its analysis. We start by defining a *hyper-edge sampler* that will be used in `HEDGE`.

Definition 12 (Hyper-edge sampler)**.** *We say that an algorithm $\mathcal{A}$ is a* ***hyper-edge sampler*** *for a function $C : 2^V \to \mathbb{R}$ if it outputs a randomly generated subset of nodes $h \subseteq V$ such that*

$$\forall S \subseteq V : \quad Pr_{h\sim\mathcal{A}}(h \cap S \neq \emptyset) = \frac{1}{\alpha}C(S),$$

where α is a normalizing factor, and independent of the set S. We call each h (sampled by $\mathcal{A}$) a random hyper-edge, or in short, a ***hyper-edge****. In this case, we say C* ***admits*** *a hyper-edge sampler.*

Our proposed algorithm `HEDGE` assumes the existence of a hyper-edge sampler and uses it in a black-box manner. Namely, `HEDGE` is oblivious to the specific mechanics of the hyper-edge sampler. The following lemma provides a simple hyper-edge sampler for BWC.

Lemma 14. *BWC admits a hyper-edge sampler.*

Proof. Let $\mathcal{A}$ be an algorithm that selects two nodes $s, t \in V$ uniformly at random, selects a s-t shortest path P, among all s-t shortest paths, uniformly at random (this can be done in linear time $O(m+n)$ using bread-first-search from s, finding the number of shortest paths from s and backward pointers; e.g. see [105]), and finally outputs the internal nodes of P (i.e., the nodes of P except s and t).

Now, suppose h is an output of $\mathcal{A}$. Since the probability of choosing each pair is $\frac{1}{n(n-1)}$, and for a given pair s, t the probability of $S \cap h \neq \emptyset$ is $\frac{\sigma_{s,t}(S)}{\sigma_{s,t}}$, for every $S \subseteq V$ we have

$$\Pr_{h\sim\mathcal{A}}(h \cap S \neq \emptyset) = \sum_{s,t\in V} \frac{1}{n(n-1)} \frac{\sigma_{s,t}(S)}{\sigma_{s,t}} = \frac{1}{n(n-1)} B(S).$$

Also note that in this case, the normalizing factor is $\alpha = n(n-1) = \Theta(n^2)$. □

For a subset of nodes $S \subseteq V$, and a set $\mathcal{H}$ of random hyper-edges, denote

$$\deg_{\mathcal{H}}(S) = |\{h \in \mathcal{H} \mid h \cap S \neq \emptyset\}|.$$

The pseudocode of our proposed algorithm `HEDGE` is given in Algorithm 4. First, it samples q hyper-edges using the hyper-edge sampler $\mathcal{A}$ and then it runs a natural greedy procedure on $\mathcal{H}$.

4.1.1 Analysis

In this section we provide our general analytical framework for `HEDGE`, which works with any hyper-edge sampler. To start, define $B_{\mathcal{H}}(S) = \frac{\alpha}{|\mathcal{H}|} \deg_{\mathcal{H}}(S)$ as the estimate

Algorithm 4: HEDGE

Input: A hyper-edge sampler $\mathcal{A}$ for BWC, number of hyper-edges q, and the size of the output set k.
Output: A subset of nodes, S of size k.

```
begin
    H ← ∅;
    for i ∈ [q] do
        h ∼ A (sample a random hyper-edge);
        H ← H ∪ {h};
    end
    S ← ∅ ;
    while |S| < k do
        u ← arg max_{v∈V} deg_H({v});
        S ← S ∪ {u};
        for h ∈ H such that u ∈ h do
            H ← H \ {h};
        end
    end
    return S;
end
```

centrality (BWC) of a set S according to the sample $\mathcal{H}$ of hyper-edges, and for a graph G let

$$q(G, \epsilon) = \frac{3\alpha(\ell + k)\log(n)}{\epsilon^2 \texttt{OPT}_k},$$

where n is the number of nodes in G, and ℓ is a positive integer. We have the following lemma:

Lemma 15. *Let $\mathcal{H}$ be a sample of independent hyper-edges such that $|\mathcal{H}| \geq q(G, \epsilon)$. Then for all $S \subseteq V$ where $|S| \leq k$, we have $Pr[|B_{\mathcal{H}}(S) - B(S)| \geq \epsilon \cdot \texttt{OPT}_k] < n^{-\ell}$.*

Proof. Suppose $S \subseteq V$ and $|S| \leq k$, and let X_i be a binary random variable that indicates whether the i-th hyper-edge in $\mathcal{H}$ intersects S. Notice that $deg_{\mathcal{H}}(S) = \sum_{i=1}^{|\mathcal{H}|} X_i$ and by the linearity of expectation $\mathbb{E}[deg_{\mathcal{H}}(S)] = |\mathcal{H}| \cdot \mathbb{E}[X_1] = \frac{q}{\alpha} B(S)$.

Using the independence assumption and the Chernoff bound, we obtain:

$$\begin{aligned}
&\Pr\left[|B_{\mathcal{H}}(S) - B(S)| \geq \delta \cdot B(S)\right] && = \\
&\Pr\left[\left|\frac{q}{\alpha}B_{\mathcal{H}}(S) - \frac{q}{\alpha}B(S)\right| \geq \frac{\delta q}{\alpha} \cdot B(S)\right] && = \\
&\Pr\left[|deg_{\mathcal{H}}(S) - \mathbb{E}\left[deg_{\mathcal{H}}(S)\right]| \geq \delta \cdot \mathbb{E}\left[deg_{\mathcal{H}'}(S)\right]\right] && \leq \\
&2\exp\left(-\frac{\delta^2}{3}\frac{q}{\alpha}B(S)\right).
\end{aligned}$$

Now, by letting $\delta = \frac{\epsilon \texttt{OPT}_k}{B(S)}$ and substituting the lower bound for $q(G, \epsilon)$ we obtain

$$\Pr\left[|B_{\mathcal{H}}(S) - B(S)| \geq \epsilon \texttt{OPT}_k\right] \leq n^{-(\ell+k)},$$

and by taking a union bound over all possible subsets $S \subseteq V$ of size k we obtain $|B_{\mathcal{H}}(S) - B(S)| < \epsilon \cdot \texttt{OPT}_k$ with probability at least $1 - 1/n^{\ell}$, for all such subsets S. □

The following theorem shows that if the number of samples, i.e. $|\mathcal{H}|$, is at least $q(G, \epsilon/2)$, then HEDGE provides a $(1 - 1/e - \epsilon)$-approximate solution.

Theorem 12. *If $\mathcal{H}$ is a sample of at least $q(G, \epsilon/2)$ hyper-edges for some $\epsilon > 0$, and S is the output of HEDGE, we have $B(S) \geq (1 - 1/e - \epsilon)\texttt{OPT}_k$, with high probability.*

Proof. Note that B is (i) monotone since if $S_1 \subseteq S_2$ then $B(S_1) \leq B(S_2)$, and (ii) submodular since if $S_1 \subseteq S_2$ and $u \in V \setminus S_2$ then $B(S_2 \cup \{u\}) - B(S_2) \leq B(S_1 \cup \{u\}) - B(S_1)$.

Similarly, $B_{\mathcal{H}}$ is monotone and submodular. Therefore, using the greedy algorithm (second part of HEDGE) we have (see [96])

$$B_{\mathcal{H}}(S) \geq (1 - 1/e)B_{\mathcal{H}}(S') \geq (1 - 1/e)B_{\mathcal{H}}(S^*),$$

where

$$S' = \arg\max_{T:|T|\leq k} B_{\mathcal{H}}(T), \quad \text{and} \quad S^* = \arg\max_{T:|T|\leq k} B(T).$$

Notice that $\texttt{OPT}_k = B(S^*)$. Since $|\mathcal{H}| \geq q(G, \epsilon/2)$, by Lemma 15 with probability $1 - \frac{1}{n^{\ell}}$ we have

$$B(S) \geq B_{\mathcal{H}}(S) - \frac{\epsilon}{2}\texttt{OPT}_k \geq \left(1 - \frac{1}{e}\right) B_{\mathcal{H}}(S^*) - \frac{\epsilon}{2}\texttt{OPT}_k$$
$$\geq \left(1 - \frac{1}{e}\right)\left(B(S^*) - \frac{\epsilon}{2}\texttt{OPT}_k\right) - \frac{\epsilon}{2}\texttt{OPT}_k \geq \left(1 - \frac{1}{e} - \epsilon\right)\texttt{OPT}_k,$$

where we used the fact $B(S^*) = \texttt{OPT}_k$, and the proof is complete. □

The total running time of HEDGE depends on the running time of the hyper-edge sampler and the greedy procedure. Specifically, the total running time is $O(t_{he} \cdot |\mathcal{H}| + (n\log(n) + |\mathcal{H}|))$, where t_{he} is the *expected* required amount of time for the sampler to output a single hyper-edge. The first term corresponds to the total required time for sampling, and the second term corresponds to an almost-linear-time implementation of greedy procedure as in [13].

Remark 2. *Note that if $\texttt{OPT}_k = \Theta(n^2)$, the sample complexity in Theorem 12 becomes $O\left(\frac{k\log(n)}{\epsilon^2}\right)$. We provide the first theoretical study on this assumption in Sect. 4.2.*

Finally, we provide a lower bound on the sample complexity of HEDGE, in order to output a $(1 - 1/e - \epsilon)$-approximate solution. This lower bound is still valid even if $\texttt{OPT}_k = \Theta(n^2)$.

Theorem 13. *In order to output a set S of size k such that $B(S) \geq (1 - 1/e - \epsilon)\texttt{OPT}_k$ w.h.p., the sample size in both HEDGE and [127]'s algorithm needs to be at least $O\left(\frac{n}{\epsilon^2}\right)$.*

Proof. Define a graph $A = (V_A, E_A)$ where

$$V_A = \left\{(i,j) \mid 1 \leq i \leq \epsilon\sqrt{n} \text{ and } 1 \leq j \leq \sqrt{n}\right\},$$

and two nodes (i,j) and (i',j') are connected if $i = i'$ or $j = j'$.[1] Note that the distance between every pair of nodes is at most 2, and there are at most 2 shortest paths between a pair of nodes in A. Let G be a graph of size n which has $(1-\epsilon)n$ isolated nodes and A as its largest connected component. We have the following lemma:

Lemma 16. *If $k = \epsilon n$, then $\texttt{OPT}_k = \sqrt{n}(\sqrt{n}-1)\cdot\epsilon\sqrt{n}(\epsilon\sqrt{n}-1) = \Theta(\epsilon^2 n^2) = \Theta(n^2)$, since ϵ is a constant.*

[1] Without loss of generality we can assume $\epsilon\sqrt{n}$ and $\sqrt{n}$ are integers, as the arguments still hold after rounding them to the closest integers.

Proof of the lemma. Obviously, all the isolated nodes in G have zero BWC, and thus, the optimal set, S^*, is A (which is already of size $k = \epsilon n$). Now, if for two nodes s, t in G, there is a shortest path with an internal node in A, we have $s, t \in V_A$ such that $s = (i, j)$ and $t = (i', j')$ where $i \neq i'$ and $j \neq j'$. In this case, there are exactly two s-t shortest paths with exactly 1 internal nodes. Finally, the number of such pairs is exactly $\sqrt{n}(\sqrt{n} - 1) \cdot \epsilon\sqrt{n}(\epsilon\sqrt{n} - 1)$. □

Note that in both HEDGE and the algorithm of [127], we first choose a pair of nodes in s, t in G, and if s and t are not in the same connected component, the returned hyper-edge is an empty set. Therefore, in order to have a non-empty hyper-edge both nodes should be chosen from V_A, which occurs with probability ϵ^2. Thus, sampling $o(n/\epsilon^2)$ hyper-edge results in reaching to at most $o(n) = o(|A|) = o(k)$ nodes, and so, the algorithm will not be able to tell the difference between the isolated nodes and many (arbitrarily large) number of nodes in A as they were not detected by any hyper-edge. □

Remark 3. *Theorem 13 implies that the number of samples $M = O(\log(n)/\epsilon^2)$ as in [127] is not sufficient.*

4.1.2 Beyond Betweenness Centrality

Suppose $C : 2^V \to \mathbb{R}^{\geq 0}$ is a centrality measure that is also defined on subset of nodes. Clearly, if C is monotone-submodular and admits a hyper-edge sampler, the algorithm HEDGE can be applied to and all the results in this section hold for C. Here, we give a couple of examples of such centrality measures, and it is easy to verify their monotonicity and submodularity.

Coverage centrality. The coverage centrality [127] for a set $S \subseteq V$ is defined as $C(S) = \sum_{(s,t)\in V^2} P_{s,t}(S)$, where $P_{s,t}(S)$ is 1 if S has an internal node on any s-t shortest path, and 0 otherwise. The coverage centrality admits a hyper-edge sampler $\mathcal{A}$ as follows: uniformly at random pick two nodes s and t. By running a breadth-first-search from s, we output every node that is on at least one shortest path from s to t. Note that for every subset of nodes $\Pr_{h\sim\mathcal{A}}(h \cap S \neq \emptyset) = \frac{1}{n(n-1)}C(S)$.

κ-Path centrality. In [5], κ-path centrality was introduced, as a centrality measure on the nodes of a network. However, it can easily be generalized to any subset of

nodes as $C(S) = \sum_{s \in V} P^s_\kappa(S)$, where $P^s_\kappa(S)$ is the probability that a *random simple path* of length κ starting at s will pass a node in S: a random simple path starting at node s is generated by running a random walk that always chooses an unvisited neighbor uniformly at random, and stops after κ of edges being traversed or if there is no unvisited neighbor. Note that κ-path centrality is a generalization of *degree centrality* by letting $\kappa = 1$ and considering sets of single nodes.

Obviously, κ-path centrality admits a hyper-edge sampler based on its definition: Let $\mathcal{A}$ be an algorithm that picks a node uniformly at random, and generates a random simple path of length at most κ, and outputs the the generated simple path as a hyper-edge. Therefore, for any subset S we have $\Pr_{h \sim \mathcal{A}}(h \cap S \neq \emptyset) = \frac{1}{n} \sum_{s \in V} P^s_\kappa(S) = \frac{1}{n} C(S)$.

4.2 On $\texttt{OPT}_k = \Theta(n^2)$ Equality

Recall that all additive approximation guarantees for BWC as well as all existing approximation guarantees for A-BWC involve an error term which grows as $\Theta(n^2)$. In this Section we provide strong theoretical evidence regarding the following question: "Why does prior work which relies heavily on the strong assumption that $\texttt{OPT}_k = \Theta(n^2)$ perform well on real-world networks?" We quote Yoshida [127]: *This additive error should not be critical in most applications, as numerous real-world graphs have vertices of centrality* $\Theta(n^2)$.

We show that the $\texttt{OPT}_k = \Theta(n^2)$ assumption holds for two important classes of graphs: graphs of bounded treewidth networks, and for certain stochastic graph models that generate scale-free and small-world networks, known as *random Apollonian networks* [43].

4.2.1 Bounded Treewidth Graphs

We start by defining the treewidth of a graph:

Definition 13 (Treewidth)**.** *For an undirected graph $G = (V, E)$, a* ***tree decomposition*** *is a tree T with nodes $V_1, \ldots, V_r$ where each V_i is (assigned to) a subset of V such that (i) for every $u \in V$ there exists at least an i where $u \in V_i$, (ii) if V_i and V_j both contains a node u, then u belongs to every V_k on the unique shortest path from V_i*

to V_j in T, and (iii) for every edge $(u, v) \in E$ there exists a V_i such that $u, v \in V_i$. The ***width*** *of the tree decomposition T is defined as $max_{1 \leq i \leq r} |V_i| - 1$, and the* ***treewidth*** *of the graph G is the minimum possible width of any tree decomposition of G.*

Now, we have the following theorem.

Theorem 14. *Let $G = (V, E)$ be an undirected, connected graph of bounded treewidth. Then $\mathtt{OPT}_k = \Theta(n^2)$.*

Proof. Suppose w is the treewidth of G, which is a constant (bounded). It is known that any graph of treewidth w has a balanced vertex separator[2] $S \subseteq V$ of size at most $w + 1$ [106]. This implies that $O(n^2)$ shortest paths pass through S. Since $|S| = w + 1 = \Theta(1)$, there exists at least one vertex $u \in S$ such that $B(u) = \Theta(n^2)$. Hence, $\mathtt{OPT}_1 = \Theta(n^2)$, and since $\mathtt{OPT}_1 \leq \mathtt{OPT}_k$ we have $\mathtt{OPT}_k = \Theta(n^2)$. □

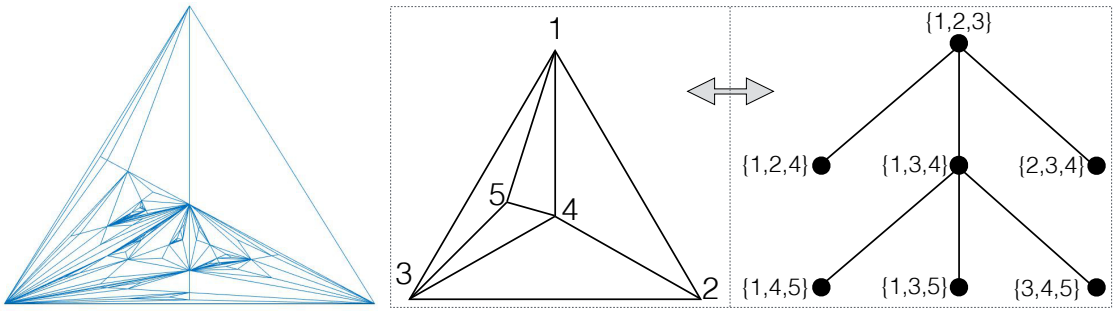
It is worth emphasizing that the classical Barabási-Albert random tree model [10, 94] belongs to this category. For a recent study of the treewidth parameter on real-world networks, see [1].

4.2.2 Scale-free, Small-world Networks

We show that $\mathtt{OPT}_k = \Theta(n^2)$ for random Apollonian networks. Our proof for the latter model relies on a technique developed by Frieze and Tsourakakis [43] and carries over for random unordered increasing k-trees [45]. A *random Apollonian network* (RAN) is a network that is generated iteratively. The RAN generator takes as input the desired number of nodes $n \geq 3$ and runs as follows:

- Let G_3 be the triangle graph, whose nodes are $\{1, 2, 3\}$, and drawn in the plane.
- **for** $t \leftarrow 4$ to n:
  - Sample a face $F_t = (i, j, k)$ of the planar graph G_{t-1} uniformly at random, except for the outer face.
  - Insert the new node t inside this face connecting it to i, j, k.

[2] Means a set of nodes Γ such that $V \setminus \Gamma = A \cup B$, where A and B are disjoint and both have size $\Theta(n)$.



(a) (b)

Figure 4.1: (a) An instance of a random Apollonian network for $n = 100$. (b) Bijection between RANs and random ternary trees.

Figure 4.1(a) shows an instance of a RAN for $n = 100$. The triangle is originally embedded on the plane as an equilateral triangle. Also, when a new node t chooses its face (i, j, k) it is embedded in the barycenter of the corresponding triangle and connects to i, j, k via the straight lines: $(i, t), (j, t)$, and (k, t). It has been shown that the diameter of a RAN is $O(\log(n))$ with high probability [39, 43].

At any step, we refer to set of candidate faces as the set of active faces. Note that there is a bijection between the active faces of a RAN and the leaves of random ternary trees as illustrated in Figure 4.1(b), and noticed first by [43].

We shall make use of the following formulae for the number of nodes (v_t), edges (e_t) and faces (f_t; excluding the outer face) after t steps in a RAN G_t:

$$v_t = t, \quad e_t = 3t - 6, \quad f_t = 2t - 5.$$

Note that $f_t = \Theta(v_t) = \Theta(t)$.

Theorem 15. *Let G be a RAN of size n. Then $\texttt{OPT}_k = \Theta(n^2)$.*

Proof. Note that removing a node from the random ternary tree $T = (V_T, E_T)$ (as in Fig. 4.1(b)) corresponds to removing three nodes from G, corresponding to a face F that existed during the RAN generation process. Clearly, the set of these three nodes is a vertex separator that separates the nodes inside and outside of F. Therefore, all nodes in the tree except for the root r correspond to a vertex separator in G. Observe that the leaves in T correspond to the set of active faces, and thus, T has $f_n = 2n - 5$ leaves after n steps.

We claim that there exists an edge $(F, F') \in E_T$ (recall that the nodes of T are active faces during the generating process of G) such that the removal of e from E_T

results in two subtrees with $\Theta(n)$ leaves. We define $g : V_T \to \mathbb{Z}$ to be the function that returns for a node $v \in V_T$ the number of leaves of the subtree rooted at v. Hence, $g(r) = 2n - 5, g(u) = 1$ for any leaf u. To find such an edge consider the following algorithm. We start from the root r of T descending to the bottom according to the following rule which creates a sequence of nodes $u_0 = r, u_1, u_2, \ldots$: we set u_{i+1} to be the child with most leaves among the tree subtrees rooted at the three children of u_i. We stop when we first find a node u_i such that $g(u_i) \geq cn$ and $g(u_{i+1}) < cn$ for some constant c. Clearly, $g(u_{i+1}) \geq cn/3 = \Theta(n)$, by pigeonhole principle. So, let $F = u_i$ and $F' = u_{i+1}$.

Now suppose $F' = \{x, y, z\}$, and consider removing x, y, and z from G. Clearly, $F' \neq r$ as F' is a child of F. Also, due to the construction of a RAN, after removing x, y, z, there are exactly two connected components, G_1 and G_2. Also, since $cn/3 \leq g(F') < cn$, the number of nodes in each of G_1 and G_2 is $\Theta(n)$.

Finally, observe that at least one of the three nodes x, y, z must have betweenness centrality score $\Theta(n^2)$, as the size of the separator is 3 and there exist $\Theta(n^2)$ paths that pass through it (connecting the nodes in G_1 and in G_2). Therefore, $\texttt{OPT}_1 \geq \max\{B(x), B(y), B(z)\} = \Theta(n^2)$, and since $\texttt{OPT}_1 \leq \texttt{OPT}_k$ we have $\texttt{OPT}_k = \Theta(n^2)$. □

We believe that this type of coupling can be used to prove similar results for other stochastic graph models.

4.2.3 On Maximum Centrality

Here, we present two examples in which the assumption of $\texttt{OPT}_k = \Theta(\alpha)$ does not hold. However, it still remains as an interesting open problem to see what properties of the graphs result in this assumption.

Complete Graph. Obviously the centrality of each set of nodes, both for Betweenness and Coverage centralities, is zero. Thus $\texttt{OPT}_k = o(n^2) \neq \Theta(n^2)$. Note that in Betweenness and Coverage centrality we had $\alpha = n(n-1) = \Theta(n^2)$.

Hypercube. The hypercube Q_r (with $n = 2^r$ nodes) is a *vertex-transitive graph*, i.e., for each pair of nodes there is an automorphism that maps one onto the other [**?**]. So, the centrality of the nodes are the same.

First consider the **coverage centrality**, and lets count how many pairs of nodes

have a shortest path that pass the node $(0, \ldots, 0)$. Note that $a, b \in \{0,1\}^r$ have a shortest path that passes the origin if and only if $\forall i \in 1, \ldots, r : a_i b_i = 0$. To count the number of such pairs, we have to first choose a subset $I \subseteq \{1, \ldots, r\}$ of the bits that are non-zero either in a or b, in $\binom{r}{|I|}$ ways, and partition the bits of I between a and b (in $2^{|I|}$ ways). Therefore, the number of $(a, b) \in V^2$ pairs that their shortest path passes the node $(0, \ldots, 0)$ is

$$\sum_{i=0}^{r} \binom{r}{i} 2^i = (1+2)^r = 3^{\log(n)} = n^{\log(3)} = o(n^2).$$

So, the maximum coverage centrality of a node is at most $n^{\log(3)}$ (since we counted the endpoints as well, but should not have). Now by submodularity of the coverage centrality we have $\texttt{OPT}_k \leq kn^{\log(3)} = O(n^{1+\log(3)}) = o(n^2)$.

Finally, since the betweenness centrality is no more than the coverage centrality, we have the similar result for betweenness centrality as well.

4.3 Experimental Results

In this section we present our experimental results. We first start by comparing `HEDGE` (our sampling based algorithm) with `EXHAUST` (the exhaustive algorithm defined in Sect. 5.4) and show that the centrality of `HEDGE`'s output is close to the centrality of `EXHAUST`'s output, with great speed-up. This part is done for 3 small graphs as `EXHAUST` cannot scale to larger graphs.

We then compare our sampling method with the method presented in [127]. We show that, although our method stores less per each hyper-edge, it does not lose its accuracy.

Equipped with our scalable algorithm, `HEDGE`, we will be able to focus on some of the interesting characteristics of the central nodes: (i) How does their centrality change over time in evolving graphs? (ii) How influential are they? and (iii) How does the size of the largest connected component change after removing them?

In our experiments, we assume the graphs are simple (no self-loop or parallel edge) but the edges can be directed. We used publicly available datasets in our experiments.[3] `HEDGE` is implemented in C++.

[3] `http://snap.stanford.edu` and `http://konect.uni-koblenz.de/networks/dblp_coauthor`

4.3.1 Accuracy and Time Efficiency

Table 4.1 shows the results of EXHAUST and HEDGE on three graphs for which we were able to run EXHAUST. The fact that EXHAUST is able to run only on networks of this scale indicates already the value of HEDGE's scalability. As we can see, HEDGE results in significant speedups and negligible loss of accuracy.

In Table 4.1 the centrality of the output sets and the speed up gained by HEDGE is given, and as shown, HEDGE gives a great speedup with almost the same quality (i.e., the centrality of the output) of EXHAUST. The centrality of the outputs are *scaled* by $\frac{1}{n(n-1)}$, where n is the number of nodes in each graph. Motivated by the result in Sect. 4.2 we run HEDGE using $k\log(n)/\epsilon^2$ hyper-edges for $\epsilon = 0.1$, and for each case, ten times (averages are reported). For sake of comparison, these experiments were executed using a single machine with Intel Xeon CPU at 2.83GHz and with 36GB RAM.

				Algorithms		
GRAPHS	#nodes	#edges	k	EXHAUST	HEDGE	speedup
ca-GrQd	5242	14496	10	0.242	0.241	2.616
			50	0.713	0.699	2.516
			100	0.974	0.951	2.217
p2p-Gnutella08	6301	20777	10	0.013	0.011	6.773
			50	0.036	0.035	6.478
			100	0.053	0.051	6.117
ca-HepTh	9877	25998	10	0.165	0.164	4.96
			50	0.498	0.497	4.729
			100	0.747	0.745	4.473

Table 4.1: HEDGE vs. EXHAUST: centralities and speedups.

4.3.2 Comparison against Yoshida's Algorithm [127]

We compare our method against Yoshida's algorithm (Y-ALG) [127] on four undirected graphs (as Y-ALG runs on undirected graphs). We use Yoshida's implementation which he kindly provided to us. Note that Yoshida's algorithm applies a different sampling method than ours: it is based on sampling random *s-t* pairs of nodes and assigning weights to *every* node that is on any *s-t* shortest path, whereas in our method we only pick one randomly chosen *s-t* shortest path with no weight on the nodes.

Y-ALG and HEDGE use $\frac{2\log(2n^3)}{\epsilon^2}$ and $\frac{k\log(n)}{\epsilon^2}$ samples, respectively, where n is the number of nodes in the graph, and we set $\epsilon = 0.1$. We also run a variation of our algorithm, $\texttt{HEDGE}_{=}$, which is essentially HEDGE but with $\frac{2\log(2n^3)}{\epsilon^2}$ samples. This allows a more fair comparison between the methods.

Table 4.2 shows the estimated centrality of the output sets, and the number of samples each algorithm uses. Surprisingly, Y-ALG does not outperform $\texttt{HEDGE}_{=}$, despite the fact that it maintains extra information. Finally, our proposed algorithm HEDGE is consistently better than the other two algorithms.

		Betw. Centrality			**# of Samples**		
GRAPHS	k	*Y-ALG*	*HEDGE*$_{=}$	*HEDGE*	*Y-ALG*	*HEDGE*$_{=}$	*HEDGE*
CA-GrQc	*10*	0.208	0.214	0.215	5278		8565
	50	0.484	0.483	0.49			42822
	100	0.569	0.568	0.577			85643
CA-HepTh	*10*	0.151	0.151	0.154	5658		9198
	50	0.403	0.4	0.409			45989
	100	0.534	0.533	0.547			91978
ego-Facebook	*10*	0.924	0.932	0.933	5121		8304
	50	0.959	0.957	0.959			41519
	100	0.962	0.96	0.964			83038
email-Enron	*10*	0.329	0.335	0.335	6445		10511
	50	0.644	0.646	0.65			52552
	100	0.754	0.756	0.762			105104

Table 4.2: Comparison against Y-ALG

4.3.3 Applications

For the next three experiments, we consider three more larger graphs that HEDGE can handle due to its scalability: email-Enron, loc-Brightkite, and soc-Epinion1, with 36692-183831, 58228-214078, and 75879-508837 number of *nodes-edges*, respectively. These experiments are based on orders defined over the set of nodes as follows: we generate $100\log(n)/\epsilon^2$ hyper-edges, where n is the number of nodes, and $\epsilon = 0.25$. Then we order the nodes based on the order HEDGE picks the nodes.

For sake of comparison, we ran HEDGE using the coverage and κ-path (for $\kappa = 2$)

centralities, since both of them admit hyper-edge sampler as we showed in Sect. 4.1.2. Also, we considered a fourth centrality that we call triangle centrality, where the centrality of a set of nodes S equals to the number of triangles that intersect with S. For the triangle centrality, we run EXHAUST as computing this centrality is easy and scalable to large graphs.[4] All these experiments are run ten times, and we report the average values.

Time evolving networks.. Leskovec *et al.* studied empirically properties of time-evolving real-world networks [82]. In this section we investigate how BWC of the most central nodes changes as a function of time.

We study two temporal datasets, the DBLP [5] and Autonomous Systems (AS) datasets. We also generate stochastic Kronecker graphs on 2^i vertices for $i \in \{8, \ldots, 20\}$, using $\left(\begin{smallmatrix} 0.9 & 0.5 \\ 0.5 & 0.2 \end{smallmatrix}\right)$ as the core-periphery seed matrix. We assume that the i-th time snapshot for Kronecker graphs corresponds to 2^i vertices, for $i = 8, \ldots, 20$. Note that in these evolving sets, the number of nodes also increases along with new edges. Also, note that the main difference between DBLP and Autonomous Systems is that for DBLP edges and nodes only can be added, where in Autonomous Systems nodes and edges can be increased and decreased.

The results are plotted in logarithmic scale (Fig. 4.2), and as shown, we observe that the centrality of the highly central set of nodes increases. Also, we observe that the model of stochastic Kronecker graphs behaves similar to the real-world evolving networks with respect to these parameters.

Influence maximization. We consider the Independent-Cascade model [68], where each edge has the probability 0.01 of being active. For computing and maximizing the influence , we consider the algorithm of [13] using 10^6 number of samples (called hyper-edge but defined differently). We compute the influence of output of HEDGE with output of [13]. As shown in Table 4.3, and as we observe, the central nodes also have high influence, which shows a great correlation between being highly central and highly influential. It is worth outlining that our main point is to show that our proposed algorithm can be used to scale heuristic uses of BWC.

[4] EXHAUST for the triangle centrality, at every iteration simply chooses a node that is incident with more number of new triangles.

[5] Timestamps are in Unix time and can be negative.

		METHODS				
GRAPHS	k	*IM*	*betw.*	*cov.*	κ-*path*	*tri.*
CA-GrQc	*10*	19.12	13.67	14.93	14.10	18.48
	50	76.65	67.28	67.44	65.06	69.30
	100	141.33	126.76	126.66	124.51	124.06
CA-HepTh	*10*	17.33	15.61	15.58	14.63	12.98
	50	77.88	70.53	69.95	67.80	63.95
	100	147.75	133.45	133.24	130.41	127.52
p2p-Gnutella08	*10*	19.61	13.05	13.71	10.39	18.06
	50	83.64	60.58	61.73	51.57	74.19
	100	148.86	118.27	118.76	103.58	132.04
email-Enron	*10*	461.84	458.70	450.34	455.25	451.53
	50	719.86	703.08	695.81	699.74	681.05
	100	887.63	863.66	858.39	865.76	830.15
loc-Brightkite	*10*	184.40	162.64	160.35	163.16	145.19
	50	402.85	372.64	360.64	366.28	330.45
	100	563.13	521.18	508.59	512.77	445.11
soc-Epinion1	*10*	343.89	81.57	111.47	14.43	311.74
	50	846.18	300.88	282.88	72.90	778.56
	100	1161.45	463.04	457.29	133.20	1062.99

Table 4.3: Comparing the *influence* of influential nodes (IM) and central nodes obtained by different centrality methods.

Graph attacks. It is a well-known fact that scale-free networks are robust to random failures but vulnerable to targeted attacks [6]. Some of the most efficient strategies for attacking graph connectivity is based on removing iteratively the most central vertex in the graph [58]. Such sequential attacks are central in studying the robustness of the Internet and biological networks such as protein-protein interaction networks [62].

We remove the nodes one-by-one (according to the order induced by these centralities and picked by `HEDGE`) and measure the size of the largest connected component. The results are plotted in Fig. 4.3. Our observation is that all the sizes of largest connected components decline significantly (almost linearly in size of S), which is compatible with our intuition of centralities. We also find that κ-path and triangle centralities can be more effective at destroying the connectivity.

4.4 Related Work

Centrality measures. There exists a wide variety of centrality measures: degree centrality, Pagerank [101], HITS [72], Salsa [80], closeness centrality [11], harmonic centrality [12], betweenness centrality [42], random walk betweenness centrality [97], coverage centrality [127], κ-path centrality [5], Katz centrality [67], rumor centrality [112] are some of the important centrality measures. Boldi and Vigna proposed an axiomatic study of centrality measures [12]. In general, choosing a good centrality measure is application dependent [46]. In the following we discuss in further detail the centrality measure of our focus, the betweenness centrality.

Betweenness centrality (BWC). is a fundamental measure in network analysis. The betweenness centrality index is attributed to Freeman [42]. BWC has been used in a wide variety of graph mining applications. For instance, Girvan and Newman use BWC to find communities in social and biological networks [47]. In a similar spirit, Iyer *et al.* use BWC to attack the connectivity of networks by iteratively removing the most central vertices [62].

The fastest known exact algorithm for computing BWC exactly requires $O(mn)$ time in unweighted, and $O(nm + n^2 \log m)$ for weighted graphs [16, 40, 109]. There exist randomized algorithms [8, 15, 105] which provide either additive error or multiplicative error guarantees with high probability.

For CMP, the state-of-the-art algorithm [127] (and the only scalable proposed algorithm based on sampling) provides a mixed error guarantee, combining additive and multiplicative error. Specifically, this algorithm provides a solution whose centrality is at least $(1 - \frac{1}{e})\texttt{OPT}_k - \epsilon n^2$, by sampling $O(\log n/\epsilon^2)$ *hyper-edges*, where each hyper-edge is a set of all nodes on any shortest path between two random nodes with some assigned weights.

As we mentioned before, CMP is APX-complete, and the best algorithm (i.e. classic greedy algorithm for maximizing the monotone-submodular functions) using *exact* computations of BWC provides $(1-1/e)$-approximation [41]. We call this greedy algorithm by `EXHAUST` algorithm, and works as follows: It starts with an empty set S. Then, at any round, it selects a node u that maximizes the *adaptive betweenness*

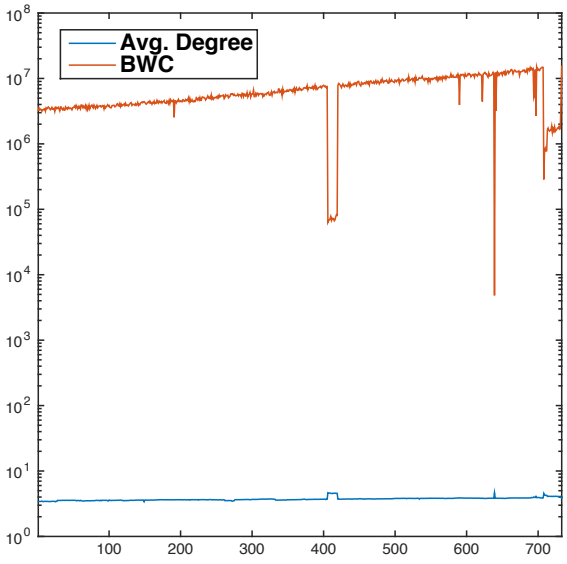
centrality (A-BWC) of u according to S defined as

$$B(u|S) = \sum_{(s,t),s,t\neq u} \frac{\sigma_{s,t}(u|S)}{\sigma_{s,t}},$$

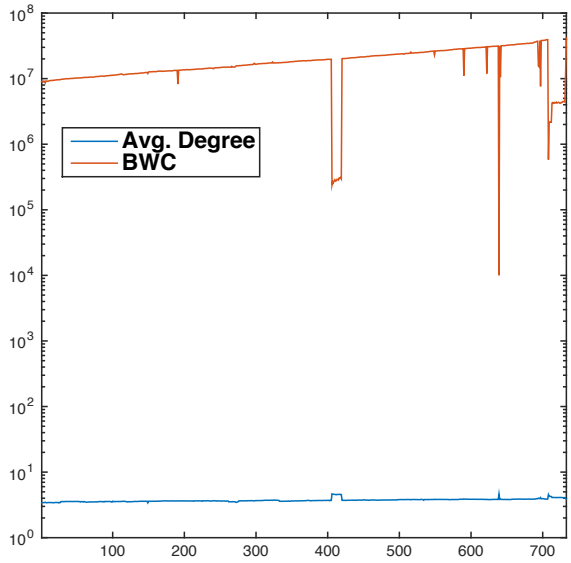
where $\sigma_{st}(u|S)$ is the number of s-t shortest paths that do not pass through any node in S and have u as an internal node. The algorithm adds u to S and stops when $|S| = k$.

Note that the A-BWC is intimately connected to the BWC through the following well-known formula [127]:

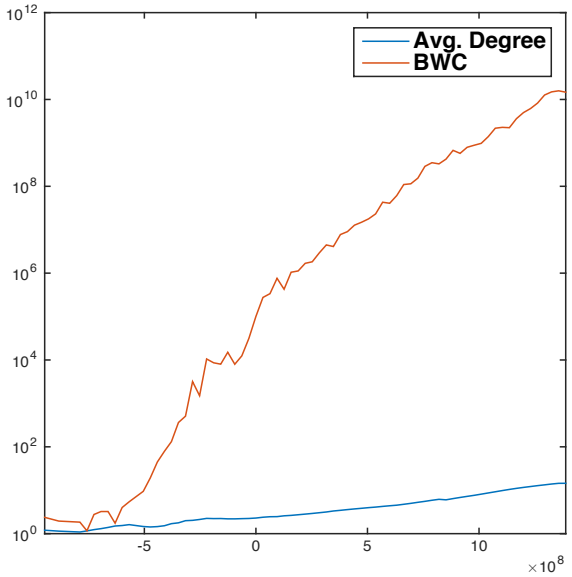
$$B(S \cup \{u\}) = B(S) + B(u|S).$$



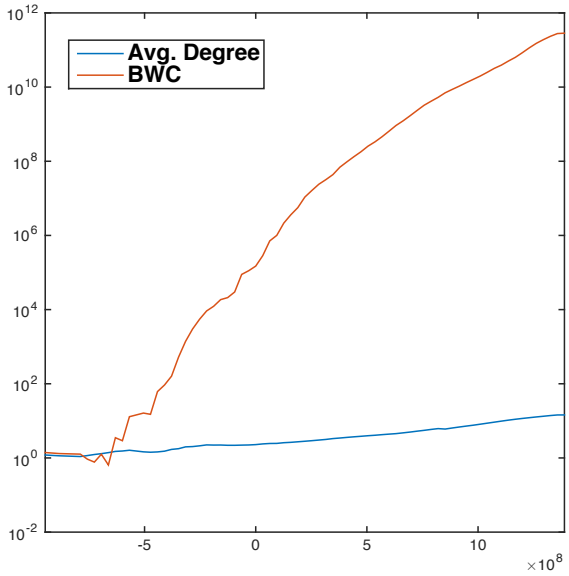
(a) AS: $k = 1$



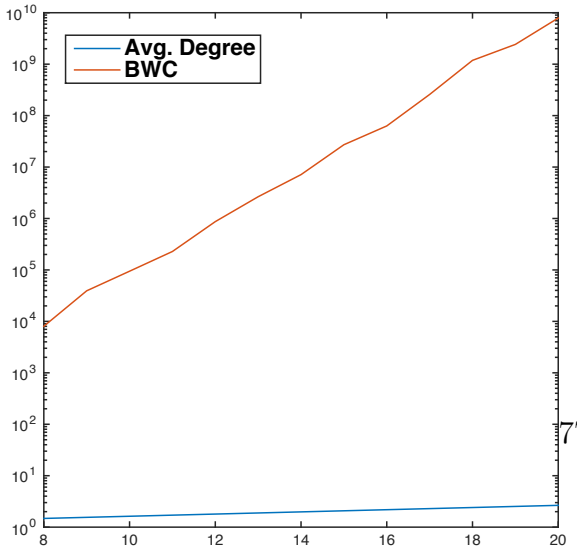
(b) AS: $k = 50$



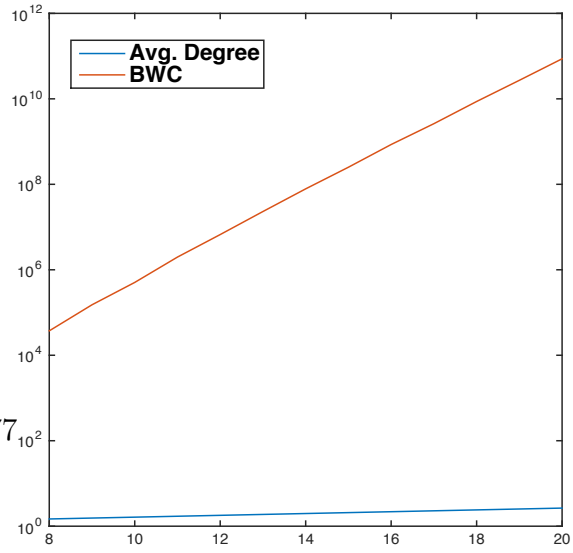
(c) DBLP: $k = 1$



(d) DBLP: $k = 50$



(e) KG: $k = 1$



(f) KG: $k = 50$

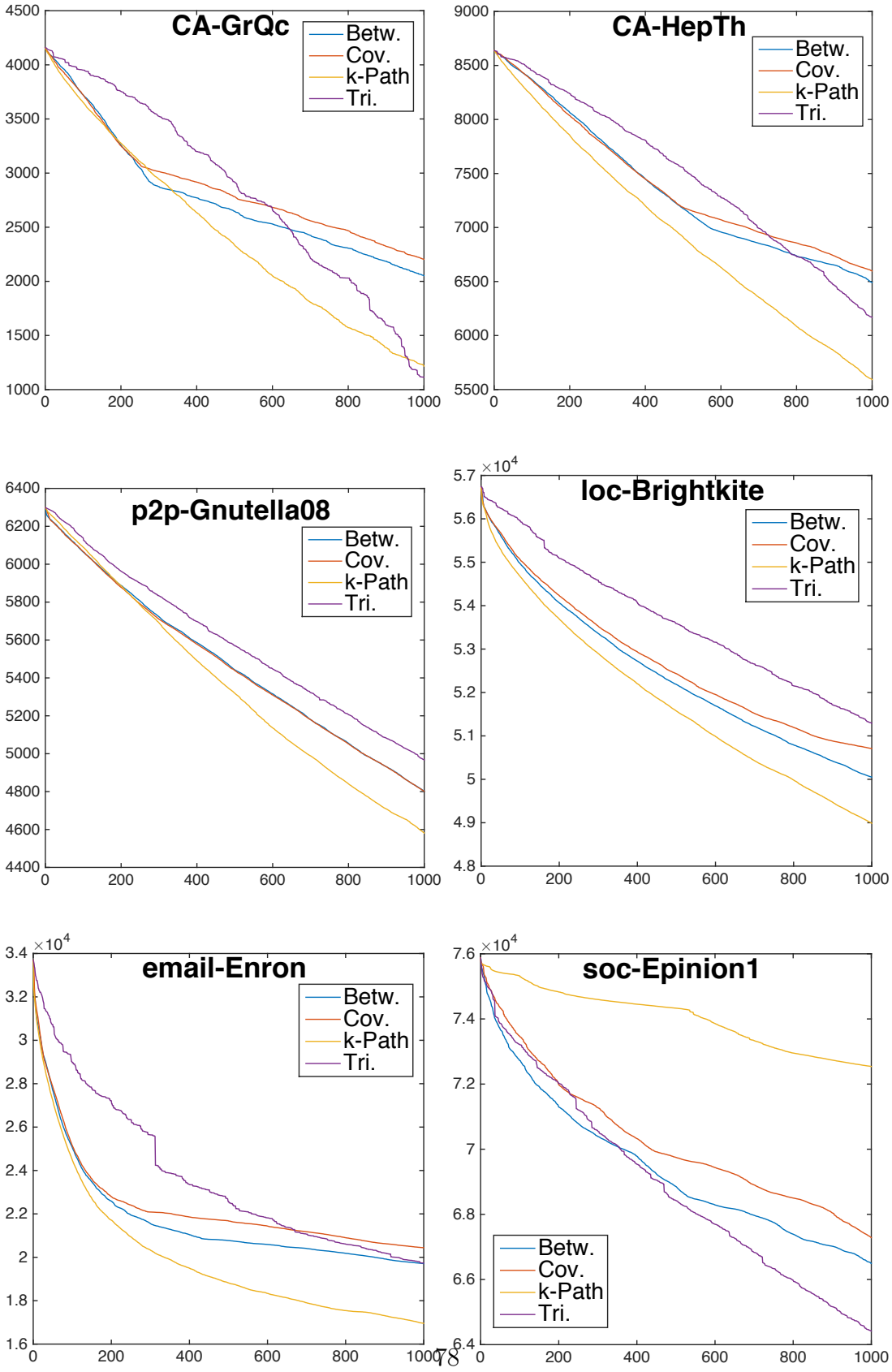


Figure 4.3: The size of the largest connected component, as we remove the first 1000 nodes in the order induced by centralities.

Part III

Influence Estimation

Chapter 5

Real-Time Targeted-Influence Queries over Large Graphs

In this chapter, we introduce the *Targeted-Influence Problem* (TIP), that aims to estimate the influence of a group of nodes (people) in the graph of social network over another group. In TIP we are given, in the *preprocessing* stage, an arbitrary graph $\mathcal{G}$ and a model of influence over $\mathcal{G}$. Then, in the *query* phase, a sequence of queries arrives for arbitrary pairs (S_i, T_i) of subsets of the nodes where S_i and T_i are the set of seed nodes and target nodes in the query, respectively. Our goal in the query stage is to estimate the influence of the set S_i over the set T_i, i.e. the expected number of influenced nodes in T_i if the nodes in S_i spread the information. To do so, as efficiently as possible, we are allowed to preprocess the graph $\mathcal{G}$ in the preprocessing stage, *before* knowing which queries will be issued during the query stage. Also, note that having access to a TIP-solver oracle is sufficient to (approximately) find the most influential node among the accessible nodes as in [130].

To the best of our knowledge, TIP is a new problem and has not been addressed before. Notice that this is a significant departure from recent works [52, 130] that proposed settings for *Influence Maximization* problem in the target context where an expensive computation is run to maximize influence over a given fixed target set. In practical settings however, the social network operator needs to optimize multiple campaigns with different targets and seed sets at the same time. For this reason our aim is instead, to allow high-quality answers in the query stage, for many distinct

(S_i, T_i) queries after an efficient preprocessing.

Naive approach. In most influence-propagation models, the TIP problem can be readily solved by standard Monte Carlo (MC) simulations: simply run (enough) MC simulations of the influence process, and measure how many nodes in T_i are influenced when the information starts propagating from the seed nodes in S_i for each query $(T_i; S_i)$. However, as we discussed above, in real settings numerous advertisers operate on the social network at any given time and each of them may have many ongoing advertisement campaigns with different target sets and potential seed sets to evaluate. Determining the best strategy even for a single campaign might involve performing numerous influence queries. As each influence query requires multiple MC simulations over a potentially huge graph, this naive approach is not able to handle many advertisers over large networks with billions of nodes and edges. For an online targeted-influence query system to be practically useful, we would like to be able to answer such queries in real-time (say in a few seconds per query). That means that we need to answer such queries in a time that is significantly shorter than what is necessary just to read the graph in input.

In this thesis we present the first real-time algorithm for TIP on large scale graphs. We present algorithms for both directed and undirected graphs: after a feasible preprocessing, our algorithms are guaranteed to answer each $(S; T)$-query in time $\tilde{O}(|S| + |T|)$, for general undirected graphs, and directed graphs under some assumptions, that are backed with experiments.[1] Note that in the naive approach, the running time at the query stage is $\Theta(|V| + |E|)$ required for just a single MC simulation. We also provide precise theoretical guarantees on the approximation error of this algorithm.

Our experiments show that our algorithms are able to answer influence queries quickly with high accuracy. We achieve several orders of magnitude speedups over the naive approach of Monte Carlo simulation while preserving good accuracy in both directed and undirected graphs.

[1] Here, $\tilde{O}(\cdot)$ notation hides the lower order terms.

5.1 Notations and Problem Definition

Throughout this paper, we assume that we are given an arbitrary graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ as input. To model the influence of a seed set $S \subseteq \mathcal{V}$ over a target set $T \subseteq \mathcal{V}$, we introduce the *Snapshot* model. In Sect. 5.2.5 we prove that the Snapshot model generalizes the Triggering model that in turn is a generalization of the classic and widely used Independent-Cascade and Linear-Threshold models [68]. Our analysis holds for this more general model, and thus, is valid for both Independent-Cascade and Linear-Threshold models. Informally, the Snapshot model is represented by an arbitrary distribution $\mathcal{M}$ over the subgraphs of $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. Each sample $G = (\mathcal{V}, E)$ (subgraph of $\mathcal{G}$ with the same set of nodes) from the distribution is a realization of the diffusion process over $\mathcal{G}$, where the edges in E represent the active influence edges in that instance. Given one realization G, a node $u \in S$ influences all the other nodes to which it is connected to by a (directed) path in G, if S is the set of seeds:

Definition 14 (Snapshot & Influence)**.** *The **Snapshot** model for a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, is a distribution $\mathcal{M}$ over all subgraphs of $\mathcal{G}$, and by $G \sim \mathcal{M}$ we mean G is a subgraph of $\mathcal{G}$ that is selected randomly via $\mathcal{M}$. We say an edge $e \in \mathcal{E}$ is **activated** in $G = (\mathcal{V}, E) \sim \mathcal{M}$, if $e \in E$. The **influence** of a seed set S over a target set T, denoted by $\mathsf{Inf}(S;T)$, is defined as*

$$\mathsf{Inf}(S;T) = \mathbb{E}_{G\sim\mathcal{M}}\left[|I_G(S) \cap T|\right],$$

where G is a random subgraph of $\mathcal{G}$ sampled according to $\mathcal{M}$ and $I_G(S)$ is the set of all reachable nodes from S in graph G. Therefore, $\mathsf{Inf}(S;T)$ is the expected number of nodes in T that can be reached from S in a random graph sampled via $\mathcal{M}$.

In this paper, we study the problem of estimating the influence of seed sets over target sets in real-time, after a preprocessing of the data to expedite the estimation of the targeted-influence. Our methods for TIP has two stages: (1) preprocessing, and (2) query stages. Obviously, in order to work with large datasets, our solution has to be scalable at the preprocessing stage, and very fast at the query stage. Formally, our goal is as follows:

Definition 15 (Targeted-Influence Problem (TIP))**.** *By preprocessing the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ in time $\tilde{O}(|\mathcal{V}| + |\mathcal{E}|)$ and using $\tilde{O}(|V|)$ space, provide an estimation of $\mathsf{Inf}(S;T)$*

*for every seed and target sets, in time $\tilde{O}(|S| + |T|)$. We call a query of a seed set S and a target set T, an $(S;T)$-**query**.*

Our Contributions. In this paper, (1) we introduce the Targeted-Influence problem, (2) we give efficient solutions to the TIP problem, both for undirected and directed graphs with theoretical guarantees, and (3) we provide extensive experiments on real world networks, and show the efficiency and accuracy of our methods.

Paper Organization. Sect. 5.4 reviews the related work. In Sect. 5.2 we provide our algorithms and theoretical results for (i) efficient preprocessing, and (ii) estimating the targeted-influence, both for undirected and directed graphs. Finally, in Sect. 5.3 we conclude by applying our proposed algorithms on real world networks.

5.2 Method

We start this section by discussing how we can estimate the targeted-influences in general settings, using Monte Carlo methods. We then introduce our algorithms.

5.2.1 Estimating Targeted-Influence Using Monte Carlo Methods

Recall that we assume the Snapshot model and denote by $\mathcal{M}$ its distribution. Also, recall that by an $(S;T)$-query we mean the query for a pair of seed set S and a target set T. By definition, the influence of S over T is $\mathsf{Inf}(S;T) = \mathbb{E}_{G\sim\mathcal{M}}[|I_G(S) \cap T|]$ which is an expectation that can be approximated using Monte Carlo simulation: given a sample of ℓ graphs $\mathcal{S} = \{G_1, \ldots, G_\ell\}$ sampled independently from $\mathcal{M}$, the (experimental) influence of S over T according to $\mathcal{S}$ is

$$\mathsf{Inf}_{\mathcal{S}}(S;T) = \frac{1}{\ell}\sum_{i=1}^{\ell} |I_{G_i}(S) \cap T|.$$

Obviously, we have $\mathbb{E}_{\mathcal{S}}[\mathsf{Inf}_{\mathcal{S}}(S;T)] = \mathsf{Inf}(S;T)$.

In order to quantify the accuracy of an estimation for an $(S;T)$-query for a given sample $\mathcal{S}$ we introduce the following definition.

Definition 16 (Impact)**.** *The **impact** of a seed set S over a target set T is the average fraction of the number of nodes in T that get influenced by S, i.e., $\frac{\mathit{Inf}(S;T)}{|T|}$. We denote the impact of S over T by $\rho(S;T)$.*

The following theorem states the sample complexity for estimating the influence of an $(S;T)$-query.

Theorem 16. *Suppose $\mathcal{S} = \{G_1, \ldots, G_\ell\}$ is a sample of ℓ independent draws from $\mathcal{M}$. If $\ell \geq \frac{\ln(2/\delta)}{2\epsilon^2\rho^2}$, where $\epsilon, \delta \in (0,1)$ and $\rho = \rho(S;T)$, with probability at least $1-\delta$ we have*

$$|\mathit{Inf}(S;T) - \mathit{Inf}_{\mathcal{S}}(S;T)| < \epsilon \cdot \mathit{Inf}(S;T)\,.$$

Proof. For $i \in \{1, \ldots, \ell\}$, let $X_i = |I_{G_i}(S) \cap T|$. Obviously , $\mathsf{Inf}_{\mathcal{S}}(S;T) = \frac{X_1 + \ldots + X_\ell}{\ell}$, $0 \leq X_i \leq |T|$, and $\mathbb{E}[X_i] = \mathsf{Inf}(S;T)$. Therefore, by Hoeffding's inequality we have

$$\begin{aligned}\Pr\left[|\mathsf{Inf}_{\mathcal{S}}(S;T) - \mathsf{Inf}(S;T)| \geq \epsilon \cdot \mathsf{Inf}(S;T)\right] \\ \leq 2 \cdot \exp\left(-\frac{2\epsilon^2 \mathsf{Inf}(S;T)^2\,\ell}{|T|^2}\right),\end{aligned}$$

and by substituting for ℓ the proof is complete. □

Note that in practice the queried seed sets are expected to be close to the target set, if not directly connected. Therefore, if we restrict the queries to those with an impact greater than a constant threshold, the sample complexity for computing the influence will be $O(\ln(1/\delta)/\epsilon^2)$. Thus a **constant** number of samples suffices for any constant probability and error parameter.

5.2.2 Intuition of Our Method

We have shown that to answer any $(S;T)$-query, it is sufficient to draw a sample $\mathcal{S} = \{G_1, \ldots, G_\ell\}$ according to $\mathcal{M}$ and then efficiently compute $|I_{G_i}(S) \cap T|$. In particular, for any graph we can obtain a good approximation for queries with at least constant impact with $O(\ln(1/\delta)/\epsilon^2)$ samples. Moreover, for $O(\ln(|V|)/\epsilon^2)$ samples we can get a $(1+\epsilon)$-approximation, with high probably, for all queries in a polynomially long sequence of queries. Notice that computing $|I_{G_i}(S) \cap T|$ at the query phase might take $\tilde{O}(|V|+|E|)$ time, as opposed to $\tilde{O}(|S|+|T|)$. To circumvent this issue, we show

how one can preprocess each random graph $G_i \in \mathcal{S}$ during the preprocessing stage so that at the query time we can estimate $|I_{G_i}(S) \cap T|$ efficiently and accurately. To do so, we present different algorithms for preprocessing and answering the queries for undirected and directed graphs. Finally, note that even for queries with small impact, we have the following immediate corollary, using Hoeffding's inequality:

Corollary 9. *Suppose $\mathcal{S} = \{G_1, \ldots, G_\ell\}$ is a sample of ℓ independent draws from $\mathcal{M}$. If $\ell \geq \frac{\ln(2/\delta)}{2\epsilon^2}$, where $\epsilon, \delta \in (0,1)$, with probability at least $1-\delta$ we have*

$$|\mathit{Inf}(S;T) - \mathit{Inf}_{\mathcal{S}}(S;T)| < \epsilon \cdot |T|.$$

5.2.3 Undirected Graphs

In this section, we assume $\mathcal{G}$ is an undirected graph,[2] and $G \sim \mathcal{M}$ is one random subgraph of $\mathcal{M}$ that needs to be preprocessed. We first describe the algorithm, and then provide its analysis.

5.2.3.1 Algorithm

We preprocess G and compute $|I_G(S) \cap T|$ as following:

Preprocessing. We obtain the list of connected components of G and for each node u we store the ID of its connected component that we denote by $\mathtt{cc}(u)$.

Computing $|I_G(S) \cap T|$. Suppose we are given the $(S;T)$-query to process. Let $\mathtt{cc}(S) = \{\mathtt{cc}(u) \mid u \in S\}$. Note that

$$|I_G(S) \cap T| = |\{v \in T \mid \mathtt{cc}(v) \in \mathtt{cc}(S)\}|.$$

This is because S reaches to a node $v \in T$ if and only if there is a node $u \in S$ such that u, v are in the same connected component. Therefore, $|I_G(S) \cap T|$ can be computed in time $O(\min|S|, |T|)$ by using hash-maps, or in time $O(\log(|S|)|S| + \log(|T|)|T|)$ by sorting and merging the lists).

5.2.3.2 Analysis

Now, we analyze the running time of our algorithm for undirected graphs, when we sample $\mathcal{S} = \{G_1, \ldots, G_\ell\}$ of randomly chosen subgraphs $G_i \sim \mathcal{M}$.

[2]Equivalently in Independent-Cascade model, each pair of nodes in an edge, have the same probability of activating it.

Preprocessing The preprocessing time is due to (i) the time to sample a random subgraph $G = (V, E)$ according to $\mathcal{M}$ (ii) finding the connected components in time $O(|V| + |E|)$. Therefore, the preprocessing algorithm can be implemented in total time $O\left(\ell[|\mathcal{V}| + |\mathcal{E}| + T_{\mathcal{M}}]\right)$, where $T_{\mathcal{M}}$ is the time to generate a sample from $\mathcal{M}$. Note that our preprocessing stage requires $O(\ell \log(|\mathcal{V}|))$ bits per node, just to store the id of the connected component of the node in each sample. Also, if $\mathcal{M}$ is the Independent-Cascade model, $T_{\mathcal{M}} = |\mathcal{E}|$. Notice that the preprocessing algorithm can be easily parallelized in $O(diam(\mathcal{G}))$ rounds of Map-Reduce, in which the communication complexity of each round is $O(\ell(|\mathcal{V}| + |\mathcal{E}|))$, and each reducer receives an input of size $O(\ell\Delta)$, where Δ is the maximum degree in $\mathcal{G}$.

Answering queries To answer each $(S; T)$-query, we need to compute the average of $|I_{G_i}(S) \cap T|$, over the sampled subgraphs, and thus, the running time for answering the queries is $O(\ell \cdot \min\{|S|, |T|\})$ by using hash-maps, or $O(\ell(\log(|S|)|S| + \log(|T|)|T|))$ by sorting and list merging as above.

Considering our analysis, our algorithm satisfies the requirements of the TIP problem, and is an accurate and efficient solution.

5.2.4 Directed Graphs

Our algorithms for directed graphs follow the same approach of the undirected one, but here the problem is complicated by the need to compute reachability in directed graphs. For this case we provide some heuristic algorithms that allow a good experimental tradeoff between the space and time required in the preprocessing phase and the precision at query time. We present our basic technique (Algorithms 5 and 6) for preprocessing and query. Later improve these algorithms with additional technique.

Here, we assume $\mathcal{G}$ is a directed graph, and we provide an efficient way to preprocess each randomly sampled subgraph $G = (\mathcal{V}, E) \sim \mathcal{M}$ and estimate $|I_G(S) \cap T|$ for any $(S; T)$-query.

5.2.4.1 Algorithm

Our method for both preprocessing and query stages are provided in Algorithms 5 and 6, and here we give an overview of these algorithms.

Preprocessing. We apply the *Bloom filter* technique (defined below), which assigns a *reachability vector* (defined below) $R_G(u)$ of length w to each node $u \in \mathcal{V}$ (see Algorithm 5).

Computing $|I_G(S) \cap T|$. Using these vectors, for each pair u, v we can estimate whether $v \in I_G(u)$, i.e., v is reachable from u or not (see Algorithm 6), which provides us a biased estimator for $\mathsf{Inf}(S;T)$.

Without loss of generality, assume $\mathcal{V} = [n]$.[3] For the directed graph G, let $\mathcal{C}(G) = \{C_1, \ldots, C_r\}$ be the partition of G into its strongly connected components (SCC).[4] Also, for each node $u \in [n]$ we denote the strongly connected component that includes u by $\mathcal{C}(u)$. Note that by grouping the nodes in each C_i, we obtain a directed-acyclic-graph (DAG), denoted by $G[\mathcal{C}]$, whose nodes are the strongly connected components in $\mathcal{C}(G)$, and C_i has an edge to C_j if there is a node in C_i that connects to a node in C_j via an edge in E. Also, without loss of generality, we always assume $C_1, \ldots, C_r$ is the topological order of C_i's according to $G[\mathcal{C}]$ (i.e. if there is a direct path from C_i to C_j with $i \neq j$ then $i < j$). We also denote the in-neighbors of C_i in $G[\mathcal{C}]$ by $N^-(C_i)$.

Definition 17 (Reachability Vector)**.** *Suppose k and w are positive integers, and let $\mathcal{H}_{k,w} = \{h_1, \ldots, h_k\}$ be a family of k independent random hash functions, such that for $i \in \{1, \ldots, k\}$ we have $h_i : \mathcal{C}(G) \to [w]$. For $C \in \mathcal{C}(G)$ define $\mathcal{H}_{k,w}(C)$ as a binary vector of length w whose i-th entry is 1 if and only if for some j we have $h_j(C) = i$. Now, the **reachability vector** of node u, denoted by $R_G(u)$ is*

$$R_G(u) = \bigvee_{v \in I_G(u)} \mathcal{H}_{k,w}(\mathcal{C}(v)),$$

where $\vee$ is the bit-wise logical `OR` operation.

Note that reachability vectors are in fact Bloom filters that verify if a strongly connected component is reachable from a given node. We later describe the tradeoff between the values of k and w and the precision of the algorithm (See in Lemma 17 and Theorem 17). Finally, in Algorithm 6 we show how we can estimate $|I_G(S) \cap T|$

[3] We denote the set $\{1, \ldots, n\}$ by $[n]$.

[4] Note that finding the partition $\mathcal{C}(G)$ can be done efficiently in time $O(|V| + |E|)$ using Tarjan's algorithm [122].

for an $(S;T)$-query. For notational convenience in Algorithm 6, we denote $\boldsymbol{x} \leq \boldsymbol{y}$ for two real vectors of the same length, if $\boldsymbol{x}$ is smaller than $\boldsymbol{y}$ entry-wise.

Algorithm 5: `PreD`(G, k, w)

input : Directed graph G, positive integers k and w.
output: Hash functions and reachability vectors.

begin
  $\mathcal{C}(G) = \{C_1, \ldots, C_r\}$ `//SCC, ordered topologically`
  $\mathcal{H} \leftarrow \mathcal{H}_{k,w}$ `//Sample random hash functions`
  for $i \leftarrow 1$ **to** r **do**
    $B(C_i) \leftarrow \mathcal{H}(C_i)$
  end
  for $j \leftarrow r$ **to** 1 **do**
    for $C_i \in N^-(C_j)$ **do**
      $B(C_i) \leftarrow B(C_i) \vee B(C_j)$
    end
  end
  for $u \leftarrow 1$ **to** n **do**
    $R_G(u) \leftarrow B(\mathcal{C}(u))$
  end
  return $\mathcal{H}$ *and* $(R_G(u))_{u\in[n]}$
end

5.2.4.2 Analysis

In this section we analyze our method for estimating $|I_G(S) \cap T|$ for an $(S;T)$-query. Let $\alpha(S)$ be the number of reachable strongly connected components from the nodes in S, i.e., $\alpha(S) = \{i \mid S \rightsquigarrow C_i\}$, where by $\rightsquigarrow$ we mean existence of a path from S to C_i in G. Similarly, we use $S \rightsquigarrow v$ for a node v if v is reachable from a node in S. We first have the following lemma:

Lemma 17. *Consider U and $\mathcal{H}$ as defined in Algorithm 6. If $w > \frac{\log(1/\delta)}{\log(1/0.7865)}\alpha(S)$ and $k = \frac{\log(2)w}{2\alpha(S)}$, for $\delta \in (0,1)$, and v is a node in G:*

(a) if $S \rightsquigarrow v$ then $\mathcal{H}(\mathcal{C}(v)) \leq U$; and

(b) if $S \not\rightsquigarrow v$, with probability at least $1-\delta$, $\mathcal{H}(\mathcal{C}(v)) \not\leq U$.

Proof. For part (a), note that if $S \rightsquigarrow v$, then there exists a node $u \in S$ such that $u \rightsquigarrow v$. First, $\mathcal{H}(\mathcal{C}(v)) \leq R_G(u)$, since $R_G(u)$ is the logical `OR` of some vectors including $\mathcal{H}(\mathcal{C}(v))$ in Algorithm 5. Secondly, $R_G(u) \leq U$ as U is the logical `OR` of some vectors including $R_G(u)$ in Algorithm 6. So, $\mathcal{H}(\mathcal{C}(v)) \leq U$.

For part (b), note that the probability that a single bit being zero in U is $(1 - 1/w)^{k\alpha(S)}$, since $\mathcal{H}$ is a family of k-independent functions. Therefore, the probability of a false positive (i.e. $\mathcal{H}(\mathcal{C}(v)) \leq U$) is

$$\begin{aligned}\Pr\left[\mathcal{H}(\mathcal{C}(v)) \leq U\right] &= \left(1 - \left(1 - \frac{1}{w}\right)^{k\alpha(S)}\right)^k \\ &\leq \left(1 - e^{-\frac{2k\alpha(S)}{w}}\right)^k \leq \left(1 - \frac{1}{2}\right)^k \leq (0.7865)^{\frac{w}{\alpha(S)}} < \delta,\end{aligned}$$

where we used the fact that $1 - x \geq e^{-2x}$ for $x \leq \frac{1}{2}$. □

Lemma 17, gives us the next Theorem:

Theorem 17. *Consider X and U as defined in Algorithm 6, and an $(S;T)$-query. If $k \geq \log_2(|T|/\delta)$ and $w \geq 2\alpha(S)$, for $\delta \in (0,1)$, we have*

(a) With probability at least $1 - \delta$, $X = |I_G(S) \cap T|$; and

(b) Always

$$|I_G(S) \cap T| \leq \mathbb{E}\left[X\right] \leq \left(1 + \frac{\delta}{\rho(S;T)}\right) \cdot |I_G(S) \cap T|.$$

Recall that $\rho(S;T)$ is the impact of S over T.

Proof. Part (a) is directly obtained from Lemma 17, and applying the union bound. For part (b) notice that

$$|I_G(S) \cap T| \leq X \leq T,$$

since (i) for every $v \in |I_G(S) \cap T|$ we have $\mathcal{H}(\mathcal{C}(v)) \leq U$, and thus, X is increased at least $|I_G(S) \cap T|$ times; and (ii) X is increased at most T times. Therefore,

$|I_G(S) \cap T| \leq \mathbb{E}[X]$, and

$$\begin{aligned}\mathbb{E}[X] &\leq \Pr[X = |I_G(S) \cap T|] \cdot |I_G(S) \cap T| \\ &\quad + \Pr[X > |I_G(S) \cap T|] \cdot |T| \\ &\leq |I_G(S) \cap T| + \delta \frac{|I_G(S) \cap T|}{\rho(S;T)} \\ &= \left(1 + \frac{\delta}{\rho(S;T)}\right) \cdot |I_G(S) \cap T|,\end{aligned}$$

where we used the fact that $\Pr[X = |I_G(S) \cap T|] \leq 1$, and the proof is complete. □

Algorithm 6: $\texttt{QueD}(S, T, \mathcal{H}, (R_G(u))_{u\in[n]})$

input : Seed set S, target set T, hash functions $\mathcal{H}$, and reachability vectors $(R_G(u))_{u\in[n]}$.
output: An estimate of $|I_G(S) \cap T|$.

```
begin
    U ← ⋁_{u∈S} R_G(u)
    X ← 0
    for v ∈ T do
        if H(C(v)) ≤ U then
            X ← X + 1
        end
    end
    return X
end
```

Theorem 17 shows that the number of bits for each node, sample pair needed to answer accurately a query is $w \geq 2\alpha(S)$. Notice that although the seed sets are in practice small sets, the $\alpha(S)$ can be a large quantity.

To reduce the memory required we introduce the following heuristic. In the preprocessing stage, we select λ strongly connected components (SCCs) with the highest out-degree in the $G[\mathcal{C}]$ DAG, i.e., SCCs with maximum number of SCCs they can reach with a directed edge. We let L to be the set of these SCCs, and call the elements of L, the **hubs** of the network. Thus, $|L| = \lambda$. We set λ to be a small constant, and in our experiments we used $\lambda = 10$.

Next, for each hub $l \in L$ (which is also a SCC) we execute two visits of the graph to obtain: 1) all the SCCs reached by l and 2) all the SCCs reaching l. The ids of

the SCCs reached by each $l' \in L$ are stored in a distinct hash set for constant query time. For each node $v \in V$, instead, we store a bitmap of λ bits representing whether the node reaches any given hub.

We finish the preprocessing stage as before (similar to Algorithm 5) but the accumulations of $B(C_i)$'s vectors stop at the components in L (which are essentially ignored from the graph).

The query algorithm, instead, proceeds as before with the following difference: we first obtain the union of all hubs reached by any nodes in S. Then in order to see whether $S \rightsquigarrow v$, for a node $v \in T$, we both check (i) $\mathcal{H}(\mathcal{C}(v)) \leq U$, and (ii) if for any hub reached by S, v belong to the set reached by that hub.

It is easy to see that using the hub trick can only decrease the false positives caused by the Bloom filter as reachability through hubs is computed exactly. It is also clear that the running time for preprocessing and query algorithms become $\tilde{O}\left(\lambda(|\mathcal{V}| + |E|)\right)$ and $\tilde{O}\left(\lambda(|S| + |T|)\right)$, respectively, and since λ is a constant we obtain the same complexity as before.

Finally, it is worth noting that this trick of using hubs significantly improves the space needed by the Bloom filters to obtain good results as we show in the experiments. This is due to the skewed reachability set size distribution of SCCs: not storing the descendants reachable from the top SSCs significantly reduces the maximum size that needs to be stored in any Bloom filter as our experiments show.

Therefore, for a general directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, we preprocess the graph and answer the queries as follows:

- `Preprocess`: generate a sample of $\mathcal{S} = \{G_1, \ldots, G_\ell\}$, where $G \sim \mathcal{M}$. Then, preprocess each G_i using using the hub heuristic (for better performance).
- `Query`: For an $(S;T)$-query, return $\frac{1}{\ell}\sum_{i=1}^{\ell} |I_{G_i}(S) \cap T|$, using the hub heuristic.

Finally we conclude this section by computing the running time and the space complexity of our approach.

Running time and space complexity The preprocessing algorithm can be implemented in total time $O(\ell(kw + \lambda) \cdot (|\mathcal{V}| + |E|))$. The space for each of ℓ samples is $O(wr + |V|(\log_2(r) + \lambda))$ where r is the number of SCCs in the sample graph. Finally the query time is $O(|S|\ell wd\lambda + \ell k\lambda(|T|))$.

5.2.5 On Snapshot Model

In this short section, we explain why the Snapshot model generalizes the Triggering model [68] and thus the independent-Cascade and the Linear Threshold model. We first start by formally defining the Triggering model:

Definition 18 (Triggering model [68])**.** *Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a graph, and for each node v let $N(v)$ be the set of v's neighbors. In the* Triggering *model, each node v is assigned with a distribution T_v over subsets of $N(v)$, and when v is influenced, it influences a subset of its neighbors sampled according to T_v, independent of other nodes. We denote a Triggering model by the vector $(T_v)_{v\in\mathcal{V}}$.*

Note that in the Triggering model, the process in which a node v chooses a subset N' of its neighbors to influence, is equivalent to sampling the edges that connect v to the nodes in N'. Therefore, the triggering model $(T_v)_{v\in\mathcal{V}}$ imposes a distribution $\mathcal{M}$ over the subsets of $\mathcal{G}$, where each subset is obtained from $\mathcal{G}$ by sampling the edges according to the joint distribution of $(T_v)_{v\in\mathcal{V}}$, independently. Therefore, clearly, our model is more general than the Triggering model as it allows for arbitrary (and hence non-independent) joint distributions for the neighbors of each node.

5.3 Experiments

In this section we provide our experimental results. First, we introduce our datasets and the influence model that we use for the experiments and discuss some important properties for these datasets in the given model. Next, we evaluate the efficiency of the preprocessing algorithms and show the speed-up we obtain for query stage with respect to using the naive Monte Carlo method for each given $(S;T)$-query. Finally, we measure the accuracy of our estimation method. Each experiment was executed using a single machine with Intel Xeon cpu at 2.83GHz and with 12GB RAM.

5.3.1 Data and Model

For sake of experiments, we consider the widely used Independent-Cascade model, which we showed is a special case of the Snapshot model. We assume that the activation probability of each edge equal to p. We show the results for $p \in \{0.05, 0.1, 0.2\}$.

In Sect. 5.2, we presented 2 approaches: one for undirected, and one for directed graphs. We refer to $\mathcal{A}_U$ as the undirected graph algorithm of Sect. 5.2.3. This algorithm is very efficient and all the experiments using $\mathcal{A}_U$ are run in the main memory.

We refer to $\mathcal{A}_D$ as the directed graph algorithm using hubs (Sect. 5.2.4). This algorithm is more expensive compared to $\mathcal{A}_U$ for the complexity of handling reachability in directed graphs. In this case for our larger graphs (com-Youtube, wiki-Talk, soc-Pokec, soc-LiveJournal1) we store the preprocessed data output in files, and use them to answer the queries.

Datasets We used publicly available datasets.[5] See Table 5.1. The direction of each edge represents the flow of the influence. We also remove any self-loop or parallel-edge.

Datasets	Direction	#nodes	#edges
email-Enron	U	36692	183831
soc-Epinions1	D	75878	508837
email-EuAll	D	265006	418956
soc-Slashdot0902	D	82168	870161
com-dblp	U	317080	1049866
ego-twitter	D	81306	1768135
com-youtube	U	1134890	2987624
wiki-Talk	D	2394367	5021410
soc-Pokec	D	1632801	30622564
soc-LiveJournal1	D	4846606	68475391

Table 5.1: The datasets, direction on their edges (**U**ndirected or **D**irected), number of nodes and edges.

Density of $G[\mathcal{C}]$ As we discussed in Sect. 5.2.4, in $\mathcal{A}_D$, we sample several subgraphs of $G \sim \mathcal{M}$, and decompose each G into its strongly connected components (SCC). Note, each reachability question can be addressed in the graph $G[\mathcal{C}]$, the DAG induced by the strongly connected components of G. Therefore, a natural question to ask how dense (or sparse!) is the graph $G[\mathcal{C}]$, since sparser graphs are easier to store and to process for reachability questions. Density of a graph is defined as the number of

[5] `http://snap.stanford.edu/`

edges divided by the number of nodes. We show the results in Table 5.2. For each graph/model we sample 10 graphs $G \sim \mathcal{M}$, and the averages are reported. As shown, the graphs are highly sparse, which motivates our approach.

Datasets	$p = 0.05$	$p = 0.1$	$p = 0.2$
email-Enron	0.905	0.944	0.984
email-EuAll	0.955	0.961	0.964
com-dblp	0.813	0.943	1.033
com-youtube	0.849	0.89	0.934
soc-Epinions1	0.954	0.952	0.973
soc-Slashdot0902	0.987	0.999	1.012
ego-twitter	1.197	1.237	1.244
wiki-Talk	1.003	1.012	1.013
soc-Pokec	1.166	1.177	1.173
soc-LiveJournal1	1.03	1.055	1.048

Table 5.2: The density of the $G[\mathcal{C}]$ graphs, for $G \sim \mathcal{M}$.

Maximum Number of Descendants As we discussed in Sect. 5.2.4, the number of bits in the each reachability vector (denoted by w) necessary to obtain a multiplicative approximation is lower-bounded by a function of $\alpha(S)$, the number of SCCs that are reachable from S. The smaller the $\alpha(S)$ the smaller the memory required.

Here we evaluate our heuristic to reduce the amount of memory need using L hubs. we consider the following cases: (1) no hubs, i.e., $L = \emptyset$, and (2) when L is the t-top degree nodes in G for $t \in \{1, 2, 3, 4, 5, 10\}$. In Fig. 5.1 we present the maximum number of descendants of any node, using different number of hubs. Each number is the average for 10 sampled random subgraph $G \sim \mathcal{M}$. As shown, by using hubs, the number of direct descendants suddenly drops, which then, improves the space required dramatically. For space limitations, we provide the plot for $p = 0.2$. However for $p = 0.05$ and $p = 0.05$ we obtain very similar results.

5.3.2 Efficiency

In this section we provide the running time for preprocessing the graphs $G \sim \mathcal{M}$, and the speed-up we obtain for answer $(S; T)$-queries, i.e. estimating $\mathsf{Inf}(S; T)$, compared to running the standard Monte Carlo simulations to estimate $\mathsf{Inf}(S; T)$. For these

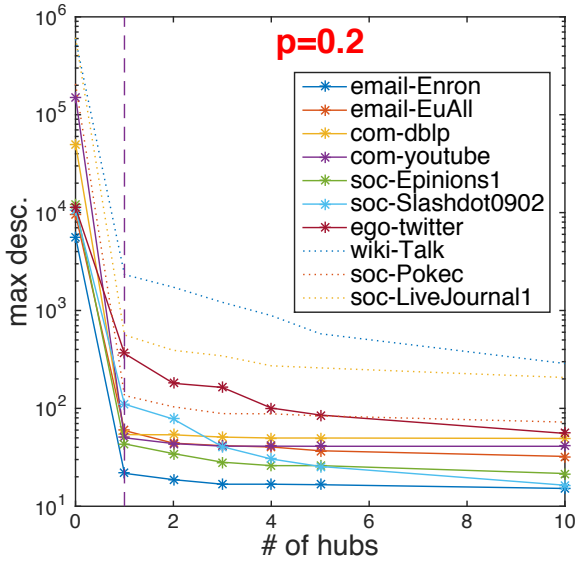


Figure 5.1: Maximum number of descendants with/without using hubs.

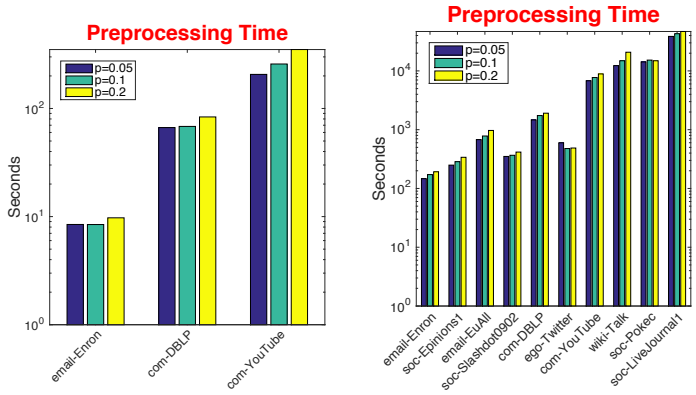
experiments we set the parameters as follows:

- $|L| = 10$: number of hubs,
- $k = 5$: number of hash functions,
- $w = 2^{14} = 16384$: number of bits for reachability vectors,
- $\ell = 50$: number of sampled random subgraphs $G \sim \mathcal{M}$,

5.3.2.0.1 Preprocessing In Figure 5.2 we show the running time for preprocessing the randomly sampled graphs $G \sim \mathcal{M}$: numbers are averaged over different samples of G.

5.3.2.0.2 Speed-ups in answering queries Here, we compare the time to answer an $(S;T)$-query using our approach T_1, to the running time of Monte Carlo simulations, T_2. In both cases we use the same number $\ell = 50$ of simulations. Here, by speed-up we mean the $\frac{T_2}{T_1}$ ratio.

For our experiments, we report the average results over 50 $(S;T)$-queries that are obtained using three different methods, in each case we set $|S| = |T| = 1000$. In the



Applying $\mathcal{A}_U$ approach Applying $\mathcal{A}_D$ approach

Figure 5.2: Preprocessing time for both approaches.

UNIF case both S and T are subsets of nodes chosen uniformly at random, among the nodes with non-zero out-degree and non-zero in-degree, respectively. In the **2-Hops** experiment S is a subset of nodes with non-zero out-degree, chosen uniformly at random. T is obtained by randomly choosing from the nodes that can be reached from S by using at most 2 edges (i.e., from 2-neighborhood of S). Finally, in the **DEG** case S and T are obtained by sampling the nodes with probability proportional to their out-degree and in-degree, respectively.

In Table 5.3 the time (seconds) to answer the queries with preprocessed data, and in Fig. 5.3 the speed up we obtain is presented. For each graph, the speed-ups are the average of 50 random $(S;T)$-queries. As shown, we obtain tremendous speed-up by using our preprocessing techniques, in both $\mathcal{A}_U$ and $\mathcal{A}_D$ approaches reducing the running time for a given query up to a 10^3 factor. Again, for lack of space, in Fig. 5.3 we present the results for UNIF only, as the other two methods give us very similar results.

5.3.3 Relative Errors

Finally in this section, we discuss the accuracy of our estimated influence of an $(S;T)$-query, by computing its relative error: For each $(S;T)$-query, we run 300 Monte Carlo simulation to estimate $\mathsf{Inf}(S;T)$ as the ground truth, and we denote this estimate by I_1. If I_2 is our estimate of $\mathsf{Inf}(S;T)$, the relative error is $\frac{|I_1-I_2|}{I_1}$.

The results are provided in Fig. 5.4. As shown, overall the relative errors are very

	UNIF			2-Hops			DEG		
Dataset $\backslash p =$	***0.05***	***0.1***	***0.2***	***0.05***	***0.1***	***0.2***	***0.05***	***0.1***	***0.2***
email-Enron(U)	0.03	0.03	0.03	0.04	0.03	0.03	0.03	0.02	0.02
com-DBLP(U)	0.41	0.05	0.03	0.17	0.05	0.05	0.05	0.04	0.03
com-YouTube(U)	1.51	0.06	0.05	0.05	0.05	0.05	1.19	0.05	0.06
soc-Epinions1(D)	2.87	2.94	3.02	2.87	2.93	3.01	3.24	2.9	2.78
email-EuAll(D)	2.78	2.65	2.78	2.83	2.7	2.77	3.31	2.95	3.06
soc-Slashdot0902(D)	2.93	2.84	2.88	3.02	2.86	2.91	3.72	2.73	2.62
ego-Twitter(D)	4.69	3.44	3.13	4.69	3.43	3.13	5.43	3.53	3
wiki-Talk(D)	10.08	10.61	11.21	10.19	10.62	10.35	11.62	13.1	11.32
soc-Pokec(D)	11.24	14.65	20.22	11.19	14.33	19.7	11.52	10.72	10.39
soc-LiveJournal1(D)	10.85	12.49	18.55	11.05	11.31	11.77	11.96	14.19	23.28

Table 5.3: The time (seconds) consumed by our algorithm for answering queries. Letters **U** and **D** correspond to $\mathcal{A}_U$ and $\mathcal{A}_D$, respectively.

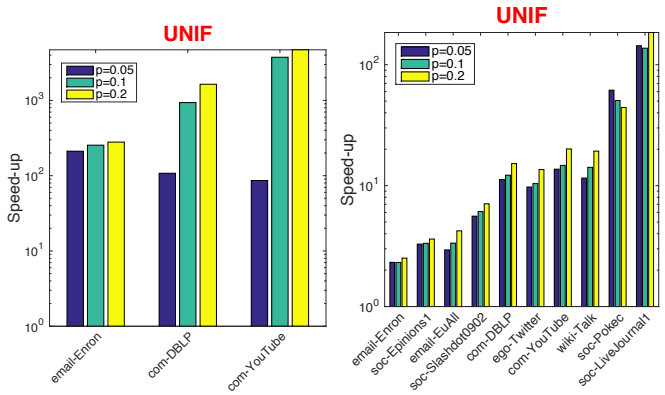


Figure 5.3: Speed-ups: Applying $\mathcal{A}_U$ and $\mathcal{A}_D$ methods.

small (true for all cases, show here only for UNIF, for space limitation), in particular, we observe that the higher the probability p, the lower the error observed. In almost all cases we obtain relative errors smaller than 10% and our relative errors can be as low as $< 0.1\%$

5.4 Related work

Kempe *et al.* [68] introduced the Influence Maximization (IM) problem. In the IM problem we want to find a subset of nodes, of a given size, that has the maximum influence over the graph. In our notation this can be defined as $\arg\max_{S:|S|=k} \mathsf{Inf}\,(S;\mathcal{V})$

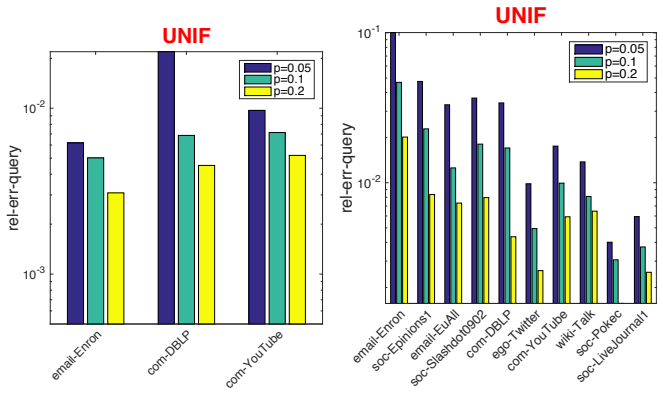


Figure 5.4: Relative error for the influence estimation in $\mathcal{A}_U$ and $\mathcal{A}_D$.

for a fixed positive integer k. They provided a constant approximation for both *Independent-Cascade* and *Linear Threshold models*. They also introduced the *Triggering* model which is a generalization of the Independent-Cascade and Linear Threshold models. Here, we introduce the *Snapshot* model, that is a generalization of Triggering model, and for which our theoretical analysis holds (see Sect. 5.2.5). Later, more efficient algorithms for the IM problem were provided based on sampling (random *hyper-edges*) [13, 120, 121]. These algorithms run in almost linear time. As mentioned before, our goal in this work is to efficiently *estimate* the targeted-influence and not maximizing it.

More related to our work, Zhou *et al.* [130] introduced the *constraint IM* problem, in which for a given positive integer k, a graph $G = (V, E)$, an accessible set $A \subseteq V$, and a target set $T \in V$, one has to find a set $S \subseteq A$ of size k that has the maximum influence over the set T. The model in [130] generalized a previous model known as *personalized influence* [52], where T is just a single node and $A = V \setminus T$. The algorithm proposed for the constraint IM in [130], is essentially the standard Monte Carlo method followed by the classic greedy algorithm. Our model/problem substantially departs from this model since, (i) we focus on computing the (targeted) influence for several seed and target sets query, and (ii) we consider a real-time setting where we need to answer each query, in the query stage, as fast as possible (i.e $\tilde{O}(|S| + |T|)$). Thus, we cannot afford to run the Monte Carlo method over the entire graph, for each query.

In [87] Lucier *et al.* studied the problem of estimating the influence of a given seed set, and proposed a parallel algorithm for this task. In contrast, our model considers more general influence queries, where we have to find the influence of a seed set over any other given target set. Also, we aim at algorithms that answer each influence query in a time that is linear in the size of the query, not the size of the graph.

Cohen *et al.* in [29], introduced a sketch-based algorithm for computing and maximizing the influence. Although this method can be applied for estimating the influence of a seed set S in the network, i.e. $\mathsf{Inf}(S;\mathcal{V})$, it cannot handle online $(S;T)$-queries for general target set T. This is because such sketch-based method allows estimating the cardinalities and not the set intersections (a significantly harder problem). A naive use of such sketches would requires storing (at the preprocessing time) a different sketch for each possible distinct target set T, where exponentially many of them are possible.

Other related work are *Topic-Aware model IM* [7], where the diffusion probabilities can be different for different items (for each item, an Independent-Cascade model is assumed), and the *Adaptive Seeding*, in which the budget for seeding the nodes is partitioned in different stages. Recently, Subbian *et al.* [116], studied the problem of finding the top-k influential nodes in graph streams. In another work, Cohen *et al.* [28] considered a different influence model based on the distances in the network. The IM problem has been also studied in continuous time diffusion models [38, 107]. In [119], Tang *et al.* considered a slightly different objective function, which is a linear combination of the influence and a *diversity* function, with the goal of maximizing influence and diversity of the influenced nodes at the same time.

Finally, our problem is closely related to the fundamental problems of *reachability* in directed graphs (RP) [30, 65, 126] and uncertain graphs (RUP) [26, 64, 71]: In RP, each query is a (u, v) pair of nodes, and we have to decide whether u can reach v. The RUP problem, for each (u, v) query, asks the *probability* that u can reach v, where edges can fail (i.e., be removed) with some probability.

Note that each method for RP can be extended to an approximate method for RUP, by generating samples from the graph, and process them individually as *certain* graphs. Also, note that to answer each $(S;T)$-query in our problem, one can naturally applies a reachability method on each (s, t) pair, where $s \in S$ and $t \in T$. However,

this causes the processing time to be at least $\Omega(\alpha \cdot |S| \cdot |T|)$, for some α that depends on the reachability method we use. But this violates our constraint for the running time in the query stage to be (almost) linear in size of the query, i.e, $\tilde{O}(|S| + |T|)$. Another important point to note is that in most of the reachability methods, either α is at least $\Omega(m)$, or the construction time is $O(mn)$, where n and m are the number of nodes and edges, respectively.

In an independent work, similar to our method [115] applied the Bloom filter method for answering reachability queries. However, our Bloom filter method is equipped with a *hub* technique that significantly improves the space needed by the Bloom filter to obtain good results as we show in the experiments.

Chapter 6

Conditional Influence Estimation

In this chapter, we study the problem of *Conditional Influence Estimation* (CIE), in which, given a source, a partial observation, and the network topology, our goal is to estimate the cascade size. To the best of our knowledge, this is the first time that the problem of cascade estimation under this more realistic assumption (i.e. partial observation) is introduced and studied, and the first work on (conditional) cascade estimation in discrete-time setting.

We first introduce the problem of *Conditional Influence Estimation* (CIE), for estimating cascade size with partial observation, in a discrete-time setting. We then provide two algorithms based on Markov Chain Monte-Carlo (MCMC) sampling, and show how they can improve upon the naive approach using rejection sampling.

6.1 Preliminaries and Problem Definition

In this section we provide the preliminary definitions and results, and formally define our problem, i.e., Conditional Influence Estimation. We first start by defining the model of cascade propagation we consider in this work, *Independent-Cascade* (IC) [68].

Definition 19 (Independent-Cascade (IC))**.** *Given a graph $G = (V, E)$ (directed or undirected) that represents a social network, the* ***Independent-Cascade*** *model assumes an* activation probability *function $p : E \to [0, 1]$ that assigns each edge $e \in E$ an* ***activation*** *probability $p(e)$. In IC model the progression of a cascade is as follows: when the cascade reaches (or starts) at node $u \in V$ it tries, only once, to pass each*

(out-going) edge $e = uv$, and reach the neighbor v, with probability $p(e)$.

Having the IC model, we now can define the influence of a source node:

Definition 20 (Influence)**.** *In a graph $G = (V, E)$, for a node $s \in V$, let $R(s)$ be the (random) set of nodes in a cascade initiated at node s. The* influence *of s, denoted by $\mathsf{Inf}(s)$, is the expected size of a cascade initiated at s, i.e., $\mathsf{Inf}(s) = \mathbb{E}[|R(s)|]$. Note that this is similar to Definition 14 when we consider the IC model, single node seed set, and the whole graph as the target set.*

The influence of a node s in a graph $G = (V, E)$ with activation probability $p : E \to [0, 1]$ can be defined equivalently: Let $H = (V, E)$ be a random subgraph of G that is obtained by sampling each edge e with probability $p(e)$. Now, if $R_H(s)$ is the set of reachable nodes from s in the subgraph H, it is easy to see that $\mathsf{Inf}(s) = \mathbb{E}_H[R_H(s)]$ (see [98]). When it is clear from the context, we may drop the subscript H from R_H or $\mathbb{E}_H$. Also, we denote this distribution over the subgraphs $H \subseteq G$ obtained by sampling the edges by $\mathcal{D}_p$.

In this work, we study the problem of estimating the conditional influence, where the influence/cascade is observed to have reached to a set of nodes, i.e., the only information we have about the cascade is the source node and some of the nodes that are in the cascade, and the goal is to estimate the average size of such cascade. The main problem of this chapter is defined formally as follows:

Definition 21 (Conditional Influence Estimation (CIE))**.** *Suppose $G = (V, E)$ is a graph, and p is its activation probability function. For a source $s \in V$ and an* ***observed set*** *of nodes $\mathcal{O} = \{t_1, \ldots, t_k\}$, the goal is to compute the expected size of a cascade initiated at s that reaches all the nodes in $\mathcal{O}$, which we call it by* ***the influence of*** *s* ***conditioned on*** *$\mathcal{O}$. The influence of s conditioned on $\mathcal{O}$ is denoted by and defined as*

$$\mathsf{Inf}(s; \mathcal{O}) = \mathbb{E}[|R(s)| \mid \mathcal{O} \subseteq R(s)].$$

We may also call the nodes in $\mathcal{O}$ as ***terminals****.*

In this section, we first discuss the naive Monte Carlo method based on rejection sampling for conditional influence estimation. We then discuss the challenges of such approach, that motivates our Markov Chain Monte-Carlo approach (see Sect. 6.2).

Suppose $G = (V, E)$ is a graph with activation probability function p, and $s \in V$ and $\mathcal{O} = \{t_1, \ldots, t_k\} \subseteq V$. Let X be a random variable that takes positive integer values, where

$$\Pr[X = k] = \Pr[R(s) = k \mid \mathcal{O} \subseteq R(s)]. \tag{6.1}$$

By the definition, $\mathbb{E}[X] = \mathsf{Inf}(s; \mathcal{O})$. Let $\mathcal{D}_X$ be the probability distribution of X over positive integers. We have the following lemma:

Lemma 18. *Suppose $X_1, \ldots, X_\ell$ are ℓ i.i.d draws from the distribution $\mathcal{D}_X$. Let $\bar{X} = \frac{X_1 + \ldots + X_\ell}{\ell}$. If $\ell \geq O(\log(n)/\epsilon^3)$, where $n = |V|$ for $\epsilon, \delta > 0$ we have*

$$Pr\left[|\bar{X} - \mathsf{Inf}(s; \mathcal{O}) \mid > \epsilon \cdot \mathsf{Inf}(s; \mathcal{O})\right] < 1 - \delta.$$

Proof. See [87, Lemma 1]. □

Lemma 18 shows that if we have access to a sampler that can generate i.i.d draws from the distribution $\mathcal{D}_X$, we would have an accurate estimator for the conditional influence with low sample complexity. This is the case for the (unconditional) influence function [87], since we can sample random subgraphs $H \sim \mathcal{D}_p$ (as described above) via the activation probability function and get $R_H(s)$ which has the same distribution as $R(s)$. However, for the conditional influence function, after sampling each random subgraph H we only can pick $|R_H(s)|$ if $\mathcal{O} \subseteq R_H(s)$, i.e., we apply **rejection sampling**. This becomes problematic when $\Pr[\mathcal{O} \subseteq R_H(s)]$ is a very small quantity, and makes the rejection sampling impractical. In practice, this is the case, and in Section 6.2 we develop methods to circumvent this issue, using Markov Chain Monte-Carlo sampling techniques.

Definition 22 (Conditional Distribution)**.** *We call the distribution defined by* (6.1) *the* conditional distribution*. Note that the goal is to design efficient sampler from the conditional distribution, as we later can apply the result from Lemma 18 to get accurate estimates.*

6.2 Algorithms

In this section we provide two methods for sampling from the conditional distribution. As mentioned in Section 6.1, our goal is to design efficient samplers from the

conditional distribution, which would enable us to have an accurate estimator (see Lemma 18). Here we provide two sampling techniques both based on the MCMC method: MACI, and MACI-HYBRID.

6.2.1 First Approach

Our first ***M**CMC based **a**lgorithm for estimating the **c**onditional **i**nfluence* function, MACI, is presented in Algorithm 7. In MACI, $C_H(s, \mathcal{O})$ denotes the set of edges in H such that, by removing any edge in $C_H(s, \mathcal{O})$, at least one node in $\mathcal{O}$ is not reachable from s using the edges in H. We also call $C_H(s, \mathcal{O})$ the *set of* $(s, \mathcal{O})$*-cut-edges* in H. MACI is consist of τ rounds of sampling an edge $e \in E$ uniformly at random, and if $e \notin C_H(s, \mathcal{O})$ we keep or add e to H with probability $p(e)$. It then returns the size of the number of reachable nodes from s in the current H, as a random draw from the conditional distribution. We can efficiently find the set of $(s, \mathcal{O})$-cut-edges (see Algorithm 8).

Note that in order to compare MACI to the rejection sampling we need to compare the expected number of steps that each method takes to generate a sample, i.e., to generate a valid sample in rejection sampling and the mixing time in MACI. There are examples in which the rejection sampling takes exponential steps between valid samples but MACI takes only (almost) linear time. In Figure 6.1, the rejection sampling takes $O\left(\left(\frac{1}{(1-(1-p)^2)}\right)^{m/2}\right)$ as it needs to pick at least one edge from each ring. However, MACI takes $O(m \log(m/\epsilon))$ number steps: consider the following coupling technique where for two states H and H' (where both are just subsets of edges) will always pick the same edge; however for each chain an edge is removed only when it does not disconnect the source node s from any terminal. It is easy to see that two chains take (1) constant time to couple inside each loop (2) $O(m \log(m/\epsilon))$ time to choose all the loops.

Algorithm 7: MACI

input : Graph $G = (V, E)$, source node $s \in V$, observation set $\mathcal{O} = \{t_1, \ldots, t_k\} \subseteq V$, and positive integer τ (for mixing time).

output: A random draw from the distribution defined in (6.1).

$H \leftarrow G$
if $\mathcal{O} \not\subseteq R_H(s)$ **then** ▷ check for zero influence
  return -1
end
for $i \leftarrow 1$ **to** τ **do**
  $e \leftarrow$ choose an edge uniformly at random from E
  if $e \notin C_H(s, \mathcal{O})$ **then** ▷ making sure the edge can be flipped
    $\alpha \leftarrow \mathsf{Unif}(0, 1)$
    if $\alpha < p(e)$ **then**
      $H \leftarrow H \cup \{e\}$
    end
    else
      $H \leftarrow H \setminus \{e\}$
    end
  end
end
return $|R_H(s)|$

Algorithm 8: Finding $(s, \mathcal{O})$-cut-edges

input : Graph $H = (V, E')$, source node $s \in V$, observation set $\mathcal{O} = \{t_1, \ldots, t_k\} \subseteq V$.

output: The set of $(s, \mathcal{O})$-cut-edges, denoted by $C_H(s, \mathcal{O})$

$R \leftarrow R_H(s)$ ▷ the set of reachable nodes from s in H
for $t \in \mathcal{O}$ **do**
  $C_t \leftarrow \emptyset$
  $P \leftarrow$ any path from s to t ▷ Note $C_H(s, \mathcal{O}) \subseteq P$
  $n \leftarrow \{t\}$ ▷ Super node
  while $s \notin n$ **do**
    $uv \leftarrow$ the edge on P connected to n for some $v \in n$
    $N^- \leftarrow$ the set of incoming edges to n from the nodes in R
    if $N^- = \{uv\}$ **then**
      $C_t \leftarrow C_t \cup \{uv\}$
      $n \leftarrow n \cup \{u\}$: contract uv to the super node
      remove self-loops at n
    end
    else
      if $\exists u'v' \in N^-$ *that* $u' \notin P$ **then**
        $n \leftarrow n \cup \{u'\}$: contract $u'v'$ to the super node
        remove self-loops at n
      end
      else
        $n \leftarrow n \cup \{u\}$: contract uv to the super node ▷ Note $uv \in P$
        remove self-loops at n
      end
    end
  end
  $C_H(s, \mathcal{O}) \leftarrow C_H(s, \mathcal{O}) \cup C_t$
end
$C_H(s, \mathcal{O}) \leftarrow \emptyset$
return $C_H(s, \mathcal{O})$

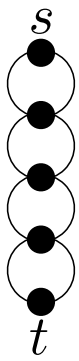


Figure 6.1: Here $\mathcal{O} = \{t\}$

6.2.2 Second Approach

The main idea behind this approach is to first sample some paths from the source to all the terminals, so that the connectivity condition is satisfied, and then sample the rest of the edges. We call this method Maci-Hybrid as it is built on two MCMC methods attached together. In this section, for simplicity we assume that the activation probability function is a constant function, i.e., $p(e) = p$ for all edges in G. The arguments can be modified for the general case.

In order to define Maci-Hybrid formally we need some notations: If $H \subseteq G$, $N_i(H)$ denotes the number of paths from s to t_i using only the edges in H, and $N(H) = N_1(H) \times \cdots \times N_k(H)$, assuming the obervation set $\mathcal{O}$ has k terminals. Also, let $m(H)$ denote the number of edges in H. Note that in order to sample a graph H from the conditional distribution, the probability of picking H should be proportional to $p^{m(H)} \cdot (1-p)^{m-m(H)}$.

Maci-Hybrid traverses a Markov chain $M_2 = (\Omega, P)$ where: (1) each state in Ω is a subgraph $H \subseteq G$ (with the same set of vertices) in which s reaches all the nodes in $\mathcal{O}$, and (2) Each state H is connected to all the other states, and makes jump by first choosing a state H' with probability $q(H')$, where $q(H') \propto N(H') \cdot p^{m(H')}(1-p)^{m-m(H')}$, and jumps with probability

$$P(H, H') = \min\left\{1, \frac{q(H) \cdot p^{m(H')}(1-p)^{m-m(H')}}{q(H') \cdot p^{m(H)}(1-p)^{m-m(H)}}\right\} = \min\left\{1, \frac{N(H)}{N(H')}\right\},$$

and a self-loop otherwise. Note that this is just the Metropolis-Hastings algorithm for sampling the subgraphs according to the conditional distribution (see [92]). So

the main challenge is to sample a random state H' with probability $q(H')$, in order to implement the jumps in M_2. To do so, we will utilize another MCMC sampler.

Now, let P_i be the set of all paths from s to t_i in graph G, and $Q(i_1, \ldots, i_k)$ be a set of edges that is obtained by picking the i_j-th path from P_j. Note that these paths are not necessarily disjoint. The algorithm for sampling subgraphs according to the probability function q is presented in Algorithm 9, which is based on `Sample-Paths` procedure that samples a $Q = Q(i_1, \ldots, i_k)$ proportional to $p^{|Q|}$. We have the following theorem:

Theorem 18. *A subgraph H returned by Algorithm 9 is sampled with probability $q(H)$.*

Proof. Note that there are $N(H)$ number of $(i_1, \ldots, i_k)$ vectors such that $Q(i_1, \ldots, i_k) \subseteq H$. Therefore, the probability of outputting a specific H is

$$\begin{aligned}\frac{1}{Z}\sum_{Q} p^{|Q|}p^{m(H)-|Q|}(1-p)^{m-m(H)} &= p^{m(H)}(1-p)^{m-m(H)}\left(\frac{N(H)}{Z}\right) \\ &\propto N(H)\cdot p^{m(H)}(1-p)^{m-m(H)},\end{aligned}$$

for a normalizing factor Z, and thus, it is equal to $q(H)$. □

Algorithm 9: Sampling H with $q(H)$

input : Graph $G = (V, E)$, source node $s \in V$, observation set $\mathcal{O} = \{t_1, \ldots, t_k\} \subseteq V$.

output: A random subgraph H sampled with probability $q(H)$.

```
H = Sample-Paths(G, s, O)
for e ∈ E \ Q do
    α ← Unif(0, 1)
    if α < p(e) then
        H ← H ∪ {e}
    end
end
return H
```

The `Sample-Paths` algorithm is an MCMC based procedure and provided in Algorithm 10, and we have the following theorem:

Theorem 19. *The* `Sample-Paths` *algorithm samples a* $Q = Q(i_1, \ldots, i_k)$ *with probability proportional to* $p^{|Q|}$.

Proof. Algorithm 10 implements the Metropolis-Hastings algorithm and its stationary distribution is the distribution we want to sample from. □

Algorithm 10: Sampling $Q(i_1, \ldots, i_k)$

input : Graph $G = (V, E)$, source node $s \in V$, observation set $\mathcal{O} = \{t_1, \ldots, t_k\} \subseteq V$, and mixing time τ.
output: $Q = Q(i_1, \ldots, i_k)$ sampled with probability $\propto p^{|Q|}$.

$Q \leftarrow \emptyset$
for $i \leftarrow 1$ **to** k **do**
  $Q_i \leftarrow$ a path (as a set of edges) from s to t_i chosen uniformly at random from P_i
  $Q \leftarrow Q \cup Q_i$
end
for $j \leftarrow 1$ **to** τ **do**
  $Q' \leftarrow \emptyset$
  for $i \leftarrow 1$ **to** k **do**
    $Q_i \leftarrow$ a path (as a set of edges) from s to t_i chosen uniformly at random from P_i
    $Q' \leftarrow Q' \cup Q_i$
  end
  $\alpha \leftarrow \mathsf{Unif}(0, 1)$
  if $\alpha < \min\left\{1, \frac{p^{|Q'|}}{p^{|Q|}}\right\}$ **then**
    $Q \leftarrow Q'$
  end
end
return Q

Since the in the Markov chains of Maci and `Sample-Paths` all the states are connected (complete graph), applying the coupling technique (simply choosing the same states at each time) we have:

- The mixing time for `Sample-Paths` is at most τ if $\max_{Q,Q'} p^{|Q'|-|Q|} < \tau$, where

the maximum is over any pair of states Q and Q'.

- The mixing time for MACI-HYBRID is at most τ if $\max_{H,H'} \frac{N(H)}{N(H')} < \tau$.

Therefore, if both of the above conditions hold for a small τ, MACI-HYBRID has a small mixing time.

Finally, Figure 6.2 shows an example for which MACI takes exponential mixing time, as it has to include both paths in order to be able to remove any of $(s, \mathcal{O})$-cut-edges (and rejection sampling takes exponential number of steps to generate a valid sample), whereas MACI-HYBRID takes only constant mixing time (since each path can be picked with probability at least 0.5 at each step).

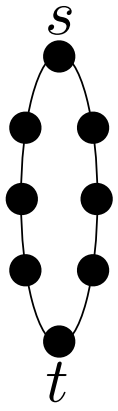


Figure 6.2: Here $\mathcal{O} = \{t\}$

6.3 Related Work

In this section, we provide an overview of the related works and topics. We first review the works in *influence estimation* and *maximization*. Next, we talk about the studies in estimating the *size of a cascade*. We then conclude this section with an overview of the work in *finding the source(s)* of a cascade.

Influence Estimation and Maximization. The Influence Maximization (IM) problem, whose goal is to find a subset of nodes, of a given size, with maximum influence was introduced in [68]. They provided a Monte-Carlo method for estimating and a constant approximation for maximizing the influence. Later, more efficient algorithms for estimation and maximizing the influence were provided based on *Reverse*

Reachable Sets (or Hyper-Edges) [13, 120, 121], or using *sketches* [29]. Also, in [87] a parallel algorithm for estimating the influence function was provided. There have been numerous variations of the IM problem, such as *Topic-Aware* model [7], *Adaptive Seeding* [110], *streaming* model [116], *Distance-Based Influence* [28], *Continuous Time Diffusion* [38, 107], or combination with *Diversity function* [119]. As mentioned before, in this work we consider the problem of estimating the influence of a source node conditioned on the event that it has influenced a given group of observed nodes, and the previous sampling techniques cannot be applied to our problem (CIE). Also note that, CIE deals with estimation and not maximization of the influence.

Cascade Size Estimation. The problem of Cascade Size Estimation has been studied in the context of *content popularity*, such as number of likes, reshares or views of a content in a social network or on a website. By applying point process models [44, 113, 129] studied the problem of predicting the final number of reshares (e.g. retweets) in online social networks, and [118] considers the number of views a a video receives on YouTube[1] or the number of votes a story gets on Digg.[2] The content popularity has been studied as a regression problem using feature-based methods to predict the popularity of memes [124], news articles [9, 70], and number of retweets in [77]. It also has been studied as a classification problem where the goal is to predict whether or not a content reaches a popularity threshold [9, 25, 53]. There have been some work to compute the probability of a content being boosted (such as being liked, reshared) in social media, that do not consider the final cascade size [2, 95, 102, 117, 128]. The most significant difference between our model and the previously studied models in the context of content popularity is that we assume we only have a limited observation of the cascade (a "given" subset of affected/infected nodes in the cascade), where the previous works assume that we have access to the full history of the cascade up to the current point. We also assume a discrete-time setting, and fully incorporate the underlying graph structure of the social network.

Lastly, there have been many studies on spotting the source(s) of a cascade [3, 24, 63, 78, 85, 103, 111] which is not directly related to this work, as we are given the source and part of the cascade, and the goal is to estimate the cascade size by

[1] `www.youtube.com`

[2] `www.digg.com`

computing the conditional influence of the source.

Bibliography

[1] Aaron B Adcock, Blair D Sullivan, and Michael W Mahoney. Tree decompositions and social graphs. *arXiv preprint arXiv:1411.1546*, 2014.

[2] Deepak Agarwal, Bee-Chung Chen, and Pradheep Elango. Spatio-temporal models for estimating click-through rate. In *Proceedings of the 18th international conference on World wide web*, pages 21–30. ACM, 2009.

[3] Ameya Agaskar and Yue M Lu. A fast monte carlo algorithm for source localization on graphs. In *SPIE Optical Engineering+ Applications*, pages 88581N–88581N. International Society for Optics and Photonics, 2013.

[4] Mahima Agumbe Suresh, Radu Stoleru, Ron Denton, Emily Zechman, and Basem Shihada. Towards optimal event detection and localization in acyclic flow networks. In *Proceedings of the 13th International Conference on Distributed Computing and Networking*, ICDCN'12, pages 179–196, Berlin, Heidelberg, 2012. Springer-Verlag.

[5] Tharaka Alahakoon et al. K-path centrality: A new centrality measure in social networks. In *Proceedings of the 4th Workshop on Social Network Systems*, page 1. ACM, 2011.

[6] Réka Albert, Hawoong Jeong, and Albert-László Barabási. Error and attack tolerance of complex networks. *nature*, 406(6794):378–382, 2000.

[7] Çigdem Aslay, Nicola Barbieri, Francesco Bonchi, and Ricardo A Baeza-Yates. Online topic-aware influence maximization queries. In *EDBT*, pages 295–306, 2014.

[8] David A Bader, Shiva Kintali, Kamesh Madduri, and Milena Mihail. Approximating betweenness centrality. In *Algorithms and Models for the Web-Graph*, pages 124–137. Springer, 2007.

[9] Roja Bandari, Sitaram Asur, and Bernardo A. Huberman. The pulse of news in social media: Forecasting popularity. In *Proceedings of the Sixth International Conference on Weblogs and Social Media, Dublin, Ireland, June 4-7, 2012*, 2012.

[10] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *science*, 286(5439):509–512, 1999.

[11] A. Bavelas. A mathematical model for group structures. *Human Organization*, 7:16–30, 1948.

[12] Paolo Boldi and Sebastiano Vigna. Axioms for centrality. *Internet Mathematics*, 10(3-4):222–262, 2014.

[13] 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*, pages 946–957. SIAM, 2014.

[14] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[15] C. Brandes and C Pich. Centrality estimation in large networks. *Int. J. Bifurcation and Chaos*, 17(7):2303–2318, 2007.

[16] Ulrik Brandes. A faster algorithm for betweenness centrality*. *Journal of Mathematical Sociology*, 25(2):163–177, 2001.

[17] Laura Bright, Avigdor Gal, and Louiqa Raschid. Adaptive pull-based policies for wide area data delivery. *ACM Transactions on Database Systems (TODS)*, 31(2):631–671, 2006.

[18] Andrei Broder and Michael Mitzenmacher. Network applications of bloom filters: A survey. *Internet mathematics*, 1(4):485–509, 2004.

[19] Jacqueline Johnson Brown and Peter H Reingen. Social ties and word-of-mouth referral behavior. *Journal of Consumer research*, pages 350–362, 1987.

[20] Mario Cataldi, Luigi Di Caro, and Claudio Schifanella. Emerging topic detection on Twitter based on temporal and social terms evaluation. In *Proceedings of the Tenth International Workshop on Multimedia Data Mining*, MDMKDD '10, pages 4:1–4:10, New York, NY, USA, 2010. ACM.

[21] Bernard Chazelle, Joe Kilian, Ronitt Rubinfeld, and Ayellet Tal. The bloomier filter: an efficient data structure for static support lookup tables. In *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 30–39. Society for Industrial and Applied Mathematics, 2004.

[22] 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*, KDD '10, pages 1029–1038, New York, NY, USA, 2010. ACM.

[23] Wei Chen, Yajun Wang, and Siyu Yang. Efficient influence maximization in social networks. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '09, pages 199–208, New York, NY, USA, 2009. ACM.

[24] Zhen Chen, Kai Zhu, and Lei Ying. Detecting multiple information sources in networks under the sir model. *IEEE Transactions on Network Science and Engineering*, 3(1):17–31, 2016.

[25] Justin Cheng, Lada Adamic, P Alex Dow, Jon Michael Kleinberg, and Jure Leskovec. Can cascades be predicted? In *Proceedings of the 23rd international conference on World wide web*, pages 925–936. ACM, 2014.

[26] Yurong Cheng, Ye Yuan, Lei Chen, and Guoren Wang. The reachability query over distributed uncertain graphs. In *Distributed Computing Systems (ICDCS), 2015 IEEE 35th International Conference on*, pages 786–787. IEEE, 2015.

[27] N. A. Christakis and J. H. Fowler. Social network sensors for early detection of contagious outbreaks. *PLoS ONE*, 5(9):e12948, 2010.

[28] Edith Cohen, Daniel Delling, Thomas Pajor, and Renato F Werneck. Distance-based influence in networks: Computation and maximization. *arXiv preprint arXiv:1410.6976*, 2014.

[29] Edith Cohen, Daniel Delling, Thomas Pajor, and Renato F Werneck. Sketch-based influence maximization and computation: Scaling up with guarantees. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, pages 629–638. ACM, 2014.

[30] Edith Cohen, Eran Halperin, Haim Kaplan, and Uri Zwick. Reachability and distance queries via 2-hop labels. *SIAM Journal on Computing*, 32(5):1338–1355, 2003.

[31] Saar Cohen and Yossi Matias. Spectral bloom filters. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 241–252. ACM, 2003.

[32] Anirban Dasgupta, Arpita Ghosh, Ravi Kumar, Christopher Olston, Sandeep Pandey, and Andrew Tomkins. The discoverability of the web. In *Proceedings of the 16th international conference on World Wide Web*, pages 421–430. ACM, 2007.

[33] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[34] Andrew Delaney. *The Growing Role of News in Trading Automation*, October 2009. `http://www.machinereadablenews.com/images/dl/Machine_Readable_News_and_Algorithmic_Trading.pdf`.

[35] Shlomi Dolev, Yuval Elovici, Rami Puzis, and Polina Zilberman. Incremental deployment of network monitors based on group betweenness centrality. *Information Processing Letters*, 109(20):1172–1176, 2009.

[36] Pedro Domingos and Matt Richardson. Mining the network value of customers. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 57–66. ACM, 2001.

[37] P Alex Dow, Lada A Adamic, and Adrien Friggeri. The anatomy of large facebook cascades. In *ICWSM*, 2013.

[38] 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*, pages 3147–3155, 2013.

[39] Ehsan Ebrahimzadeh, Linda Farczadi, Pu Gao, Abbas Mehrabian, Cristiane M Sato, Nick Wormald, and Jonathan Zung. On the longest paths and the diameter in random apollonian networks. *Electronic Notes in Discrete Mathematics*, 43:355–365, 2013.

[40] Dóra Erdos, Vatche Ishakian, Azer Bestavros, and Evimaria Terzi. A divide-and-conquer algorithm for betweenness centrality. SIAM, 2015.

[41] Martin Fink and Joachim Spoerhase. Maximum betweenness centrality: approximability and tractable cases. In *WALCOM: Algorithms and Computation*, pages 9–20. Springer, 2011.

[42] Linton C Freeman. A set of measures of centrality based on betweenness. *Sociometry*, pages 35–41, 1977.

[43] Alan Frieze and Charalampos E Tsourakakis. Some properties of random apollonian networks. *Internet Mathematics*, 10(1-2):162–187, 2014.

[44] Shuai Gao, Jun Ma, and Zhumin Chen. Modeling and predicting retweeting dynamics on microblogging platforms. In *Proceedings of the Eighth ACM International Conference on Web Search and Data Mining*, pages 107–116. ACM, 2015.

[45] Yong Gao. The degree distribution of random k-trees. *Theoretical Computer Science*, 410(8):688–695, 2009.

[46] R. Ghosh, S.H. Teng, K. Lerman, and X. Yan. The interplay between dynamics and networks: centrality, communities, and cheeger inequality. KDD, 2014.

[47] Michelle Girvan and Mark EJ Newman. Community structure in social and biological networks. *PNAS*, 99(12):7821–7826, 2002.

[48] Jacob Goldenberg, Barak Libai, and Eitan Muller. Talk of the network: A complex systems look at the underlying process of word-of-mouth. *Marketing letters*, 12(3):211–223, 2001.

[49] Jacob Goldenberg, Barak Libai, and Eitan Muller. Using complex systems analysis to advance marketing theory development: Modeling heterogeneity effects on new product growth through stochastic cellular automata. *Academy of Marketing Science Review*, 2001:1, 2001.

[50] Daniel Golovin, Matthew Faulkner, and Andreas Krause. Online distributed sensor selection. In *Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks*, IPSN '10, pages 220–231, New York, NY, USA, 2010. ACM.

[51] Deutsche Börse Group. Alphaflash trader automated trading based on economic events. `http://www.alphaflash.com/product-info/alphaflash-trader`, 2015.

[52] Jing Guo, Peng Zhang, Chuan Zhou, Yanan Cao, and Li Guo. Personalized influence maximization on social networks. In *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management*, pages 199–208. ACM, 2013.

[53] Ruocheng Guo, Elham Shaabani, Abhinav Bhatnagar, and Paulo Shakarian. Toward order-of-magnitude cascade prediction. In *Proceedings of the 2015 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2015*, pages 1610–1613. ACM, 2015.

[54] Godfrey H Hardy. *Divergent series*, volume 334. American Mathematical Soc., 1991.

[55] Godfrey H Hardy. *Divergent series*, volume 334. American Mathematical Soc., 1991.

[56] W. Hart and R. Murray. Review of sensor placement strategies for contamination warning systems in drinking water distribution systems. *Journal of Water Resources Planning and Management*, 136(6):611–619, 2010.

[57] Miguel Helft. *Google: 1 Million Advertisers in 2007, More Now*, January 2009. http://bits.blogs.nytimes.com/2009/01/08/google-1-million-advertisers-in-2007-more-now/?_r=0.

[58] Petter Holme, Beom Jun Kim, Chang No Yoon, and Seung Kee Han. Attack vulnerability of complex networks. *Physical Review E*, 65(5):056109, 2002.

[59] Bradley Hope. How computers trawl a sea of data for stock picks. *The Wall Street Journal*, April 2015. http://www.wsj.com/articles/how-computers-trawl-a-sea-of-data-for-stock-picks-1427941801?KEYWORDS=computers+trawl+sea.

[60] Roxana Horincar, Bernd Amann, and Thierry Artières. Online refresh strategies for content based feed aggregation. *World Wide Web*, pages 1–35, 2014.

[61] Vatche Ishakian, Dóra Erdös, Evimaria Terzi, and Azer Bestavros. A framework for the evaluation and management of network centrality. In *SDM*, pages 427–438. SIAM, 2012.

[62] Swami Iyer, Timothy Killingback, Bala Sundaram, and Zhen Wang. Attack robustness and centrality of complex networks. *PloS one*, 8(4):e59613, 2013.

[63] Jiaojiao Jiang, Sheng Wen, Shui Yu, Yang Xiang, and Wanlei Zhou. Identifying propagation sources in networks: State-of-the-art and comparative studies. *IEEE Communications Surveys & Tutorials*, 2016.

[64] Ruoming Jin, Lin Liu, Bolin Ding, and Haixun Wang. Distance-constraint reachability computation in uncertain graphs. *Proceedings of the VLDB Endowment*, 4(9):551–562, 2011.

[65] Ruoming Jin, Yang Xiang, Ning Ruan, and Haixun Wang. Efficiently answering reachability queries on very large directed graphs. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 595–608. ACM, 2008.

[66] Kyomin Jung, Wooram Heo, and Wei Chen. IRIE: Scalable and robust influence maximization in social networks. *arXiv preprint arXiv:1111.4795*, 2011.

[67] L. Katz. A new status index derived from sociometric index. *Psychometrika*, pages 39–43, 1953.

[68] David Kempe, 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*, pages 137–146. ACM, 2003.

[69] David Kempe, Jon Kleinberg, and Éva Tardos. Influential nodes in a diffusion model for social networks. In *Proceedings of the 32Nd International Conference on Automata, Languages and Programming*, ICALP'05, pages 1127–1138, Berlin, Heidelberg, 2005. Springer-Verlag.

[70] Yaser Keneshloo, Shuguang Wang, Eui-Hong Han, and Naren Ramakrishnan. Predicting the popularity of news articles. In *Proceedings of the 2016 SIAM International Conference on Data Mining*, pages 441–449. SIAM, 2016.

[71] Arijit Khan, Francesco Bonchi, Aristides Gionis, and Francesco Gullo. Fast reliability search in uncertain graphs. In *EDBT*, pages 535–546, 2014.

[72] Jon M Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM (JACM)*, 46(5):604–632, 1999.

[73] A. Krause, J. Leskovec, C. Guestrin, J. VanBriesen, and C. Faloutsos. Efficient sensor placement optimization for securing large water distribution networks. *Journal of Water Resources Planning and Management*, 134(6):516–526, 2008.

[74] Andreas Krause and Carlos Guestrin. Submodularity and its applications in optimized information gathering. *ACM Trans. Intell. Syst. Technol.*, 2(4):32:1–32:20, July 2011.

[75] Andreas Krause, Carlos Guestrin, Anupam Gupta, and Jon Kleinberg. Robust sensor placements at informative and communication-efficient locations. *ACM Trans. Sen. Netw.*, 7(4):31:1–31:33, February 2011.

[76] Andreas Krause, Ram Rajagopal, Anupam Gupta, and Carlos Guestrin. Simultaneous placement and scheduling of sensors. In *Proceedings of the 2009*

International Conference on Information Processing in Sensor Networks, IPSN '09, pages 181–192, Washington, DC, USA, 2009. IEEE Computer Society.

[77] Andrey Kupavskii, Liudmila Ostroumova, Alexey Umnov, Svyatoslav Usachev, Pavel Serdyukov, Gleb Gusev, and Andrey Kustarev. Prediction of retweet cascade size over time. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, pages 2335–2338. ACM, 2012.

[78] Theodoros Lappas, Evimaria Terzi, Dimitrios Gunopulos, and Heikki Mannila. Finding effectors in social networks. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1059–1068. ACM, 2010.

[79] Noam Lemelshtrich Latar. The robot journalist in the age of social physics: The end of human journalism? In *The New World of Transitioned Media*, pages 65–80. Springer, 2015.

[80] Ronny Lempel and Shlomo Moran. Salsa: the stochastic approach for link-structure analysis. *TOIS*, 19(2):131–160, 2001.

[81] Alberto Leon-Garcia. *Probability, Statistics, and Random Processes For Electrical Engineering (3rd Edition)*. Prentice Hall, 3 edition, 1 2008.

[82] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graph evolution: Densification and shrinking diameters. *TKDD*, 1(1):2, 2007.

[83] Jure Leskovec, Andreas Krause, Carlos Guestrin, Christos Faloutsos, Jeanne M. VanBriesen, and Natalie S. Glance. Cost-effective outbreak detection in networks. In Pavel Berkhin, Rich Caruana, and Xindong Wu, editors, *KDD*, pages 420–429. ACM, 2007.

[84] David Liben-Nowell and Jon Kleinberg. Tracing information flow on a global scale using internet chain-letter data. *Proceedings of the National Academy of Sciences*, 105(12):4633–4638, 2008.

[85] Andrey Y Lokhov, Marc Mézard, Hiroki Ohta, and Lenka Zdeborová. Inferring the origin of an epidemic with a dynamic message-passing algorithm. *Physical Review E*, 90(1):012801, 2014.

[86] Eagle Alpha Ltd. Discovering the web's hidden alpha. `http://www.eaglealpha.com/whitepaper_pdf`, June 2014.

[87] 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*, pages 735–744. ACM, 2015.

[88] Vijay Mahajan, Eitan Muller, and Frank M Bass. New product diffusion models in marketing: A review and directions for research. *The journal of marketing*, pages 1–26, 1990.

[89] Michael Mathioudakis and Nick Koudas. Twittermonitor: Trend detection over the twitter stream. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, SIGMOD '10, pages 1155–1158, New York, NY, USA, 2010. ACM.

[90] Wes McKinney. *Structured Data Challenges in Finance and Statistics*, November 2011. `http://www.slideshare.net/wesm/structured-data-challenges-in-finance-and-statistics`.

[91] Gautam Mitra and Leela Mitra. *The handbook of news analytics in finance*, volume 596. John Wiley & Sons, 2011.

[92] Michael Mitzenmacher and Eli Upfal. *Probability and computing: Randomized algorithms and probabilistic analysis*. Cambridge university press, 2005.

[93] Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, 2005.

[94] Tamás F Móri. The maximum degree of the barabási–albert random tree. *Combinatorics, Probability and Computing*, 14(03):339–348, 2005.

[95] Nasir Naveed, Thomas Gottron, Jérôme Kunegis, and Arifah Che Alhadi. Bad news travel fast: A content-based analysis of interestingness on twitter. In *Proceedings of the 3rd International Web Science Conference*, page 8. ACM, 2011.

[96] George L Nemhauser and Leonard A Wolsey. Best algorithms for approximating the maximum of a submodular set function. *Mathematics of operations research*, 3(3):177–188, 1978.

[97] Mark EJ Newman. A measure of betweenness centrality based on random walks. *Social networks*, 27(1):39–54, 2005.

[98] Naoto Ohsaka, Takuya Akiba, Yuichi Yoshida, and Ken-ichi Kawarabayashi. Fast and accurate influence maximization on large networks with pruned monte-carlo simulations. In *AAAI*, pages 138–144, 2014.

[99] Marilena Oita and Pierre Senellart. Deriving dynamics of web pages: A survey. In *TWAW (Temporal Workshop on Web Archiving)*, 2011.

[100] A. Ostfeld, J. Uber, E. Salomons, J. Berry, W. Hart, C. Phillips, J. Watson, G. Dorini, P. Jonkergouw, Z. Kapelan, F. di Pierro, S. Khu, D. Savic, D. Eliades, M. Polycarpou, S. Ghimire, B. Barkdoll, R. Gueli, J. Huang, E. McBean, W. James, A. Krause, J. Leskovec, S. Isovitsch, J. Xu, C. Guestrin, J. VanBriesen, M. Small, P. Fischbeck, A. Preis, M. Propato, O. Piller, G. Trachtman, Z. Wu, and T. Walski. The battle of the water sensor networks (BWSN): A design challenge for engineers and algorithms. *Journal of Water Resources Planning and Management*, 134(6):556–568, 2008.

[101] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: bringing order to the web. 1999.

[102] Sasa Petrovic, Miles Osborne, and Victor Lavrenko. Rt to win! predicting message propagation in twitter. In Lada A. Adamic, Ricardo A. Baeza-Yates, and Scott Counts, editors, *ICWSM*. The AAAI Press, 2011.

[103] B Aditya Prakash, Jilles Vreeken, and Christos Faloutsos. Spotting culprits in epidemics: How many and which ones? In *Data Mining (ICDM), 2012 IEEE 12th International Conference on*, pages 11–20. IEEE, 2012.

[104] Matthew Richardson and Pedro Domingos. Mining knowledge-sharing sites for viral marketing. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 61–70. ACM, 2002.

[105] Matteo Riondato and Evgenios M Kornaropoulos. Fast approximation of betweenness centrality through sampling. WSDM, 2014.

[106] Neil Robertson and Paul D. Seymour. Graph minors. ii. algorithmic aspects of tree-width. *Journal of algorithms*, 7(3):309–322, 1986.

[107] Manuel Gomez Rodriguez and Bernhard Schölkopf. Influence maximization in continuous time diffusion networks. *arXiv preprint arXiv:1205.1682*, 2012.

[108] Everett M Rogers. *Diffusion of innovations*. Simon and Schuster, 2010.

[109] Ahmet Erdem Sarıyüce, Erik Saule, Kamer Kaya, and Ümit V Çatalyürek. Shattering and compressing networks for centrality analysis. *arXiv preprint arXiv:1209.6007*, 2012.

[110] Lior Seeman and Yaron Singer. Adaptive seeding in social networks. In *FOCS*, pages 459–468. IEEE Computer Society, 2013.

[111] Devavrat Shah and Tauhid Zaman. Rumors in a network: Who's the culprit? *IEEE Transactions on information theory*, 57(8):5163–5181, 2011.

[112] Devavrat Shah and Tauhid Zaman. Rumor centrality: a universal source detector. In *ACM SIGMETRICS Performance Evaluation Review*, volume 40, pages 199–210. ACM, 2012.

[113] Hua-Wei Shen, Dashun Wang, Chaoming Song, and Albert-László Barabási. Modeling and predicting popularity dynamics via reinforced poisson processes. *arXiv preprint arXiv:1401.0778*, 2014.

[114] Ka Cheung Sia, Junghoo Cho, and Hyun-Kyu Cho. Efficient monitoring algorithm for fast news alerts. *Knowledge and Data Engineering, IEEE Transactions on*, 19(7):950–961, 2007.

[115] Jiao Su, Qing Zhu, Hao Wei, and Jeffrey Xu Yu. Reachability querying: Can it be even faster? *IEEE Transactions on Knowledge and Data Engineering*, 2016.

[116] Karthik Subbian, Charu C Aggarwal, and Jaideep Srivastava. Querying and tracking influencers in social streams. In *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining*, pages 493–502. ACM, 2016.

[117] Bongwon Suh, Lichan Hong, Peter Pirolli, and Ed H Chi. Want to be retweeted? large scale analytics on factors impacting retweet in twitter network. In *Social computing (socialcom), 2010 ieee second international conference on*, pages 177–184. IEEE, 2010.

[118] Gabor Szabo and Bernardo A Huberman. Predicting the popularity of online content. *Communications of the ACM*, 53(8):80–88, 2010.

[119] Fangshuang Tang, Qi Liu, Hengshu Zhu, Enhong Chen, and Feida Zhu. Diversified social influence maximization. In *Advances in Social Networks Analysis and Mining (ASONAM), 2014 IEEE/ACM International Conference on*, pages 455–459. IEEE, 2014.

[120] 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*, pages 1539–1554. ACM, 2015.

[121] Youze Tang, Xiaokui Xiao, and Yanchen Shi. Influence maximization: Near-optimal time complexity meets practical efficiency. *arXiv preprint arXiv:1404.0900*, 2014.

[122] Robert Tarjan. Depth-first search and linear graph algorithms. *SIAM journal on computing*, 1(2):146–160, 1972.

[123] Sasu Tarkoma, Christian Esteve Rothenberg, and Eemil Lagerspetz. Theory and practice of bloom filters for distributed systems. *Communications Surveys & Tutorials, IEEE*, 14(1):131–155, 2012.

[124] Lilian Weng, Filippo Menczer, and Yong-Yeol Ahn. Predicting successful memes using network and community structure. *arXiv preprint arXiv:1403.6199*, 2014.

[125] Joel L Wolf, Mark S Squillante, PS Yu, Jay Sethuraman, and Leyla Ozsen. Optimal crawling strategies for web search engines. In *Proceedings of the 11th international conference on World Wide Web*, pages 136–147. ACM, 2002.

[126] Yosuke Yano, Takuya Akiba, Yoichi Iwata, and Yuichi Yoshida. Fast and scalable reachability queries on graphs by pruned labeling with landmarks and paths. In *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management*, pages 1601–1606. ACM, 2013.

[127] Yuichi Yoshida. Almost linear-time algorithms for adaptive betweenness centrality using hypergraph sketches. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1416–1425. ACM, 2014.

[128] Tauhid R Zaman, Ralf Herbrich, Jurgen Van Gael, and David Stern. Predicting information spreading in twitter. In *Workshop on Computational Social Science and the Wisdom of Crowds, NIPS*, volume 104, pages 17599–601. Citeseer, 2010.

[129] Qingyuan Zhao, Murat A Erdogdu, Hera Y He, Anand Rajaraman, and Jure Leskovec. Seismic: A self-exciting point process model for predicting tweet popularity. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1513–1522. ACM, 2015.

[130] Chuan Zhou and Li Guo. A note on influence maximization in social networks from local to global and beyond. *Procedia Computer Science*, 30:81–87, 2014.