Efficient Computations on Uncertain Graphs by Group Testing, Streaming and Recycling

by

Glenn Steven Bevilacqua

B.Sc., The University of British Columbia, 2012

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

Doctor of Philosophy

in

THE FACULTY OF GRADUATE AND POSTDOCTORAL STUDIES
(Computer Science)

The University of British Columbia
(Vancouver)

April 2022

The following individuals certify that they have read, and recommend to the Faculty of Graduate and Postdoctoral Studies for acceptance, the thesis entitled:

Efficient Computations on Uncertain Graphs by Group Testing, Streaming and Recycling

submitted by **Glenn Steven Bevilacqua** in partial fulfillment of the requirements for the degree of **Doctor of Philosophy** in **Computer Science**.

Examining Committee:

Laks V.S. Lakshmanan, Professor, Computer Science, UBC
Supervisor

Raymond Ng, Professor, Computer Science, UBC
Supervisory Committee Member

David Poole, Professor, Computer Science, UBC
Supervisory Committee Member

William Evans, Professor, Computer Science, UBC
University Examiner

Daniel J. McDonald, Associate Professor, Statistics, UBC
University Examiner

Sourav S. Bhowmick, Associate Professor, Nanyang Technological University
External Examiner

Abstract

Uncertain graphs, where the presence of connections between nodes is probabilistic, have received a great deal of attention in a wide range of fields. Despite the progress made by prior works, the application of existing algorithms to solve the important problems of coverage maximization, also known as Influence Maximization (IM), and reachability estimation, also known as reliability, on uncertain graphs remains constrained by their computational costs in the form of both the running time and memory needed for one to achieve high quality solutions. In Chapter 2 we address the issue that when performing sampling on large networks the majority of random draws of edges are being effectively wasted as the majority of edges will not be live. We resolve this by introducing an approach that through the application of group testing of edges scales only logarithmically with the number of failed edges in the worst case as opposed to linearly. In Chapter 3 we tackle the problem of the exorbitant memory required to store the Reverse Influence Sample (RIS) collection used by existing approaches to solve the IM problem becoming prohibitive. We avoid this by developing a new non-greedy approach that avoids storing the RIS collection by streaming it instead. In Chapter 4 we note that a key cause of Monte Carlo simulation, along with existing approaches that build upon it, becoming ineffective for estimating low probability reachability is caused by the number of samples it needs to attain a desired relative error depending on the probability that is being estimated. We remedy this by developing a technique of recycling of random draws which enables us to develop an algorithm whose relative error does not depend on the probability being estimated and as such does not suffer from this limitation. In all cases we perform experiments on real datasets to empirically validate the effectiveness of the algorithms and techniques we develop.

Lay Summary

Many real networks such as protein-protein interaction (PPI) networks, peer-to-peer (P2P) networks and social networks exhibit uncertainty: each connection from one node, be it a computer or person, to another has an associated probability that it is live. Reachability estimation involves determining with what probability there exists a path connecting a source node to a target node. Example applications are estimating the probability two nodes in a computer network can reach each other and determining to whom information may spread in a social network. Coverage maximization requires identifying a set of source nodes that reach the maximum number of nodes in the network in expectation. This is critical for the placement of information sources in viral marketing or sensors for outbreak detection. We develop efficient algorithms that improve on the state-of-the-art and enable addressing these important problems using less computer running time and memory.

Preface

All research presented in this dissertation was done under the supervision of Prof. Laks V.S. Lakshmanan. Chapter 2 and Chapter 3 contain material that has been published [6] in the June 2021 Special Issue on Big Graph Data Management and Processing in *The VLDB Journal*. I designed the algorithm presented, developed the associated theory, and conducted the experiments under the guidance of Prof. Laks V.S. Lakshmanan.

Table of Contents

List of Tables

List of Figures

Acknowledgments

I would like to express my gratitude to my supervisor Professor Laks V.S. Lakshmanan. I am indebted to him for his patience and willingness to spend time every week to work through challenges together. This dissertation would not have been possible without his supervision and continuous guidance.

I must also thank my supervisory committee members Professor David Poole and Professor Raymond Ng for their time and feedback. I am very thankful for Professor William Evans, Professor Daniel J. McDonald and Professor Sourav Bhowmick examining my dissertation and providing very helpful comments. Professor Mark Schmidt, who chaired my thesis proposal defense, and Professor Boris Stoeber, who chaired my final defense, I thank for their time and help.

I must also acknowledge all of the DMM lab members through the years and everyone that attended our reading group who accompanied me on this journey. Most importantly, I am greatly indebted to my parents whose support enabled me to focus on completing this dissertation and without whom none of this would have been possible.

Chapter 1

Introduction

Uncertain graphs, where the presence of connections between nodes is probabilistic, have received attention from a wide range of fields. The ability to express uncertainty inherent in data is valuable in a wide range of settings including sensor networks [30], protein-protein interaction (PPI) networks [48, 56], road networks [44], peer-to-peer (P2P) networks [48] and social networks [56]. Fundamental problems are those of reachability and coverage.

Reachability, also known as reliability, on uncertain graphs is not a simple YES / NO query but rather asks with what probability does a path exist connecting two nodes of interest. Estimating the probability of two nodes being connected is of importance for many applications. In particular, it may be used to capture the probability of a route existing from a source to a destination in sensor, computer or road networks [30, 44]. In addition, it provides valuable information on how frequently nodes may interact in a direct or indirect manner. This is of importance for analysis of protein-protein interaction (PPI) networks and social networks [48, 56]. In the typically studied setting, where connections between nodes of the network are present (i.e. live) or absent (dead) with independent probability, the exact calculation of the reachability probability is known to be #P-hard [86]. Consequently, focus is instead given to approximation algorithms that can scale to realistic size networks.

Coverage, or spread, is defined to be the expected number of nodes in the network that are reachable from a source node or set of source nodes. Estimation of

coverage is critical for assessing the effectiveness of a placement of information sources for viral marketing [55] or sensors for outbreak detection [61] in uncertain networks. Exact calculation of coverage under standard diffusion models, which characterize the probability with which nodes reach each other, is also known to be #P-hard [15, 87]. As with reachability, the goal is development of efficient approximation algorithms. Optimization of the selection of nodes so to maximize coverage gives rise to the extensively studied problem known as *influence maximization*.

The problem of influence maximization (IM), introduced as a discrete optimization problem by Kempe et. al. [55], has found a wide range of applications including infection containment [61], feed ranking [46], personalized recommendation [82] and regulatory cell cycle analysis [31] beyond its initial application to viral marketing [20, 77]. The IM problem asks: subject to a given upper limit cardinality constraint on the number of nodes that may be selected, find the set of nodes, referred to as a 'seed set', that achieves maximum coverage, also known as the influence spread. Even if one were to have an oracle that could provide the exact coverage of a set of nodes, finding the optimal seed set that achieves maximum coverage is known to be NP-hard [55]. Despite this, the coverage of sets of nodes under typical diffusion models exhibits properties that enable a simple greedy algorithm to attain a constant factor approximation. Unfortunately the effectiveness of the greedy algorithm does nothing to address the underlying difficulty of coverage estimation. Kempe et al. [55] initially tackled this using Monte-Carlo (MC) simulation but this approach does not scale to large networks and seed set sizes [84]. Reverse influence sampling (RIS) introduced by Borgs et al. [7] has led to a class of current state-of-the-art algorithms [45, 71, 83–85]. While these approaches can often scale to large networks they become very memory intensive.

The focus of this dissertation is on addressing key efficiency challenges involved in coverage estimation, influence maximization and reachability estimation.

1.1 Reachability and Diffusion on Uncertain Graphs

Given a graph a diffusion model specifies how and with what probability nodes may reach each other. For reliability estimation the standard setting involves each edge having an associated independent probability with which transmission may

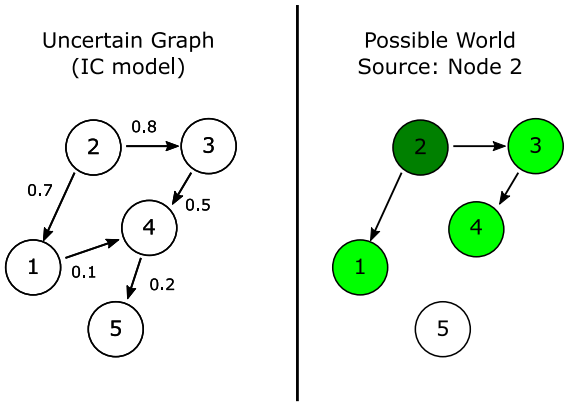


Figure 1.1: Independent Cascade (IC) Model Example

occur over it [86]. Existing literature in influence maximization (IM) primarily focuses on the Independent Cascade (IC) and Linear Threshold (LT) *discrete time* models [55]. The IC model studied in the literature on IM is equivalent to the setting considered in the literature on reliability estimation.

It has been argued that a *continuous time* model may be more appropriate for modeling diffusion of product adoptions and infections and that it is significantly more accurate than discrete time models [21, 35–37]. Discrete time models lack the ability to capture the time needed for propagation to occur across links, which can differ between links. As such, if the goal is to attain a desired influence spread within a given time frame, discrete time models lack the ability to capture this precisely.

Diffusion Models. Here we will consider the independent cascade (IC) model, as it pertains to a variety of reachability applications, and the continuous-time independent cascade (CTIC) model for coverage estimation as needed for influence maximization. These models are formally defined in the literature as follows.

Definition 1 (Independent Cascade (IC) Model [55])**.** *Given a directed graph, $G = (V,E)$, the IC model includes a function, $p : E \to [0,1]$, that assigns an independent probability to each edge of the graph. The probability $p((u,v))$ associated with an edge $(u,v) \in E$ specifies the probability with which the edge (u,v) is present in the graph. Equivalently, it can be understood as the probability with which node*

u can transmit to v over the edge. The resulting uncertain graph can be seen as a probability distribution, $\mathcal{G}$, over deterministic graphs, also known as possible worlds, where the probability associated with a graph, $G' = (V, E')$, $G' \sim \mathcal{G}$, where $E' \subseteq E$ is:

$$\Pr[G'] = \prod_{e \in E'} p(e) \prod_{e \in E \setminus E'} (1 - p(e)) \tag{1.1}$$

As each possible world is deterministic, one can then find for a source node s the set of nodes that are reachable from it in the possible world. Shown in Figure 1.1 is a toy example of an uncertain graph on the left and a possible world drawn from it on the right. The nodes that are reachable from node 2 in this possible world are highlighted in green.

The IC model is a fundamental model that has been studied extensively in the literature on Influence Maximization[7, 55, 71, 83–85, 87] originating from the context of marketing[33, 34]. The IC model has the property that the probabilities that edges are present in the graph are *independent* of each other. It is also the case that the IC model corresponds to the setting studied in the literature on reliability[4, 48, 54, 86], which we refer to as reachability estimation, where it is standard to assume that the events of individual links being present or absent are independent.

Problem 1 (Reachability Estimation)**.** *Let, $\mathbb{I}_{G'}[s,t]$, be an indicator function that is 1 if s can reach the node t in a given deterministic graph G' and 0 otherwise. Then the reachability probability, $R(s,t)$, (also known as the reliability) of s reaching t on the uncertain graph characterized by $\mathcal{G}$ is:*

$$R(s,t) = \mathbb{E}_{G' \sim \mathcal{G}} [\mathbb{I}_{G'}[s,t]] \tag{1.2}$$

where each deterministic graph, G', is drawn according to its probability, $\Pr[G']$.

It has been argued in the literature that a *continuous time* model may be more appropriate for modeling diffusion and that it can be significantly more accurate than discrete time models [21, 35–37]. Specifically, discrete time models lack the ability to capture the time needed for propagation to occur across links, which can differ for different links. As such, if the goal is to reach some coverage within a given time frame, discrete time models lack the ability to capture this precisely.

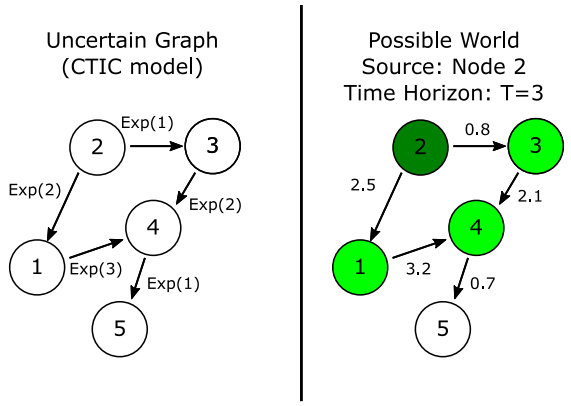


Figure 1.2: Continuous-time Independent Cascade (CTIC) Model Example

Definition 2 (Continuous-time Independent Cascade (CTIC) Model [35, 78])**.** *Given a directed graph, $G = (V, E)$, the CTIC model associates an edge length probability distribution $\mathscr{L}(e)$ with each edge $e \in E$. Under the CTIC model a possible world corresponds to an edge length being drawn for each edge, e, from its associated edge length distribution, $\mathscr{L}(e)$. Each edge length that is drawn specifies how much time is required for propagation to occur across that edge in the possible world.*

For instance the exponential distribution may be used to specify the time needed for diffusion to occur across each edge, $\mathscr{L}(e)$, as done in [78]. Each edge, e, then has an associated rate parameter, λ_e, which parametrizes its edge length distribution, $\mathscr{L}(e)$, which is specified by $\text{Exp}(\lambda_e)$. The probability density, $f(G')$, associated with a possible world, $G' = (V, E, t(\cdot))$, for which each edge has an associated time length, $t(e)$, to traverse then is:

$$f(G') = \prod_{e \in E} \lambda_e \exp(-\lambda_e t(e)) \tag{1.3}$$

Definition 3 (Coverage [78])**.** *Let, $\mathbb{I}_{(G',T)}[S, v]$, be an indicator function that is 1 if there is a node in the set of nodes, S, that can reach the node v via a path in the deterministic graph G' whose edges' total time length is less that T and 0 otherwise. The coverage, $\sigma(S;T)$, also known as spread or influence spread, of the*

set of nodes S under the CTIC model is then:

$$\sigma(S;T) = \sum_{v \in V} \left[\mathbb{E}_{G' \sim \mathscr{G}} \left[\mathbb{I}_{(G',T)}[S,v] \right] \right] \tag{1.4}$$

where each deterministic graph is drawn by performing an edge length draw for each edge of the uncertain graph. The parameter T, which is implicit in the context, is often dropped and the coverage is referred to as simply, $\sigma(S)$.

Although the CTIC model was previously proposed [35, 78] in a manner that involves time, the time horizon, T, can also be understood as the maximum path length and the time needed for propagation to occur across an edge in a possible world as its edge length. The coverage, $\sigma(S;T)$, is then the expected number of nodes that can be reached from S with a path length less than T.

Shown in Figure 1.2 is a toy example of an uncertain graph which has exponentially distributed edge lengths (left) and a possible world drawn from it (right). For a time horizon of $T = 3$, there are a total 4 nodes covered (shown highlighted in green) by the source node 2 in the possible world.

Diffusion Approximation. Since exact calculation of the reachability probability and coverage is known to be #P-hard [15, 86, 87] it is critical to have an efficient means to approximate the diffusion. It is worth noting that the reachability probability is not the same as the probability of the source reaching the target via a random walk. At an intuitive level, random walks allow each path taken by the random walk to redraw the edges encountered (i.e. for different paths that share the same edge some paths may find the edge to be live while others find it to be dead). In the context of diffusion analysis, existing applications of the multivariate Hawkes process exhibit this behavior (e.g. [25]). In contrast, in reachability all paths share the same edge draw result (i.e. if the edge is dead then all paths through it also fail). It is this sharing of edge results among paths that makes the reachability probability more realistic but also is the underlying cause of it being difficult to estimate. The most immediate means to address this is to utilize the connection to possible worlds.

This results in the Monte Carlo sampling approach that is used as a baseline and a building block for reachability estimation [48, 54]. This approach is sometimes referred to as 'direct' or 'crude' Monte Carlo to distinguish it for other Monte Carlo

based techniques [27, 57]. This also is the approach applied by [55] for influence spread estimation that is referred to Monte Carlo simulation by subsequent works, e.g. [14, 15, 84, 87].

Definition 4 (Monte Carlo (MC) Simulation [14, 27, 55])**.** *Monte Carlo simulation (also called Monte Carlo sampling) samples K possible worlds, $G' \sim \mathscr{G}$, from the distribution over graphs, which are used to estimate the expectation over the distribution. For instance, let $G'_1, ..., G'_K \sim \mathscr{G}$ then the estimate:*

$$\hat{R}(s,t) = \frac{1}{K} \sum_{i=1}^{K} \left[\mathbb{I}_{G'_i}[s,t] \right] \tag{1.5}$$

may be used to approximate the reachability probability, $R(s,t)$ (see Problem 1).

MC simulation also can be understood as running several simulations of the diffusion process and averaging their results. Each simulation of the diffusion process is equivalent to sampling one possible world.

While the approximation may be made as accurate as needed, doing so comes at a very high computational cost. In the following we elaborate on why this is the case. A significant limitation of MC simulation for reachability estimation is that the relative error of the approximation is impacted by the value of the probability being estimated. We focus on relative error because an absolute error that may be acceptable when the quantity being estimated is large can make the estimate useless when the quantity being estimated is small. It is known [76] that the relative error of MC sampling when using a number of possible world samples, K, that satisfies, $K \geq \frac{3}{\varepsilon^2 R(s,t)} \ln\left(\frac{2}{\delta}\right)$ is,

$$\mathbf{Pr}[|\hat{R}(s,t) - R(s,t)| \geq \varepsilon R(s,t)] \leq \delta \tag{1.6}$$

That is, using this result, we can certify a relative error of at most ε, with probability $1-\delta$, using a sample of $\frac{3}{\varepsilon^2 R(s,t)} \ln\left(\frac{2}{\delta}\right)$ possible worlds. However, this is problematic because $R(s,t)$ is not known and is in fact the quantity that we are estimating. Furthermore, if $R(s,t)$ is small, it can result in a greatly inflated number of possible world samples needed. This is because the number of possible world samples which is needed is inversely proportional to $R(s,t)$. If the number of possible

world samples used is not increased accordingly then the relative error will not be ε and instead will be much greater. For instance, if one were to use $K = \frac{3}{\varepsilon^2} \ln\left(\frac{2}{\delta}\right)$ this result implies that the resulting relative error will be $\varepsilon/\sqrt{R(s,t)}$, which can be much larger than ε if $R(s,t)$ is small.

This issue is somewhat less problematic for coverage estimation, since the selected nodes themselves count towards the coverage. The coverage of a set of nodes, S, is trivially lower bounded by $|S|$ thus preventing the possibility of the quantity to be estimated being arbitrarily small. However, when one is interested in finding a set of nodes that attains maximum coverage simply estimating the spread of one set of nodes is insufficient. This is because MC simulation, as used by [55] and subsequent works [61], needs to start from scratch when estimating the coverage of a new set of nodes. Due to overlap between the coverage of nodes it is not possible to compute the coverage of a set of nodes simply from the expected coverage that the nodes attain individually. A way around this is to keep all of the sampled possible worlds in memory instead of only storing the number of nodes that were reached in the world. While such an approach has been considered in [73] the number of sampled possible worlds that can be practically stored in memory (at most several hundred) limits the estimation accuracy such an approach can attain. Meanwhile the approach of rerunning the MC simulation quickly requires a prohibitive amount of computation time, especially if one wishes to select a seed set that isn't very small. Specifically, prior works focus on sets of nodes of size less than one hundred. In contrast, more recent works consider sets of sizes up to tens of thousands [71].

Reverse influence sampling (RIS) introduced by Borgs et al. [7], also referred to as rr-sets [84], provides an alternative means for obtaining an estimate of coverage in a manner that allows the samples drawn to be used repeatedly to estimate the coverage of various sets of nodes. It is based on equivalently representing the coverage attained on the graph under the diffusion model as instead the fraction of sets covered (see Lemma 1) drawn from an appropriately constructed distribution of sets. Specifically, the sets, known as reverse influence sets or reverse reachability sets, are constructed as follows:

Definition 5 (Reverse influence sampling (RIS) [7, 84])**.** *Let v be a node randomly*

selected from V. Let G' be a possible world sampled from $\mathscr{G}$. Then the resulting RIS sample, also known as a reverse reachable set or rr-set, is:

$$R = \{u \in V \mid \mathbb{I}_{G'}[u,v] = 1\} \tag{1.7}$$

where $\mathbb{I}_{G'}[u,v]$ is 1 if u reaches v in G' and 0 otherwise. Each rr-set is generated by resampling both v and G'. Let the resulting distribution over rr-sets be, $\mathscr{D}$, (i.e. $R \sim \mathscr{D}$).

The coverage under the IC and CTIC models along with other diffusion models that have been considered in the literature has been shown to be representable in this manner [85]. When this is the case, the coverage $\sigma(S)$, of a set S under the diffusion model and the associated probability distribution over reverse influence sets, $\mathscr{D}$, satisfies:

Lemma 1 (Corollary 1 [84])**.** $\sigma(S)/n = \mathbb{E}_{R\sim\mathscr{D}}\mathbb{I}[R \cap S \neq \emptyset] = \mathbb{E}_{\mathbf{r}\sim\mathscr{D}}[\min(\langle \mathbf{s}, \mathbf{r}\rangle, 1)]$, *where* $\mathbf{r}$ *and* $\mathbf{s}$ *are binary vector representations of R and S, $\langle\cdot,\cdot\rangle$ is the inner product, $\sigma(S)/n$ is the normalized influence spread and n is the number of nodes in the network.*

What this means is that the fraction of rr-sets that contain at least one of the nodes in S may be used to estimate the normalized coverage, $\sigma(S)/n$, which is the same as the normalized influence spread considered in [84]. The probability distribution over sets that satisfies this is sampled by using the procedure described in Definition 5. The key advantage of this approach compared to MC simulation (see Definition 4) is that the RIS collection may be used to estimate the coverage of any given set of nodes. This allows them to be reused for multiple different sets of nodes. In contrast, the simulations performed by MC simulation only give the coverage achieved by the source node set that the diffusion simulation was started from, and as such cannot be used to determine the coverage of other node sets.

Influence Maximization. Optimization of the selection of nodes so to maximize coverage gives rise to the extensively studied problem known *influence maximization*.

Problem 2 (Influence Maximization [55])**.** *Given a coverage function, $\sigma(S)$, also known as the spread or influence spread, that applies to a graph, $G = (V,E)$, and*

an integer k, find S^* such that:

$$S^* \in \underset{S \subseteq V \ s.t. \ |S| \leq k}{\arg\max} \sigma(S) \tag{1.8}$$

There exists a vast array of existing works on the influence maximization (IM) problem ranging from more efficient versions of Kempe et. al. approach such as [61] to a wide array of heuristics developed for specific diffusion models such as [15, 87]. Recently developed approaches based on reverse influence sampling (RIS) have consistently achieved state-of-the-art results [45, 71, 83–85] out-competing existing approaches not just in computational running time but also in solution quality, as measured by achieved coverage of returned solutions, theoretical guarantees and empirical guarantees. Furthermore, RIS acts as an abstraction layer that enables algorithms developed for one propagation model to be easily applied to others simply by providing the appropriately generated rr-set. In particular, it is straightforward to adapt RIS based algorithms for the CTIC model [85]. In contrast, algorithms specifically designed for a particular diffusion model (e.g. the IC model) would need to be rebuilt from scratch and may not even be applicable at all if one were to want to apply them to the CTIC model. This is a consequence of dependence of these algorithms on properties that are unique to the diffusion model they were designed for. For these reasons the portion of this dissertation that discusses influence maximization will focus on these state-of-the-art approaches that are built on reverse influence sampling.

The use of RIS to solve the IM problem by existing works may be summarized as: 1) Determining the appropriate number of RIS samples, for a desired relative error of coverage estimation. 2) Constructing and storing a collection of the said number of RIS samples. 3) Applying the Greedy algorithm to select a set of nodes that achieve high coverage, using the RIS collection for node set coverage estimation. The Greedy algorithm is a simple myopic approach that incrementally builds up the set of nodes starting from the empty set and at each step adding the node to the set that has the greatest *marginal gain*. The marginal gain of a node, u, is defined as the increase in the coverage that results from adding it to the selected set, S, which is $\sigma(S \cup \{u\}) - \sigma(S)$. While simple it has been shown that the greedy algorithm attains a $(1 - 1/e)$-approximation of the optimal solution in the case of no

coverage estimation inaccuracy [55]. Furthermore, this approximation guarantee is in effect optimal due to an inapproximability result by Feige [26], which states that for any $\varepsilon > 0$, it is NP-hard to approximate the optimal coverage within a factor of $(1 - 1/e + \varepsilon)$.

It may be noted that the RIS perspective produces a *max-k cover* instance that in expectation is equivalent to the IM instance.

Problem 3 (Max-k Cover [43])**.** *The max-k cover problem, also known as the maximum coverage problem, is a classic problem where one is given an integer k and a collection of sets* $\mathbf{A} = \{A_1, A_2, \ldots, A_n\}$ *and the objective is to find a collection* $\mathbf{A}^*$ *of k sets in* $\mathbf{A}$ *such that their union has the largest cardinality, i.e.,*

$$\mathbf{A}^* \in \underset{\mathbf{A}' \subseteq \mathbf{A} \; s.t. \; |\mathbf{A}'| \leq k}{\arg\max} \left| \bigcup_{A_i \in \mathbf{A}'} A_i \right| \tag{1.9}$$

The mapping to the max-k cover problem is as follows. According to Corollary 1 if one samples a RIS collection, $\mathbf{R}$, the coverage of a set of nodes, S, may be estimated by, $\hat{\sigma}(S) = 1/|\mathbf{R}| \sum_{R \in \mathbf{R}} \mathbb{I}[R \cap S \neq \emptyset]$. As such, for each node, u, one can identify the set of RIS samples that it covers, $\mathbf{A}_u = \{R \in \mathbf{R} \mid u \in R\}$. This is the mapping from the nodes of the IM problem, as represented using RIS sampling, to the sets of the max-k cover problem. Notice that, for a set of nodes S, $|\bigcup_{u \in S} \mathbf{A}_u| = \sum_{R \in \mathbf{R}} \mathbb{I}[R \cap S \neq \emptyset]$. The cardinality constraint, k, is the same as that from the IM problem. As each set, $\mathbf{A}_u$, corresponds to a node the set of sets that is the solution to the max-k cover problem can be directly mapped backed to the set of nodes that are the corresponding solution to the IM problem.

The max-k cover problem objective function has the key properties of *monotonicity* and *submodularity*. Monotonicity is the property that the coverage can only be increased by selecting more nodes.

Definition 6 (Monotonic)**.** *A set function,* $f : 2^V \to \mathbb{R}$*, is said to be monotonic if whenever,* $A \subseteq B$*, for* $A, B \subseteq V$*, then* $f(A) \leq f(B)$*.*

Submodularity is the property of diminishing returns. Adding a node to a set of nodes can only have an equal or smaller marginal gain as adding that node to a subset of that set.

Definition 7 (Submodular)**.** *A set function, $f : 2^V \to \mathbb{R}$, is said to be submodular if for every $A, B \subseteq V$, if $A \subseteq B$ for all $u \in V \setminus B$, $f(A \cup \{u\}) - f(A) \geq f(B \cup \{u\}) - f(B)$.*

As noted by [55] these properties are sufficient for the Greedy algorithm to provide the approximation guarantee. Consequently any diffusion model representable by RIS sampling permits the resulting IM problem to be solved to within a $(1 - 1/e - \varepsilon)$-approximation using the Greedy algorithm, where the additional loss comes from the RIS sampling error. As a result this approach is taken by the majority of existing approximation algorithms for IM.

1.2 Challenges and Key Contributions

Despite the progress made by prior works, the application of existing algorithms remains constrained by their computational costs in the form of both the running time and memory needed for one to achieve high quality solutions for the important problems of coverage maximization and reachability estimation on uncertain graphs.

We identify and develop new approaches to address key inefficiencies that limit existing approaches. Specifically we: i) Identify that when performing sampling on large networks the majority of random draws are being effectively wasted as the majority of edges will not be live (i.e. do not enable diffusion to occur across them). We address this by introducing an approach that, in the worst case, scales only logarithmically with the number of failed edges as opposed to linearly; ii) Observe that the memory required to store the RIS collection used by existing approaches to solve the IM problem can become prohibitive. We resolve this by developing a new non-greedy approach that avoids storing the RIS collection by streaming it instead; iii) Determine that a key cause of MC simulation becoming ineffective for estimating low probability reachability is that the number of samples needed to attain a desired relative error depends on the probability being estimated. We remedy this by developing a technique whose relative error does not depend on the probability being estimated and as such does not suffer from this limitation.

1.2.1 Group Testing for Efficient Sample Generation

Sampling is a core tool used when performing estimation on uncertain graphs. While it is may be simplest to sample a full possible world and only check reachability after, no practical implementation of MC simulation (see Definition 4) or RIS sampling (see Definition 5) does this. Typically the state of many edges are not needed when determining what nodes are reachable. This is leveraged by only sampling edges when they are encountered. 'Forwards' diffusion simulation, as is typically the case when using MC simulation to sample the outcome on a possible world, involve random trials of edge out of nodes reached so far to identify the set of nodes that are reachable. Similarly, 'reverse' reachability simulation, as used in RIS generation, involves random trials of edge into nodes reached so far to identify the set of nodes that are reverse reachable. The most direct manner that this may be implemented is that whenever an edge is encountered a random trial is performed so as to determine whether or not the diffusion successfully traverses the edge or not. In the case of the IC model this involves performing a random trial to check if the edge is live or dead and for the CTIC model it involves drawing the edge's time length and checking if it is less than time remaining until the time horizon.

A clear lower bound on the computational cost of constructing a sample is the number of nodes that are reached as this is the size of the set that must be returned in the case that we are constructing an rr-set. However, the computation time needed to construct such a sample can be more than an order of magnitude greater than this. This is because of the computation time spent on testing edges that fail which do not add to the reachable set size and yet can dominate the running time. Existing works have tackled this problem for the IC model [41, 65] however these techniques are specific to this model and do not generalize to the CTIC model.

In Chapter 2, we develop a more general approach that is based on efficiently testing groups of edges together to determine if any edge in the group succeeds. By assembling these group tests into a hierarchy we are able to efficiently identify the edges that succeed. Our approach has time complexity that scales linearly with the number of edges that succeed but only logarithmically with the number of edges that fail. This is in contrast with the standard approach of testing each edge independently which results in a time complexity that scales linearly with the number

of edges that fail. We apply our technique to the CTIC model to accelerate the edge testing needed to generate RIS samples. We demonstrate the practical effectiveness of our approach on a variety of real world data sets where we achieve a speedup of over an order of magnitude compared to independently testing the encountered edges. The ability to greatly accelerate RIS sample generation is valuable for efficiently solving the IM problem, as the running time of the existing state-of-the-art algorithms is dominated by the time needed to generate the collection of RIS samples required, as we report in Section 3.7.

1.2.2 Memory Efficient Influence Maximization via Streaming

Existing RIS based approaches [45, 71, 83–85] fundamentally depend on storing a large collection of RIS samples so that the greedy algorithm may be applied to solve the resulting max-k cover problem instance that is produced. Despite the successes that the application of RIS for solving the IM problem has enjoyed, it is prone to becoming highly memory intensive under certain circumstances. Specifically, the worst case memory complexity is known to scale linearly with the number of nodes one wants to select, as such doubling the number of nodes one wants to select will result in the require memory being doubled. This may not be viewed as a concern in the traditional setting of viral marketing that typically only considers seed sets of sizes of up to a hundred. However, if one wants to extend the the problem of IM to considering the impact of online advertising, which can easily involve thousands or tens of thousands of impressions, existing algorithms will be prone to running out of memory unless solution quality is sacrificed as we show in Section 3.7.

In Chapter 3, we present a novel algorithm based on optimization of a pair of fractional objectives that enables us to process the RIS samples as a stream and in doing so completely avoid the need to store them. In addition, we establish a means to compute both a lower bound on the coverage of the set of nodes our algorithm returns as well as an instance specific upper bound on the coverage of the optimal solution. This upper bound enables us to provide instance specific guarantees on the quality of the returned solution compared to the optimal solution. In our experiments that we perform on a variety of real data sets we show that not only does

our approach completely eliminate the large memory cost associated with storing the large RIS collection but it does so in a manner that its running time and attained coverage remains competitive with existing state-of-the-art algorithms. In addition, the instance specific guarantees that we are able to provide are found to be superior to that which the best existing approach achieves. This is attributed to the upper bound on the optimal solution that we are able to establish via the optimization of a fractional objective being tighter than the best upper bound on the optimal solution that can be established using information from the greedy algorithm.

1.2.3 Recycling random draws for Efficient Reachability Estimation

Reachability estimation, also known as reliability, has primarily been addressed via MC simulation with more advanced approaches hybridizing sampling with analytical diffusion accounting [48, 62]. While such approaches achieve substantial efficiency improvements over simply applying MC simulation they are ultimately still subject to the same limitation, namely the relative error of their estimates being impacted by reachability probability being estimated. Interestingly, a technique of utilizing geometric sampling to perform sampling under the IC model [65] more efficiently was found to be surprisingly competitive with other approaches designed specifically for reliability in [54]. Note that this approach is not fundamentally different than MC simulation. It simply makes the generation of samples much more efficient by avoiding needing to perform repeated Bernoulli draws and instead drawing from a geometric variable which determines the next trial on which the edge will be live. However, it turns out that the amount of random trials that do not need to be performed as a result of utilizing geometric sampling in the place of Bernoulli draws increases the smaller the edge probabilities are. As a consequence the MC simulation samples require less computational work to construct which helps to offset the need for more such samples.

In Chapter 4, we develop a novel approach that is designed to have the key property that both it's relative error and running time is not impacted by the probability that is being estimated. To do this we take inspiration from how geometric sampling successfully achieves this property at the edge level (i.e. the number of geometric samples from an edge needed to produce an estimate with a desired rel-

ative error is not impacted by the edge's probability). This is made possible by only successful samples, i.e. when the edge is live, contributing to the running time; cases were the edge is dead incur no running time cost. We successfully achieve similar behavior for graph reachability using a technique that we refer to as *random draw recycling*. Specifically, we are able to efficiently generate unbiased samples of the *indices* of the first ℓ possible worlds in which the source node reaches the target node in a manner that the computation cost scales only with ℓ, not the probability of the source reaching the target. As with geometric sampling the success indices, i.e. indices of worlds in which the source reaches the target, can be used to estimate the probability of the source reaching the target. We perform experiments on a variety of real datasets and demonstrate the effectiveness of our approach. In particular, we investigate not only the average relative error, as done by prior works, but also the *distribution* of relative errors over a variety of randomly selected source-target (s-t) pairs. This highlights the ability of our approach to not only achieve low average relative error but to achieve consistently low relative error over all s-t pairs, in contrast to existing approaches.

1.3 Outline

In Chapter 2 we present our approach to accelerate sample generation on uncertain graphs through the use of testing groups of edges together. In Chapter 3 we describe our novel approach to the IM problem that enables processing RIS samples as a stream and in doing so avoid the large memory cost incurred by existing approaches that require storing a collection of RIS samples. In Chapter 4 we propose a novel technique for estimating reachability in a manner that provides consistently accurate estimates whose relative error does not depend on the probability being estimated, in contrast with existing approaches. Finally, in Chapter 5 a summary of the dissertation is given and future research directions are discussed.

Chapter 2

Efficient Sample Generation

2.1 Introduction

Sampling provides a fundamental way to operate on uncertain graphs in a tractable manner. Under the *possible world* semantics an uncertain graph may be understood as a distribution over deterministic graphs. A deterministic graph has no uncertainty in what edges are present and which nodes each node can reach. Sampling amounts to drawing a possible world (i.e. deterministic graph) at random from this distribution. Importantly, the graph properties of reachability, does there exist a path from a source node to a destination node, and coverage, how many nodes does a source node set reach, can be efficiently computed on a *deterministic graph*. In contrast, exact computation of the reachability probability, also known as reliability, as well as the coverage, or spread, of a node or node set on an *uncertain graph* are both known to be #P-hard [15, 86, 87]. Hence, evaluating the expectation over the distribution of deterministic graphs, represented by the uncertain graph, is intractable to compute. However, we can approximate this quantity by draw samples of possible worlds and computing an average across these possible worlds.

Some Context. Monte Carlo (MC) simulation (see Definition 4), as used by Kempe et al. [55] and subsequent works for tackling the influence maximization (IM) problem (see Problem 2), corresponds to sampling possible worlds so to identify for a given set of source nodes how many other nodes in the graph they

are able to reach. This enables approximation of the coverage, also known as *influence spread*, of a candidate set of sources. This is fundamentally needed to solve the IM problem, which asks one to find a set of nodes, referred to as a seed set, subject to a cardinality constraint that achieves maximum coverage. Unfortunately, the coverage estimation of each new set of sources restarts from scratch. The standard application of the greedy algorithm to solve the IM problem [55] requires coverage estimation of many candidate source sets to find the candidate source node that when added to the currently seed set results in the largest marginal gain in coverage. Consequently, the use of MC simulation for coverage estimation quickly become very computationally expensive.

Recently, reverse influence sampling (RIS) (see Definition 5), also referred to as rr-sets [84], introduced by Borgs et al. [7] has provided a new perspective for coverage approximation and led to a class of current state-of-the-art algorithms for the IM problem [45, 71, 83–85]. MC simulation may be viewed as simulating diffusion through the graph in the 'forward' direction; starting from the set of source nodes the nodes that they are able to reach are identified. In contrast, RIS sampling operates in the 'reverse' direction; for a randomly selected target node the set of nodes that are able to reach it in the sampled possible world are identified. Interestingly, the coverage of set of nodes can be approximated by its coverage of RIS samples as shown by [7, 84] (see Lemma 1). The critical advantage of RIS over direct application of MC simulation is the ability to reuse the RIS samples to approximate the coverage of any given set of nodes. In this chapter we will focus on improving the efficiency of generating RIS samples. However, due to the symmetry between performing forward and reverse diffusion simulation the techniques developed in this chapter are also applicable to accelerating MC simulation.

Efficient Sampling. To begin with, it is important to notice that it is not necessary to sample a full possible world to determine the set of nodes that can reach the RIS target node. This is leveraged in existing works by only performing random trials when edges are encountered by the reverse graph traversal that is started at the target node. In the case of the independent cascade (IC) model this involves performing a random trial to check if the edge succeeds, (i.e. it is 'live') or fails (is dead). For the continuous-time independent cascade (CTIC) model it involves drawing the edge's time length and checking if it is less than the time remaining to

the time horizon. As such, the fundamental operation that needs to be performed for each newly reverse-reached node is: determine the subset of the edges in-bound on the node that succeed (i.e. enable diffusion to occur across them).

The simplest way to do this is test each edge independently. However, this can become highly inefficient on large graphs if the vast majority of edges do not succeed. Such a situation is in fact not unlikely given that large social networks are highly connected yet one does not expect a single user to be able to typically start a spread of information that reaches the majority of users in the network. Hence, it can be inferred that, on average, edges have a low chance of succeeding. Ideally, the time complexity of constructing an RIS sample should only scale with the number of reverse reached nodes, or similarly the number of edges that succeed. This is a straightforward lower bound on the required compute time as the set of reverse reached nodes must be returned. In contrast, the approach of testing each edge independently has a time complexity that scales with the total number of edges encountered. In particular, it scales linearly with the number of failed edges, which may vastly outnumber the number of succeeding edges. This causes the time needed to test failed edges to dominate the running time.

The key to addressing this and enabling sampling to be performed in an efficient manner is to structure edge trials so as to be able to efficiently determine and eliminate the cases where edges fail without needing to consider each case individually. Existing works have tackled this problem for the IC model, which we discuss in more detail in Section 2.2. However, these approaches depend on specific properties of the IC model and as such cannot be extended to the CTIC model. In Section 2.3 we will present a more general approach that is based on the notion of testing a group of edges together in a manner that enables determining if there is any edge in the group of edges that succeeds. By structuring these group tests into a hierarchy we develop an approach that enables efficiently identifying the edges that succeed in a manner only depends logarithmically on the number of failed edges. In Section 2.4 to demonstrate the effectiveness of our technique we apply it to accelerating RIS sample generation under the CTIC model on a variety of real networks.

2.2 Related Work

Here we will review existing techniques for accelerating sampling under the IC model. These prior works also tackled this problem for the purpose of improving the efficiency of RIS sample generation.

Geometric Sampling. Li et al. [65] proposed an approach that they refer to as lazy propagation sampling which only probes the edges when they are activated (i.e. succeed). This is built on being able to determine the next trial on which each edge will be active. The drawing of geometric random variables from distributions parametrized by each edge's probability of being live enables directly sampling on which trial the edge's next activation will occur. It is straightforward to show that there is no statistical difference between using this technique compared to repeatedly performing Bernoulli trials to test if the edge is active or not. This is because of the Geometric distribution's definition which is based on Bernoulli trials. Observe that this approach cannot be extended to the CTIC model as it depends on the probability of an edge activating each time it is tested being identical. This is not the case for the CTIC model as the probability of an edge succeeding depends on the amount of time remaining until the time horizon is reached. The amount of time remaining will vary between cases where an edge is encountered. As such, the probability that an edge succeeds will also vary. This prohibits the technique of geometric sampling from being successfully extended to the CTIC model.

Subset Sampling. In [41] a connection is made between subset sampling and the determining of which edges to a node's in-neighbors are live. In the case that all of the in-neighbor edges have the same probability of being live the problem of identifying the live edges can be mapped directly to the subset sampling problem. Furthermore the authors observe that when the probabilities are the same, the subset sampling can be effectively solved with geometric distribution sampling. The main advantage is that this enables skipping nodes that are not sampled (i.e. the edges that are dead), saving computational time. The authors mention that this approach may be extended to general edge weights by adding an edge pre-processing step. However, this fundamentally relies on the probability of an edge being live each time it is encountered being the same. As has already been discussed this is not the case under the CTIC model and as such this approach is not applicable to

it.

2.3 Our Approach

Reverse influence sampling (RIS) involves the construction of samples (also referred to as rr-sets). Under the continuous-time independent cascade (CTIC) model the core operation that this requires is drawing an edge's edge length from its associated edge length distribution and checking if the drawn length is less than the remaining time horizon. This determines whether or not the edge is traversed and then the reverse-reached node is added to the set of reverse-reachable nodes that make up the RIS sample. We will say an edge test succeeds if its edge length is less then the remaining time horizon; otherwise it fails. Ideally, the construction cost of an influence sample should be proportional to its size. However, when the probability of an edge succeeding is low, the time spent on testing edges that fail can dominate the running time. Only each succeeding edge can add a node to the sample (i.e. the set of reverse reachable nodes), hence *the construction cost of an influence sample can be much greater than its size if each edge is tested independently*. To mitigate this, we present an approach for efficiently finding the edges that succeed.

The key idea is to form a binary hierarchy of edge groups, with each hierarchy node corresponding to a test to see whether any edge in the *group* succeeds. The root corresponds to the set of all in-bound edges E_u of a current node u. The left and right child of a node x represent a disjoint partition of the group of edges associated with x, and so on recursively (example in Figure 2.1). This tree will enable us to find all succeeding in-bound edges of node u efficiently:

Theorem 1. *Group edge testing enables identification of the succeeding in-bound edges of node u in time $O(b \log_2 |E_u|)$, where b is the number of edges that succeed, provided each group test is a constant time operation.*

Proof. To see this, consider starting from an instance where all edges fail. Observe that this base case requires only one group test as the root group test immediately determines all edges in the group to be dead. We will now bound the number of additional group tests that are performed as the number of succeeding edges is increased. Observe that the addition of a succeeding edge results in a path of

succeeding group tests from the root of the tree to the leaf that represents that edge. The length of a path from the root of the tree to a leaf is $\log_2 |E_u|$. Hence, at most $O(\log_2 |E_u|)$ additional group tests succeed for each succeeding edge that is added. Note that as more succeeding edges are added the number of additional succeeding group tests added per additional succeeding edge can only be less than this amount due to their paths overlapping. Furthermore, for each succeeding group test there is at most one additional failing group test. This failing group test rules out the branch of the tree that the succeeding edge is not in and prevents further group tests from being needed. It follows that if there are b succeeding edges they can be identified by performing $O(b \log_2 |E_u|)$ group test. If each group test is performed in constant time then the overall running time is $O(b \log_2 |E_u|)$. □

In order to realize this, we must be able to pre-compute a test that can be used to check, *in constant time*, whether there exists any edge in a group that succeeds. The details of this test depend on the edge length distribution at hand. We consider the Weibull distribution that has been previously used in works on CTIC model [22, 85]. The Weibull distribution also has often been used in survival analysis to model lifetime events [24]. The Weibull distribution is a generalization of both the exponential and the rayleigh distributions. The exponential distribution in survival analysis corresponds to a constant hazard rate (i.e. the instantaneous probability of an active node activating a neighbor node is constant over time). The shape parameter of the Weibull distribution enables representation of a hazard rate that either increases or decreases with time. This gives it more flexibility and expressive power. Prior work has applied the exponential, rayleigh and power-law distributions for modeling the spread of information (specifically 'memes'; short textual phrases) in social media [37]. In Section 2.5 we will discuss the possibility of extending our techniques to other distributions.

2.3.1 Group Test Construction

The Weibull distribution probability density (pdf) and cumulative density functions (cdf) with parameters λ, ℓ, are,

$$f(x) = \frac{\ell}{\lambda}\left(\frac{x}{\lambda}\right)^{\ell-1} e^{-(x/\lambda)^{\ell}} \tag{2.1}$$

$$F(x) = 1 - e^{-(x/\lambda)^{\ell}} \text{ if } x \geq 0, \text{ and 0 otherwise.} \tag{2.2}$$

Weibull distributions have the following property.

Lemma 2 ([8]). *If $X_i \sim$ Weibull(λ_i, ℓ), for $i = 1, 2, ..., n$ and $X_1, X_2, ..., X_n$ are independent random variables, then,*

$$\min\{X_1, X_2, ..., X_n\} \sim \textit{Weibull}\left(\left[\sum_{i=1}^{n}(1/\lambda_i)^{\ell}\right]^{-1/\ell}, \ell\right)$$

Using this property for Weibull distributions that have different scale parameters λ_i, but the same shape parameter ℓ, we can construct an efficient group test using the probability $\Pr[\min\{X_1, X_2, ..., X_n\} > T]$, where T is the remaining time until the time horizon. Clearly, if the minimum edge length in a group of edges exceeds the threshold (remaining time), it implies that all edges fail and so none of the edges needs to be considered further (e.g., see step ⑧, Figure 2.1). We will refer to the scale parameter of the Weibull distribution that results from taking the minimum over the Weibull random variables in the group (i.e. $\left[\sum_{i=1}^{n}(1/\lambda_i)^{\ell}\right]^{-1/\ell}$) as the *group edge length scale*.

By pre-computing the group edge length scales, groups of edges can be tested in constant time. For an arbitrary group node in the hierarchy (see Figure 2.1 for an example), let L and R be the set of edges of its left and right children respectively. Then we can pre-compute the following:

$$a_L = \sum_{i \in L}(1/\lambda_i)^{\ell}, \qquad a_R = \sum_{i \in R}(1/\lambda_i)^{\ell}$$

Having these quantities precomputed enables the group test, i.e., checking whether the group's minimum edge length is less than a given remaining time limit, to be done in constant time. This is done using a random value r in $U(0,1)$, representing the randomness in length of the minimum edge, which we can then use in

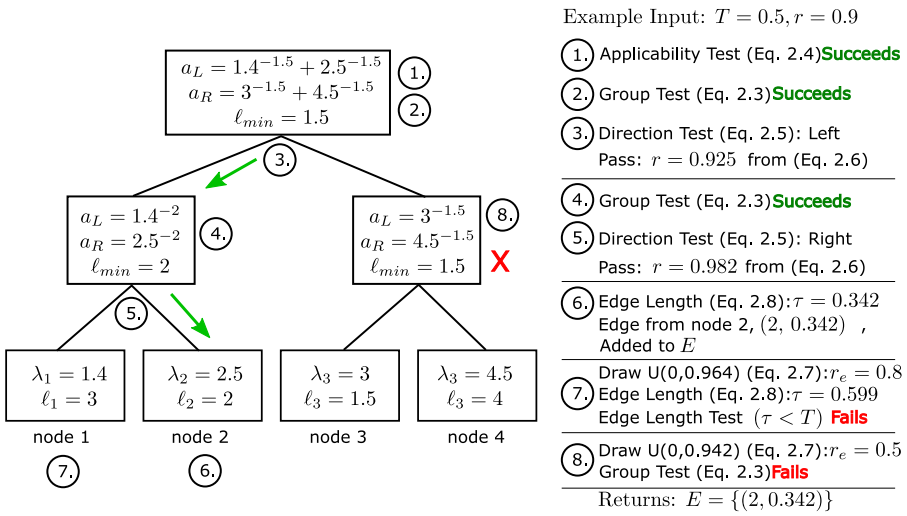


Figure 2.1: Group edge testing Example showing the interaction of the applicability test, group test and direction test as the edges that succeed are identified. Edges associated with nodes 3 and 4 do not need to be tested directly as the group test failing rules out the possibility that either has an edge length less than the time limit.

the following test:

$$\begin{aligned} &\Pr[\min\{X_1, X_2, ..., X_n\} > T] < r \text{ where,} \\ &\Pr[\min\{X_1, X_2, ..., X_n\} > T] = \exp(-T^\ell \textstyle\sum_{i \in L \cup R} (1/\lambda_i)^\ell) \\ &\qquad\qquad = \exp(-(a_L + a_R) T^\ell) \end{aligned} \tag{2.3}$$

In general, the shape parameters may not be the same. To deal with this, we can use the fact that for any λ parameterizing, for a random variable $X \sim \text{Weibull}(\lambda, \ell)$, the probability that the edge length is greater than the time limit T, is $\Pr[X > T] = e^{-(T/\lambda)^\ell}$, which is monotonically increasing in ℓ when $(T/\lambda) < 1$. As such, we may consider random variables, $Y_i \sim \text{Weibull}(\lambda_i, \min_j \ell_j)$ for which we have,

$$\Pr[\min\{Y_1, Y_2, ..., Y_n\} > T] \leq \Pr[\min\{X_1, X_2, ..., X_n\} > T]$$

provided, $T \cdot [\sum_{i=1}^{n} (1/\lambda_i)^{\ell}]^{1/\ell} < 1$ or equivalently,[1]

$$(a_L + a_R)T^{\ell} < 1 \tag{2.4}$$

Below, we will refer to the condition in Eq. 2.4 as the *group test applicability condition*. *This should not be confused with the group test itself.* The minimum shape can only underestimate the minimum edge length when this property is satisfied. This ensures that the group test retains the property that it succeeds whenever there is an edge in the group that succeeds. In addition, we have the following result:

Lemma 3. *If the group test applicability condition succeeds for a group it also succeeds for all of its subgroups.*

Proof. Without lost of generality we will consider the left child of a group. From the applicability condition succeeding we have, $T \cdot [\sum_{i \in L \cup R} (1/\lambda_i)^{\ell}]^{1/\ell} < 1$. Using this and the fact that the scale parameters are non-negative we have that, $T \cdot [\sum_{i \in L} (1/\lambda_i)^{\ell}]^{1/\ell} < 1$. What remains to be shown is that increasing ℓ can only decrease the left hand side (the minimum shape of a child group may only be equal to or larger than the minimum shape of the parent group).

Consider ℓ' such that $\ell \leq \ell'$ and let $\sum_{i \in L} (1/\lambda_i)^{\ell}]^{1/\ell} < c$ for $c > 0$ then, $\sum_{i \in L} (1/\lambda_i)^{\ell} < c^{\ell}$. For all i, $(1/\lambda_i)^{\ell} < c^{\ell}$ must hold, since all terms are positive, and hence $(1/\lambda_i) < c$ must also hold. Now multiplying both sides by $c^{\ell'-\ell}$ gives, $\sum_{i \in L} (1/\lambda_i)^{\ell} c^{\ell'-\ell} < c^{\ell'}$. Since $(1/\lambda_i) < c$ for all i it follows that $(1/\lambda_i)^{\ell'-\ell} < c^{\ell'-\ell}$. From this we have, $\sum_{i \in L} (1/\lambda_i)^{\ell'} < c^{\ell'}$, which gives, $[\sum_{i \in L} (1/\lambda_i)^{\ell'}]^{1/\ell'} < c$. □

2.3.2 Group Test Application

Algorithm 1 makes use of the group test hierarchy to efficiently find the edges that do succeed, i.e., they have time lengths less than the given time horizon, T, as well as the time lengths of these edges. Recall, as an influence sample is constructed it repeatedly needs to identify the edges that can be traversed from a node within the remaining time. As opposed to naively checking each edge from a node independently, group edge testing (Algorithm 1) is used. No further changes to the

[1] Notice that $1/\lambda = [\sum_{i=1}^{n} (1/\lambda_i)^{\ell}]^{1/\ell}$.

Algorithm 1 GroupTest

Input: $A, T, r,$ check_applicability
Output: E

$E \leftarrow \emptyset$;
if is_leaf(A) **then**
  $(v, \lambda, \ell) = A$; {Expand node as tuple}
  $\tau = \lambda(-\log(r))^{1/\ell}$;
  if $\tau < T$ **then** {Edge Length Test}
    $E \leftarrow E \cup \{(v, \tau)\}$;
else
  if check_applicability **then**
    $(A_L, A_R, a_L, a_R, \ell_{min}) = A$; {Expand node as tuple}
    if $(a_L + a_R)T^{\ell_{min}} < 1$ **then** {Applicability Test (Eq. 2.4)}
      $E \leftarrow E \cup \text{GroupTest}(A, T, r_e, \text{false})$;
    else
      $r_L \sim U[0,1]$; $r_R \sim U[0,1]$;
      $E \leftarrow E \cup \text{GroupTest}(A_L, T, r_L, \text{true})$;
      $E \leftarrow E \cup \text{GroupTest}(A_R, T, r_R, \text{true})$;
  else
    $(A_L, A_R, a_L, a_R, \ell_{min}) = A$; {Expand node as tuple}
    if $r > \exp(-(a_L + a_R)T^{\ell_{min}})$ **then** {Group Test (Eq. 2.3)}
      $r_L = \exp(\log(r) \cdot a_L/(a_L + a_R))$;
      $r_R = \exp(\log(r) \cdot a_R/(a_L + a_R))$;
      $r_d \sim U[0,1]$;
      if $r_d < a_L/(a_L + a_R)$ **then** {Direction Test (Eq. 2.5)}
        $E \leftarrow E \cup \text{GroupTest}(A_L, T, r_L, \text{false})$;
        $r_e \sim r_R \cdot U[0,1]$;
        $E \leftarrow E \cup \text{GroupTest}(A_R, T, r_e, \text{false})$;
      else
        $E \leftarrow E \cup \text{GroupTest}(A_R, T, r_R, \text{false})$;
        $r_e \sim r_L \cdot U[0,1]$;
        $E \leftarrow E \cup \text{GroupTest}(A_L, T, r_e, \text{false})$;

rest of influence sample generation algorithm are needed. A group test node A in the hierarchy is represented by the tuple $(A_L, A_R, a_L, a_R, \ell_{min})$ where A_L and A_R are the left and right children of the group node and the values a_L, a_R, ℓ_{min} have been precomputed as described in §2.3.1. Notice that a leaf node of the tree corresponds to an edge of the original influence graph G and is represented by the tuple (v, λ, ℓ)

where v is the id of the node of G that the edge is out-bound from, and λ and ℓ are the scale and shape parameters of the Weibull distributions. The argument r of Algorithm 1 is set to a random value drawn from $U[0,1]$ by the initial invocation. This argument is important for the recursive calls as the hierarchy is traversed, which we will describe below.

Algorithm 1 checks the group test applicability condition on line 10 (the condition is the same as Eq. 2.4). If the applicability test fails, the group is split and handled by recursive calls on the children of the group (lines 13-15). If the applicability test succeeds, then the check_applicability flag is cleared enabling the group test to performed on the following recursive call. Line 18 checks the group test condition. If this condition fails, then the minimum length edge out of all of those that make up the current group exceeds the time limit and as such the group does not need to be considered any further. *This is the key step that allows ruling out groups of edges without needing to test edges individually.*

As Algorithm 1 is performing the group tests, the random draw that is initially provided must be propagated down the tree as it specifies the length of the minimum length edge. However, we must identify which edge in the group it corresponds to. To propagate the min length edge down the tree at each node it must be determined which branch it originates from. This is decided according to the probability $\Pr[\tau_L < \tau_R]$, i.e., the probability that the min length edge in the left group is less than the min length edge from the right group. We have, $\tau_L \sim \text{Weibull}(a_L^{-1/\ell}, \ell)$, $\tau_R \sim \text{Weibull}(a_R^{-1/\ell}, \ell)$, where $\ell = \min_{i \in L \cup R} \ell_i$. Notice that $\ell = \ell_{min}$. As such,

$$\begin{aligned}
\Pr[\tau_L < \tau_R] &= \int_0^\infty \int_x^\infty f_L(x) f_R(y) dy dx \\
&= \int_0^\infty f_L(x)(1 - F_R(x)) dx \\
&= \int_0^\infty a_L \ell x^{\ell-1} e^{-a_L x^\ell} e^{-a_R x^\ell} dx \\
&= -a_L/(a_R + a_L) e^{-(a_L + a_R)x^\ell} \Big|_0^\infty \\
&= a_L/(a_R + a_L) \qquad (2.5)
\end{aligned}$$

A new random draw is used to determine the branch that the min edge is from according to this probability. This direction test is performed on line 22 and steps

③ and ⑤ in Figure 2.1.

Given the branch that the min edge came from, the random drawn value must be rescaled to ensure that the min length retains the same length even though it is now determined to be from a different distribution. Without loss of generality, assume the min length edge came from the left branch. Let r be the random value that was used in the group test at the parent node. Then the random value that will be used in the left branch group, r_L, must satisfy, $S_L^{-1}(r_L) = S_{L\cup R}^{-1}(r)$ where $S_L(x) = 1 - F_L(x)$ and $S_L^{-1}(x) = (-\log(x)/a_L)^{1/\ell}$, which gives the time length produced by the Weibull distribution from a random value $x \sim [0,1]$. Solving for r_L in terms of r,

$$\begin{aligned}(-\log(r_L)/a_L)^{1/\ell} &= (-\log(r)/(a_L+a_R))^{1/\ell} \\ \log(r_L)/a_L &= \log(r)/(a_L+a_R) \\ r_L &= \exp(\log(r)\cdot a_L/(a_L+a_R)) \end{aligned} \tag{2.6}$$

This is used to compute the value passed in step ③ of Figure 2.1 (the symmetric right case is used for step ⑤).

In addition, we must condition the other branch on the fact that its minimum edge length now must be larger than this minimum edge length. This is done by restricting the range that the random values are drawn from. Similar to r_L, we have r_R, which is the random value that would produce the minimum edge length.

$$r_R = \exp(\log(r)\cdot a_R/(a_L+a_R)) \tag{2.7}$$

Instead of drawing the random value for the right branch from $[0,1]$ we will now draw from $[0,r_R]$ which will ensure that random edge lengths that can be produce satisfy the condition. This is shown by range of the draws performed in steps ⑦ and ⑧ of Figure 2.1. These values are computed on lines 19-20. The branch that contains the min edge is recursed on using the existing random value while the branch that doesn't contain the min edge is recursed on as a new group with the the new random value r_e drawn such that all possible edge lengths from this group satisfy the length lower bound.

Once a random draw has been propagated down to a leaf node it is used to

Dataset	type	nodes	edges	memory
Youtube	undirected	1.1M	3.0M	120MB
Pokec	directed	1.6M	30.6M	410MB
LiveJournal	directed	4.8M	69.0M	1.0GB
Orkut	undirected	3.1M	117.2M	2.8GB
Twitter	directed	41.7M	1.5G	18.3GB

Table 2.1: Datasets (M = Million, G = Billion)

compute the edge length to the edge's source node (Lines 3-6 and steps ⑥ and ⑦ of Figure 2.1).

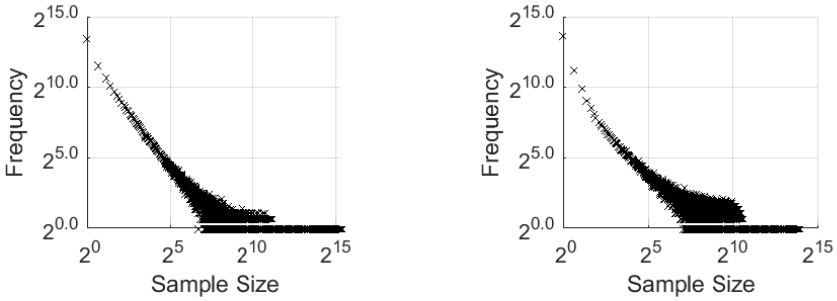
$$\tau = \lambda(-\log(r))^{1/\ell} \tag{2.8}$$

Since ℓ_{min} increases as the hierarchy is descended even though the upper level group tests determined there was an edge that is below the threshold in a group it may be ruled out by the lower level group tests or the final leaf level test that checks the final edge length against the threshold (line 5). All of the edges that succeed and their associated time lengths are collected in the set E which on completion of the group test is returned to the sample construction algorithm.

2.4 Experiments

The goal of our experiments is to assess the effectiveness of our proposed group edge testing approach at accelerating reverse influence sampling (RIS) sample generation under the continuous-time independent cascade model (CTIC). As the main application of RIS is for tackling the problem of influence maximization (IM) our experiments focus on real world social networks and configure the diffusion model to align with non-trivial instances of the IM problem. In particular, it is necessary to avoid configurations that results in very little diffusion occurring or cases where diffusion saturates, i.e. covers the entire network.

Datasets. We perform experiments on five real and public social networks (see Table 2.1). For the undirected networks, Youtube and Orkut, we replace each edge by two directed edges enabling each direction to have different parameters. *YouTube*, *Pokec*, *LiveJournal* and *Orkut* can be obtained from [60] and *Twitter* from [58].



(a) LiveJournal T=0.42 (Avg. Inf. Sample Size 256)

(b) YouTube T=0.6 (Avg. Inf. Sample Size 1024)

Figure 2.2: Size distribution of 1M Influence Samples

Diffusion Model. Under the CTIC model [35] each edge has an associated length distribution. We use the *Weibull distribution* as done in [22, 85], which has the following probability density function: $f(x) = \frac{\ell}{\lambda}\left(\frac{x}{\lambda}\right)^{\ell-1} e^{-(x/\lambda)^{\ell}}$. For each directed edge we sample uniformly at random the parameter ℓ from $[1,10]$ and λ from $[0,10]$. Figure 2.2 shows examples of the resulting influence sample frequency distributions. The distribution should be non-degenerate (i.e., singleton samples should not dominate) as such a situation corresponds to a trivial instance of very little diffusion occurring. Furthermore, if the distribution is skewed right (i.e., large samples dominate) such a situation corresponds to network saturation, which is the other case we wish to avoid. In [22, 85] ℓ (referred to as a) was selected from $[0,10]$, which was perhaps appropriate for the small networks they considered. However, we observed that on large networks this resulted in irregular influence sample frequency distributions, having multiple modes, due to a non-negligible number of edges having extremely short time lengths with high probability. This corresponds to one of the undesirable cases of large samples dominating, which is indicative of the network saturating. Consequently, we sample ℓ uniformly at random from $[1,10]$, which we found to avoid this issue.

In addition to edge lengths, the CTIC model also has a time horizon parameter T. For given edge length distributions and network structure, increasing the time horizon T increases the expected depth to which influence will propagate from the seed set. To investigate the effect of this on the performance of the sample

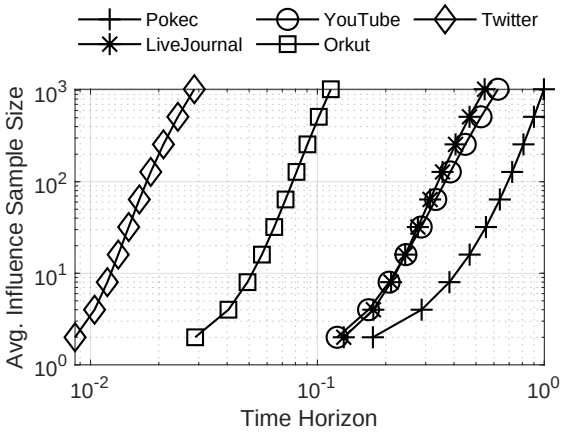


Figure 2.3: Effect of time horizon on Influence Sample size.

generation, we consider a range of time horizons for each dataset. Importantly, a reasonable range of time horizons must be identified specifically for each network and associated edge length distributions. An inappropriate range of time horizons would result in increasing from very little coverage to network saturation in a single increment, providing little insight into the performance in the intermediate regime. To avoid these issues, we calibrate the time horizons using the average RIS sample size. The time horizons considered for each dataset are those that yield average sample sizes of approximately $\{2^1, 2^2, ..., 2^9\}$. The use of exponentially spaced average sample sizes is to enable efficiently investigate a wide range of configurations. Figure 2.3 shows the correspondence between average influence sample size and the time horizon for each dataset. It is apparent that the time horizons that are appropriate for the smaller datasets (e.g. Pokec), if used on the larger datasets (e.g. Twitter), would yield impractically large RIS sample sizes, which corresponds to network saturation.

It is worth noting that increasing the time horizon under the CTIC model has an equivalent effect as scaling the time length distributions of the edges. As such, considering a range of time horizons is sufficient to investigate the effectiveness of the proposed sample generation in such situations. Ultimately what impacts how sample generation behaves is how far through the network diffusion propagates which is impacted in an identical manner by edge length scaling and time horizon

	Pokec	LiveJournal	Youtube	Orkut	Twitter
Average	2.3	3.8	4.3	9.1	32.2
Min	2.0	2.3	2.3	6.9	16.3
Max	3.1	4.7	5.8	10.9	46.1

Table 2.2: Summary of speedups of proposed group edge testing compared to independent testing of edges across datasets. Average, min and max reported is over the range of time horizons considered for each dataset.

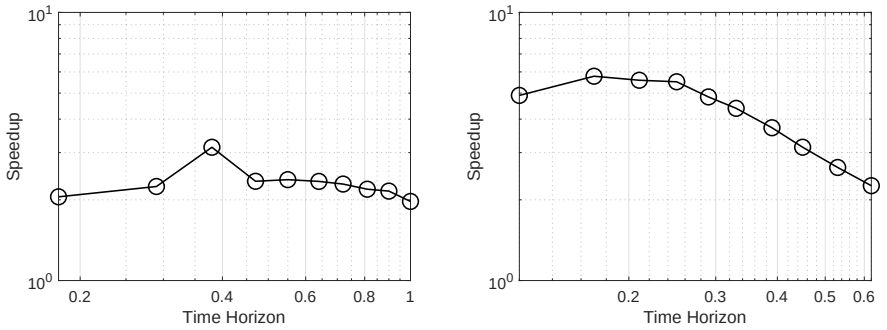
scaling.

Figures 2.4a, 2.4b and 2.4c show the speedup achieved by our proposed approach compared to the prior existing approach, which involves testing edges independently. The speedup is measured by comparing the time required to generate one million influence samples. It can be observed that our approach achieves a substantial speedup on all datasets across the wide range of time horizons considered. Although the speedup is observed to decrease at high time horizons the speedup remains appreciable. Furthermore, this is expected as once the fraction of encountered edges that fail begins to diminish the benefit of avoiding testing edges that fail will diminish.

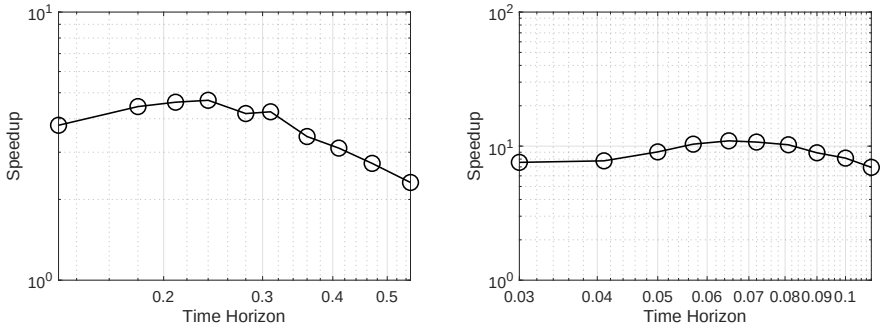
Table 2.2 summarizes the speedups attained on each dataset over the range of time horizons considered for that dataset. Recall that the running time of our approach only scales linearly with the number of succeeding edges (not the failed edges) while the baseline also scales linearly with the number of failed edges. As such, the larger the fraction of the edges in the graph that fail the greater the speedup of our approach may be expected to be. It can be inferred that a large fraction of the edges in the large graphs must fail otherwise the diffusion would saturate. Consistent with this, it is observed that group edge testing yields the greatest benefit on the larger datasets. The speedup on these datasets is very substantial – nearly an order of magnitude for Orkut and well over that for Twitter.

2.5 Discussion and Conclusions

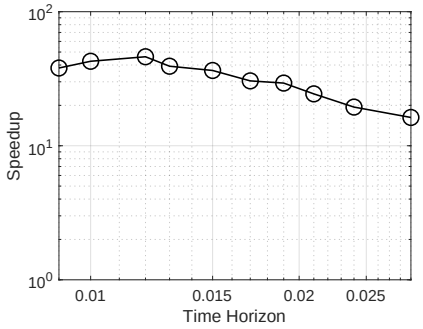
We have proposed a technique, that we refer to as group edge testing, for improving the efficiency with which sampling can be performed on uncertain graphs. We have demonstrated its effectiveness in speeding up the generation of RIS samples



(a) Speedup on Pokec (left) and Youtube (right).



(b) Speedup on Live Journal (left) and Orkut (right).



(c) Speedup on Twitter.

Figure 2.4: Speedup of proposed group edge testing compared to independent testing of edges. Speedup is assessed w.r.t. time horizon which controls the length of time diffusion occurs for. Higher time horizons yield larger RIS samples.

under the CTIC model. Efficient RIS sample generation is valuable for improving the state-of-the-art approaches to solving the influence maximization (IM) problem. As will be shown in Section 3.7 state-of-the-art approaches for IM have a running time that is dominated by the time needed to generate RIS samples. Consequently, the speedups that we attain for RIS generation will directly translate to nearly equivalent speedups of IM algorithms based on RIS. Although there are existing works that tackle accelerating RIS sample generation none of them are applicable to the CTIC model. It is also worth noting that the technique proposed here is not limited to RIS sample generation. For instance identifying the successful edges is also the fundamental operation required by 'forward' MC simulation of diffusion from a source node set. As such it would benefit in a similar manner. Although MC simulation is not used as a component of state-of-the-art approaches to IM it remains a useful technique for evaluating the coverage of a specific set of nodes of interest.

While we chose to focus on the CTIC model and the Weibull distribution for specifying the edge lengths the technique of group testing is not fundamentally dependent on these particular choices. The key property that is needed is the ability to establish an efficient test that checks if there is no edge in a predefined group of edges that succeeds. For the CTIC model this more concretely amounts to being able to sample the shortest edge length from a group of edges using compute time that does not scale with the number of edges in the group. We expect that the required group tests can be constructed for any edge length distributions for which this is the case in a similar manner as we have done here for the Weibull distribution. Furthermore, this is not a property that is fundamentally linked to the CTIC model. For instance, it would not be difficult to apply our technique to the IC model. It is straightforward to pre-compute the probability that every edge in a group of edges under the IC model does not succeed, which is sufficient information to establish the needed group test. We have chosen to focus on the CTIC model as it is not handled by prior works where as the IC model is already sufficiently addressed by simpler existing techniques.

Chapter 3

Memory Efficient Influence Maximization

3.1 Introduction

The problem of *Influence maximization* (IM) is to find k nodes in a network $G = (V,E)$, with vertices V and edges E, which can recursively influence the largest number of nodes, in expectation, under a given stochastic diffusion model. The k nodes are known as *seeds* or *initial adoptors*. This problem is motivated by applications such as viral marketing [20, 77], infection containment [61], feed ranking [46], personalized recommendation [82], and regulatory cell cycle analysis [31]. Since the seminal study of IM by Kempe et al. [55] as a discrete optimization problem, this problem has become popular in network analysis and has been extensively studied [7, 13–15, 17, 22, 38, 45, 49, 61, 63, 71, 73, 83–85, 87].

Some background. Kempe et al. [55] studied IM under discrete time diffusion models including the independent cascade (IC) model and the linear threshold (LT) model and their variants, drawn from the sociological and marketing literature [33, 34, 39]. It has been argued that a *continuous time* model may be more appropriate for modeling diffusion of product adoptions and infections and that it is significantly more accurate than discrete time models [21, 35–37]. In particular, discrete time models lack the ability to capture the time needed for propagation to

occur across links, which can differ for different links. As such, if the goal is to reach some influence spread within a given time frame, discrete time models lack the ability to capture this precisely.

IM is well known to be a computationally hard problem (see below). While scalable approximation algorithms have been proposed for IM over discrete time diffusion models, there are relatively few works that develop such algorithms for continuous time models [22, 85]. Unfortunately, even their performance evaluation has been restricted to relatively small networks. A recent study found that approaches that perform well on particular diffusion models (e.g., LT or IC with 1/in-degree edge weights) do not necessarily do so on others (e.g., IC with constant edge weights) [2]. In particular, the memory footprint of many algorithms was found to be prohibitive. Our experiments on the continuous time independent cascade (CTIC) model reveal similar memory limitations of the existing approximation algorithms, as observed in prior work [2, 74]. In this chapter, we propose a novel approach that resolves this memory limitation, without sacrificing running time and solution quality guarantee.

Hardness and Approximation. The IM problem is known to be NP-hard in general [55]. However, the objective function of expected number of influenced nodes, called *expected spread* or just *spread*, under these diffusion models, satisfies the key properties of *monotonicity* and *submodularity*. These properties enable a simple Greedy algorithm to attain a $(1-1/e)$-approximation to IM, if the exact expected spread for a given seed set is known. However, computing the spread of a given seed set is a hard problem, shown to be #P-hard [15, 87]. Kempe et al. [55] estimate the spread using Monte-Carlo (MC) simulations, thus ensuring a $(1-1/e-\varepsilon)$-approximation to the optimal solution to IM, but this approach does not scale to large networks and seed set sizes [84]. Reverse influence sampling (RIS) introduced by Borgs et al. [7] has led to a class of current state-of-the-art algorithms, which retain the $(1-1/e-\varepsilon)$-approximation guarantee [45, 71, 83–85] while often scaling to large networks. The main line of this series of works has been refining the theory that prescribes how many samples are required to attain a $(1-1/e-\varepsilon)$-approximate solution.

Online IM. In a recent paper, Tang et al. [83] study *online influence maximiza-*

tion, where they track an empirical solution quality guarantee w.r.t. the number of influence samples used. While they refer to their approach as using the reverse influence samples in a "streaming fashion", it involves *storing all of the samples received*. In this chapter, we use the term *streaming* in its standard usage [68], whereby *each sample is processed right after it is received, then it is removed from memory*. It has been observed in experiments that the attainable empirical guarantees can greatly exceed the theoretical worst case [61, 83]. Thus, merely aiming for a solution quality of $(1-1/e-\varepsilon)$ can result in solutions that significantly underperform what is efficiently attainable. As such, we will primarily focus on this *online IM* setting. We develop a fundamentally different approach to computing the empirical guarantee, which we empirically find to be superior.

Large seed sets. Traditionally, seeding a user for viral marketing is viewed as being expensive (e.g., providing a free product to the user), arguing for a small seed set size. On the other hand, online display advertising in social networks coupled with the lower cost of ad impressions, enables an advertiser to reach a substantially larger set of users, at a relatively small cost. E.g., typical CPM (Cost-per-thousand impressions) for Instagram ad and Facebook Ads is around \$5 and \$10 respectively [81]. Targeting of online display ads in social networks relies not only on the response of individual users, but also leverages the ripple effect produced by social influence [59, 88, 89]. As such, scaling influence maximization to be able to handle targeting of ad campaigns that involve placement of large numbers of impressions requires handling of very large seed sets.

Memory bottleneck. All existing algorithms that utilize influence sampling are prone to prohibitive memory usage as they need to store all the samples generated. The worst case memory complexity is known to be $O(k(m+n)\log(n)/\varepsilon^2)$ where n and m are respectively the number of nodes and edges in the network [71, 83–85]. The seed set size k has the most pronounced effect on the required memory.

We find that existing algorithms become infeasible at large seed set sizes unless solution quality is sacrificed. This memory complexity is a consequence of the number of samples required to accurately estimate the influence function. As such, even on "easy" instances, where empirical solution quality greatly exceeds the theoretical worst case guarantee, a relatively large number of samples are needed for

the empirical solution quality to saturate. Furthermore, to attain higher influence spread under the CTIC model, one may wish to allow the diffusion to continue to occur for a longer duration. However, this results in significantly larger reverse influence samples which exacerbates the memory bottleneck.

While a number of techniques have been proposed to address the memory overhead, they either rely on specific properties of the diffusion process [70, 75] or they incur additional inaccuracy in simplifying the IM instance, which can further degrade the solution quality[74]. We discuss these approaches in greater detail in §3.8. In this chapter, we seek to address this memory limitation by developing an approach that does not need to store the generated influence samples. Instead, they are processed as they are generated and then removed from memory (i.e., in a *streaming* manner). By doing so, *our proposed approach has memory complexity that is independent of the desired solution quality, diffusion process, and seed set size, in contrast to existing works.*

Breaking away from Greedy. To accomplish this, we need an algorithm that can progressively improve the solution quality (i.e., increase its expected spread), while it is running. The Greedy algorithm, used by the vast majority of existing works, is based on sequential selection which is not amenable to progressive improvement; once a node is added to the seed set it can not be removed. We thus break away from the Greedy approach and propose a novel algorithm, a *Fractional Approach to Influence Maximization* (FAIM), that builds on two fractional objectives – the *multi-linear extension* and the *fractional relaxation*, of the expected influence spread objective. The fractional relaxation (FR) is a concave function and its optimal solution upper bounds the optimal binary solution. On the other hand, the multi-linear extension (ME) is non-concave, its local optima are binary, and its global optimum coincides with the optimal binary solution, i.e., the optimal expected spread. Although the ME solution converges towards a local optimal binary solution, the FR solution converges towards *its* global optimal solution. We leverage the latter to ensure that the ME solution does not get stuck in an arbitrarily poor local optimum.

Our approach. To provide an online instance-dependent empirical guarantee, we must maintain an upper bound on the influence spread attained by the optimal

solution and a lower bound on that attained by our best candidate solution. The FR objective, being concave and having an optimal solution that upper bounds the optimal binary solution, provides a means to upper bound the optimal expected spread. However, prior to convergence, the current fractional solution to FR objective is *not* a valid upper bound. The gradient at the current solution could be used to compute a first-order Taylor approximation of the function, which by concavity overestimates the objective function everywhere. The optimal solution to the resulting constrained linear maximization problem can be easily obtained and would yield an upper bound. However, we only have an approximation of the influence function via influence samples, and do not have access to the exact gradient. We address this challenge by deriving an upper bound that holds with high probability and only requires a cumulative over the gradients produced by the influence samples. In addition, we maintain a lower bound by generating and evaluating candidate solutions to the IM problem.

Interestingly, we can leverage the linear maximization, used for computing the upper bound, for both generating candidate solutions and optimizing the fractional solutions. As we will show, the solution is always a binary vector that represents a seed set of size k. This may be interpreted as a candidate solution to the IM problem. In addition, we can construct the current fractional solution as the average over rounds of the linear maximization. Doing so ensures that the fractional solutions always remains feasible.

An important consideration is the time needed to process an influence sample and update the current solution. Ideally we would like sample processing to be sufficiently efficient to be dominated by its construction time (i.e., proportional to its construction cost). Since the seed set size can exceed the average influence sample size, a naive approach to adding a selected seed set to the cumulative solution would be impractical for large seed sets. To address this, we develop an efficient *lazy update* that is equivalent, but only requires time proportional to the influence sample size and at worst logarithmic in the seed set size.

Existing approaches have their time complexity dominated by influence sample construction cost. Applying the Greedy algorithm requires at most time proportional to the aggregate size of the samples. We observed that the running time needed to construct influence samples by naively testing each edge independently

could be dominated by testing a large number of edges that do not succeed. To address this we will make use of the efficient sample generation approach we developed in Chapter 2.

Our main contributions are as follows:

1. A non-greedy algorithm, FAIM, based on optimizing a pair of fractional objectives that avoids storing influence samples by processing them as a stream (§3.3 and §3.5). This resolves the memory bottleneck of existing approaches.

2. A high-probability upper bound on the optimal solution using the fractional relaxation and concentration bounds. This bound is found to be tighter than the best existing bound and yields instance-specific solution quality guarantees that far exceed the theoretical worst case (§3.4).

3. An efficient parallel implementation of FAIM (§3.6).

4. We conduct extensive experiments using a variety of diffusion models on networks of sizes up to 41M nodes and 1.5B edges. We test online processing performance of various algorithms and instance guarantee certificates for seed sets of over 70K nodes. Our experiments demonstrate the effiency and effectiveness of FAIM (§3.7).

Preliminary notions used in the paper are provided in §3.2, while related work is discussed in §3.8. We summarize the paper and discuss future work in §3.9.

3.2 Preliminaries

3.2.1 Influence Maximization

We review background on influence maximization (IM) and the problem as formulated under state-of-the-art approaches.

Influence Diffusion and Maximization. Given a graph $G = (V, E)$, a diffusion model is used to specify how propagation of influence occurs in the network G. Existing large scale IM experiments have primarily focused on the Independent Cascade (IC) and Linear Threshold (LT) *discrete time* models [55], where influence weights or probabilities are associated with edges. For a given set of seeds, the

(expected) influence spread (or just *spread*) is defined as the expected number of nodes that activate at the end of the propagation, as governed by the underlying diffusion model. Unfortunately, exact computation of the expected spread is #P-hard [15, 87].

A continuous time version of the IC model, the CTIC model (see Definition 2), has been studied in the papers [21, 35–37]. Under the CTIC model, each directed edge e in the graph is associated with an edge length probability distribution, $\mathscr{L}(e)$. Diffusion occurs up until a specified time horizon T; edges are traversed (by influence) so long as the total length traversed from a seed node is less than the horizon. As such, the spread of a seed set for a specified time horizon is the number of nodes reached by paths of length less than the horizon, in expectation. Expected spread under the aforementioned diffusion models is *monotone* (see Definition 6) and *submodular* (see Definition 7).

The IM problem requires finding a seed set of size at most k, for a given number k, that attains the maximum spread over a given network, under a given diffusion model (see Problem 2).

Reverse Influence Sampling. The key property of reverse influence sampling (RIS) (see Definition 5) is that the reverse samples, also referred to as rr-sets [84], provide a means for obtaining an unbiased estimate of the influence function (see Lemma 1).

Applying RIS to the influence function produces a max-k set cover problem (see Problem 3). As the number of samples used in this problem instance is increased its optimal solution approaches that of the IM problem instance. Importantly, any IM problem where the underlying diffusion model admits reverse influence sampling can be solved using a unified max-k set cover representation. Not only do RIS based techniques lead to state-of-the-art efficient approximation algorithms, they even admit extension beyond the discrete time models they were initially designed for. Indeed, we extend [71] and [83], current state-of-the-art algorithms for discrete-time models, to the CTIC model using the technique developed in [85], for the purpose of comparison against our proposed algorithm.

Influence Maximization as Max-k Cover. The RIS perspective produces a max-k cover instance that is equivalent in expectation to the IM instance. Max-k cover is

solved to within a $(1-1/e)$-approximation using the standard Greedy algorithm to sequentially select nodes. The approximation guarantee is in effect optimal due to an inapproximability result by Feige [26]. This achieves a $(1-1/e-\varepsilon)$-approximation of the original IM problem with the additional loss coming from the RIS estimation accuracy. This approach is taken by the majority of existing approximation algorithms for IM. In principle one could utilize any technique that is capable of solving max-k cover. For example TipTop [63] utilized integer linear programming, whereas most previous work uses the classic Greedy approach for solving the max-k cover [71, 83–85]. However, *any approach that requires the max-k cover instance to be materialized suffers from the space overhead of storing all the generated influence samples.* As an example, on the twitter dataset (41M nodes and 1.5B edges), selecting 15,625 seeds while guaranteeing a solution quality of $\varepsilon = 0.1$ with high probability, takes up to 237GB memory, approximately $13\times$ that of the input graph.

Streaming Max-k Cover. Given that the storage required to materialize the max-k cover can be substantial, we consider a setting where the interface to the max-k cover instance is in the form of a *stream*. Variants of streaming coverage problems have been considered in the literature: e.g., streaming set cover is studied in [10, 19, 42] and streaming max-k cover is considered in [3, 79]. However, they focus on the case where the objects to be selected are streaming. In the problem we are interested in, it is the *sets of objects to be covered* (i.e., reverse influence samples) that are streaming. A more general, *edge*-arrival, streaming setting is first considered in [5], which could in principle be applied to our problem. However, it involves constructing and storing an intermediate sketch. The space required by this sketch rapidly grows as the error tolerance required of the solution, ε, decreases, specifically as $O(1/\varepsilon^3)$. In addition, large constant factors make its application to our problem impractical. E.g., on the Twitter ($n = 41.7$M nodes) dataset we experimented on, for an error tolerance of $\varepsilon = 0.1$, the sketch required by Algorithm 3 of [5] would contain more than $24 \cdot (12/\varepsilon)^3 \cdot n$ edges (approximately 1.7×10^{15}). This would require more memory than available on our machine used to run experiments by several orders of magnitude! *Our goal is to devise a solution whose memory usage does not depend on the desired solution quality, properties of the diffusion process, and seed set size.*

3.2.2 Fractional Objectives

We introduce two fractional objective functions as well as some of their known properties which we will make use of.

Multi-Linear Extension. We first consider the multi-linear extension (ME) to the objective function in Problem 2. It treats a fractional solution as a probability distribution over seed sets. Each fractional entry $\mathbf{x}[i]$ is rounded to 1 (i.e. node i is selected) w.p. $\mathbf{x}[i]$ and rounded down to 0 w.p. $1-\mathbf{x}[i]$.

$$\begin{aligned} M(\mathbf{x}) &= \mathbb{E}_{S\sim\mathbf{x}}[\sigma(S)/n] = \mathbb{E}_{\mathbf{s}\sim\mathbf{x}}\mathbb{E}_{\mathbf{r}\sim\mathscr{D}}[\min(\langle\mathbf{s},\mathbf{r}\rangle,1)] \\ &= \mathbb{E}_{\mathbf{r}\sim\mathscr{D}}(1-\textstyle\prod_{i|\mathbf{r}[i]=1}(1-\mathbf{x}[i])) \\ &\text{Let } M(\mathbf{x};\mathbf{r}) = (1-\textstyle\prod_{i|\mathbf{r}[i]=1}(1-\mathbf{x}[i])). \\ &\text{Then } M(\mathbf{x}) = \mathbb{E}_{\mathbf{r}\sim\mathscr{D}}M(\mathbf{x};\mathbf{r}) \end{aligned} \tag{3.1}$$

This approach of interpreting a fractional solution as a probability distribution has been previously considered for general submodular functions in [9]. Importantly, it preserves the value of binary solutions. Furthermore, any local/global optimal solution is binary (see [9], Lemma 3). As such, *the multi-linear objective retains the same optimal solutions as the original objective.* Unfortunately, the objective is non-concave. Due to the presence of local optima, direct optimization of this objective could lead to poor solutions.

Fractional Relaxation. The fractional relaxation (FR) naturally arises from formulating the maximum coverage problem as an integer linear program (ILP) and then dropping the integrality constraint on the solution.

$$\begin{aligned} &F(\mathbf{x}) = \mathbb{E}_{\mathbf{r}\sim\mathscr{D}}\min(\langle\mathbf{r},\mathbf{x}\rangle,1) \\ &\text{Let } F(\mathbf{x};\mathbf{r}) = \min(\langle\mathbf{r},\mathbf{x}\rangle,1). \\ &\text{Then } F(\mathbf{x}) = \mathbb{E}_{\mathbf{r}\sim\mathscr{D}}F(\mathbf{x};\mathbf{r}) \end{aligned} \tag{3.2}$$

The min operation ensures that no extra credit is given for covering a set more than once. Since the fractional relaxation relaxes the integrality constraint on the solution, it follows that the optimal fractional solution is an upper bound on the optimal binary solution. Furthermore, the objective is concave enabling the optimal

solution to found efficiently. In addition, for any fractional solution $\mathbf{x}$, $M(\mathbf{x}) \geq (1-1/e)F(\mathbf{x})$. This result has previously appeared in [1, 32].

3.3 Approach

We present an overview of our approach, deferring technical details, algorithm design, and efficient implementation considerations to §3.4, §3.5, and §3.6 respectively.

Application of Fractional Objectives. The multi-linear extension (ME) of Problem 2 results in a fractional objective, which retains the same optimal solution as the original objective, i.e., its optimal solutions are binary. However, we must overcome the limitations caused by its non-concavity. On the other hand, unlike the ME objective, the fractional relaxation (FR) is concave. Hence, a $(1-\varepsilon)$-approximation to the optimal solution to the FR objective can be found efficiently, as the only unavoidable loss is the influence function approximation error. However, in general the optimal solution to the FR objective is fractional, which cannot be used as a solution to the original IM problem. We propose to optimize *both* fractional objectives together so that they complement each other.

Optimization of Fractional Objectives. Our choice of technique for optimizing the fractional objectives is driven by two main considerations: (1) To provide an online instance dependent empirical guarantee we must maintain an upper bound on the optimal solution. (2) We need candidate solutions that are feasible w.r.t. the original IM problem, not just the fractional objectives, i.e., the solutions must be binary seed sets of size $\leq k$.

The FR objective, being concave and having an optimal solution that upper bounds the optimal binary solution, partially achieves the first goal. However, a challenge is that prior to convergence, the current fractional solution is not a valid upper bound. To resolve this, we could use the gradient at the current solution to compute a first-order Taylor approximation of the function, which by concavity overestimates the objective function everywhere. The optimal solution to the resulting constrained linear maximization problem can be efficiently found (details in §3.6), and it yields an upper bound. *The challenge is that we only have an approximation of the influence function via influence samples, so we do not have access to*

Symbol	Meaning
$G = (V, E)$	Graph with nodes V, edges E.
$\mathscr{L}(e)$	Edge length distribution.
n	Number of nodes in G, $n = \|V\|$
k	Seed set size control parameter
ε	Solution quality control parameter.
δ	Failure probability control parameter.
$\mathbf{s}$	Seed set (binary vector format).
$\sigma(\cdot)$	Expected influence objective.
$\mathbf{r}$	Influence sample (binary vector format).
$\mathscr{D}$	RIS distribution from diffusion model on G.
$M(\cdot), M(\cdot;\mathbf{r})$	Multi-linear extension (ME) objective (Eq. 3.1).
$F(\cdot), F(\cdot;\mathbf{r})$	Fractional relaxation (FR) objective (Eq. 3.2).
$\mathbf{x}, \mathbf{z}$	FR, ME solution.
$\bar{\mathbf{x}}, \bar{\mathbf{z}}$	FR, ME cumulative solution.
$\bar{\mathbf{r}}, \bar{\mathbf{g}}$	FR, ME cumulative of stochastic gradients.
$\mathbf{x}^*, \mathbf{z}^*$	Optimal solution to FR, ME objective.
S^*	Optimal solution to influence objective.
ν	Optimal FR value, $\nu = F(\mathbf{x}^*)$.
ν^-	High probability lower bound on $F(\mathbf{x}^*)$.
ν^+	High probability upper bound on $F(\mathbf{x}^*)$.
$\nabla F(\cdot;\mathbf{r}), \nabla M(\cdot;\mathbf{r})$	FR, ME gradient w.r.t. $\mathbf{r}$.
$\phi(\cdot;\mathbf{r})$	Offset function satisfying: $\forall_{\mathbf{x},\mathbf{r}} F(\mathbf{x};\mathbf{r}) = \langle \mathbf{x}, \nabla F(\mathbf{x};\mathbf{r}) \rangle + \phi(\mathbf{x};\mathbf{r})$

Table 3.1: Notation for Sections 3.3, 3.4 and 3.5

the exact gradient. We address this by deriving an upper bound which holds with high probability and which only requires a cumulative over the gradients produced by the influence samples (see §3.4).

The constrained linear maximization when used against the ME objective also provides a means to generate candidate binary solutions to the original IM problem. This is because the linear maximization amounts to identifying the top-k coordinates in the cumulative gradient (see §3.6.1). Thus, although the solution is not constrained to be binary, as a consequence of the linear maximization it always will be. This provides a means to generate a candidate binary solution even when

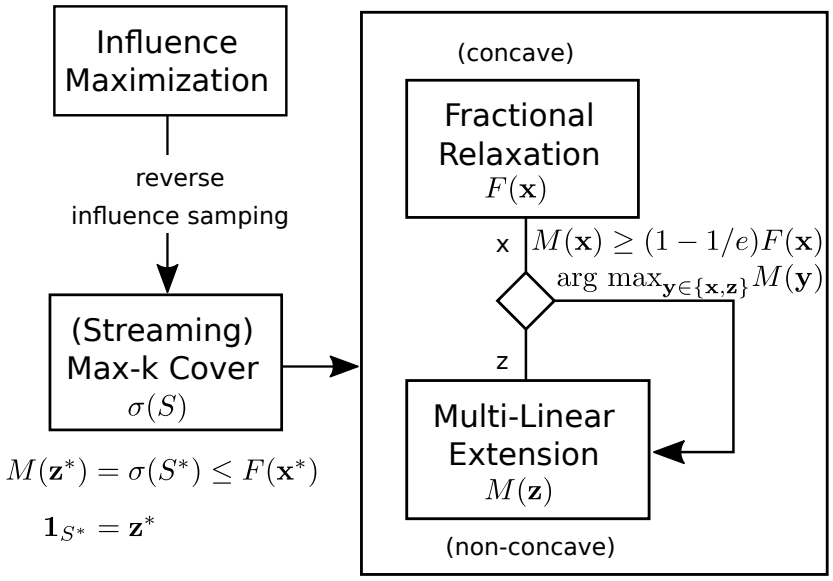


Figure 3.1: Overview of applying the multi-linear extension (ME) and fractional relaxation (FR) to influence maximization.

the *current* solution to the ME objective is not binary. Efficiently generating candidate binary solutions which can then be evaluated is key to measuring empirical progress of the algorithm online.

Finally, the solutions to the constrained linear maximization can be used to construct and update the fractional as the cumulative of these solutions. A valuable property this has is that since each individual solution satisfies the constraints, the resulting fractional solution produced by normalizing their cumulative will also always be a feasible fractional solution. This avoids the need to perform a potentially computationally expensive operation to project an infeasible fractional solution back on to the feasible space.

Combined Optimization. Simultaneously optimizing the ME objective with the FR objective provides a means to escape any poor local optima encountered. This is made possible by the connection between the ME objective and the FR objective indicated in Figure 3.1, namely for any fractional solution $\mathbf{x}$, $M(\mathbf{x}) \geq (1-1/e)F(\mathbf{x})$. Notice that finding $\mathbf{x}$ that approaches the value of $F(\mathbf{x}^*)$ is tractable,

due to concavity of $F(\cdot)$, and that $F(\mathbf{x}^*) \geq \sigma(S^*)/n$. Hence, taking the max of $\mathbf{x}$ and $\mathbf{z}$ on $M(\cdot)$ helps ensure that $M(\mathbf{z}) \geq (1-1/e)\sigma(S^*)/n$. In addition, the FR objective will provide a means to assess the quality of the solution attained by way of providing an upper bound on the optimal solution.

Even though $\mathbf{z}$ is optimized against the ME objective, $M(\cdot)$, since $M(\cdot)$ is non-concave it may happen that $\mathbf{x}$ attains higher value on $M(\cdot)$, i.e., $M(\mathbf{x}) > M(\mathbf{z})$. All binary vectors are local attractors of $M(\cdot)$ and may trap $\mathbf{z}$ in a poor local optima. In contrast, as $\mathbf{x}$ is being optimized on $F(\cdot)$, it will converge to the global optimal solution on this objective and the loss in value when considering this solution's value on the ME objective is limited by $M(\mathbf{x}) \geq (1-1/e)F(\mathbf{x})$. We estimate the values that the running fractional solutions attain on the ME objective. Should the fractional solution to the ME objective under-perform that of the FR objective, the solution to the ME objective is overwritten by that of the FR objective. *This is enables the optimization of the ME objective to continue at a point that is past the local optima.*

3.4 Theory

Here we discuss the upper and lower bounds that are used by the instance-dependent empirical guarantee and for assessing convergence of the fractional solution. We consider the FR objective (Eq. 3.2): its optimal solution upper bounds the optimal solution to the original binary objective of Problem 2. Furthermore, its value coincides with that of the original objective on solutions that are binary. Let ν be the optimal solution value of the FR objective.

$$\nu = \max_{\mathbf{x}\in[0,1]^n, \|\mathbf{x}\|_1 \leq k} \mathbb{E}_{\mathbf{r}\sim\mathscr{D}} F(\mathbf{x};\mathbf{r}) \text{ where } F(\mathbf{x};\mathbf{r}) = \min(\langle \mathbf{r}, \mathbf{x}\rangle, 1) \tag{3.3}$$

The FAIM algorithm processes a sequence of samples $\mathbf{r}_1, \mathbf{r}_2, ..., \mathbf{r}_t$, and produces a sequence of fractional seed sets $\bar{\mathbf{x}}_i$, where $\bar{\mathbf{x}}_i$ is a function of $\mathbf{r}_1, ..., \mathbf{r}_{i-1}$. From this, we attain upper and lower bounds on ν that can be computed online.

3.4.1 Bounds Derivation

To compute upper and lower bounds on the values of the optimal FR objective solution and candidate solutions, we will make use of martingale bounds on the influence sample coverage estimates. Recall that this is needed for providing empirical guarantees. We first introduce the concept of a *martingale difference sequence* and show that the influence sample coverage estimates form a martingale difference sequence. Previously, Tang et al. [85] used martingale sequences in their analysis. Their analysis is *not* applicable here since the running solution $\bar{\mathbf{x}}_i$ depends on previous influence samples. We use *martingale difference sequence* instead, which then enables the application of the associated concentration bounds.

Definition 8 (Martingale Difference Sequence)**.** *A sequence of random variables $Y_1, Y_2, ..., Y_t$ is a martingale difference sequence w.r.t. the sequence of random variables $Z_1, Z_2, ..., Z_t$ if for all i:*

1. $\mathbb{E}[|Y_i|] < \infty$.
2. *there exists a deterministic function $g_i(\cdot)$ such that,*

$$Y_i = g_i(Z_1, Z_2, ..., Z_i)$$

3. $\mathbb{E}[Y_i | Z_1, Z_2, ..., Z_{i-1}] = 0$.

To establish the correspondence between the sampling results and a martingale difference sequence, let $\mathbf{R}$ be a sequence of samples, $\mathbf{r}_1, ..., \mathbf{r}_t$. We now define a sequence of random variables, $Y_i := F(\bar{\mathbf{x}}_i; \mathbf{r}_i) - \mathbb{E}_{\mathbf{r}}[F(\bar{\mathbf{x}}_i; \mathbf{r})]$ where $\bar{\mathbf{x}}_i$ may depend on $\mathbf{r}_1, ..., \mathbf{r}_{i-1}$. In this section, we will abbreviate the expectation taken over the distribution of all influence samples, $\mathbb{E}_{\mathbf{r} \sim \mathscr{D}}[\cdot]$, as $\mathbb{E}_{\mathbf{r}}[\cdot]$, since all influence samples are drawn from the same distribution. We will also use $\mathbb{E}_{\mathbf{r}_{1:t}}[\cdot]$ to represent expectation under a sequence of t influence samples being drawn.

Lemma 4. *$Y_1, ..., Y_t$ is a martingale difference sequence w.r.t. $\mathbf{r}_1, ..., \mathbf{r}_t$.*

Proof. We verify the three conditions to be a martingale difference sequence below:

1. $\mathbb{E}_{\mathbf{r}_{1:i}}[|Y_i|] < \infty$:

$$\mathbb{E}_{\mathbf{r}_{1:i}}[|Y_i|] = \mathbb{E}_{\mathbf{r}_{1:i}}[|F(\bar{\mathbf{x}}_i;\mathbf{r}_i) - \mathbb{E}_{\mathbf{r}}[F(\bar{\mathbf{x}}_i;\mathbf{r})]|] \leq 1$$

since $F(\bar{\mathbf{x}}_i;\mathbf{r}_i)$ is bounded between 0 and 1.

2. $Y_i = g_i(\mathbf{r}_1,...,\mathbf{r}_i)$ for deterministic function $g_t(\cdot)$:

$$Y_i = F(\bar{\mathbf{x}}_i,\mathbf{r}_i) - \mathbb{E}_{\mathbf{r}}[F(\bar{\mathbf{x}}_i;\mathbf{r})].$$

Notice that $\bar{\mathbf{x}}_i$ is determined with $\mathbf{r}_1,...,\mathbf{r}_{i-1}$ specified, and with $\mathbf{r}_i$ also specified, $F(\bar{\mathbf{x}}_i,\mathbf{r}_i)$ is determined.

3. $\mathbb{E}[Y_i|\mathbf{r}_1,...,\mathbf{r}_{i-1}] = 0$:

$$\mathbb{E}[Y_i|\mathbf{r}_1,...,\mathbf{r}_{i-1}] = \mathbb{E}_{\mathbf{r}_i}[F(\bar{\mathbf{x}}_i;\mathbf{r}_i) - \mathbb{E}_{\mathbf{r}}[F(\bar{\mathbf{x}}_i;\mathbf{r})]] = 0$$

Only $\mathbf{r}_i$ is unspecified and bound by the outer expectation. The two terms are then identical and cancel. □

Lemma 4 enables the application of the concentration bounds associated with martingale difference sequences, specifically McDiarmid's inequality, as specified below in Lemma 5.

Lemma 5 (Theorem 3.12 [66])**.** *Let $Y_1,Y_2,...,Y_t$ be a martingale difference sequence with $-a_i \leq Y_i \leq 1-a_i$ for each i, for suitable constants a_i; and let $a = \frac{1}{t}\sum_{i=1}^{t} a_i$.*

1. *For any $b \geq 0$, $\Pr[|\sum_{i=1}^{t} Y_i| \geq b] \leq 2e^{-2b^2/t}$*
2. *For any $\varepsilon > 0$, $\Pr[\sum_{i=1}^{t} Y_i \geq \varepsilon at] \leq e^{-\frac{\varepsilon^2 at}{2(1+\varepsilon/3)}}$*
3. *For any $\varepsilon > 0$, $\Pr[\sum_{i=1}^{t} Y_i \leq -\varepsilon at] \leq e^{-\frac{1}{2}\varepsilon^2 at}$*

Using the definition of the martingale random variables Y_i, and letting $a_i := \mathbb{E}_{\mathbf{r}}[F(\bar{\mathbf{x}}_i;\mathbf{r})]$ then immediately $-a_i \leq Y_i \leq 1-a_i$ holds by observing that $Y_i + a_i = F(\bar{\mathbf{x}}_i;\mathbf{r}_i)$ and recalling that $F(\bar{\mathbf{x}}_i;\mathbf{r})$ is bounded between 0 and 1. From this, Corollary 1 immediately follows by renaming of a to μ, replacing Y_i with its definition, to give $\sum_{i=1}^{t} Y_i = \sum_{i=1}^{t} F(\bar{\mathbf{x}}_i;\mathbf{r}_i) - \mu$, and adding μ to both sides inside the probability.

Corollary 1 (Concentration). *Let* $\mu = \frac{1}{t}\sum_{i=1}^{t}\mathbb{E}_{\mathbf{r}}[F(\bar{\mathbf{x}}_i;\mathbf{r})]$ *then,*

$$\Pr[\textstyle\sum_{i=1}^{t} F(\bar{\mathbf{x}}_i;\mathbf{r}_i) \geq (1+\varepsilon)\mu t] \leq e^{-\frac{\varepsilon^2\mu t}{2(1+\varepsilon/3)}}$$
$$\Pr[\textstyle\sum_{i=1}^{t} F(\bar{\mathbf{x}}_i;\mathbf{r}_i) \leq (1-\varepsilon)\mu t] \leq e^{-\frac{1}{2}\varepsilon^2\mu t}$$

Recall that, the purpose of these bounds is to determine online what the current error is at runtime. Importantly, at runtime we will know the sampling estimate, $\sum_{i=1}^{t} F(\bar{\mathbf{x}}_i;\mathbf{r}_i)$. The following Lemma 6 and Lemma 7 can be derived by algebraic manipulation of the bounds in Corollary 1. At a high level, one solves (the quadratic equation) for ε in terms of X and δ with $X = \sum_{i=1}^{t} F(\bar{\mathbf{x}}_i;\mathbf{r}_i)$. The detailed proof is provided in Appendix A.1.

Lemma 6. *If* $\Pr[X \geq (1+\varepsilon)\mu t] \leq e^{-\frac{\varepsilon^2\mu t}{2(1+\varepsilon/3)}} = \delta$ *then,*

$$\Pr[\mu > LB(X,\delta)/t] \geq 1-\delta$$

where $LB(X,\delta) \coloneqq X + \frac{2}{3}\log(\frac{1}{\delta}) - \sqrt{\frac{2}{9}\log(\frac{1}{\delta})(9X + 2\log(\frac{1}{\delta}))}$

Lemma 7. *If* $\Pr[X \leq (1-\varepsilon)\mu t] \leq e^{-\frac{1}{2}\varepsilon^2\mu t} = \delta$ *then,*

$$\Pr[\mu < UB(X,\delta)/t] \geq 1-\delta$$

where $UB(X,\delta) \coloneqq X + \log(\frac{1}{\delta}) + \sqrt{\log(\frac{1}{\delta})(2X + \log(\frac{1}{\delta}))}$

Using these results we now introduce our lower bound (Theorem 2) and upper bound (Theorem 3) which enable us to assess the convergence of the current fractional solution towards to the optimal fractional solution. This will also provide a means to give an empirical solution quality guarantee and judge when the desired quality has been attained. We will begin by stating and proving the lower bound.

Theorem 2 (Lower bound). *Let* $\mathbf{r}_1,...,\mathbf{r}_t$ *be a sequence of samples and* $\bar{\mathbf{x}}_i$ *be sequence of fractional seeds where* $\bar{\mathbf{x}}_i$ *may depend on* $\mathbf{r}_1,...,\mathbf{r}_{i-1}$. *Define,*

$$\hat{v}_t^- \coloneqq \textstyle\sum_{i=1}^{t} F(\bar{\mathbf{x}}_i;\mathbf{r}_i)/t$$
$$v_t^- \coloneqq LB\left(t\cdot\hat{v}_t^-,\delta\right)/t$$

then, $\Pr[\mathbb{E}_{\mathbf{r}_{1:t}}[\hat{v}_t^-] > v_t^-] \geq 1-\delta$ *and* $v \geq \mathbb{E}_{\mathbf{r}_{1:t}}[\hat{v}_t^-]$.

Recall, v is defined to be the value attained by the optimal factional solution (see Eq. 3.3). v_t^- is the lower bound that we compute online which holds with at least probability $1-\delta$. $\mathbb{E}_{\mathbf{r}_{1:t}}[\hat{v}_t^-] = \frac{1}{t}\sum_{i=1}^{t}\mathbb{E}_{\mathbf{r}_i}[F(\bar{\mathbf{x}}_i;\mathbf{r}_i)]$ is the average expected value of the sequence of fractional solutions, $\bar{\mathbf{x}}_1,...,\bar{\mathbf{x}}_t$.

Proof. Using Corollary 1 and Lemma 6 where we have
$X = \sum_{i=1}^{t} F(\bar{\mathbf{x}}_i;\mathbf{r}_i) = t\cdot\hat{v}_t^-$ and $\mu = \frac{1}{t}\sum_{i=1}^{t}\mathbb{E}_{\mathbf{r}}[F(\bar{\mathbf{x}}_i;\mathbf{r})]$,

$$\Pr\left[\tfrac{1}{t}\textstyle\sum_{i=1}^{t}\mathbb{E}_{\mathbf{r}}[F(\bar{\mathbf{x}}_i;\mathbf{r})] > \mathrm{LB}\left(t\cdot\hat{v}_t^-,\delta\right)/t\right] \geq 1-\delta \tag{3.4}$$

$$\Pr\left[\tfrac{1}{t}\textstyle\sum_{i=1}^{t}\mathbb{E}_{\mathbf{r}_i}[F(\bar{\mathbf{x}}_i;\mathbf{r}_i)] > v_t^-\right] \geq 1-\delta \tag{3.5}$$

$$\Pr\left[\mathbb{E}_{\mathbf{r}_{1:t}}[\hat{v}_t^-] > v_t^-\right] \geq 1-\delta \tag{3.6}$$

Since $\bar{\mathbf{x}}_i$ doesn't depend on $\mathbf{r}_i$, $\mathbb{E}_{\mathbf{r}}[F(\bar{\mathbf{x}}_i;\mathbf{r})]$ is the same as $\mathbb{E}_{\mathbf{r}_i}[F(\bar{\mathbf{x}}_i;\mathbf{r}_i)]$. Equation 3.5 and 3.6 are then equivalent to equation 3.4 as they are simply using the definitions of v_t^- and $\mathbb{E}_{\mathbf{r}_{1:t}}[\hat{v}_t^-]$ respectively.

We now need to establish a connection between
$\frac{1}{t}\sum_{i=1}^{t}\mathbb{E}_{\mathbf{r}}[F(\bar{\mathbf{x}}_i;\mathbf{r})]$ and the optimal fractional value v.

$$v = \max_{\mathbf{x}\in[0,1]^n,\|\mathbf{x}\|_1\leq k}\mathbb{E}_{\mathbf{r}}[F(\mathbf{x};\mathbf{r})] \tag{3.7}$$

$$\geq \tfrac{1}{t}\textstyle\sum_{i=1}^{t}\mathbb{E}_{\mathbf{r}}[F(\bar{\mathbf{x}}_i;\mathbf{r})],\ \because\text{ no }\bar{\mathbf{x}}_i\text{ can exceed optimal }\mathbf{x}. \tag{3.8}$$

It follows that $v \geq \mathbb{E}_{\mathbf{r}_{1:t}}[\hat{v}_t^-]$ □

Notice, the lower bound in Eq. 3.8 is also valid for the case that the $\bar{\mathbf{x}}_i$'s are binary. As such, in addition to providing a means for lower bounding the value attained by the fractional solution it may also be used to lower bound the value attained by a given binary solution.

Before introducing the upper bound, we will first prove a step that enables us to upper bound the optimal fractional solution using only a given number of samples. Let,

$$F_{\mathbf{R}}(\mathbf{x}) := \tfrac{1}{t}\textstyle\sum_{i=1}^{t}F(\mathbf{x};\mathbf{r}_i) \tag{3.9}$$

be the estimated coverage of a fractional solution $\mathbf{x}$ using the samples in $\mathbf{R}$. We now consider the fractional solution that attains the maximum value on $F_{\mathbf{R}}(\mathbf{x})$, i.e., the optimal solution of $F_{\mathbf{R}}(\cdot)$. Let this optimal fractional solution be $\mathbf{x}^+$. The key point that we use is that *with respect to the samples considered,* $\mathbf{R}$, $\mathbf{x}^+$ has at least as much coverage as $\mathbf{x}^*$, i.e., $F_{\mathbf{R}}(\mathbf{x}^+) \geq F_{\mathbf{R}}(\mathbf{x}^*)$, which follows by construction.[1] This then can be related back to the optimal influence coverage, $\nu = F(\mathbf{x}^*)$, via the concentration result: the following result follows from Corollary 1.

Lemma 8. *Let* $\mathbf{x}^+ \in \arg\max_{\mathbf{x}\in[0,1]^n, \|\mathbf{x}\|_1 \leq k} F_{\mathbf{R}}(\mathbf{x})$ *then,*

$$\Pr[F_{\mathbf{R}}(\mathbf{x}^+) \leq (1-\varepsilon)F(\mathbf{x}^*)] \leq e^{-\frac{1}{2}\varepsilon^2 tF(\mathbf{x}^*)}$$

Proof. Using the second bound of Corollary 1 and setting all $\bar{\mathbf{x}}_i$ to $\mathbf{x}^*$ we have,

$$\Pr[\textstyle\sum_{i=1}^{t} F(\mathbf{x}^*;\mathbf{r}_i) \leq (1-\varepsilon)\mu t] \leq e^{-\frac{1}{2}\varepsilon^2\mu t} \tag{3.10}$$

where $\mu = \frac{1}{t}\sum_{i=1}^{t} \mathbb{E}_{\mathbf{r}}[F(\mathbf{x}^*;\mathbf{r})] = \mathbb{E}_{\mathbf{r}}[F(\mathbf{x}^*;\mathbf{r})]$. Notice that $tF_{\mathbf{R}}(\mathbf{x}^*) = \sum_{i=1}^{t} F(\mathbf{x}^*;\mathbf{r}_i)$, which follows from the definition in Equation 3.9. In addition, $F(\mathbf{x}^*) = \mathbb{E}_{\mathbf{r}}[F(\mathbf{x}^*;\mathbf{r})] = \mu$. From this, we have

$$\Pr[tF_{\mathbf{R}}(\mathbf{x}^*) \leq (1-\varepsilon)tF(\mathbf{x}^*)] \leq e^{-\frac{1}{2}\varepsilon^2 tF(\mathbf{x}^*)} \tag{3.11}$$

By dividing both sides inside the probability by t and using the fact that $F_{\mathbf{R}}(\mathbf{x}^+) \geq F_{\mathbf{R}}(\mathbf{x}^*)$ we have the following.

$$\Pr[F_{\mathbf{R}}(\mathbf{x}^+) \leq (1-\varepsilon)F(\mathbf{x}^*)] \leq e^{-\frac{1}{2}\varepsilon^2 tF(\mathbf{x}^*)}$$

□

While from Lemma 8, $F_{\mathbf{R}}(\mathbf{x}^+)$ can be used to upper bound the optimal fractional solution, ν, $\mathbf{x}^+$ cannot be easily determined, especially since we are not storing $\mathbf{R}$, but rather streaming it. To derive an easily computable upper bound for ν, we use a linear overestimate to the concave objective. For this purpose, let $\nabla F(\mathbf{x};\mathbf{r}) = \mathbb{I}[\langle \mathbf{x},\mathbf{r}\rangle \leq 1]\cdot\mathbf{r}$ and $\phi(\mathbf{x};\mathbf{r}) = \mathbb{I}[\langle \mathbf{x},\mathbf{r}\rangle > 1]$. Using this we can express

[1] Recall that $\mathbf{x}^*$ is the optimal fractional solution to F (see Eq. 3.2).

$F(\mathbf{x};\mathbf{r})$ as

$$F(\mathbf{x};\mathbf{r}) = \langle \mathbf{x}, \nabla F(\mathbf{x};\mathbf{r}) \rangle + \phi(\mathbf{x};\mathbf{r}) \tag{3.12}$$

We will now show how ν_t^+, used in Theorem 3, which is easily computed, is a valid upper bound, by relating it to $F_{\mathbf{R}}(\mathbf{x}^+)$.

Theorem 3 (Upper bound)**.** *Let* $\mathbf{r}_1, ..., \mathbf{r}_t$ *be a sequence of samples and* $\bar{\mathbf{x}}_i$ *be a sequence of fractional seeds where* $\bar{\mathbf{x}}_i$ *may depend on* $\mathbf{r}_1, ..., \mathbf{r}_{i-1}$*. Define,*

$$\hat{\nu}_t^+ \coloneqq \frac{1}{t}\left(\max_{\mathbf{x}\in[0,1]^n, \|\mathbf{x}\|_1 \leq k} \langle \mathbf{x}, \sum_{i=1}^{t} \nabla F(\bar{\mathbf{x}}_i;\mathbf{r}_i) \rangle + \sum_{i=1}^{t} \phi(\bar{\mathbf{x}}_i;\mathbf{r}_i) \right)$$
$$\nu_t^+ \coloneqq UB\left(t \cdot \hat{\nu}_t^+, \delta\right)/t$$

then, $\Pr[\nu < \nu_t^+] \geq 1-\delta$

Proof. Applying Lemma 7 with Lemma 8 using $X = F_{\mathbf{R}}(\mathbf{x}^+)$ and $\mu = F(\mathbf{x}^*) = \nu$ yields,

$$\Pr\left[\nu < \mathrm{UB}(t \cdot F_{\mathbf{R}}(\mathbf{x}^+), \delta)/t\right] \geq 1-\delta \tag{3.13}$$

We now establish a connection between $F_{\mathbf{R}}(\mathbf{x}^+)$ and $\hat{\nu}_t^+$.

$$tF_{\mathbf{R}}(\mathbf{x}^+) = \max_{\mathbf{x}\in[0,1]^n, \|\mathbf{x}\|_1 \leq k} \textstyle\sum_{i=1}^{t} F(\mathbf{x};\mathbf{r}_i) \tag{3.14}$$

Recall that $\phi(\mathbf{x};\mathbf{r}_i)$ is defined such that:
$F(\mathbf{x};\mathbf{r}_i) = \langle \mathbf{x}, \nabla F(\mathbf{x};\mathbf{r}_i) \rangle + \phi(\mathbf{x};\mathbf{r}_i)$. Thus,

$$tF_{\mathbf{R}}(\mathbf{x}^+) = \max_{\mathbf{x}\in[0,1]^n, \|\mathbf{x}\|_1 \leq k} \textstyle\sum_{i=1}^{t} (\langle \mathbf{x}, \nabla F(\mathbf{x};\mathbf{r}_i) \rangle + \phi(\mathbf{x};\mathbf{r}_i)) \tag{3.15}$$

Since $F(\mathbf{x};\mathbf{r}_i)$ is concave, the value that $\mathbf{x}$ attains on a linearization of $F(\mathbf{x};\mathbf{r}_i)$ at any point $\mathbf{y}$, which is of the form $\langle \mathbf{x}, \nabla F(\mathbf{y};\mathbf{r})\rangle + \phi(\mathbf{y};\mathbf{r})$, is minimized when $\mathbf{y} = \mathbf{x}$. Hence,

$$tF_{\mathbf{R}}(\mathbf{x}^+) = \max_{\substack{\mathbf{x}\in[0,1]^n, \\ \|\mathbf{x}\|_1 \le k}} \sum_{i=1}^{t} \min_{\mathbf{y}\in[0,1]^n} (\langle \mathbf{x}, \nabla F(\mathbf{y};\mathbf{r}_i)\rangle + \phi(\mathbf{y};\mathbf{r}_i)) \tag{3.16}$$

Now, using $\bar{\mathbf{x}}_i$ for $\mathbf{y}$ instead of the minimizer results in the upper bound below. The last equality in Eq. 3.17 follows by pushing the sum inward and using the definition of $\hat{v}_t^+$.

$$tF_{\mathbf{R}}(\mathbf{x}^+) \le \max_{\substack{\mathbf{x}\in[0,1]^n, \\ \|\mathbf{x}\|_1 \le k}} \sum_{i=1}^{t} (\langle \mathbf{x}, \nabla F(\bar{\mathbf{x}}_i;\mathbf{r}_i)\rangle + \phi(\bar{\mathbf{x}}_i;\mathbf{r}_i)) \tag{3.17}$$

$$= t \cdot \hat{v}_t^+ \tag{3.18}$$

In addition, $\mathrm{UB}(X,\delta)$ is monotone increasing in X. Thus, $\mathrm{UB}(t \cdot F_{\mathbf{R}}(\mathbf{x}^+),\delta) \le \mathrm{UB}(t \cdot \hat{v}_t^+,\delta)$. Hence,
$\Pr[v < \mathrm{UB}(t \cdot \hat{v}_t^+,\delta)/t = v_t^+] \ge 1-\delta$ □

3.4.2 Bounds Failure Probability

We make repeated use of these upper and lower bounds throughout our algorithm in order to assess online, the progress made on both the empirical guarantee and the fractional solution optimality gap. Since each individual bound only holds with high probably it is essential to ensure that the probability of any of the bounds *failing* to hold is accounted for. By applying the union bound, we can upper bound the probability of any of the bounds failing to hold, by the sum of their failure probabilities. However, to apply this requires knowing the number of times they will be invoked.

At each stopping test, the empirical guarantee and the fractional solution optimality gap are determined, for which all of the bounds involved must hold with high probability. To limit the number of stopping tests performed, and hence number of bounds which must hold, we will consider the number of samples processed to be doubled between tests. This strategy of sample doubling has been used in

prior works [71, 85] and is a common technique to ensure only a small number of stopping tests are needed. Let δ be the tolerated probability of any of the stopping tests being incorrect. Then we need to identify δ' the probability with which *each* individual upper and lower bound needs to hold. Assume Λ_0 is the number of samples processed before the first test and θ_0 is the maximum number of samples that are to be processed. Then the number of doublings, and hence stopping tests performed, is $\log_2(2\theta_0/\Lambda_0)$. Each stopping test will require 3 bounds – 2 lower bounds (one fractional solution and one binary solution lower bound) and 1 upper bound – to simultaneously hold, and each of them fails to hold with probability δ'. By the union bound, the probability of any of these bounds failing to hold is at most $3\log_2(2\theta_0/\Lambda_0)\delta'$. Hence, setting $\delta' = \delta/(3\log_2(2\theta_0/\Lambda_0))$ ensures an overall failure probability of at most δ.

Even if we do not know θ_0, the maximum number of samples that may be used, we can start with an initial guess value of $\theta_0 \geq \Lambda_0$ and still ensure that all bounds required by the stopping tests hold with high probability by doing the following. First, we set δ' as,

$$\delta' = \delta/(6\log_2(2\theta_0/\Lambda_0)) \tag{3.19}$$

This ensures that $\delta/2$ failure probability is left after θ_0 samples. Then in the event that the number of samples needed exceeds θ_0, δ' is reduced such that the total failure probability will always remain below δ. This is accomplished by reducing δ' in such a manner that it follows a geometric series: each run of $\log_2(2\theta_0/\Lambda_0)$ stopping tests results in the failure probability of any bound in the run to be $\frac{\delta}{6}, \frac{\delta}{12}, \frac{\delta}{24}$, ... As such, the probability of any bound in any run failing is at most $6\log_2(2\theta_0/\Lambda_0)\delta' \leq \delta$.

3.4.3 Parameters

The number of samples processed before the first stopping test, Λ_0, is set as the minimum number of samples that could potentially enable the stopping criteria to pass. This corresponds to the minimum number of samples the Stopping Rule Algorithm [18] would require in the case that $k = 1$ and all samples that are drawn

are covered. This gives,

$$\Lambda_0 = (1 + (1 + \varepsilon') \log(2/\delta)) \cdot \varepsilon'^{-2} \tag{3.20}$$

where $\varepsilon' = \varepsilon/(1 - 1/e)$ and ε and δ are input parameters. A smaller value for Λ_0 would result in unnecessary stopping tests being performed. A larger value may result in the use of more samples than are needed to reduce the error to ε.

The setting of θ, which is used to guess an upper limit on the number of samples needed, is based on Lemma 3 in [84]. Here we replace unknown influence of the optimal binary solution with the trivial lower bound of k, which gives,

$$\theta_0 = \tfrac{n}{k} \cdot (8 + 2\varepsilon)(\log \tfrac{2}{\delta} + \log \tbinom{n}{k} + \log 2) \cdot \varepsilon^{-2} \tag{3.21}$$

As a result, this upper bound is highly pessimistic. However, this is intentional so that θ_0 will not be exceeded in practice. Although in §3.4.2 we ensured that we can handle cases where θ_0 is exceeded, doing so requires reducing δ'. This is undesirable as it results in the gap between the the upper and lower bounds being increased. This is a consequence of the bounds being required to hold with higher probability: it forces the bounds to be further from the true value due to the uncertainty associated with it.

3.5 Algorithm

The FAIM algorithm is divided into Algorithms 2-5. Algorithm 2 is the main algorithm which invokes Algorithms 3-4, which handle initialization and processing a new influence sample respectively. Algorithm 5 is a subroutine used by Algorithm 4 for updating the solutions to the FR and ME objectives. We describe these algorithms in detail below.

Initialization. Algorithm 3 initializes a number of variables and constants used by the FAIM algorithm. The variables $\bar{\mathbf{g}}$ and $\bar{\mathbf{r}}$ store the cumulative stochastic gradients while $\bar{\mathbf{z}}$ and $\bar{\mathbf{x}}$ are the cumulative solution of the ME and FR objective functions respectively. Furthermore, $\hat{M}_{\mathbf{x}}$, $\hat{M}_{\mathbf{z}}$, $\hat{F}_{\mathbf{x}}$, $\hat{F}_{\mathbf{s}}$ are used to approximate $M(\bar{\mathbf{x}})$, $M(\bar{\mathbf{z}})$, $F(\hat{\mathbf{x}})$ and $F(\hat{\mathbf{s}})$ respectively. $\hat{\phi}$ is needed for the upper bound calculation: it counts the number of samples that have been completely covered by the fractional

Algorithm 2 FAIM

Input: $G, k, \varepsilon, \delta$
Output: $\hat{\mathbf{s}}, \alpha$

1: Initialization(); {Algorithm 3}
2: **repeat**
3: $\hat{\mathbf{s}} \leftarrow \arg\max_{\mathbf{x}\in[0,1]^n, \|\mathbf{x}\|_1 \le k} \langle \mathbf{x}, \bar{\mathbf{g}} \rangle$; $\hat{\mathbf{x}} \leftarrow \bar{\mathbf{x}}/t$;
4: $\hat{F}_{\mathbf{s}}, \hat{F}_{\mathbf{x}}, i \leftarrow 0$;
5: **while** $i < \Lambda$ **do**
6: $\mathbf{r} \leftarrow$ Generate an Influence Sample from graph G;
7: ProcessSample($\mathbf{r}$); {Algorithm 4}
8: $i \leftarrow i+1$; $t \leftarrow t+1$;
9: $\nu_{\mathbf{s}}^- = \mathrm{LB}(\hat{F}_{\mathbf{s}}, \delta')/\Lambda$; $\nu_{\mathbf{x}}^- = \mathrm{LB}(\hat{F}_{\mathbf{x}}, \delta')/\Lambda$;
10: $\nu^- = \max(\nu_{\mathbf{x}}^-, \nu_{\mathbf{s}}^-)$;
11: $\nu^+ = \mathrm{UB}(\max_{\mathbf{x}\in[0,1]^n, \|\mathbf{x}\|_1 \le k} \langle \mathbf{x}, \bar{\mathbf{r}} \rangle + \hat{\phi}, \delta')/t$;
12: **if** $\hat{M}_{\mathbf{z}} < \hat{M}_{\mathbf{x}}$ **then** $\bar{\mathbf{g}} \leftarrow \bar{\mathbf{r}}$; $(\bar{\mathbf{z}}, \hat{M}_{\mathbf{z}}) \leftarrow (\bar{\mathbf{x}}, \hat{M}_{\mathbf{x}})$;
13: **if** $t \ge \theta$ **then** $\delta' \leftarrow \delta'/2$; $\theta \leftarrow (\theta_0/\Lambda_0) \cdot \theta$;
14: $\Lambda \leftarrow 2 \cdot \Lambda$;
15: **until** $\nu^+ - \nu^- \le \varepsilon' \cdot \nu^-$ **and** $(1 - 1/e - \varepsilon)\nu^+ \le \nu_{\mathbf{s}}^-$
16: **return** $(\hat{\mathbf{s}}, \nu_{\mathbf{s}}^-/\nu^+)$

solution at the iteration in which the sample was generated. The parameters Λ_0, Λ, θ_0, θ and δ' represent the number of samples that are to be processed in the initial round, the number of samples that are to be processed in the current round, the initial maximum number of samples, the revised maximum number of samples, and the allowed bound failure probability respectively. The parameters Λ_0, θ_0 and δ' are set as discussed in §3.4.2 and §3.4.3.

Sample Processing. Shown in Algorithm 4 is an iteration used to process a new influence sample, $\mathbf{r}$. $\hat{\phi}$, which maintains $\sum_{i=1}^{t} \phi(\bar{\mathbf{x}}_i; \mathbf{r}_i)$, is updated as needed by the upper bound (Theorem 3). $\hat{F}_{\mathbf{x}}$ and $\hat{F}_{\mathbf{s}}$ estimate the value attained by the fractional and binary solutions, $\hat{\mathbf{x}}$ and $\hat{\mathbf{s}}$, that are to be evaluated. The terms $\hat{F}_{\mathbf{x}}$ and $\hat{F}_{\mathbf{s}}$, which are used in the lower bounds, are of the form $\sum_{i=1}^{t} F(\bar{\mathbf{x}}_i; \mathbf{r}_i)$. Each has a new term added to it by evaluating $\hat{\mathbf{x}}$ and $\hat{\mathbf{s}}$ on the FR objective using the new influence sample. In the same manner, the terms $\hat{M}_{\mathbf{x}}$ and $\hat{M}_{\mathbf{z}}$ are updated using the values attained by $\bar{\mathbf{x}}$ and $\bar{\mathbf{z}}$ on the ME objective respectively. Algorithm 5 is then used (lines 4 and 5, Algorithm 4) to update the current solutions to the ME and FR

Algorithm 3 Initialization

1: $\bar{\mathbf{g}}, \bar{\mathbf{r}} \leftarrow \mathbf{0}$; $\bar{\mathbf{z}}, \bar{\mathbf{x}} \leftarrow \mathbf{0}$; $i, t, \hat{M}_{\mathbf{x}}, \hat{M}_{\mathbf{z}}, \hat{\phi}, \hat{F}_{\mathbf{x}}, \hat{F}_{\mathbf{s}} \leftarrow 0$;
2: $\varepsilon' = \varepsilon/(1-1/e)$;
3: $\Lambda_0 = (1+(1+\varepsilon')\log\frac{2}{\delta})\cdot\varepsilon'^{-2}$; $\Lambda \leftarrow \Lambda_0$;
4: $\theta_0 = \frac{n}{k}\cdot(8+2\varepsilon)(\log\frac{2}{\delta}+\log\binom{n}{k}+\log 2)\cdot\varepsilon^{-2}$; $\theta \leftarrow \theta_0$;
5: $\delta' \leftarrow \delta/(6\cdot\log_2(\theta_0/\Lambda_0))$;

Algorithm 4 ProcessSample

Input: $\mathbf{r}$
1: $\hat{\phi} \leftarrow \hat{\phi} + \phi(\bar{\mathbf{x}}/t;\mathbf{r})$;
2: $\hat{F}_{\mathbf{x}} \leftarrow \hat{F}_{\mathbf{x}} + F(\hat{\mathbf{x}};\mathbf{r})$; $\hat{F}_{\mathbf{s}} \leftarrow \hat{F}_{\mathbf{s}} + F(\hat{\mathbf{s}};\mathbf{r})$;
3: $\hat{M}_{\mathbf{x}} \leftarrow \hat{M}_{\mathbf{x}} + M(\bar{\mathbf{x}}/t;\mathbf{r})$; $\hat{M}_{\mathbf{z}} \leftarrow \hat{M}_{\mathbf{z}} + M(\bar{\mathbf{z}}/t;\mathbf{r})$;
4: $(\bar{\mathbf{z}}, \bar{\mathbf{g}}) \leftarrow$ FractionalUpdate$(M(\cdot), \bar{\mathbf{z}}, \bar{\mathbf{g}}, \mathbf{r}, t)$; {Algorithm 5}
5: $(\bar{\mathbf{x}}, \bar{\mathbf{r}}) \leftarrow$ FractionalUpdate$(F(\cdot), \bar{\mathbf{x}}, \bar{\mathbf{r}}, \mathbf{r}, t)$; {Algorithm 5}

Algorithm 5 FractionalUpdate

Input: $H(\cdot), \bar{\mathbf{w}}, \bar{\mathbf{f}}, \mathbf{r}, t$
Output: $\bar{\mathbf{w}}, \bar{\mathbf{f}}$
1: $\bar{\mathbf{f}} \leftarrow \bar{\mathbf{f}} + \nabla H(\bar{\mathbf{w}}/t;\mathbf{r})$; {Alg. 6 used to read lazily maintained $\bar{\mathbf{w}}$.}
2: $\mathbf{s} \leftarrow \arg\max_{\mathbf{x}\in[0,1]^n, \|\mathbf{x}\|_1 \le k} \langle \mathbf{x}, \bar{\mathbf{f}} \rangle$; {In §3.6 lines 2-3 are replaced
3: $\bar{\mathbf{w}} \leftarrow \bar{\mathbf{w}} + \mathbf{s}$; with call to Alg. 7}

objectives.

Fractional Solution Update. Algorithm 5 shows the fractional update procedure. It is used to update the current solutions to both the ME and FR objective functions. The first parameter to the procedure, $H(\cdot)$, is the objective that is to be updated and can be either $M(\cdot)$ or $F(\cdot)$. Similarly, $\bar{\mathbf{w}}$ and $\bar{\mathbf{f}}$ take on the value of either $\bar{\mathbf{z}}$ or $\bar{\mathbf{x}}$ and either $\bar{\mathbf{g}}$ or $\bar{\mathbf{r}}$ respectively. The stochastic gradient induced by the given influence sample is calculated (line 1). The stochastic gradient for $F(\mathbf{x};\mathbf{r})$ is simply $\mathbf{r}\cdot\mathbb{I}[\langle\mathbf{x},\mathbf{r}\rangle \le 1]$. For $M(\mathbf{z};\mathbf{r})$ it is constructed as i s.t. $\mathbf{r}[i]=1$ is set to $(\prod_{j|\mathbf{r}[j]=1}(1-\mathbf{z}[j]))/(1-\mathbf{z}[i])$ and all other coordinates are 0. The product in the numerator can be computed once and the result reused for all coordinates of the gradient. Using this, the cumulative stochastic gradient is updated to give an updated approximation of the gradient. Following this, using the approximate gradient as a linearization of the objective, the optimal solution against this under the constraints is found (line 2). Note the solution to the maximization is simply a

Symbol	Meaning
i	# of influence samples so far in current round.
Λ	Current round influence sample limit.
t	Total # of influence samples processed.
$\hat{\mathbf{s}}, \hat{\mathbf{x}}$	Candidate binary and fractional solutions.
$\hat{F}_{\mathbf{x}}, \hat{F}_{\mathbf{s}}$	Cumulative estimate of $F(\hat{\mathbf{x}}), F(\hat{\mathbf{s}})$.
$\hat{M}_{\mathbf{z}}, \hat{M}_{\mathbf{x}}$	Cumulative estimate of $M(\bar{\mathbf{z}}), M(\bar{\mathbf{x}})$.
$\hat{\phi}$	Cumulative estimate of $\mathbb{E}_{\mathbf{r}\sim\mathscr{D}}[\phi(\bar{\mathbf{x}};\mathbf{r})]$
$\nu_{\mathbf{s}}^{-}$	High probability lower bound on $F(\hat{\mathbf{s}})$.
$\nu_{\mathbf{x}}^{-}$	High probability lower bound on $F(\hat{\mathbf{x}})$.
α	Empirical guarantee: $\sigma(\hat{\mathbf{s}}) \geq \alpha \cdot F(\mathbf{x}^*)$.

Table 3.2: Additional Notation for Section 3.5

binary vector with the coordinates that correspond to the Top-k entries in $\bar{\mathbf{g}}$ set to 1 and all others to 0. In §3.6.1 we discuss this in more detail and show how it can be efficiently computed. Finally, this solution is added to the cumulative solution which makes up the current solution to the fractional objective $H(\cdot)$ (line 3).

FAIM Algorithm. Having discussed the subroutines we now go over Algorithm 2. $\hat{\mathbf{s}}$ and $\hat{\mathbf{x}}$ are the current candidate binary and fractional solution respectively. $\hat{F}_{\mathbf{x}}$ and $\hat{F}_{\mathbf{s}}$ maintain the estimated value of these candidate solutions through the round. Lines 5-8 generate and process influence samples until the number of samples that have been processed in the round reaches Λ, the current round sample limit. ProcessSample (Algorithm 4) revises the current fractional solutions, $\bar{\mathbf{z}}$ and $\bar{\mathbf{x}}$, as well as the cumulative of stochastic gradients, $\bar{\mathbf{g}}$ and $\bar{\mathbf{r}}$, as influence samples are processed. In addition, it also maintains the estimates $\hat{F}_{\mathbf{x}}$, $\hat{F}_{\mathbf{s}}$, $\hat{M}_{\mathbf{x}}$ and $\hat{M}_{\mathbf{z}}$.

Following this, the upper and lower bounds are updated using the resulting estimates produced by the processed samples. $\nu_{\mathbf{s}}^{-}$ and $\nu_{\mathbf{x}}^{-}$ are lower bounds on the value of the candidate binary solution and the candidate fractional solution respectively. These are calculated according to Theorem 2 which makes use of the $\mathrm{LB}(\cdot)$ function introduced in Lemma 6. As both of these are valid lower bounds to the optimal fractional solutions, ν^{-} is set to be the maximum of the two. The upper bound, ν^{+}, is calculated according to Theorem 3 which makes use of the $\mathrm{UB}(\cdot)$ function introduced in Lemma 7.

In lines 12-13 the estimated values of fractional solutions to the FR and ME objectives, evaluated on the ME objective, i.e., $\hat{M}_{\mathbf{x}}$ and $\hat{M}_{\mathbf{z}}$, are compared. Should $\hat{M}_{\mathbf{z}}$ have lower value than $\hat{M}_{\mathbf{x}}$, then the current solution to the ME objective, $\bar{\mathbf{z}}$, and its cumulative gradient, $\bar{\mathbf{g}}$, are replaced by those of the FR objective. This ensures that $\mathbf{z}$ can escape poor local optima, as discussed in §3.3. Line 14 handles updating of δ' should θ be exceeded. After each round, the number of samples to process on the next round is doubled (line 15). This is done according to §3.4.2.

Termination is based on the conjunction of two conditions. The primary termination condition checks if the relative gap between the fractional upper and lower bounds is at most ε', i.e., $\nu^+ - \nu^- \leq \varepsilon' \cdot \nu^-$. The second termination condition, $(1-1/e-\varepsilon)\nu^+ \leq \nu_{\mathbf{s}}^-$, ensures that the candidate binary solution meets the worst-case approximation guarantee of $(1-1/e-\varepsilon)$ times the optimum. On the other hand, the first stopping criterion prevents the algorithm from stopping prematurely on instances on which solutions vastly superior to the theoretical worst case can be easily obtained within a reasonable number of rounds. More specifically, it aims to ensure that the solution quality lost due to inaccuracy in the influence function approximation is small such that any sub-optimality of the returned solution in excess of ε' is due to the combinatorial hardness the problem, and as such cannot be easily reduced further.

Theorem 4. *On termination, Algorithm 2 ensures that the returned seed set* $\hat{\mathbf{s}}$ *satisfies* $\sigma(\hat{\mathbf{s}}) \geq (1-1/e-\varepsilon)\sigma(\mathbf{s}^*)$ *with probability at least* $1-\delta$.

Proof. In §3.4.1, we established $\nu_{\mathbf{s}}^-$ to be a lower bound on $F(\hat{\mathbf{s}})$ (see Theorem 2) and similarly ν^+ to be an upper bound on ν (see Theorem 3). Recall, $\nu = F(\mathbf{x}^*)$ where $\mathbf{x}^*$ is the optimal fractional solution and that $F(\mathbf{x}^*) \geq \sigma(\mathbf{s}^*)/n$. Also we have that $F(\hat{\mathbf{s}}) = \sigma(\hat{\mathbf{s}})/n$ since $\hat{\mathbf{s}}$ is binary. Furthermore, in §3.4.2 we ensured that the probability of any of these bounds failing to hold on any iteration is at most δ. As such, it follows from the second termination condition, namely $(1-1/e-\varepsilon)\nu^+ \leq \nu_{\mathbf{s}}^-$, that on termination, $\sigma(\hat{\mathbf{s}})/n = F(\hat{\mathbf{s}}) \geq \nu_{\mathbf{s}}^- \geq (1-1/e-\varepsilon)\nu^+ \geq (1-1/e-\varepsilon)F(\mathbf{x}^*) \geq (1-1/e-\varepsilon)\sigma(\mathbf{s}^*)/n$, holds with probability at least $1-\delta$. □ □

Time Complexity. The time complexity of Algorithm 2 may be decomposed into (i) the time required to generate an influence sample along with the time needed to

process the sample, which make up the per iteration computation cost; and (ii) the iteration complexity of attaining a desired solution quality. We address efficient construction of influences samples in Chapter 2. Naive sample processing has a time complexity of $O(\max(k,|R|))$ due to the computational cost of updating the fractional solutions, which is problematic for large k. In §3.6.1 we improve this to being only logarithmic in k, specifically $O(|R|\log_2(k))$, by lazily updating the fractional solutions. Together, the per iteration time complexity is upper bounded by $O(|R|(\log_2(k)+\log_2(d_{\max})))$.

Unfortunately characterizing the iteration complexity of the FAIM algorithm is an open problem. A major challenge here is that the FR objective function is non-smooth, making the convergence rate of the fractional solution unamenable to standard convex analysis techniques. There are also significant similarities to the Fictitious Play algorithm from game theory: specifically, two-player zero-sum games where the maximizer chooses seed nodes and the minimizer decides whether or not to play a drawn influence sample. The Fictitious Play algorithm has been conjectured (over 60 years ago!) to have a convergence rate of $O(t^{-1/2})$ [53]. However, the only rate that has been proven to date is $O(t^{-1/(p+q+2)})$ [80]. As the action space (size of p and q) in our problem is $\Omega(n)$ where n is the number of nodes in the network, this convergence rate has no value for the IM application. One could alter the FAIM algorithm to make its convergence rate easier to analyze. However, several natural modifications that we tried tend to degrade its empirical performance.

Memory Complexity. The memory required by Algorithm 2 remains constant. As the algorithm runs, only existing state is updated, no additional state is added over iterations. The vectors representing the fractional solutions, $\bar{\mathbf{x}}$ and $\bar{\mathbf{z}}$, and the gradient approximations, $\bar{\mathbf{g}}$ and $\bar{\mathbf{r}}$, are all of length n and so require $O(n)$ memory. Since only one sample needs to be held at a time, it requires at most $O(n)$ memory as well. As a result, the memory complexity is dominated by that needed to hold the graph, which requires $O(n+m)$ memory.

3.6 Efficient Implementation

In this section, we describe an efficient implementation of the algorithm outlined in the previous section. Existing state-of-the-art algorithms based on RIS simply store all the generated influence samples, with the final greedy max-k cover taking a small fraction of the time needed for generating influence samples. To be competitive with this, in our approach, the total *per-iteration computation cost needs to be comparable to the time needed to generate one influence sample*.

We also describe a parallel implementation. The influence sample generation phase of existing approaches is perfectly parallelizable on a shared-memory multiprocessor, as all influence samples may be constructed independently. Since our approach streams the influence samples, a major challenge is that our proposed algorithm needs to share information between threads. Despite this, we devise an approach that attains comparable speed-up from multi-threading.

3.6.1 Lazy Update

The minimum per-iteration compute time is lower bounded by the time needed to generate a single influence sample, which is proportional to its size. We aim to have the per-iteration computation scale at most linearly in the size of the influence sample generated in that iteration. We now discuss an efficient means to implement lines 2-3 in Algorithm 5.

Line 2 amounts to identifying the Top-k entries in the cumulative stochastic gradient. This can be observed by noting that the gradient of $\langle \mathbf{x}, \bar{\mathbf{f}} \rangle$ with respect to $\mathbf{x}$ is simply $\bar{\mathbf{f}}$. As such, under the constraints $\mathbf{x} \in [0,1]^n, \|\mathbf{x}\|_1 \leq k$ the coordinates that correspond to the highest entries in $\bar{\mathbf{f}}$ will be increased to their maximum value first. Since k is an integer, each coordinate of $\mathbf{x}$ has a maximum of 1 under the constraints, and all coordinates of $\bar{\mathbf{f}}$ are non-negative, the vector $\mathbf{x}$ that sets the coordinates that correspond to the Top-k entries in $\bar{\mathbf{f}}$ to 1 and all other entries to 0 attains the maximum value under constraints. Despite this, recomputing the Top-k from scratch in each iteration is expensive, since it costs at a minimum $\Omega(k)$ and more realistically $O(n \log k)$ time. Line 3, which simply adds the Top-k seed set to the cumulative solution is also an $O(k)$ operation. Thus, the operations on line 2 and 3 are both expensive: notice that k may be far larger than the influence sample

size and as such these operations could dominate the overall running time.

To speed up line 2, we maintain a Top-k set across iterations and detect when swaps are required by the additions to the cumulative stochastic gradient. For line 3, we will only update the cumulative solution when a node is swapped out of the Top-k. By doing this, we can still compute the correct cumulative solution value for any coordinate. This is done by adjusting the cumulative solution value by how long (i.e., for how many iterations) the currently selected Top-k nodes have been in the Top-k set. To support this, we use a vector $\mathbf{h}$ that records the iteration on which every node currently in the Top-k set entered the Top-k. This efficient implementation is captured by Algorithms 6 and 7 which are used to improve Algorithm 5. In place of Algorithm 5 line 1, which reads the vector $\bar{\mathbf{w}}$ directly, we use Algorithm 6. In addition, Algorithm 5 lines 2-3 are replaced by a call to Algorithm 7.

To efficiently maintain the Top-k nodes on the cumulative stochastic gradient we utilize a min-heap, $\mathscr{H}$, of the current Top-k nodes. When a non-Top-k node has its value increased, it is tested against the minimum value in the min-heap and a swap occurs if it exceeds it. Note that all stochastic gradient coordinates of both fractional objective functions (i.e., the ME and FR objectives) are non-negative. Moreover, the number of coordinates that are increased is precisely the number of nodes in the influence sample.

Algorithm 7 performs the updates required to process an addition to the cumulative stochastic gradient. As with Algorithm 5, the same procedure applies to updating the current solutions to both the ME and FR objectives. We illustrate this algorithm with a toy example, which will serve as a running example.

Example 1 (Illustrating Lazy Update)**.** *Consider an instance containing 5 nodes. Suppose nodes 1 and 2 are currently selected (i.e., $k = 2$) and the influence sample to be processed,* $\mathbf{r}$*, is* $\{3,4\}$*. Suppose the current solution to the FR objective is being updated (i.e., $H(\cdot) = F(\cdot), \bar{\mathbf{w}} = \bar{\mathbf{x}}, \bar{\mathbf{f}} = \bar{\mathbf{r}}$). Let the entry iteration of node 1 be iteration 0 and node 2's be iteration 3 (i.e.,* $\mathbf{h}[1] = 0, \mathbf{h}[2] = 3$*, for a node, i, not in the current Top-k, the entry iteration is not defined, i.e., $\mathbf{h}[i] = \perp$). Let the current iteration be 5, with* $\bar{\mathbf{f}} = [1,2,1,0,1]$ *and* $\bar{\mathbf{w}} = [0,0,0,0,3]$*. To begin with, the nodes in the influence sample will have their values in the cumulative stochastic gradient updated (Line 1 in Algorithm 5). As the cumulative solution is updated*

Algorithm 6 Lazy-Read (replacing read access to $\bar{\mathbf{w}}[i]$)

```
1: if i ∈ 𝒦 then
2:     return w̄[i] + t − h[i];
3: else
4:     return w̄[i];
```

Algorithm 7 Lazy-Update (replacing Alg. 5 Lines 2-3)

```
1: for i s.t. ∇H(w̄/t; r)[i] > 0 do
2:     if i ∉ 𝒦 then
3:         while f̄[i] > 𝒦.minValue() do
4:             j ← 𝒦.minKey();
5:             if 𝒦.minValue() = f̄[j] then
6:                 w̄[j] ← w̄[j] + t − h[j]; h[i] ← t;
7:                 𝒦.pop(); 𝒦.push(i, f̄[i]);
8:                 break;
9:             else
10:                𝒦.update(j, f̄[j]);
```

in a lazy manner, Algorithm 6 is used instead of reading entries in $\bar{\mathbf{w}}$ directly, for retrieving the correct value. This adds a correction accounting for how long nodes have been in the current Top-k, using their entry iteration recorded in $\mathbf{h}$. Doing so provides the same value as if the cumulative solution had been incremented on each iteration. In our example, the effective $\bar{\mathbf{w}}$ used in the stochastic gradient calculation is $[5,2,0,0,3]$. After adding $\nabla H(\bar{\mathbf{w}}/t;\mathbf{r})$ the updated $\bar{\mathbf{f}}$ is $[1,2,2,1,1]$. Next, the nodes that have increased in value are considered for being swapped into the Top-k.

The value of a node j in the Top-k min-heap, $\mathscr{K}$, is only refreshed to its current value in the cumulative stochastic gradient, $\bar{\mathbf{f}}[j]$, when it is under consideration for removal (Line 10). If the min-heap's node with minimum value matches its value in $\bar{\mathbf{f}}$ and is still exceeded by the candidate node's value then a swap occurs (Lines 4-8). Node 1 will appear at the top of the min-heap with a value of 1. Node 3 now has a higher value than that of node 1 and a swap will occur where by node 1 is removed from the min-heap and node 3 is added. The node that is removed has its cumulative solution value, $\bar{\mathbf{w}}[j]$, updated using how long the node has been in the Top-k (Line 6). The number of iterations is determined from the current iteration,

t, and the iteration on which the node entered the Top-k, which is recorded in $\mathbf{h}[j]$. *The current iteration is recorded as the entry iteration for the node that is swapped into the Top-k (Line 6) and it is entered into* $\mathscr{H}$ *with its current value in the cumulative stochastic gradient (Line 7). To finish the example, using the entry iteration of node 1* $\bar{\mathbf{w}}$ *is updated to be* $[5,0,0,0,3]$ *and the entry iteration of node 3 is recorded to be iteration 5 (i.e.,* $\mathbf{h}[3] = 5$*).* □

The worst case processing time for an influence sample, $\mathbf{r}$, is $O(\|\mathbf{r}\|_1 \log k)$, which occurs in the case that all increases trigger swaps. In practice, very few of the increases are found to trigger swaps. Moreover, because cumulative increases are of limited size (max 1) a newly swapped in node taking the place of the worst node is unlikely to move many positions, making the $\log k$ cost for a swap highly pessimistic. By using a binary vector indicating whether each node is currently in the Top-k, reading a corrected element of $\bar{\mathbf{w}}$ via Algorithm 6 remains an $O(1)$ operation and as such does not affect the running time complexity.

3.6.2 Parallel Implementation

We next describe a multi-threaded implementation of the FAIM algorithm. Our first observation is that the threads need to share some information. Otherwise, they will end up doing redundant work. We have two basic pieces of information: min-heap and cumulative of stochastic gradients. Swaps in the seed set will need to be synchronized with the min-heap, so if it is shared, it will result in excessive communication and contention. Thus, each thread maintains a local version of the min-heap, $\mathscr{H}$, representing what it determines to be the Top-k nodes in the cumulative of stochastic gradients. Hence, each thread also maintains a local version of $\bar{\mathbf{w}}$, as the local Top-k sets can differ, producing different cumulative solutions. We make the threads share the cumulative of stochastic gradients, $\bar{\mathbf{f}}$. This choice is motivated by the observation that on an average, each influence sample gradient hits only a tiny fraction of the nodes. As such, contention resulting from possible simultaneous access to $\bar{\mathbf{f}}$ by multiple threads is expected to be rare and limited.

Recall, $\bar{\mathbf{w}}$ (resp. $\bar{\mathbf{f}}$) is used as a generic variable that can represent $\bar{\mathbf{z}}$ or $\bar{\mathbf{x}}$ (resp. $\bar{\mathbf{g}}$ or $\bar{\mathbf{r}}$), depending on whether the ME or FR objective is being considered. As such, each of the two objectives has a copy of the discussed data structures associated

with it to help maintain its current solution.

The main challenge is to keep the local Top-k set that each thread has from deviating too long from the true global Top-k. Should this happen then in the extreme case it would be as if each thread were progressing independently, resulting in large amounts of redundant computation. This would prevent parallelization from yielding any speedup. Discrepancies can occur due to the fact that the Top-k sets are maintained by checking whether nodes that have had their value increased in the cumulative gradient are now in the Top-k. With multiple threads incrementing the globally shared cumulative of stochastic gradients, the global Top-k can change due to increments caused by other threads. To account for this, we use a globally shared queue, Q, that all the threads use, to propose nodes that they swap into their local Top-k. We now demonstrate this by extending Example 1.

Example 2 (Illustrating Parallel Update)**.** *Consider there to be 2 threads both of which initially have the same Top-k set (nodes 1 and 2). Let thread 1 generate and process the influence sample considered previously in Example 1. This results in the true Top-k changing, as the global* $\bar{\mathbf{f}}$ *has changed, but so far only thread 1 that caused the change will have revised its Top-k. In addition to performing the swap handling discussed previously, thread 1 now also enters node 3 in to the global queue. Since a critical increment that changes the Top-k is performed by one of the threads, it will locally trigger a swap and propose the associated node in the globally shared queue. All threads check whether nodes placed in the queue are either in their Top-k or are worse than those in their Top-k. When thread 2 checks for newly proposed nodes in the queue it will see node 3 and test whether it exceeds the value of its local worst Top-k node. This enables thread 2 to then also recognize that node 3 needs to be swapped into its Top-k, replacing node 1 and enabling it to update to the true Top-k. Once a proposed Top-k node has been checked by all threads it is removed from the queue. This ensures that if influence sample generation is stopped and all threads are caught up on checking proposed nodes (i.e., the shared queue is empty), they will arrive at Top-k sets that have total value equal to the true Top-k and will match each other, as long as the Top-k is unique.* □

Empirically, we have found this approach to be highly effective. Our approach

has running time competitive with state-of-the-art baselines even when both are run with 16 threads (see §3.7). This is despite the baselines being trivially parallelizable, achieving effectively linear speed-up.

Importantly, the proposed approach to parallelization does not adversely affect the calculation of the upper or lower bounds. The upper bound is not affected as the true Top-k can be found on the shared cumulative stochastic gradient at the check points. As such the upper bound calculation doesn't change. The lower bound only requires a sum over results from testing candidate solutions against generated influence samples. Since the candidate solutions are chosen at the start of each round, each thread can compute local aggregates over the influence samples that they individually processed. These local aggregates are then accumulated when a checkpoint is reached, enabling the computation of the lower bounds using the results from all of the threads.

3.7 Experiments

The aim of our experiments is to assess the computational resource efficiency (time and memory) and solution quality attained by major state-of-the-art approximation algorithms on challenging instances of the IM problem. We test the algorithms on a variety of large networks, over a wide range of seed set sizes as well as on a range of expected diffusion depths. All these factors greatly impact the performance of IM algorithms. To our knowledge, the last of of these factors has received little attention.

We expect FAIM to use dramatically less memory than existing algorithms on account of not storing the generated influence samples. As such, a key question is: *Can* FAIM *find seed sets with comparable or better guarantees and solution quality, without increasing the number of influence samples processed and the running time, in spite of streaming the samples?*

Our experiments have been designed to answer the above question. The rest of this section is organized as follows. (1) In §3.7.2, we evaluate the online processing performance of the algorithms and their empirical guarantee. (2) In §3.7.3, we consider performance in the conventional setting, where a specific solution quality target is specified. (3) In §3.7.4, we summarize the findings from §3.7.2 and §3.7.3

about the relative pros and cons of the compared algorithms. (4) In §3.7.5, we assess the effect of our efficient implementation strategies – parallelization and group testing. We begin by describing the experimental configurations in the next section.

3.7.1 Experimental Configurations

Datasets. We perform experiments on the five real and public social networks considered in Chapter 2 (see Table 2.1).

Diffusion Model. We focus on the *continuous time independent cascade* (CTIC) model [35]. The CTIC model is configured in the same manner as described in Chapter 2 Section 2.4.

Algorithms. Our goal is to compare our algorithm, FAIM, with state-of-the-art scalable approximation algorithms for IM, namely IMM [85], DSSA [71] and OPIM [83]. Of these, only IMM has been proposed and tested on the CTIC model and attained state-of-the-art performance[2]. However, DSSA and OPIM have been shown to outperform IMM on *discrete time models*. Since they share the same sampling framework as IMM, we note that they can be extended to the CTIC model in the same manner, which we do for the sake of comparison. Prior work [71, 83, 85] has shown these approaches to be state-of-the-art, not only attaining superior solution quality but also lower running time than all other approaches, including heuristics. Despite there being a wide range of heuristics designed for discrete time models [13–16, 29, 38, 49] there does not appear to be a clear way to apply them to the CTIC model, as they don't allow a notion of a time horizon.

Influence Sample Generation. All the algorithms compared involve influence sample (i.e., rr-set) generation and on large datasets such as Orkut and Twitter, this can take a long time. To enable fair comparisons between all IM approaches, we give equal advantage to all, i.e., allow the existing approaches to use our efficient influence sample generation (see Chapter 2).

In addition, to further accelerate all of the algorithms, we run influence sample generation in parallel for FAIM, DSSA and OPIM on the shared memory multi-processor on which we conducted experiments. As sample generation constitutes

[2]We utilize a version of IMM that corrects the issue raised by [12].

the bulk of the running time for DSSA and OPIM on large datasets and is trivially perfectly parallizable, they receive near linear speed-up with the number of threads used. The time needed for Greedy selection that is run sequentially was found to be still dominated by the sample generation time even with sample generation being run in parallel on 16 threads. As a representative example, for DSSA on Twitter with $T = 0.017$, the sample generation made up 77%, 72%, 83%, 96.5% of the total running time for seed set sizes 5, 125, 3,125, 78,125. The total running times were 84s, 57s, 143s, 3,234s respectively.

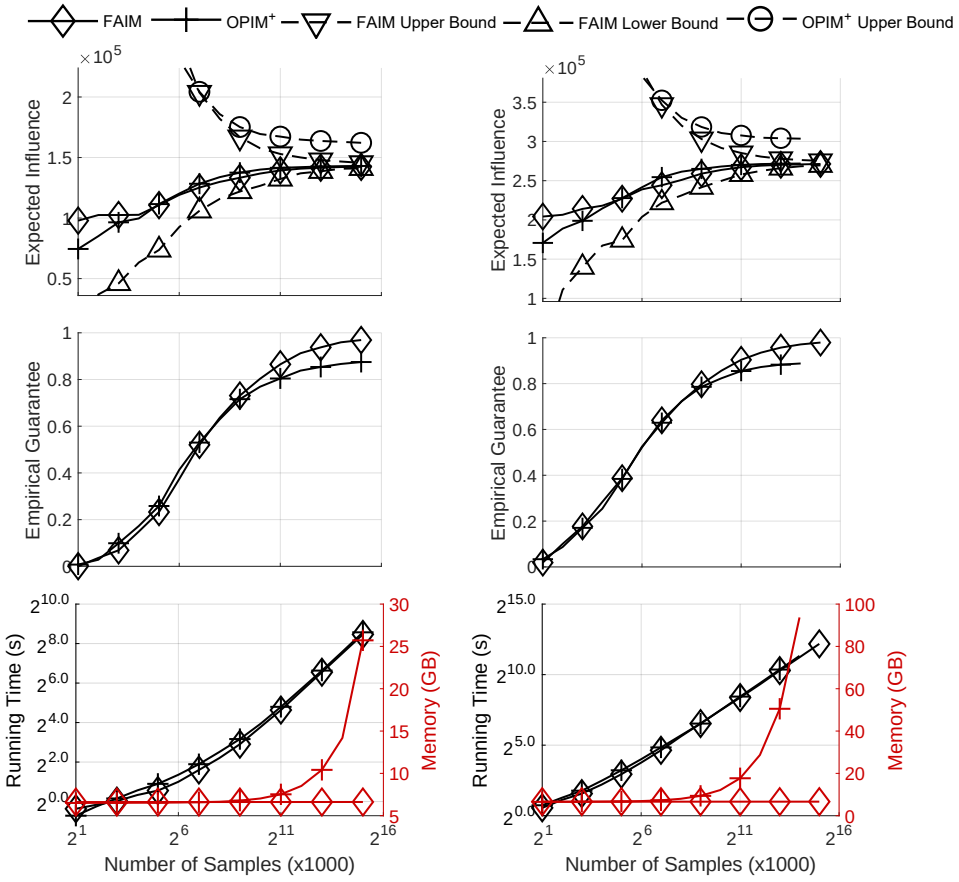
Seed Set Evaluation. The influence spread attained by the seed sets output by the various algorithms is assessed using influence sampling in conjunction with the Stopping Rule algorithm proposed in [18]. Each reported influence spread estimate is unbiased and has at most 0.01 relative error with probability at least $1 - 10^{-6}$.

Test Environment. All algorithms were run on an Intel(R) Xeon(R) CPU E5-2690 @ 2.90GHz server with 16 physical cores and 256GB of RAM running openSUSE Leap 15.0. FAIM and our implementations of OPIM and DSSA, as well as the code for IMM are in C++[3]. The time needed to load the dataset into memory is excluded from reported running times, as it is common among all considered algorithms.

3.7.2 IM Experiments: Online Setting

One approach for measuring the accuracy-efficiency tradeoff is to vary the accuracy parameter ε and study its impact on the running time and memory. Here, ε essentially affects how many samples the various algorithms use. However, ε is interpreted differently by different algorithms, and the solution quality achieved by the algorithms can differ considerably, for a given ε. Consequently, we believe that measuring the empirical solution quality w.r.t. the number of samples processed is a more precise and useful way of calibrating the algorithms on how they address the accuracy-efficiency tradeoff, as has also been noted by [72]. Thus, we directly measure how the solution quality is affected by the number of samples processed. Indeed, the solution quality of IMM, DSSA, and OPIM only differs due to their stopping criteria and how it is determined. Thus, for a *given* number of samples to be processed, their solution quality behaves identically. OPIM was

[3] All implementations were compiled using Intel compiler ICC 18.0.1 using optimization level -O3. OpenMP is used for parallel execution.

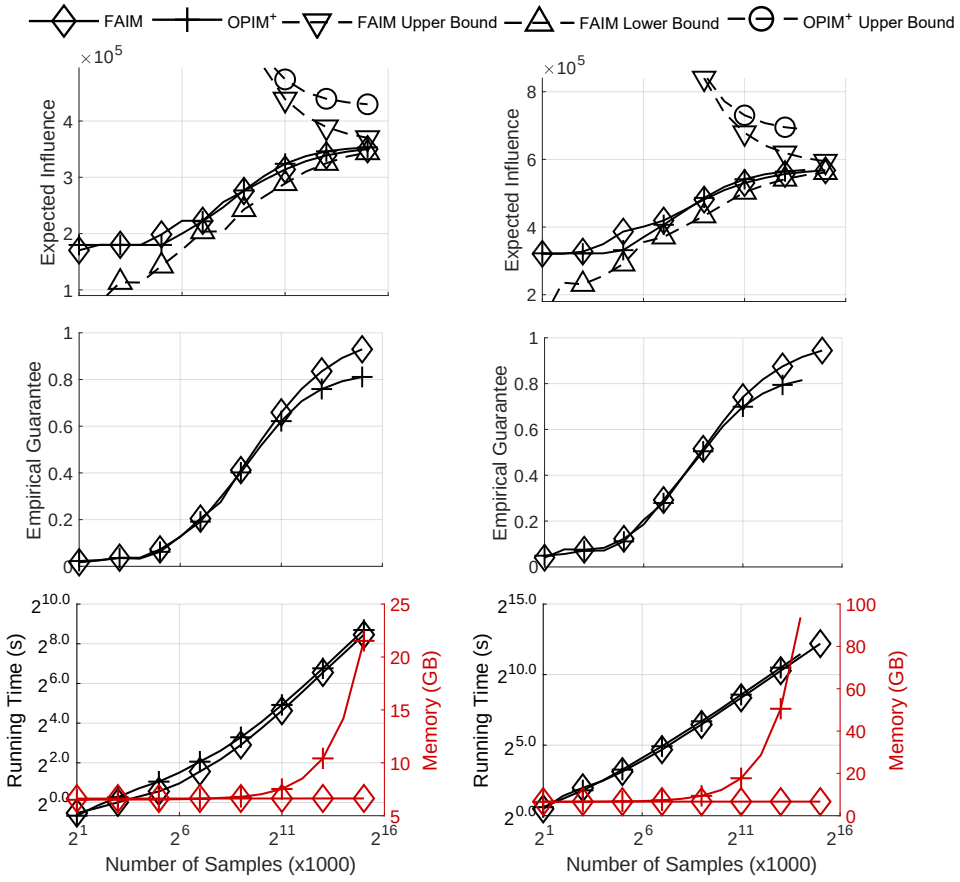


(a) $T = 0.072$ **(b)** $T = 0.11$

Figure 3.2: Online processing results on Orkut with $k = 625$ (Plots in columns share x-axis, every second marker is omitted). Top: Shows the expected influence spread and the associated upper and lower bounds of our and the competing approach. Middle: Shows the empirical guarantees that each approach can certify using their upper and lower bounds. Bottom: Shows the running time and memory used.

indeed proposed for this setting and we use it as the representative of this family of algorithms.

Figures 3.2-3.7 (a-b) are each organized column-wise. Each column corresponds to specific values of seed set size k and time horizon T, and shows: the empirical solution quality along with its upper and lower bounds (top); empirical

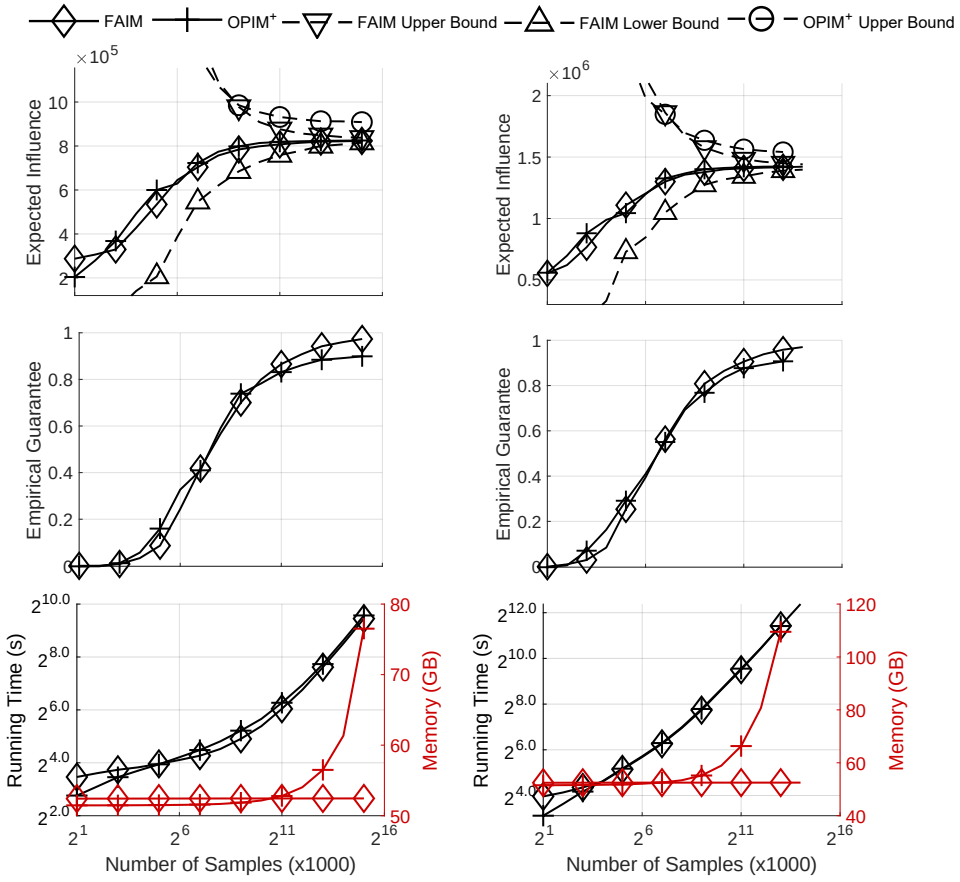


(a) $T = 0.072$ **(b)** $T = 0.11$

Figure 3.3: Online processing results on Orkut with $k = 15,625$ (Plots in columns share x-axis, every second marker is omitted). Top: Shows the expected influence spread and the associated upper and lower bounds of our and the competing approach. Middle: Shows the empirical guarantees that each approach can certify using their upper and lower bounds. Bottom: Shows the running time and memory used.

guarantee (middle); and running time / memory used, w.r.t. the number of samples processed[4] (bottom). Running time is reported by a black line and memory by a red line. All plots in a column have a common x-axis (number of samples processed) and a shared legend above the figure.

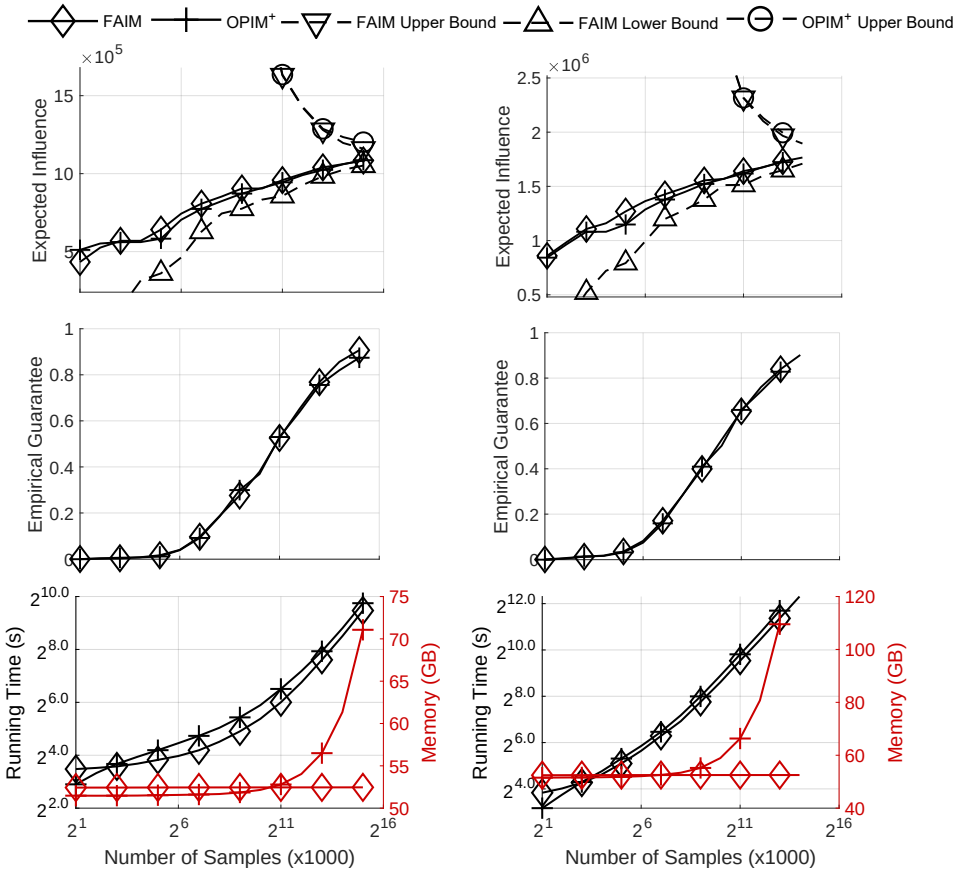
[4]Samples that OPIM uses in its validation phase are not counted.



(a) $T = 0.017$ **(b)** $T = 0.029$

Figure 3.4: Online processing results on Twitter with $k = 625$ (Plots in columns share x-axis, every second marker is omitted). Top: Shows the expected influence spread and the associated upper and lower bounds of our and the competing approach. Middle: Shows the empirical guarantees that each approach can certify using their upper and lower bounds. Bottom: Shows the running time and memory used.

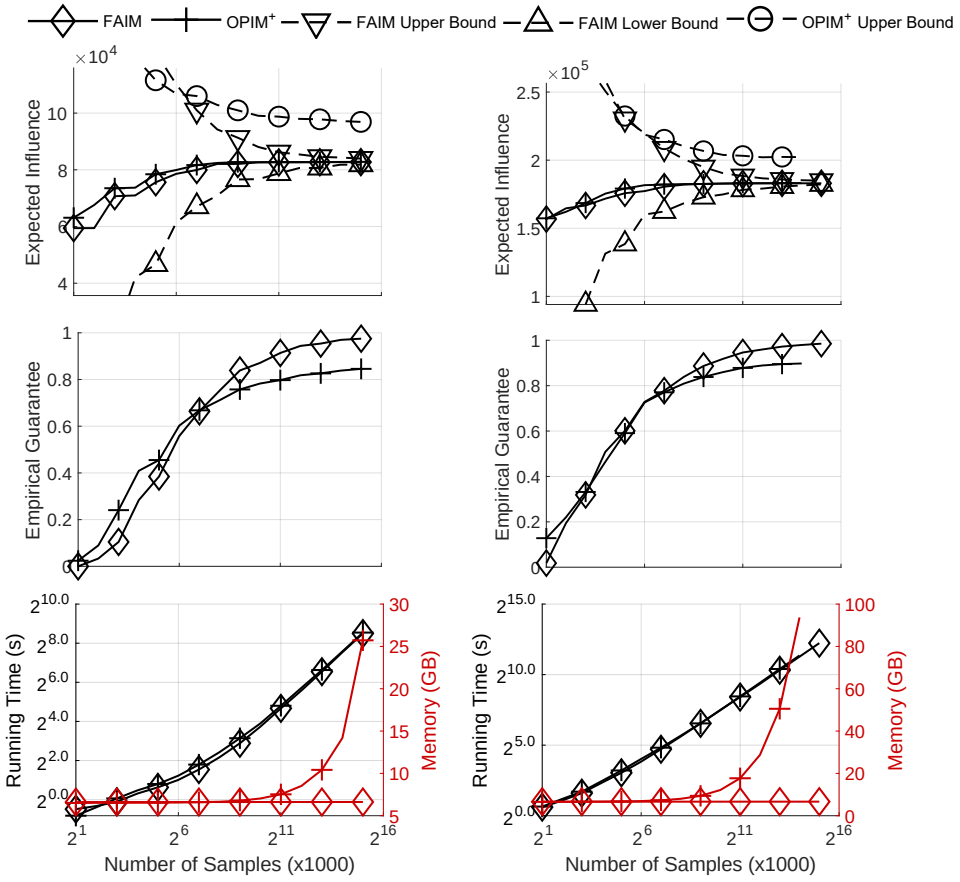
The main property of interest is what can be regarded as *sample efficiency*: how quickly the expected influence of the seed set output by each algorithm improves as the number of samples processed by the algorithm is increased. It is worth noting that the number of samples processed determines the running time – the generation of samples constitutes the bulk of the computational work for all algorithms, as



(a) $T = 0.017$ **(b)** $T = 0.029$

Figure 3.5: Online processing results on Twitter with $k = 15,625$ (Plots in columns share x-axis, every second marker is omitted). Top: Shows the expected influence spread and the associated upper and lower bounds of our and the competing approach. Middle: Shows the empirical guarantees that each approach can certify using their upper and lower bounds. Bottom: Shows the running time and memory used.

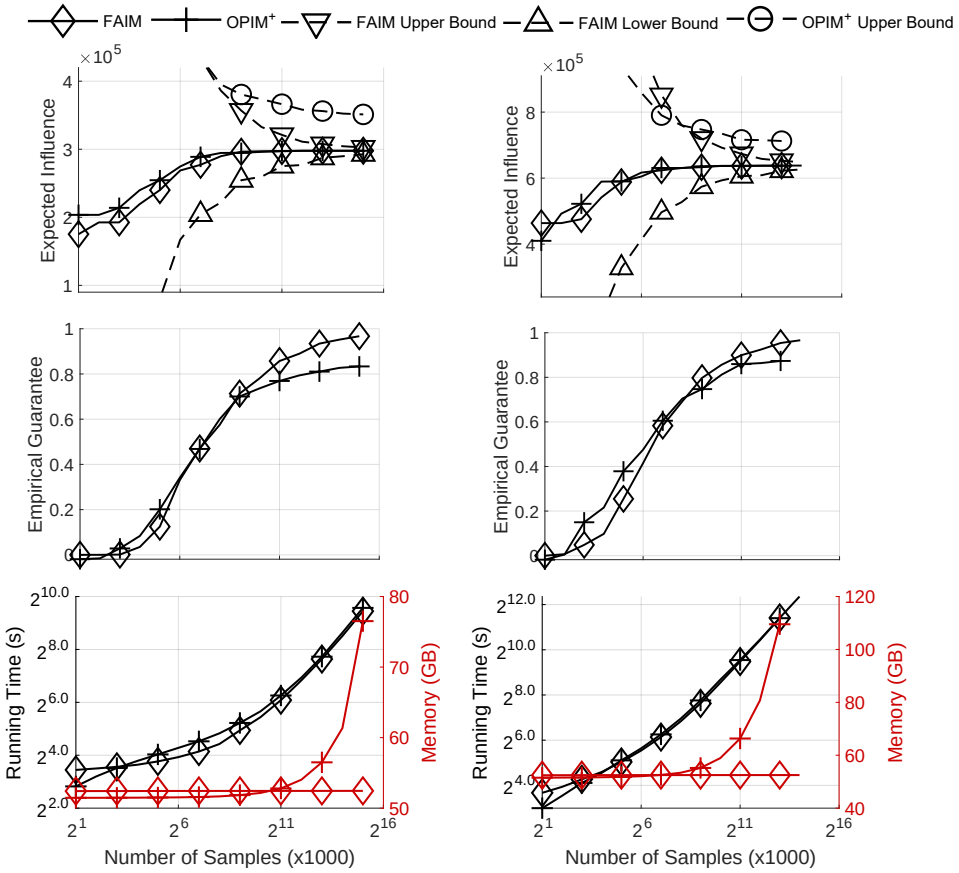
noted earlier. It also determines the memory usage, for those algorithms that need the samples to be stored. Furthermore, given that FAIM does not store the generated samples, an important question is whether this adversely affects the sample efficiency of FAIM. We can see that the empirical solution quality of FAIM is consistently comparable to that of OPIM across different configurations (see Fig-



(a) $k = 25, T = 0.072$ **(b)** $k = 25, T = 0.11$

Figure 3.6: Online processing results on Orkut with $k = 25$ (Plots in columns share x-axis, every second marker is omitted) Top: Shows the expected influence spread and the associated upper and lower bounds of our and the competing approach. Middle: Shows the empirical guarantees that each approach can certify using their upper and lower bounds. Bottom: Shows the running time and memory used.

ures 3.2-3.7 (a-b) (top)). Tables 3.3 and 3.4 shows the expected spread and bounds at the maximum number of samples that the algorithms successfully complete for all seed set sizes and time horizons considered. We can conclude that there is no noticeable difference in their sample efficiency, even though FAIM, unlike OPIM, does not store the samples and streams them.



(a) $k = 25, T = 0.017$ **(b)** $k = 25, T = 0.029$

Figure 3.7: Online processing results on Twitter with $k = 25$ (Plots in columns share x-axis, every second marker is omitted). Top: Shows the expected influence spread and the associated upper and lower bounds of our and the competing approach. Middle: Shows the empirical guarantees that each approach can certify using their upper and lower bounds. Bottom: Shows the running time and memory used.

FAIM is capable of providing empirical optimality guarantee certificates that are vastly better than the $(1-1/e) \approx 0.63$*-approximation guarantee that is the best attainable in the theoretical worst case.* In fact, the upper bound is seen to nearly converge to the lower bound. The final empirical guarantees attained by FAIM (resp., OPIM) on the seed set sizes and time horizons shown in Figures 3.2-3.3 (a-

	k	25		625		15,625	
	T	0.072	0.11	0.072	0.11	0.072	0.11
Expected Influence $\times 10^5$	FAIM	0.827	1.83	1.42	2.70	3.39	5.57
	OPIM	0.827	1.83	1.43	2.72	3.46	5.64
	FAIM Upper Bound	0.846	1.86	1.48	2.78	3.89	6.19
	OPIM Upper Bound	0.978	2.02	1.64	3.04	4.39	6.95
	FAIM Lower Bound	0.807	1.81	1.39	2.66	3.25	5.42

Table 3.3: Online Processing expected influence (in hundreds of thousands) at $2^{13} \times 1000$ samples on Orkut.

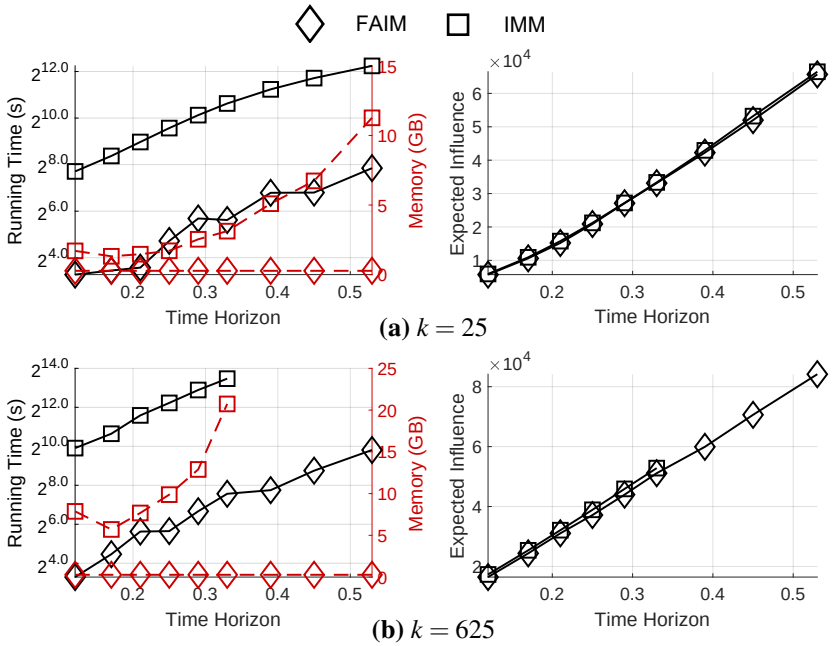
	k	25		625		15,625	
	T	0.017	0.029	0.017	0.029	0.017	0.029
Expected Influence $\times 10^5$	FAIM	2.98	6.38	8.20	14.2	10.4	17.3
	OPIM	2.98	6.37	8.23	14.2	10.2	17.2
	FAIM Upper Bound	3.08	6.53	8.49	14.5	12.8	19.7
	OPIM Upper Bound	3.56	7.13	9.12	15.4	12.9	19.9
	FAIM Lower Bound	2.87	6.23	8.00	13.9	9.83	16.5

Table 3.4: Online Processing expected influence (in hundreds of thousands) at $2^{13} \times 1000$ samples on Twitter.

b) are: **0.969 (0.875), 0.979 (0.888), 0.929 (0.811), 0.944 (0.815)**; and those shown in Figures 3.4-3.5 (a-b) are: **0.973 (0.899), 0.970 (0.907), 0.907 (0.874), 0.901 (0.828)**. The empirical guarantee provided by FAIM exceeds that which OPIM can certify by an appreciable margin in many cases. This improved empirical guarantee is a result of the tighter upper bound provided by FAIM compared to OPIM. While in some plots they are comparable, namely Figure 3.5, it is apparent that in others, e.g., Figures 3.2-3.3, the upper bound from FAIM is asymptotically considerably tighter than that of OPIM. This is reflected in the empirical guarantee that FAIM can certify compared to that of OPIM. The lower bound of OPIM is omitted to reduce clutter as it was found to be very close to that of FAIM.

The memory efficiency of FAIM over OPIM is clear from the bottom plot of each figure. While the memory needed by OPIM grows rapidly with the number of samples processed, that of FAIM remains constant. On both Orkut (Figures 3.2-3.3 (b)) and Twitter (Figure 3.4-3.5 (b)) the run of OPIM had to be cut short since

the memory needed to store influence samples exceeded 256GB. This is a substantial limitation of OPIM considering that processing further samples can continue to improve both the empirical guarantee and empirical solution quality, e.g., see Figures 3.4-3.5 (b). In addition, the running time of FAIM is consistently comparable to that of OPIM for any given number of samples to be processed. This is made possible by our efficient lazy updates to the fractional solutions maintained by FAIM.



(a) $k = 25$

(b) $k = 625$

Figure 3.8: Running time and memory usage (left) and expected influence spread attained (right) in conventional setting on Youtube dataset where all algorithms are set to have $\varepsilon = 0.1$ and $\delta = 1/n$. n is the number of nodes in the network.

3.7.3 IM Experiments: Conventional Setting

We consider seed set sizes that are powers of 5 ranging from 5 up to 78,125 seeds. In accordance with previous work [71, 83], for all algorithms, we set $\varepsilon = 0.1$ and $\delta = 1/n$, where n is the number of nodes in the network. However, it is worth noting that equal ε does not equate to equivalent empirical solution quality. This

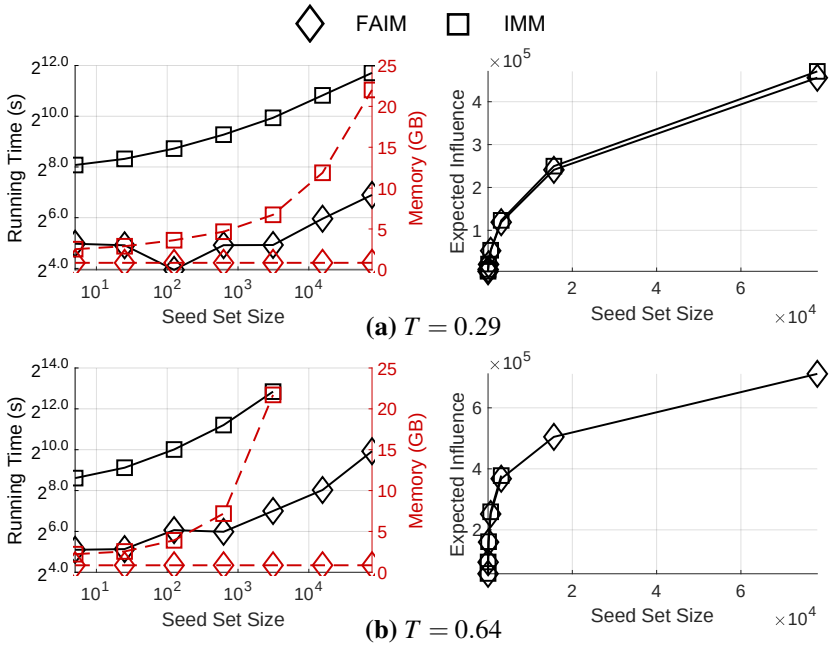
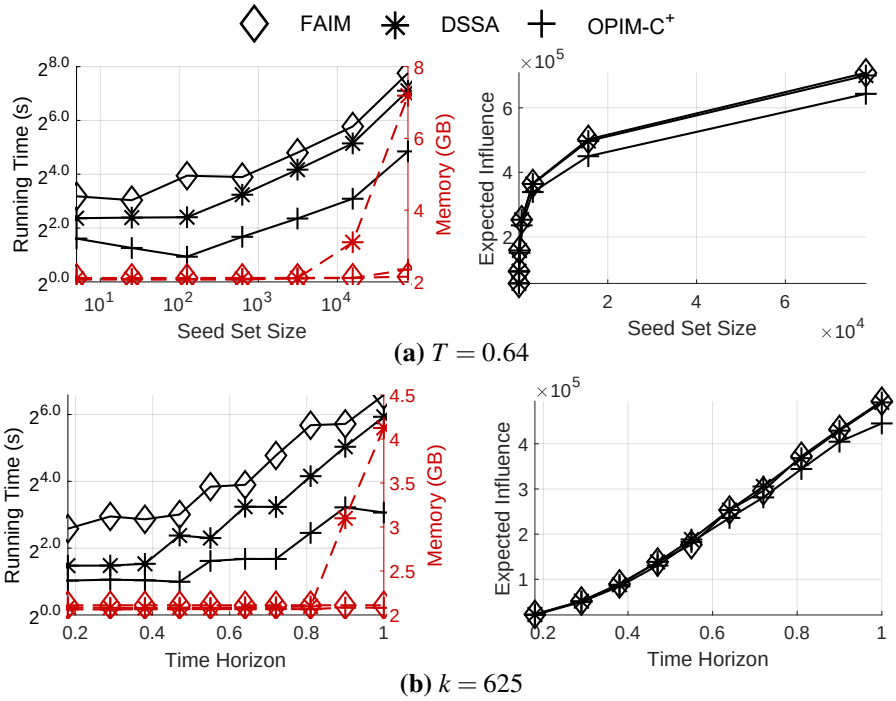


Figure 3.9: Running time and memory usage (left) and expected influence spread attained (right) in conventional setting on Pokec dataset where all algorithms are set to have $\varepsilon = 0.1$ and $\delta = 1/n$. n is the number of nodes in the network.

is primarily because different algorithms determine the number of samples to be processed for a given ε, in different ways.

It was shown in [85] that IMM incurs more running time and memory compared to DSSA and OPIM and is thus dominated by those algorithms. Nevertheless we first compare FAIM with IMM in this section, given that of these algorithms, only IMM has been formally proposed for continuous time models. For these tests, because IMM is single-threaded, FAIM is also run single threaded for fair comparison. As this takes a relatively large running time, we restrict this comparison to the smaller datasets Pokec and Youtube.

In Figures 3.8 and 3.9, IMM is observed to generally require $(2^2\text{-}2^6)\times$ more time and memory than FAIM. In addition, the memory usage of IMM increases with respect to both the seed set size and the time horizon, while that of FAIM is constant and much smaller. In these tests, a CPU time limit of 3 hours was



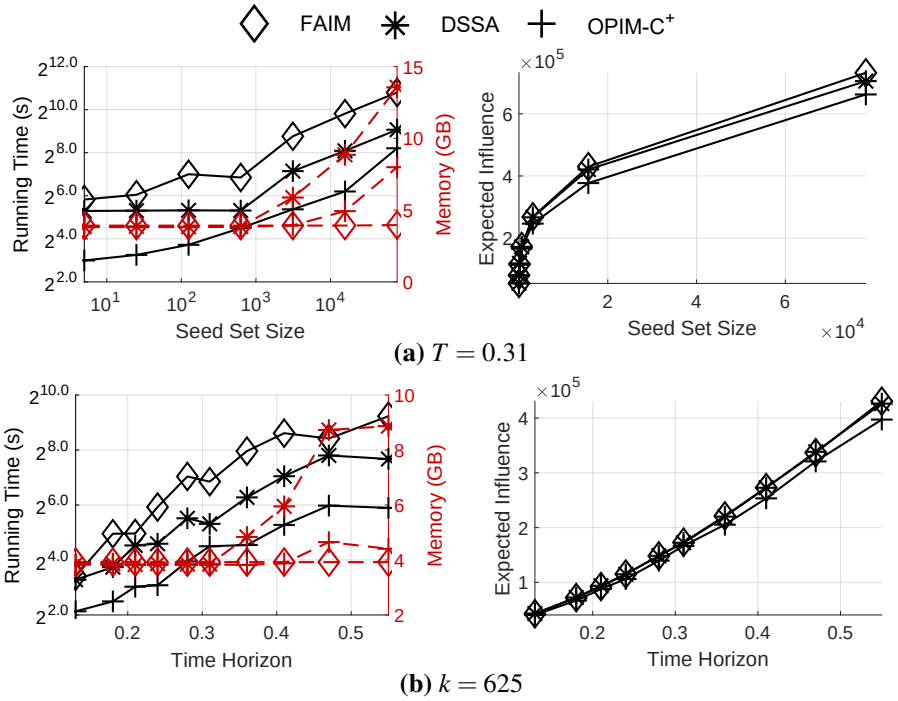
(a) $T = 0.64$

(b) $k = 625$

Figure 3.10: Running time and memory usage (left) and expected influence spread attained (right) in conventional setting on Pokec dataset where all algorithms are set to have $\varepsilon = 0.1$ and $\delta = 1/n$. n is the number of nodes in the network.

enforced, which resulted in IMM timing out on some of the larger seed set sizes and higher time horizons (see 3.8b and Figures 3.9b). IMM does attain a marginally better influence spread, albeit at much high computational cost. For instance, in Figure 3.9a, for $k = 78,125$ the influence spread attained by IMM is 3.3% higher than that of FAIM; however IMM incurs 28× the running time and uses 26× the memory.

We now compare FAIM, against DSSA and OPIM. All algorithms are implemented using 16 threads to generate influence samples in parallel; making use of all available physical cores. We allow 72 hours of CPU time (approx. 5 hours real time) and all 256GB of memory. As seen in Figures 3.10-3.13, we find that FAIM generally requires approximately 2× the running time of DSSA, while its memory usage is constant. In contrast, DSSA is observed to typically require 2-4× more



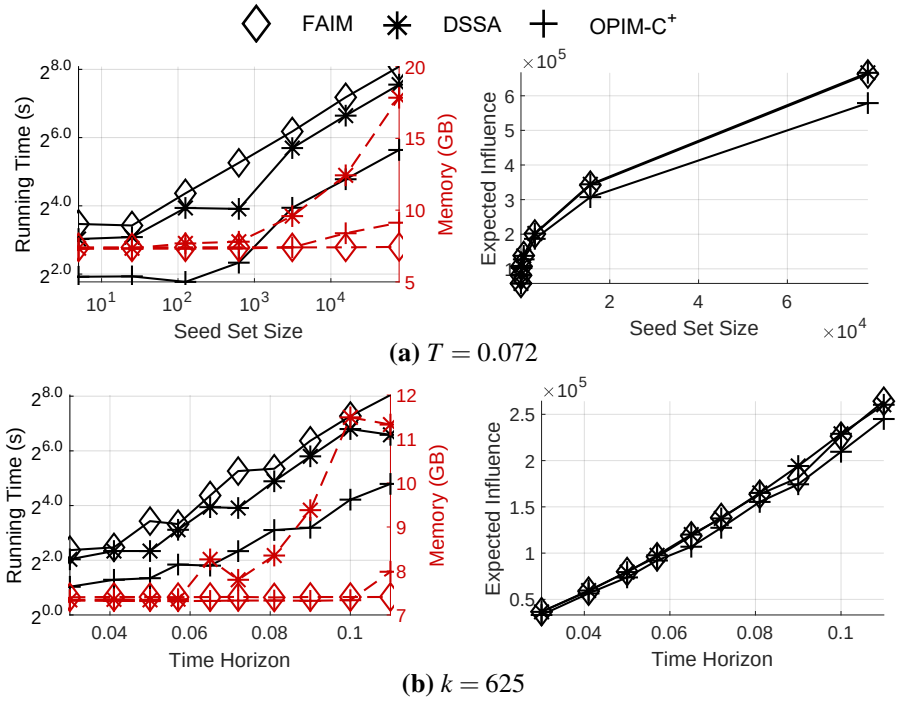
(a) $T = 0.31$

(b) $k = 625$

Figure 3.11: Running time and memory usage (left) and expected influence spread attained (right) in conventional setting on LiveJournal dataset where all algorithms are set to have $\varepsilon = 0.1$ and $\delta = 1/n$. n is the number of nodes in the network.

memory in cases when the seed set size is large or the time horizon is high. However, the difference can be much higher. For instance, on Youtube for $T = 0.62$ $k = 78,125$, DSSA used more than $32\times$ the memory. This trend would continue, hence providing more compute time to handle larger seed set sizes or higher time horizons would further enlarge the memory usage difference.

OPIM has considerably lower running time than FAIM and DSSA but the output seed set attains lower expected influence spread. This is most pronounced when the seed set size is large. While the influence spread is comparable for small seed sets, for the two largest seed set sizes considered, 15,625 and 78,125, OPIM is observed to attain approximately 5-10% less influence spread than that of FAIM and DSSA. For instance, in Figure 3.13a, when $k = 15,625$ and $k = 78,125$, FAIM achieves 9.6% and 11.1% more influence spread than OPIM, while using 94%

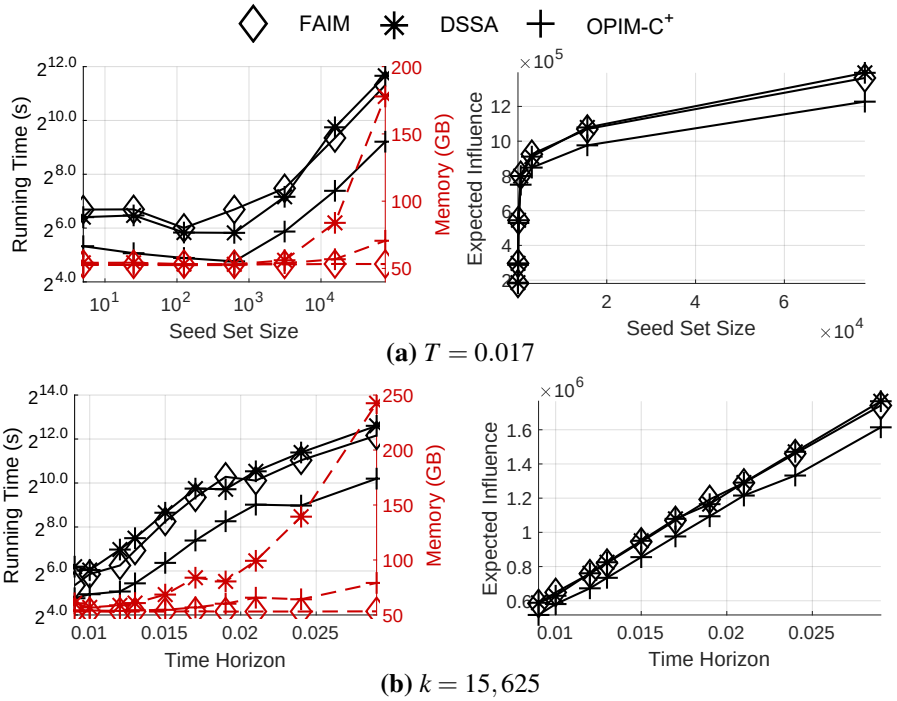


(a) $T = 0.072$

(b) $k = 625$

Figure 3.12: Running time and memory usage (left) and expected influence spread attained (right) in conventional setting on Orkut dataset where all algorithms are set to have $\varepsilon = 0.1$ and $\delta = 1/n$. n is the number of nodes in the network.

and 76% of the memory; however it requires 5.4× and 5.1× the running time. The relative influence spread of OPIM and DSSA compared to FAIM is shown in Figure 3.14. OPIM consistently attaining 5-10% less influence than FAIM, particularly at large seed set sizes, is significant as it cannot be explained simply by randomness in the influence estimates because they are accurate to within 1%, with high probability. Since it has been seen in §3.7.2 that FAIM and OPIM produce solutions of comparable quality when processing the same number of samples, it can be concluded that OPIM's reduced computation cost is entirely due to opting for lower solution quality.



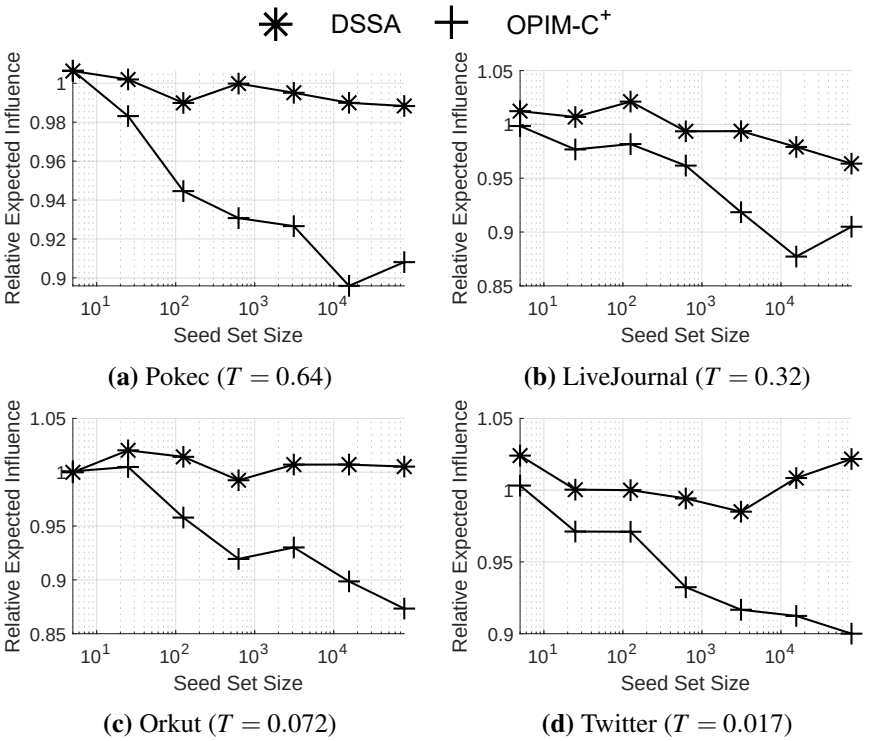
(a) $T = 0.017$

(b) $k = 15,625$

Figure 3.13: Running time and memory usage (left) and expected influence spread attained (right) in conventional setting on Twitter dataset where all algorithms are set to have $\varepsilon = 0.1$ and $\delta = 1/n$. n is the number of nodes in the network.

3.7.4 Summary of IM experiments

We can broadly divide the space of IM problem configurations into online vs conventional setting; and easier instances vs challenging instances (e.g., large graph, large seed set size, high time horizon). In the conventional setting, DSSA may not be applicable on challenging instances, unless large amounts of memory are available: it can require 100's of GB of memory, as seen on Twitter for large seed set sizes (see Figure 3.13). OPIM is unlikely to run into memory issues, however it avoids the issues at the expense of reduced solution quality. As for the online setting, OPIM is the only existing algorithm that is applicable. In this setting, OPIM *can* run out of memory: as the required empirical quality guarantee increases, the amount of memory needed to store the corresponding number of samples grows.



(a) Pokec ($T = 0.64$)

(b) LiveJournal ($T = 0.32$)

(c) Orkut ($T = 0.072$)

(d) Twitter ($T = 0.017$)

Figure 3.14: Relative Expected Influence w.r.t. FAIM in conventional setting where all algorithms are set to have $\varepsilon = 0.1$ and $\delta = 1/n$. n is the number of nodes in the network.

For both online and conventional settings, the advantages of FAIM are most apparent when the problem instance is challenging or when a high quality solution and a corresponding empirical guarantee are required. In sum, FAIM fills the void where OPIM and DSSA become inoperable due to memory limitations, while also providing a competitive alternative in less challenging settings (online or conventional) and offering an improved empirical guarantee.

3.7.5 Effect of Efficient Implementation

We conducted additional experiments to gauge the effect of efficient implementation strategies, namely parallel implementation and group edge testing.

Parallelization Tests. For existing IM algorithms, the sample generation can be

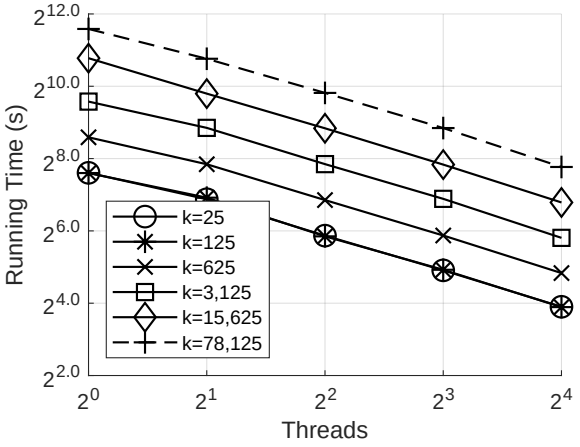


Figure 3.15: LiveJournal T=0.18 Parallelization test results

trivially run in parallel. However, FAIM needs to process the samples on the fly, i.e., as they are generated, which can potentially impact the speed-up attained. We next test the effectiveness of our parallel implementation of FAIM. Figure 3.15 provides a representative example of the effect of increasing the number of threads used by FAIM on its running time. We can see that our technique of integrating samples generated in parallel into a single fractional solution attains *near linear speed-up* at the level of parallelism considered: $1 - 16$ threads. It can also be seen that the speed-up is similar regardless of the size of the seed set to be selected. We did not find any noticeable impact of parallelization on the quality of the final output seed set.

3.8 Related Work

Research in a diverse set of topics is relevant to efficient influence maximization, which we discuss below.

Influence Estimation. Influence of a candidate seed set can be estimated using Monte-Carlo (MC) simulation or using reverse influence sampling. The classic CELF approach of Leskovec et al. [61] seeks to reduce the number of MC simulations, exploiting submodularity of the objective function. However, it fails to scale to large networks. This prompted the development of a plethora of heuristics

[13–15, 38, 49, 87] that trade worst-case solution guarantees for improved computational efficiency. The development of reverse influence sampling (RIS) by Borgs et al. [7], subsequently refined by a series of works [45, 71, 84, 85], has led to approaches with solution quality guarantees *and* computational efficiency that is superior to many existing heuristics. The sample construction cost of RIS can be non-trivial. Recent works have improved the efficiency of RIS construction under discrete time models [40, 64]. However, these approaches rely on properties of discrete time diffusion and as such are not applicable to the continuous time setting that this work addresses. An alternative approach to influence estimation resorts to sampling possible worlds [17, 22, 73], sometimes referred to as snapshots. Although their empirical results suggest that these approaches can be competitive when the number of possible worlds sampled is *set heuristically*, doing so would sacrifice the theoretical solution quality guarantee. In fact, recent work comparing the empirical solution quality w.r.t. running time indicates that RIS is significantly more efficient than sampling possible worlds for large graphs [72].

Continuous Submodular Maximization. There are a number of existing techniques that approach submodular maximization as a continuous optimization problem. However, many require the objective function to be smooth (e.g., [67] [11]) while the fractional relaxation in our setting is *not* smooth owing to the presence of the *min* function. There exist methods to smooth a given objective function, e.g., [23]. However, smoothing this way introduces unnecessary inefficiency: a direct application to our setting would introduce a large scaling of the perturbation according to the seed set size, k. This would degrade practical performance since the noise added to the cumulative gradient would dominate the added stochastic gradients for a non-negligible number of iterations and delay progress towards the optimal solution. One approach that is applicable in principle is [52]: in fact, the authors even consider IM as a possible application. Unfortunately, their approach incurs a very expensive projection step on each iteration, which limits their approach to very small networks and renders it completely impractical in our setting. An important property of our approach is that an update never takes the solution outside of the feasible space, obviating the need for the expensive projection step. It shares this property with the Frank-Wolfe optimization algorithm [28, 47]. A key difference, however, is that unlike Frank-Wolfe, we only have an approxima-

tion of the influence function as estimated via influence samples, and as such do not have access to the exact gradient. Consequently, the upper bound on the optimal solution produced by the Frank-Wolfe algorithm does not hold. Hence, there is a need for deriving a new upper bound.

Memory Usage Reduction. Given that RIS based methods are highly memory intensive [2, 72, 74], recently there have been efforts at trimming their memory requirements. It has been observed that under certain diffusion models and edge probabilities, a large number of the samples contain only a single node. We can either avoid storing such samples [75] or avoid generating them [70]. However, the memory savings from doing away with singleton samples in this way is highly sensitive to the manner in which diffusion occurs. E.g., singleton samples represent a failure of any diffusion to occur. To illustrate the limitation of approaches like SKIS, consider the two test cases for which the sample size distribution is shown in Figure 3. Despite making up 80.1% and 67.6% of the samples, because the average sample size is 606 and 288 nodes respectively for cases a) and b), the memory savings of not storing singleton samples would be just 0.1% and 0.2%! In [75], the authors use web-graph node id compression to reduce the memory required to store the samples. For the resulting savings to be significant, this requires the frequency of node ids in RIS samples to be near power-law distributed and align with their frequency in the original graph, a property that may not always hold. An alternative approach taken by [74] is to coarsen the network before applying IM. While this makes the IM problem easier to solve, it introduces another source of inaccuracy, the extent of which is not easily determined, which can degrade the quality of the returned solution. In contrast to existing works that only reduce the memory used by influence samples by a constant factor, our approach completely eliminates the need to store them by processing them in a streaming manner.

Empirical Guarantees. One can straightforwardly produce lower bounds on the optimal solution quality by evaluating generated candidate seed sets. Hence, the key to attaining instance-dependent empirical guarantees is to upper bound the value of the optimal solution. In [61], this is done using the value attained by the final Greedy solution and the remaining highest marginal gains. This is at most twice the true value of the optimal solution. The upper bound used in [83] is similarly based on the Greedy solution but considers all prefixes of it and in doing

so tightens the bound to at most $1/(1-1/e)$ times optimal. The ratio between a candidate seed set's lower bound and the upper bound yields the empirical guarantee. In practice, the upper bounds have been found to be much tighter than their theoretical worst case. This is what enables the empirical guarantees produced in this manner in [61] and [83] to significantly exceed $(1-1/e)$. By using an ILP solver, [63] is also able to provide empirical guarantees that exceed the theoretical worst case. While use of ILP is elegant, this approach has its limitations. The resulting ILPs fall well outside typical ILP solver use-cases due to their exorbitant size: reaching 10's of millions of variables and 100's of millions of non-zeros in the constraint matrix. This is caused by the large number of influence samples that they must represent. Our upper bound, which comes from the fractional relaxation, is theoretically at least as tight as that of [83] and is empirically found to be superior. Furthermore, all existing approaches require storing the influence samples, incurring high memory overhead, while ours does not.

3.9 Discussion and Conclusions

In this chapter, we present FAIM, an efficient IM algorithm that provides a worst-case theoretical guarantee of $(1-1/e-\varepsilon)$ like previous work, and an empirical solution quality guarantee that is superior to previous work. FAIM achieves this while incurring small, constant memory overhead, which does not depend on the desired solution quality, properties of the diffusion process, and seed set size, thanks to its novel approach that breaks away from the Greedy paradigm. We use two fractional objective functions, namely a fractional relaxation and a multi-linear extension, which enable us to process RIS samples as a stream, avoiding the need to store them. We derive an upper bound on the optimal solution which is empirically found to be tighter than the best existing upper bound. FAIM can be run in online mode, and with very strong instance-specific solution quality guarantees. Our experiments demonstrate the effectiveness of our algorithm on a variety of large networks and settings.

Our approach is built on the RIS technique which we improve for the CTIC model by greatly reducing the time required to generate RIS samples. Furthermore, the RIS abstraction enables our approach, developed to process samples as a

stream, to be straightforwardly extended to other influence models for which RIS samples can be efficiently generated. Notably, this is the case for the discrete time models IC and LT. The value of avoiding the need to store RIS becomes more pronounced the larger the average RIS gets, as seen by the effect of varying the time horizon under the CTIC model. Under discrete time models, the graph edge probabilities can dramatically impact the average RIS size. We expect that settings that yield similar average RIS sizes as those considered here will benefit from our approach, which enables processing the samples as a stream.

Chapter 4

Edge Sample Recycling for Reachability Estimation

4.1 Introduction

Reachability is a fundamental problem in graph analysis that asks if there exists a path from a source node to a target node in the graph. This problem becomes dramatically more challenging on uncertain graphs. As each edge in an uncertain graph exists only with a specified probability, the question of reachability becomes a probabilistic one: With what probability does there exist a path that connects the source to the target?

This probability of interest can be formalized by viewing the uncertain graph using the perspective of *possible worlds*. A possible world corresponds to one instantiation of the edges of the graph to being either live or dead. Each possible world occurs with a probability determined from its edge states and the probabilities associated with these edges. In each possible world, a path of live edges from the source to the target either does or doesn't exists. The probabilistic reachability then is the aggregate probability of all of the possible worlds in which such a path exists.

Many real life networks have edges that are uncertain. Examples include computer networks [30], protein interaction networks [48, 56] and social networks [56]. In these settings reachability on the corresponding uncertain graph may be used to

address important questions such as measuring the reliability of connections between two terminals, finding other proteins that are highly probable to be connected with a particular protein of interest or estimating how information may diffuse by the word-of-mouth effect.

Unfortunately, exact calculation of this probability is known to be #P-Hard [86]. As such, to tackle real-world scale graphs it is necessary to turn our attention to approximate solutions. Of particular importance are *unbiased* estimators for which the expectation of the estimator is equal to the reachability probability. The error of approximate solutions generated by such estimators is then solely due to their variance.

Monte Carlo (MC) sampling (see Definition 4) is a fundamental approach that can provide such approximate solutions. However, it is prone to requiring exorbitant amounts of computational running time to reduce the variance of the estimate to an acceptable level. Subsequently, more advanced sampling methods have been proposed [48, 62, 65]. These seek to either reduce the variance by more intelligent allocation of sampling work or make sample generation more efficient.

While these approaches improve substantially over MC sampling they remain susceptible to either producing poor quality estimates or requiring a prohibitive amount of computational running time. In particular, we care about producing estimates that have low relative error. While a moderate variance may be acceptable when the reachability probability to be estimated happens to be large the same variance can make the estimate useless if the true probability is in fact small. As opposed to focusing on variance, which is comparable to the squared error of the estimate, we argue the focus should be on the relative error of the estimate or similarly the *coefficient of variation*. The coefficient of variation is the ratio of the standard deviation to the mean. We find that existing approaches produce highly inconsistent quality solutions with respect to this criteria. This is a consequence of them being ineffective at estimating the reachability probability when the source only reaches the target with a low probability.

In this chapter we present unbiased estimators that avoid this shortcoming of existing work. We present an approach for which its relative error does not depend on the probability of the source reaching the target. As a consequence, it is able to provide consistently accurate estimates. To do this efficiently, the relative error of

intermediate nodes will be controlled by ensuring that edges are not oversampled, in which case they would be estimated to unnecessarily high accuracy. Existing approaches are prone to wasting computational work estimating the reachability of intermediate nodes, that have substantially higher probability of being reached than the target, to a lower relative error than is necessary. To see this, consider a trivial case of a path composed of four edges of probability 0.5 connecting the source to the target. Observe that under MC simulation for every time the fourth edge is reached and found to succeed (i.e. the full path is formed and the source reaches the target) the first edge will have in expectation succeeded 8 times and been tested 16 times. Consequently, the reachability probability estimate of the first intermediate node will be far more accurate than that of the target node. We find the work spent repeatedly testing the first edge to be wasteful.

To avoid this we introduce a technique that we refer to as *edge sample recycling*. Edge sample recycling allows us to only sample an edge to a desired level of accuracy after which we efficiently emulate further samples using those drawn so far. We show that the resulting estimator remains unbiased. In addition we demonstrate experimentally that its relative error is not impacted by reachability probability, in contrast to existing approaches.

The contributions of this work are,

- We present the novel concept of recycling random draws which we prove to yield an unbiased estimator.
- We show empirically that our approach is highly effective and achieves more consistent accuracy as measured by relative error. The existing state-of-the-art suffers from inconsistent s-t accuracy depending on the reachability probability of s-t pair that is chosen for evaluation.

4.2 Related Work

Monte Carlo Simulation. As discussed in Section 1.1 the most direct way one can tackle approximating the reachability probability is through the connection to possible worlds. Sampling possible worlds and then averaging over whether the source reaches the target in each possible world provides an unbiased estimate of

the reachability probability. This naive approach can be improved by avoiding sampling full possible worlds and instead only testing an edge when it is encountered by a forward traversal from the source. It is easy to see that this will yield the same reachability result for each possible world as if the entire possible world were realized however at a much lower computational cost, as in practice only a small fraction of the edges of the graph will typically need to be tested. Unfortunately, it remains the case that the number of possible worlds that one must sample for a desired relative error depends on the reachability probability (see Section 1.1). Clearly this is undesirable as it means that one cannot know how many possible world samples are needed beforehand, since the reachability probability is what we are trying to approximate. Furthermore the computation time needed to achieve a desired relative error may be greatly inflated if the probability to be estimated is small. Despite these limitations Monte Carlo (MC) simulation remains a key building block to the existing approaches to reachability probability estimation.

Recursive Sampling. Recursive sampling [48] improves on MC sampling by following a divide-and-conquer technique. An edge is said to be expandable if it starts from a reached node. Initially the source is the only reached node. The algorithm proceeds by repeatedly picking an extendable edge e and then dividing the samples into two groups: one group where the edge is live (in which case the destination node becomes reachable) and the other where it is dead. Let K be the number of samples. As opposed to sampling the edge K times the samples are analytically divided into the two cases according to the edge's probability.[1] When the remaining samples allocated to a branch is below a pre-defined threshold, a non-recursive sampling method such as the basic Monte Carlo, which is equivalent to the optimal Hassen-Hurvitz estimator, (this is referred to as RHH) or a more sophisticated Horvitz-Thomson estimator (referred to RHT) is used to sample the remaining edges [48, 50].

Recursive stratified sampling [62] may be viewed as a generalization of recursive sampling. Instead of having a branching factor of 2, as is the case for recursive

[1] In [48] Algorithm 2 shows the number of samples being divided according to $\lfloor Kp(e) \rfloor$ and $K - \lfloor Kp(e) \rfloor$ however this results in the introduction of bias into the estimate as the floored branch can be systematically under-weighted. The actual implementation provided by the authors does not have this issue as it uses randomized rounding instead of floor.

sampling since there are always two cases when dividing on edges one at a time, recursive stratified sampling considers the general case of having a branching factor of 2^r, which results from dividing on r edges simultaneously. r is a hyper parameter of recursive stratified sampling. Note that when $r = 1$ it follows the same division as recursive sampling.

The effectiveness of both recursive sampling approaches are heavily dependent on the selection of edges to expand. Consequently, while selection of the next edge to expanded may be done randomly, in [48] it is guided using a selection procedure that favors expanding edges if they are near the target. Unfortunately it can be noted that if the probability of reaching the target is small, regardless of the edge expansion order, the branch that contains the live edges needed to reach the target will be prone to having its number of remaining samples falling below the threshold. This follows from the remaining samples being at most the initial samples multiplied by the probabilities of the edges needed to form a path from the source to the target, which can potentially be very small. Due to reverting back to MC sampling when this occurs these approaches still have their accuracy severely degraded if the probability to be estimated is small.

Geometric Sampling. The geometric sampling based approach proposed by Li et al. [65] was found to be surprisingly competitive with other approaches designed specifically for reachability estimation (i.e. reliability) in [54]. The approach is not fundamentally different than MC simulation. However, it makes the generation of samples much more efficient by avoiding needing to perform repeated Bernoulli draws by instead drawing from a geometric distribution, which determines the next trial on which the edge will be live. It turns out that the amount of random trials that do not need to be performed as a result of utilizing geometric sampling in the place of Bernoulli draws increases the smaller the edge probabilities are. As a consequence the MC simulation samples require less computational work to construct on low probability graphs, which helps to offset the need for more such samples. However, this advantageous speedup only occurs at the level of edges not paths. This limitation will be demonstrated in Figure 4.1 where we will show that while geometric sampling can efficiently handle the presence of one low probability edge as soon as there are multiple such edges it too becomes inefficient.

4.3 Our Approach

A key goal of our approach is to be able to achieve consistent relative error *regardless* of the value of the reachability probability that is to be estimated. Importantly this should be attained without the running time being dramatically increased when the reachability probability is small. To achieve this we take inspiration from geometric sampling.

Geometric sampling when used to estimate the probability of a single edge has the property that the number of sample draws needed to achieve a desired relative error does not depend on the edge's probability. Geometric sampling is equivalent to performing Bernoulli trails except only the occurrence of successful cases are drawn. This is done by way of the geometric draws specifying the number of trials until the next successful trial. As such, the expected number of geometric draws needed to be equivalent to K Bernoulli draws is $p(e)K$, where $p(e)$ is the probability of the edge being live (i.e., succeeding). Recall that if $K \geq \frac{3}{\varepsilon^2 p(e)} \ln\left(\frac{2}{\delta}\right)$ Monte Carlo simulation trials (i.e. Bernoulli draws) are performed then the estimate has ε relative error with probability at least $1-\delta$ (see Section 1 Diffusion Approximation). It follows that the expected number of geometric draws that would be needed is $\frac{3}{\varepsilon^2} \ln\left(\frac{2}{\delta}\right)$. Observe that the required geometric draws, i.e. the number of *successful trials*, required is independent of the probability to be estimated, $p(e)$. We seek to extend this to the general problem of reachability estimation.

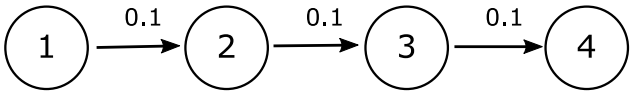
The key intuition is to only sample the worlds that are *successful*. That is, the worlds in which the source node has a path to the target node. If we can identify the points at which a desired number of successful worlds occur without spending computation work that depends on the number of failed worlds between them then we will be able to achieve the goal of efficiently attaining consistent relative error regardless of the reachability probability. As discussed, geometric sampling achieves this for a single edge but *extending this behavior to reachability is non-trivial.* A key problem when extending to paths is that the expected number of times prior edges in a path need to succeed before the entire path succeeds grows exponentially with the path length. Consequently, just identifying when individual edges are successful is insufficient. Notice that because the earlier edges in the path have to succeed far more times than the later edges it is the case that the

reachability probability of these intermediate nodes in the path will have far lower relative error under existing approaches than the relative error of the reachability probability of the target. A key insight is that since we do not care about the reachability probability to these intermediate nodes other than as a means to estimate the reachability probability to the target, estimating them to such excessive accuracy can be seen as a massive waste of computational work. This raises the question, how to minimize this wasted computational work?

4.3.1 Edge Draw Recycling

To address this, we propose a technique that we refer to as *edge draw recycling*. Notice that the reason we need to draw the earlier edges so many times is that they must succeed many times before the later edges do. Instead of repeatedly sampling these earlier edges, we propose to emulate doing so using a limited number of samples. Specifically, consider determining the state of an edge for some number of worlds, K', through sampling. Now assume we want to know if the edge is live for world i where $i > K'$. Instead of performing a new edge draw, we will reuse the samples that have already been performed by replicating the K' draw results. Doing so is equivalent to taking $i \mod K'$ and then using the result to index into the K' samples. It is obvious that the replicated draw results remain an unbiased estimate of the edge's probability as the results they are copying from are unbiased. Observe that this provides a means to efficiently emulate the edge's state at any number of worlds without needing to perform any further actual draws. Armed with this technique it becomes easy to efficiently address the simple case of a path. We need to know how many tries each edge needs to succeed in, which we can efficiently get from geometric sampling. In addition, we need to know after how many worlds the edge will be tried this many times. This is what edge draw recycling enables us to do efficiently. Specifically, we know how many times the edge is reached in the limited number of samples. As such, we can efficiently determine how many replications are needed for the edge to be reached the required number of times.

To demonstrate the benefit of this technique consider the toy example in Figure 4.1. We will assess how many random draws are used by the various techniques to estimate the probability that node 4 is reachable from node 1. Assume



Reach Set	Probability	Expected # of BFS MC Edge Draws	Expected # of Geometric Draws	Expected # of Edge Draw Recycling Draws
$\{1\}$	0.9	9,000	0	0
$\{1, 2\}$	0.09	1,800	900	0
$\{1, 2, 3\}$	0.009	270	180	0
$\{1, 2, 3, 4\}$	0.001	30	30	30
Total Draws		11,100	1,110	30

Figure 4.1: Toy example for comparing amount of random draws performed by BFS MC, Geometric sampling, and Edge Draw Recycling (our approach).

that we want to draw sufficiently many samples such that in expectation at least 10 of them succeed (i.e. the path from node 1 to node 4 is realized). For this toy example it is easy to see that the probability of the path being formed is 0.001. As such using $K = 10,000$ will result in 10 successful possible worlds in expectation. Naive Monte Carlo, which samples all edges for every possible world, will perform 30,000 random Bernoulli draws. These random draws are used to determine whether or not each of the 3 edges is live or dead in each world. For breadth-first search Monte Carlo (BFS MC) the amount of random draws performed depends on the set of nodes reached. In Figure 4.1 we show the expected number of edge draws used on all reach set sizes. For instance the reach set containing $\{1,2\}$ occurs with probability 0.09 as the first edge must succeed and the second fail. As such, there will in expectation be 900 such cases out of the 10,000 and on each such case BFS MC will have tested 2 edges for a total of 1800 edge trials. The total number of edge trials performed is the sum over all the reach set cases which is 11,000 for BFS MC. Using geometric sampling in place of performing Bernoulli edge trials reduces the number of random trials performed by an order of magnitude. Notice that there are no occurrences of the reach set $\{1\}$ for geometric sampling as performing geometric draws will only draw the cases where an edge

is live. However, after that in 90% of the samples in which node 2 is reached node 3 will not be reached. This is because geometric sampling only ensures the success at the level of individual edges. It does not extend to paths. Consequently, 900 of the samples will contain the reach set $\{1,2\}$ and one geometric draw will have been used on each of them. In contrast, with edge draw recycling, instead of redrawing the edge $(1,2)$ many times we will draw it only 10 times, after which we will emulate further draws using recycling. For instance, assume we get the following success indices $\{4,29,32,43,53,54,69,71,81,98\}$ for the first edge out of 100 worlds. Assume that the first success index of the edge from node 2 to node 3 is 14 then we determine that this occurs on world 143 by way of recycling the draws from the first edge. Note that the success indices of the edge (1,2) need to be recycled once for the node 2 to be reached 14 times, as there are only 10 success indices in the 100 worlds. As such, the next 10 success indices produced by recycling are $\{104,129,132,143,153,154,169,171,181,198\}$. It is on the 4th of these (i.e. 143) that the node 2 will have been reached 14 times. By only performing in expectation 10 draws on each edge we are able to construct the desired 10 successful worlds where there exists a path from node 1 to node 4. Hence, our approach only uses 30 random draws in contrast with BFS MC and geometric sampling which need 11,100 and 1,110 respectively.

It is worth noting that the number of random draws required by both BFS MC and geometric sampling to produce at least 10 successful samples in expectation grows exponentially in the path length while that needed by our edge draw recycling approach grows only linearly. To see this, consider adding a 5th node with an edge of probability 0.1 to it from node 4 of this example. The probability of the path from node 1 to the new node 5 is now 0.0001 and as such $K = 100,000$ is needed to attain 10 successful possible worlds in expectation. Consequently, the number of random draws needed by BFS MC and geometric sampling rise to 111,110 and 11,110 respectively. This can be seen by noting that the number of random draws in the first 3 rows of the table for BFS MC and geometric sampling with be increased by 1 order of magnitude since K was increased by one order of magnitude. The third row will be changed to occurring with probability 0.009 and result in 270 and 180 random draws being used by BFS MC and geometric sampling respectively. In addition, a new row is added corresponding to the case of

the path from node 1 to node 5 being formed. This occurs with probability 0.0001 and used 40 random draws for both BFS MC and geometric sampling. In contrast, number of random draws need by edge recycling only increases to 40. This is because we do not need to perform any additional random draws on the previous edges and we only required an expected number of 10 successful trials of the newly added edge which can be identified by performing 10 geometric random draws in expectation.

We will now focus on the challenges that arise in the more general problem that are not present when only considering the simple case of one path. The most significant challenge is caused when paths from the source to the target overlap. This is in fact the underlying cause of the problem being intractable to solve exactly. It is necessary to correctly account for the correlation of the paths caused by them sharing edges. This happens naturally under Monte Carlo simulation when paths co-occur in worlds and only count once towards the reachability probability estimate. We must ensure that this remains the case when we apply edge draw recycling to ensure that the produced estimates remain unbiased. We find that we can in fact do so, which we will prove to be the case in Section 4.4, however there are two additional criteria that the application of edge draw recycling must satisfy.

The first we will refer to as the *alignment property*. The alignment property requires that the number of worlds represented by each edge (i.e. K', the number of actual 'real' samples performed before the application of recycling) must divide evenly into the number of worlds represented by any other edges. For instance it is sufficient to make the number of worlds represented by each edge to be a power of 2, as we will do in our algorithm. This ensures that edge co-occurrence frequency is preserved under edge recycling. The intuitive reason is that the set of live and dead edges repeats every m steps where m is the largest common multiple of the number of worlds represented by each edge. This is made more formal in Section 4.4. The second criteria we will refer to as the *edge consistency property*. The consistency property requires that when an edge's draws are emulated using edge draw recycling all occurrences of that edge's draws must use edge draw recycling and it must use the same underlying samples and hence number of worlds. This ensures that whenever an edge's state is queried for a given world the result is always the same. Together these allow us to prove that if we use edge draw recycling

to emulate edge draws the resulting estimates that we get for reachability are unbiased. In fact, the result applies to any function computed from the edge states, although we only focus on reachability in this work. The details of how we build our algorithm that uses the edge recycling technique is discussed in Sections 4.5 and 4.6.

Edge draw recycling leaves open how one generates the underlying samples that it then uses for replication. The main requirement is that we will want that the underlying arrays that are replicated to contain a desired number of successes. We will consider two approaches both of which will be proven to yield unbiased estimates when used with edge draw recycling. The first is geometric sampling and the second is an approach that we refer to as *expectation matching sampling*. Geometric sampling is the obvious choice to consider because as already discussed it is efficient for determining after how many trials the next success will occur. Furthermore we can set the number of worlds represented such that the expected number of successes is in expectation the desired number. Where expectation matching sampling differs from geometric sampling is that it will reduce the randomness in the number of successes that occur. Specifically, expectation matching will set the number of successes that occur to its expectation and then randomize on which worlds these successes occur within the range of worlds considered. This is in essence a variance reduction technique that removes a source of unnecessary randomness. Empirically we find that it greatly reduces the relative error of the produced estimates (see Section 4.7).

4.4 Theory

In this section we will first prove that edge draw recycling produces unbiased estimates and show that this applies to the two sampling approaches that we will be using to generate the samples that edge draw recycling will replicate. We will then discuss how the two requirements, the *alignment property* and *consistency property*, are being used in the proof and their implications for the algorithm that we design. Following this we attempt to characterize the variance and associated relative error that results from applying edge draw recycling to estimate the reachability probability.

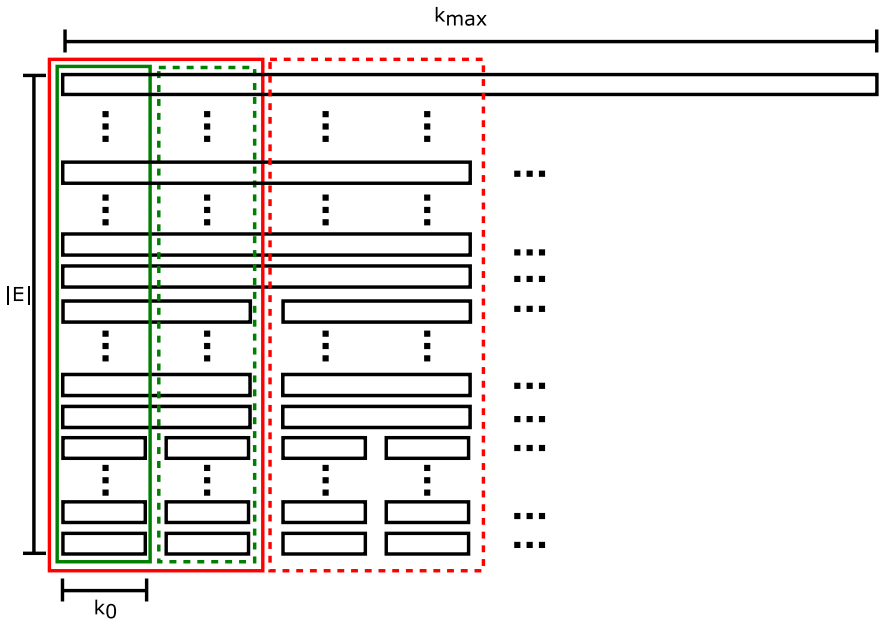


Figure 4.2: Edge Draw Recycling Example

4.4.1 Unbiased Estimation

We will now prove that the reachability estimate produced when using edge draw recycling is unbiased.

Theorem 5. *Let $\mathscr{G}$ be the distribution over possible worlds (i.e. deterministic graphs) represented by an uncertain graph, $\mathscr{G}'$ be the distribution over possible worlds produced when each edge's draws may be emulated through edge draw recycling, and $f(G)$ be a function that when given a possible world G evaluates to one or zero (e.g. one if the source reaches the target and zero otherwise) then $\mathbb{E}_{G\sim\mathscr{G}}[f(G)] = \mathbb{E}_{G\sim\mathscr{G}'}[f(G)]$, i.e., edge draw recycling is unbiased.*

Proof. We will prove this result through induction on the number of worlds (discretized into powers of two). To start off with, consider sorting all of the edges according to the number of samples drawn from them (i.e., the number of worlds their state is determined for) before the application of replication. Let k_i correspond to the i'th largest number of worlds represented in ascending order. (e.g. Suppose

there are 6 edges with array sizes $[16, 16, 32, 64, 64, 64]$, then $k_0 = 16$, $k_1 = 32$, $k_3 = 64$). As discussed in Section 4.3 all of the underlying numbers of worlds are constructed to be divisible into each other. In the following it will be assumed that this requirement is satisfied by all of the numbers of worlds being powers of 2, as this is what we will be using in our algorithm. We will reference the example diagram shown in Figure 4.2 which has the edges with shortest array length on the bottom and longest on the top. Recall that edge draw recycling can be thought of as replicating the smaller arrays of samples to produce full worlds up to the number needed by the largest array length (e.g. the length 16 arrays will be replicated 4 times and the length 32 array twice to match the length 64 array).

Base Case. Observe that if we only consider the k_0 first possible worlds these worlds are generated identical to MC simulation as no edge draws have been recycled yet (green boxed region in Figure 4.2). As MC simulation is known to be unbiased the reachability probability estimate produced by averaging whether or not s reaches t in each of these worlds is an unbiased estimate (same argument applies for an arbitrary function $f(G)$). It is important to note that for MC simulation to provide an unbiased estimate it fundamentally depends on the assumption that the possible worlds are independent. This is the case for the k_0 first possible worlds but this assumption is no longer satisfied once we begin recycling edge draws.

Inductive Step. Assume the estimate produced using worlds up to k_i is unbiased. Consider the worlds up to k_{i+1}. Observe that the k_{i+1} worlds may be split evenly into segments of length k_i and that these segments are indistinguishable from each other (i.e. the makeup of each segment in terms of the edge array replicas it contains are identical). This perfect segmentation always happens because all array lengths are powers of 2 and as such, it is always the case that $2^c \cdot k_i = k_{i+1}$ for some integer c. In Figure 4.2 the first application of the inductive step corresponds to considering the green boxed region and the green dashed boxed region. By the inductive hypothesis we already know the first segment provides an unbiased estimate and since all segments are indistinguishable each segment individually also provides an unbiased estimate. The estimate for the k_{i+1} worlds is equivalent to averaging the estimates of the 2^c segments. Since the estimate of each segment is unbiased the resulting estimate from the k_{i+1} worlds is unbiased (i.e. the red boxed region is unbiased).

Repeated application of the inductive step starting from the base case gives the result that the estimate produced using as many worlds as the length of the largest array provides an unbiased estimate. □

This result showing that edge draw recycling is unbiased immediately implies that edge draw recycling when using *geometric sampling* is also unbiased. It is only necessary to show that geometric sampling is unbiased on the base case, which is before any edge draw recycling has occurred. Geometric sampling being unbiased on the base case (i.e. using Geometric sampling for MC simulation in the place of Bernoulli draws) follows from the definition of the Geometric distribution in terms of Bernoulli trials, which implies that whether Bernoulli or Geometric trials were used is indistinguishable.

A similar argument also is sufficient to show that edge recycling using what we refer to as *expectation matching sampling* is also unbiased. Observe that we only need to revise the base case after which the same induction argument applies. Notice that expectation matching results in the possible worlds becoming correlated. This may be seen by considering that the knowledge that in the first world an edge is dead increases the probability that the edges in the other worlds are live as together the number of live edges is matched to the expectation. However, this cross world correlation does not introduce any bias, as we will now explain. Consider one of the possible worlds in isolation. Observe that the probability that an edge is live in that world using the expectation matching sampling is equal to the probability that it is live under independent sampling. This follows from the expectation matching construction. As such each individual possible world provides an unbiased estimate. Furthermore averaging unbiased estimates always produces an unbiased average estimate. Importantly, the fact that the estimates that are being averaged are correlated cannot introduce bias. It then follows that the base case also holds for expectation matching sampling and hence edge draw recycling using expectation matching sampling produces unbiased estimates.

In addition to the *alignment property* that is explicitly used in the proof it should be noted that it is implicitly asserted that each edge also satisfies the *consistency property* (i.e. when $f(G)$ is applied it only sees edges through the edge draw recycling). While this may seem obvious it results in some important limitations on

how the algorithm we develop can manipulate how it uses edge draw recycling. The main consequence of the consistency requirement is that if we are to alter the edge draw recycling of an edge (e.g. say we want to reduce the number of 'real' samples and increase replication for efficiency reasons) this must be reflected everywhere that edge's success indices have been indirectly used.

4.4.2 Variance Characterization

We would like to characterize the variance so that we can understand how it depends on the number of success indices, ℓ, used and the probability of source reaching the target, $R(s,t)$. In particular, we would like to be able to show that the variance of our approach depends on $R(s,t)$ in a manner that results in the relative error not depending on $R(s,t)$. Recall that existing approaches suffer from their relative error increasing as $R(s,t)$ decreases unless more samples, and hence more running time, are used.

Currently we are only able to formally characterize the variance that results from the application of edge draw recycling for a path, which we present in the following. However it is likely the case that the variance of the reachability estimate behaves similarly. Our experiments present in Section 4.7 also provide empirical evidence to support this. The intuitive argument why the variance of the reachability estimate should be similar is that it may be viewed as requiring the estimation of the cardinality of a set of success indices, representing the worlds in which the source reached the target, which is produced by taking the union of the success indices of all paths from the source to the target. We show that each path's set of success indices has its cardinality variance bounded. Furthermore the corresponding *set* of success indices that reach the target will in fact be known, not just cardinality estimations. As such the union to find the number of success indices present at the target, representing any of the paths succeeding, may be performed explicitly.[2] Consequently we conjecture that the variance of the cardinality of the union is no more than that of the path with the maximum variance. In fact, we

[2]Paths' success indices are unioned also at intermediate nodes. This corresponds to determining the worlds in which intermediate nodes are reached by the source. The worlds in which a node's in-neighbours are reached by the source along with the corresponding edge's success indices may be used to determine in which worlds the node is reached by the source. Doing so avoids the presence of exponentially many paths in general from becoming a problem.

believe it is likely to be closer to a weighted average of the variances of the paths where the weight associated with each path's variance is based on the number of entries it contributes to the union.

Path Variance Analysis. Consider a path containing edges $e_1, ..., e_i, ..., e_d$ with associated probabilities $p(e_i)$. Let ℓ be the number of success indices that is used. This controls the number times edges are sampled before edge recycling is applied. Consequently, the number of worlds, K, that the first edge, e_1, is sampled in is at least, $K \geq \ell / p(e_i)$. The algorithm will use this many worlds rounded up to nearest power of 2.

Let $X_{i1}, X_{i2}, ...X_{iK}$ be the K independent Bernoulli random variables each with $\mathbb{E}[X_{ij}] = p(e_i)$ that correspond to the state of an edge e_i in K worlds. Let $Y_i = \sum_{j=1}^{K} X_{ij}/K$ be the resulting random variable representing the average. We will analyze the variance of Y_i.

As the variance of each independent Bernoulli random variable is $\mathrm{Var}(X_{ij}) = p(e_i)(1 - p(e_i))$ if we consider the first edge we have:

$$\mathrm{Var}(Y_1) = p(e_1)(1 - p(e_1))/K \tag{4.1}$$

$$\leq p(e_1)^2(1 - p(e_1))/\ell \tag{4.2}$$

$$\leq p(e_1)^2/\ell \tag{4.3}$$

The first line follows from the variance of the average of K identically distributed independent random variables, which is known to be the variance of the individual random variables reduced by a factor of K. Following that, the inequality, $K \geq \ell / p(e_i)$, is used to substitute for K.

When the subsequent edges are considered the number of worlds considered is increased according to $K_i \geq \ell / \prod_{j=1}^{i} p(e_j)$ for the number of worlds considered at edge e_i. This is done to compensate for later edges only being tested when the edges before them succeed. It follows that the variance of the estimation of each

edge's probability is also:

$$\begin{aligned}
\mathrm{Var}(Y_i) &= p(e_i)(1-p(e_i))/\left(K_i\prod_{j=1}^{i-1}p(e_j)\right) && (4.4)\\
&\leq p(e_i)^2(1-p(e_i))/\ell && (4.5)\\
&= p(e_i)^2(1-p(e_i))/\ell && (4.6)\\
&\leq p(e_i)^2/\ell && (4.7)
\end{aligned}$$

The multiplier $\prod_{j=1}^{i-1} p(e_j)$ is the probability that all edges prior to the edge i in the path succeed. Only when this happens is the edge e_i tested. The effect of this multiplier is canceled out by K_i.

We will now analyze the variance of the estimate of the probability of the path being formed, $\prod_{i=1}^{d} Y_i$. Since the individual edges are independent the variance of the path is:

$$\begin{aligned}
\mathrm{Var}(\prod_{i=1}^{d}Y_i) &= \mathbb{E}[\prod_{i=1}^{d}Y_i^2]-\mathbb{E}[\prod_{i=1}^{d}Y_i]^2 && (4.8)\\
&= \prod_{i=1}^{d}\mathbb{E}[Y_i^2]-\prod_{i=1}^{d}\mathbb{E}[Y_i]^2 && (4.9)\\
&= \prod_{i=1}^{d}(\mathrm{Var}(Y_i)+\mathbb{E}[Y_i]^2)-\prod_{i=1}^{d}\mathbb{E}[Y_i]^2 && (4.10)\\
&\leq \prod_{i=1}^{d}(p(e_i)^2/\ell+p(e_i)^2)-\prod_{i=1}^{d}p(e_i)^2 && (4.11)\\
&= \prod_{i=1}^{d}p(e_i)^2\left((1/\ell+1)^d-1\right) && (4.12)
\end{aligned}$$

The first line follows from the definition of variance. The following line uses the fact that the edges are independent. After that the definition of variance is used again to replace $\mathbb{E}[Y_i^2]$. The fact that $\mathbb{E}[Y_i] = p(e_i)$ along with the result for $\mathrm{Var}[Y_i]$ is then substituted in.

We can express the quantity, $(1/\ell+1)^d - 1$, using the Taylor series of the

binomial approximation:

$$(1/\ell+1)^d - 1 = \sum_{i=1}^{\infty}\prod_{j=0}^{i-1}(d-j)/(i!\ell^i) \tag{4.13}$$

Provided $d < \ell$, using infinite geometric series, this can be simplified to

$$\sum_{i=1}^{\infty}\prod_{j=0}^{i-1}(d-j)/(i!\ell^i) \leq 2\sum_{i=1}^{\infty} d^i/(2\ell)^i \tag{4.14}$$

$$= (d/\ell)/(1-(d/(2\ell))) \tag{4.15}$$

$$= 2d/(2\ell-d) \tag{4.16}$$

It can be observed that the variance is scaled by the square of the path's probability, $\prod_{i=1}^{d} p(e_i)^2$. As such the coefficient of variation, defined as the standard deviation divided by the expected value, is,

$$c_v = \sqrt{2d/(2\ell-d)} \tag{4.17}$$

This characterizes how the relative error depends on the path's probability, however, it can be observed that there is no dependence on path probability. This happens because the variance depends on $\prod_{i=1}^{d} p(e_i)^2$, which is the square of the probability of the full path being formed (i.e. the probability of the source reaching the target). Consequence of the variance decreasing in this manner as the probability of the path being formed decreases the relative error remains the same regardless of the probability of the path being formed. There is however a dependence on the length path, d. Notice that, d the path length, only grows linearly, in contrast to the path probability that in general can decrease exponentially. What this means is that for an approach who's relative error depends inversely on the probability of the path being formed, e.g. MC simulation, the amount of samples, and consequently running time, needs to be increased exponentially with the path length for the resulting relative error to not increase. In contrast, the number of success indices, ℓ, used by our approach only needs to be increased linearly with the path length, regardless of the edge probabilities, to keep the relative error from increasing.

4.5 Algorithm

Our algorithm, WIR, is divided into three main components: 1. The main loop which selects the next node to process (Algorithm 8). 2. The node processing procedure (Algorithm 9). 3. The success index sampling which may use either Algorithm 10 or Algorithm 11. The input to the main algorithm (Algorithm 8) is the source node, s, target node, t, probabilistic graph, G, desired number of success indices, ℓ, and number of repetitions to average, K. The returned value, $\bar{p}$, is the estimated probability that there exists a path from s to t. Both ℓ and K control the estimation accuracy with K increasing the accuracy at the cost of running time and ℓ increasing the accuracy at the cost of less running time than K but also increasing the memory used. This is investigated in our experiments (Section 4.7). We also defer a number of implementation details and techniques to improve efficiency to Section 4.6.

Algorithm 8 WIR

Input: s, t, G, ℓ, K
Output: $\bar{p}$

for $i = 1$ **to** K **do**
  for all $u \in V$ **do**
    $\mathscr{I}[u] = \emptyset$; $\mathscr{P}[u] = \emptyset$;
  Set r_{limit} according to numerical accuracy;
  (e.g. if 64-bit integers are used for storing success indices, $r_{limit} = 2^{64}$)
  Let A contain success index 0 and represent 1 world;
  (i.e. $A.indices = \{0\}$, $A.worlds = 1$)
  $\mathscr{I}[s] \leftarrow \mathscr{I}[s] \cup \{A\}$;
  $\mathscr{H}$.push$((1, s))$;
  while !$\mathscr{H}$.isempty() **do**
    $(r, u) \leftarrow \mathscr{H}$.pop();
    ExpandNode$(u, \min(r, r_{limit}), G, \mathscr{H}, \mathscr{I}, \mathscr{P}, \ell, t)$;
  $p_i = p[t]$;
$\bar{p} = \sum_{i=1}^{K} p_i / K$
return $\bar{p}$

The WIR algorithm performs K repetitions each of which produces an estimate of the reachability probability, p_i. These independent estimates are averaged to produce the returned estimate, $\bar{p}$, to reduce variance. We will now focus on

the creation of one such estimate (lines 2-11). The main data structures that are used are referred to as *success index sets*. Each success index set, A, has a value specifying the number of worlds that it represents, $A.worlds$, and a set of success indices, $A.indices$, that indicates in which of these worlds a success occurred (i.e. the source node reached the node that the success index set applies to). Each node, u, has a set of success index sets, $\mathscr{I}[u]$, that together represent the worlds in which the source node reaches the node u. As discussed in Section 4.3, even though each success index set only represents a particular number of worlds *through the use of replication,* whether a success occurs at at *any* given possible world is determined by $\mathscr{I}[u]$. In addition, each node has a set of success index sets, $\mathscr{P}[u]$, which are the success index sets that have been *processed*. Processed success index sets contain the success indices that have been considered for propagation to the node's neighbors. In addition, the success index sets in $\mathscr{P}[u]$ are maintained to be non-overlapping. The implementation of this is discussed in detail in Section 4.6.

Initially, $\mathscr{I}[u]$ for all nodes except the source node are empty. The collection of success index sets for the source, $\mathscr{I}[s]$, is initialized to contain one success index set which contain success index 0 and represents 1 world. Notice that this success index set represents that the source reaches itself in all possible worlds as every world in the success index set corresponds to a success index. We will say a success index set of a node has been processed once its success indices have been considered for propagation to neighbors of the node by Algorithm 9. The algorithm is designed to process success index set in order of increasing number of worlds. Intuitively, success index sets with smaller numbers of worlds correspond to higher probability events. This processing order is also required for implementation reasons that are discussed in Section 4.6 Success Index Deduplication. To support this a min-heap, $\mathscr{H}$, is used to order nodes by the minimum number of worlds one of their unprocessed success index sets represents. To begin with the source is added (line 7). Following this, the algorithm continues to remove the node that is at the top of the min-heap and calls Algorithm 9 on it until the heap is empty. The vector $p[u]$ contains the estimate of the probability of the source reaching each node (i.e. R(s,u) for each u). This will be incrementally updated each time a success index set at a node is processed. Once all success index sets have been processed $p[t]$ contains the reachability probability that we wanted to estimate.

Algorithm 9 ExpandNode

Input: $u, r, G, \mathscr{H}, \mathscr{I}, \mathscr{P}, \ell, t$

1: **if** $r < r_{limit}$ **then**
2: $C = \{A \mid A \in \mathscr{I}[u] \wedge A.worlds = r\}$;
3: $I = \bigcup_{A \in C} A.indices$;
4: **else**
5: $I = \{i \mid i \in \bigcup_{A \in \mathscr{I}[u]} A.indices \wedge i < r_{limit}\}$;
6: Remove indices in I that overlap with those in the success index sets in $\mathscr{P}[u]$.
7: **if** $I \neq \emptyset$ **then**
8: Let $P.indices = I$ and $P.worlds = r$;
9: $\mathscr{P}[u] \leftarrow \mathscr{P}[u] \cup \{P\}$;
10: $p[u] \leftarrow p[u] + |P.indices|/P.worlds$;
11: **if** $u \neq t$ **then**
12: **for all** $v \in G[u]$ **do**
13: Let $w = p(u, v)$;
14: $r_{new} = \min(r_{limit}, \max(r, 2^{\lceil \log_2(\ell \cdot r/(w|I|)) \rceil}))$;
15: Let $A.indices$ = SampleIndices(I, r, r_{new}, w);
16: Let $A.worlds = r_{new}$;
17: $\mathscr{I}[v] \leftarrow \mathscr{I}[v] \cup \{A\}$;
18: $\mathscr{H}$.push((r_{new}, v));
19: **if** $v == t$ **then**
20: $r_{limit} \leftarrow r_{new}$;

We will now discuss Algorithm 9, which handles the processing of success index set and propagation of success indices to a node's neighbors. To begin with, if $r < r_{limit}$ all of the success indices of the success index sets for the node u that represent r worlds that haven't been processed are merged into the set I. If r has reached r_{limit} then all unprocessed success index set are merged into the set I. In addition, any success indices exceeding r_{limit} are filtered out. By construction it is not possible for there to be an unprocessed success index set that represents less than r worlds as all nodes that contain such success index sets must be processed before r can be increased. Furthermore when an success index set is processed it can only create success index set with equal or higher r_{new} or with a r_{new} value equal to r_{limit}, as will be discussed in the following. The success indices that overlap with those already present in $\mathscr{P}[u]$ (i.e. corresponds to worlds where node u is already reached by the source via an alternative path) are removed. The reason this

deduplication is only with respect to the processed success index sets is discussed in Section 4.6 Success Index Deduplication. Provided I is not empty the success a new processed success index set is created using the success indices and added to $\mathscr{P}[u]$. The probability estimate of u being reached by the source is updated to reflect the newly processed success index set. Following this if u is not the target node the success indices are considered for propagation to each neighbor ($v \in G[u]$) of u (lines 10-20). Line 14 computes r_{new} which is the minimum number of worlds that is a power of 2 that if considered ensures that the expected number of success indices propagated across the edge is at least ℓ in expectation. This follows from there being currently $|I|$ success indices in r worlds and in expectation $p(u,v)$ will pass over the edge. Note that r_{new} is not allowed to be smaller than r. As discussed in Section 4.4 the recycling pattern of edges may not be changed unless it is changed globally. As such, the number of worlds represented must be kept at r or higher unless it is lowered globally, as is done by r_{limit}. r_{limit} is a global limit on the number of worlds considered that is initially set according to technical limitations (see Algorithm 8 line 4) and then lowered once the target node has been reached (line 20). Having determined how many worlds are need the SampleIndices procedure is invoked to determine the success indices that pass over the edge (i.e. in what worlds in which u has been reached by the source does u reach v). We have two different techniques that may be applied to produce the success indices (Algorithms 10 and 11) that will be discussed shortly. Once we have the success indices these along with r_{new} are used to form A a new success index set. The success index set is added $\mathscr{I}[v]$ and the node is added to the heap with its corresponding value being the number of worlds of the newly created success index set, r_{new}. Note that a node may be registered in the heap multiple times. This ensures that the node is processed according to the success index set addition that requires processing soonest (i.e. has added a success index set with lowest r_{new}). If the target node is reached then we now have established an upper limit on the number of worlds that are needed to reach the target node with the desired number of success indices, namely r_{new}. This is used to lower the value of r_{limit}. We will now demonstrate how the algorithm, as discussed so far, works with a toy example.

Example 3. Shown in Figure 4.3 is toy graph for which we will walk through how

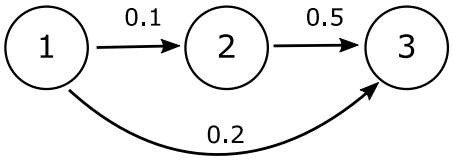


Figure 4.3: Toy example for demonstrating the algorithm.

Algorithms 8 and 9 estimate the probability of node 1 reaching node 3. Initially, in Algorithm 8 $\mathscr{H}$ contains only (1, node 1), as node 1 is the source node. As such, Algorithm 9 will be called on node 1 with $r = 1$. From the initialization, $\mathscr{I}$[(node 1)] contains one element, A, with $A.indices = \{0\}$ and $A.worlds = 1$. As such, $I = \{0\}$. Note that $\mathscr{P}$[(node 1)] thus far is empty. Lines 8-10 create P using I and then add it to $\mathscr{P}$[(node 1)]. p[(node 1)] is updated to its existing value, 0, plus $|I|/1$, which gives 1. Since node 1 is not the target node, propagation is performed across its two edges. Consider the edge from node 1 to node 2. For this example suppose $\ell = 5$. w is 0.1 which gives $r_{new} = 2^{\lceil \log_2(5 \cdot 1/(0.1 \cdot 1)) \rceil} = 2^{\lceil \log_2(50) \rceil} = 64$. Note that $2^{\lceil \log_2(50) \rceil}$ has the effect of rounding 50 up to the nearest larger power of 2. Suppose invoking SampleIndices returns the set $\{4, 16, 20, 51, 62\}$. The operation of SampleIndices will be discussed in detail in Section 4.5.1. This is used to create a success index array that is added $\mathscr{I}$[(node 2)]. (64, node 2) is then added to $\mathscr{H}$. The edge from node 1 node 3 is processed similarly. w is 0.2 which gives $r_{new} = 32$. Suppose invoking SampleIndices in this case returns the set $\{3, 10, 15, 20, 31\}$. This is used to create a success index array that is added to $\mathscr{I}$[(node 3)]. (32, node 3) is then added to $\mathscr{H}$. As node 3 is the target r_{limit} is set to 32.

Next, Algorithm 9 will be called on node 3 with $r = 32$. $I = \{3, 10, 15, 20, 31\}$ from the one element in $\mathscr{I}$[(node 3)] and since $\mathscr{P}$[(node 3)] is so far empty. Lines 8-10 create P using I and then add it to $\mathscr{P}$[(node 3)]. p[(node 3)] is updated to its existing value, 0, plus $|I|/32$, which gives 0.15625. Node 3 is the target so we return to Algorithm 8. Next, Algorithm 9 will be called on node 2 with $r = 32$. Note that r that was associated with node 2 has been overridden by r_{limit}, as it is smaller. As $r = r_{limit}$ is now the case the success indices will now be restricted

down to r_{limit} worlds. Consequently, $I = \{4, 16, 20\}$. The other success indices now exceed the number of worlds under consideration. Lines 8-10 create P using I and then add it to $\mathscr{P}$[(node 2)]. p[(node 2)] is updated to its existing value, 0, plus $|I|/32$, which gives 0.09375. Since node 2 is not the target node propagation is performed across its one edge. Consider the edge from node 2 to node 3. Since $r = r_{limit}$ it is the case that $r_{new} = r_{limit}$. Suppose invoking SampleIndices in this case returns the set $\{16, 20\}$. This is used to create a success index array that is added to $\mathscr{I}$[(node 3)]. (32, node 3) is then added to $\mathscr{H}$. Finally, Algorithm 9 will be called on node 3 with $r = 32$. $I = \{16\}$ after removal of the success indices that overlap with those in $\mathscr{P}$[(node 3)]. Notice that the success index 20 was removed. This corresponds to the case where node 3 has been reached simultaneously via the path from node 1 through node 2 to node 3 and also the edge from node 1 directly to node 3. Lines 8-10 create P using I and then add it to $\mathscr{P}$[(node 3)]. p[(node 3)] is updated to its existing value, 0.15625, plus $1/32$, which gives 0.1875. Node 3 is the target so we return to Algorithms 8. The heap is now empty and the value p[(node 3)] $= 0.1875$ is the result for this repetition.

The described procedure is repeated K times and the average of the p[(node 3)] results from each repetition is returned as the final estimate. The randomness in this procedure originates from the the success indices returned by the invocations of SampleIndices. □

4.5.1 Success Indices Sampling

Algorithm 10 implements the SampleIndices procedure using geometric sampling where as Algorithm 11 does so using a sampling strategy that we refer to as *expectation matching*. Both use edge draw recycling as needed to extend the number of worlds given r_u, to the desired number of worlds, r_v. The inputs are the success indices, I_u, and corresponding number of worlds, r_u, the requested number of worlds to test, r_v, and the edge's probability w. The output is the success indices I_v that correspond to the request number of worlds and represent the worlds in which a success index from I_u was present and the edge was live. These success indices correspond to the worlds in which the source reaches the node v via the edge in consideration.

Algorithm 10 SampleIndices-Geometric

Input: I_u, r_u, r_v, w
Output: I_v

1: $I_v = \emptyset$
2: $i = 0$; $i_w = 0$;
3: **while** $i_w < r_v$ **do**
4: $\quad j \sim \text{Geom}(w)$;
5: $\quad i \leftarrow i + j$;
6: $\quad ind = i \mod |I_u|$;
7: $\quad i_w = r_u(i - ind)/|I_u| + I_u[ind]$;
8: $\quad$ **if** $i_w < r_v$ **then**
9: $\quad\quad I_v \leftarrow I_v \cup \{i_w\}$;
10: $\quad i \leftarrow i + 1$;
11: **return** I_v;

Algorithm 10's use of geometric sampling is an efficient and equivalent way to implement the following more easily understood procedure. By construction it is the case that r_u and r_v are powers of 2 with $r_u \leq r_v$. As such r_v/r_u is a positive integer. The application of edge recycling can be understood as taking the possible worlds represented by I_u and r_u and replicating them r_v/r_u times. The result is a new set I'_u that corresponds to r_v worlds and contains $(r_v/r_u)|I_u|$ success indices with the replica indices generated by offsetting the original indices in I_u by $c|I_u|$ for the c'th replica. The set I_v is then constructed by for each element in $|I'_u|$ performing a Bernoulli trial with probability w. If the trial is 1 this corresponds to the edge being live and the success index is added to I_v otherwise it is not. While simple, this direct way of constructing I_v quickly becomes inefficient if the number of replications is large since the amount of work performed scales with $(r_v/r_u)|I_u|$. Recall from Algorithm 9 line 14 the number of worlds requested, and hence number of replications, is controlled by the edge's probability, w. Hence, the number of replications will only be large if the edge's probability is small. Geometric sampling enables producing a result that is identical to the Bernoulli trial approach but only with $|I_v|$ work.

In Algorithm 10 i_w is the most recently generated success index, so long as it is less than the desired number of worlds, r_v, the next success index is generated. The variable i tracks the number of times the edge has been tested (i.e. the number

of Bernoulli trials that would have been performed so far if geometric sampling wasn't used). The geometric random variable draw on line 4 determines how many times, j, the edge is tested and fails before the next time it is tested and it succeeds. In addition to the number of failed attempts i also needs to be incremented for each time the edge succeeds. This is accounted for by the increment on line 10. i_w is computed from i by first computing the remainder from $ind = i \mod |I_u|$ and then using this to compute $(i - ind)/|I_u|$ which determines which array replica it is from. Once which replica to use has been determined the resulting success index is produced by combining the replica offset, $r_u(i - ind)/|I_u|$, which accounts for the number of replicas that have been cycled over, and then indexing into the current replica, $I_u[ind]$. If the resulting success index, i_w, is less than the desired number of worlds, r_v, then it is added to I_v otherwise the procedure finishes returning I_v. We will now go over an example.

Example 4. Suppose the input is $I_u = \{3,6,13\}, r_u = 16, r_v = 32, w = 0.4$. Suppose the 1st geometric random draw is $j = 1$. Hence, $i = 1$ and $ind = 1 \mod 3 = 1$. i_w is, 1) 0 (since $i - ind = 0$), 2) plus $I_u[1] = 6$, which gives 6. As, $i_w = 6$ is less than r_v, 6 is added to I_v. i is then incremented to 2. Suppose the 2nd geometric random draw is $j = 3$. Hence, $i = 5$ and $ind = 5 \mod 3 = 2$. i_w is, 1) 16 (from the number of array replications that have been pasted over, $(i - ind)/|I_u|$, which is 1, multiplied by the array length, r_u, which is 16), 2) plus $I_u[2] = 13$, which gives 29. As, $i_w = 29$ is less than r_v, 29 is added to I_v. i is then incremented to 6. Suppose the 3rd geometric random draw is $j = 0$. Hence, $i = 6$ and $ind = 6 \mod 3 = 0$. i_w is, 1) 32 (from the number of array replications that have been pasted over, $(i - ind)/|I_u|$, which is 2, multiplied by the array length, r_u, which is 16), 2) plus $I_u[0] = 3$, which gives 35. $i_w = 35$ is not less than r_v. As such the algorithm returns $I_v = \{6,29\}$. □

We will now discuss our second technique for the SampleIndices procedure that we refer to as expectation matching sampling. In contrast to geometric sampling, where the resulting number of success indices can vary substantially about its expected value, in expectation matching sampling the number of success indices is constructed to be essentially equal to the expectation. Doing so does not introduce any bias into the estimation, as shown in Section 4.4, and serves to reduce the variance of the estimation. First the expected number of success indices, a, is

Algorithm 11 SampleIndices-ExpectationMatching

Input: I_u, r_u, r_v, w
Output: I_v

1: $I_v = \emptyset$
2: $a = w \cdot (|I_u| \cdot r_v / r_u)$
3: Probabilistically round a to integer.
(Round up with probability $a - \lfloor a \rfloor$, round down otherwise.);
4: $I_r \leftarrow$ Draw a times uniformly at random without replacement from multiset containing integers in range $[0, (r_v/r_u) - 1]$ where each integer is replicated $|I_u|$ times.
5: **for all** $i_r \in I_r$ **do**
6: Draw i uniformly at random from I_u excluding entries that have already been drawn from the i_r'th replica.
7: $i_w = i_r \cdot r_u + i$;
8: $I_v \leftarrow I_v \cup \{i_w\}$;

computed on line 2. Since the number of success indices is discrete this value is rounded using randomized rounding as described on line 3. Having determined the number of success indices these must be drawn uniformly at random without replacement from the r_v/r_u replicas. This is done by first drawing which replica each success index is from (line 4). After that each of these is drawn from their associated replica (line 6). Put together these give the success index, i_w, that is computed on line 7 and added to the set I_v, which is returned once all the success indices have been generated.

Example 5. Suppose the input is $I_u = \{3,6,13\}, r_u = 16, r_v = 32, w = 0.4$. Hence, $a = 0.4 \cdot (3 \cdot 32/16) = 2.4$. As such, a is rounded up to 3 with probability 0.4 and rounded down to 2 with probability $1 - 0.4$. Suppose a is rounded down to 2. The multiset that is drawn from on line 4 is $\{0,0,0,1,1,1\}$. Suppose that the multiset $I_r = \{1,1\}$ is drawn. Suppose the first entry drawn from replica 1 is 13. The resulting i_w is then 16 (from $i_r \cdot r_u$) plus 13 which gives 29. This is added to I_v. Suppose the second entry drawn from replica 1 is 3. Note that the only possibilities are 3 and 6 as 13 has already been drawn from this replica. The resulting i_w is then 16 (from $i_r \cdot r_u$) plus 3 which gives 19. This is added to I_v. The algorithm then returns $I_v = \{19,29\}$. □

4.6 Implementation Details

We will now discuss a number of implementation details.

Success Index Deduplication. A very important step in Algorithm 9 is the removal of success indices that overlap. This corresponds to identifying worlds where the node has been reached by the source via multiple paths. Deduplication prevents double counting which if allowed to happen would clearly result in the reachability estimate being incorrect. The reason deduplication is not as simple as set union is that the success index set correspond to varying number of worlds and the deduplication must be done against their replicated representation *without performing the replication*. A trivial solution is to replicate the sets till their sizes match however that is clearly impractical. To get around this, notice that it is relatively straightforward to deduplicate via removal of success indices from a success index set that is of a larger size (i.e. represents more worlds) against one of smaller size. This is because we can take the modulo of the success index with respect to the number of worlds represented by a smaller success index set and then check if the resulting remainder is in the set or not. However, this does not work in the other direction. Notice that attempting to remove a success index from a success index set that represents fewer worlds will cause the success index of the smaller success index set to 'fragment'. This occurs because the replication pattern of the smaller success index set is disrupted by the removal of a success index that is replicated at lower frequency (i.e., is from an success index set that represents more worlds). Note that simply removing success indices from the larger success index set is not an option as it is the unprocessed success index set that must have the success index removed or it would incorrectly result in a success index being reprocessed.

We avoid this problem by processing the success index sets in order of increasing number of represented possible worlds. This ensures that all the processed success index sets will represent equal or smaller success index sets than the one currently under consideration. This corresponds to the case where the deduplication can be performed efficiently. Processing the success index sets in this order is supported by the heap $\mathscr{H}$ which allows us to identify and process the nodes that have success index sets which represent the smallest number of worlds.

Example 6. Shown in Figure 4.4 is a toy graph that we will use to demonstrate the

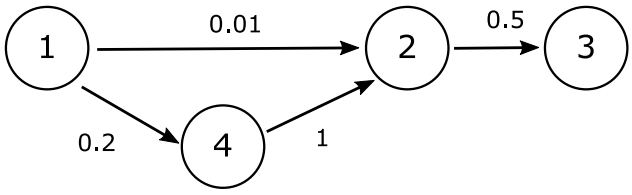


Figure 4.4: Toy example for demonstrating deduplication.

importance of the order the success index sets are processed in. Suppose that we want to find the probability of node 1 reaching node 3. We will now show what goes wrong if if success index sets are not processed in order of those that represent the smallest number of worlds.

Initially, in Algorithm 8 $\mathscr{H}$ contains only (1, node 1), as node 1 is the source node. As such, Algorithm 9 will be called on node 1 with $r = 1$. From the initialization, $\mathscr{I}$[(node 1)] contains one element, A, with $A.indices = \{0\}$ and $A.worlds = 1$. As such, $I = \{0\}$. Since node 1 is not the target node propagation is performed across its two edges. Consider the edge from node 1 to node 2. For this example suppose $\ell = 5$. w is 0.01 which gives $r_{new} = 512$. Suppose invoking SampleIndices returns the set $\{71, 96, 164, 266\}$. This is used to create a success index array that is added $\mathscr{I}$[(node 2)]. (512, node 2) is then added to $\mathscr{H}$. The edge from node 1 node 4 is processed similarly. w is 0.2 which gives $r_{new} = 32$. Suppose invoking SampleIndices in this case returns the set $\{3, 10, 15, 20, 31\}$. This is used to create a success index array that is added $\mathscr{I}$[(node 4)]. (32, node 4) is then added to $\mathscr{H}$.

Now consider what happens if node 2 is process before node 4. That is, Algorithm 9 will be called on node 2 with $r = 512$. Our algorithm will not do this as the smallest success index set size associated with node 4 is 32 which is smaller than that of node 2, which is 512. $I = \{71, 96, 164, 266\}$ from the one element in $\mathscr{I}$[(node 2)]. Note that P is created using I and then added to $\mathscr{P}$[(node 2)] Consider the edge from node 2 to node 3. w is 0.5 which gives $r_{new} = 1024$. Suppose invoking SampleIndices returns the set $\{164, 266, 778\}$. This is used to create a success index array that is added $\mathscr{I}$[(node 3)]. (1024, node 3) is then added to $\mathscr{H}$. As node 3 is the target r_{limit} is set to 1024. Next, suppose node 4 with $r = 32$ is processed by Algorithm 9. $I = \{3, 10, 15, 20, 31\}$ from the one element in

$\mathscr{I}$[(node 2)]. Since the edge probability is 1, $r_{new} = 32$ and invoking SampleIndices returns $\{3, 10, 15, 20, 31\}$. This is used to create a success index array that is added $\mathscr{I}$[(node 3)]. (32, node 2) is then added to $\mathscr{H}$.

Now, when Algorithm 9 is called on node 2 with $r = 32$ a problem will be encountered. The issue is with line 6. Note that prior to line 6, $I = \{3, 10, 15, 20, 31\}$. $\mathscr{P}$[(node 2)] contains one element, P, with $P.indices = \{71, 96, 164, 266\}$ and $P.worlds = 12$.Observe that 266 in P overlaps with 10 in I because 266 mod 32 = 10. The problem is that 266 can not be removed at this point. It has already been propagated across the edge when node 2 was processed previously. Furthermore, removing 10 from I is not correct. Doing so would represent 10 overlapping in all copies but this is not the case only 266 overlaps which is only one case of overlap out of 16 copies. At this point the only way to proceed would be to replicate the success index set 16 times and only removal the case that should be removed. However, this obviously is very inefficient as there will then be a greatly increased number of success indices that need to be propagated.

This problem does not occur if the nodes are processed in the correct order. Processing the nodes in increasing order of number of worlds (i.e. r) ensures that the situation of r being less than $P.worlds$ for any processed success index set cannot happen. As a result, this problematic case where deduplication cannot be performed efficiently will not happen. □

Eagerly Eliminating Target Success Indices. Once a success index has been found at the target any further copies of this success index reaching the target will be removed by deduplication. Continuing to propagate such success indices through the network is unnecessary as removing them cannot change the resulting reachability probability estimate. Once processed the target's success index set can be checked whenever a node is expanded to prevent success indices that overlap with those already present at the target from being propagated. Specifically, the indices in I are checked for overlap with the processed success index sets in $\mathscr{P}[t]$ by adding the following line to Algorithm 9 after line 6:

Remove indices in I that overlap with those in the success index sets in $\mathscr{P}[t]$.

Limiting the number of Worlds. The trivial initialization of the global maximum

number of worlds, r_{limit}, to 2^{64}, shown Algorithm 8, simply uses the maximum that can be represented when using 64-bit unsigned integers. While it is eventually revised when a path is found to the target, having an unnecessarily high r_{limit} will result in the generation of success indices that will ultimately be discarded when the r_{limit} is reduced. To improve efficiency we would like a tighter initial value for the r_{limit}. Since we will typically want to perform more than one repetition (i.e. $K > 1$) we can reuse the r_{limit} from the first repetition as opposed to completely restarting from scratch. However, this does not help with the first repetition. For this purpose we use the probability of the most probable path from the source to the target. Specifically, let p_{max_path} be the probability of the most probable path from the source to the target being present, which is simply the product of the probabilities of the edges needed to form it. We can then use, $r_{limit} = 2^{\lceil \log_2(\ell / p_{max\_path}) \rceil}$. While still relatively loose in general it is still dramatically better than the technical limit. Furthermore, the path with maximum probability can be efficiently found by applying Dijkstra's algorithm.

In addition to the global maximum number of worlds we would like more fine-grained control to keep the number of success indices at intermediate nodes, not just at the target node, at the desired number of success indices. Notice that the number of worlds used when processing each edge is aggressively increased so to ensure that even if there is only one path to a node it will attain the desired number of success indices. However, the side effect of this is that if a node is reached via many alternative paths it may end up with far more than the requested number of success indices. If we knew the probability with which the source reached each intermediate node then we could be more conservative in increasing the number of possible worlds considered. Specifically, if we knew the reachability probability for each intermediate node v, $R(s,v)$, then we could limit the increase of r_{new} to at most $2^{\lceil \log_2(\ell / R(s,v)) \rceil}$. Doing so accounts for the node being reached by multiple paths and as such it is not necessary to have ℓ success indices from each path. Of course, we do not know $R(s,v)$. However, even a loose underestimate can be useful in preventing unnecessary work on a low probability edge if the node has already been reached by a higher probability path. The values $p[u]$ that are computed on the fly by using their processed success indices can serve this purpose. The calculation

of r_{new} can then be revised to be the following,

$$r_{new} = \min(r_{limit}, \max(r, \min(2^{\lceil \log_2(\ell/p[v]) \rceil}, 2^{\lceil \log_2(\ell \cdot r/(w|I|)) \rceil}))); \tag{4.18}$$

Note that $p[v]$ will be an underestimate until all of the success index sets across all nodes complete processing, as it progressively incorporates alternative paths. Despite this, it can be used to help prevent the number of worlds used from being excessively increased.

Recall that the key properties that we must ensure for edge draw recycling to be unbiased are the *alignment property* and the *consistency property*. The alignment property continues to be satisfied as the number of worlds, r_{new}, continues to be a power of 2 because $\ell/p[v]$ is rounded up to the nearest larger power of 2 by the calculation, $2^{\lceil \log_2(\ell/p[v]) \rceil}$. The consistency property satisfied because r_{new} is at least r. The only way the consistency property can be violated is if the number of worlds, r_{new}, is less than the number of worlds used at previous edges that were crossed to reach this node. Specifically, if the number of worlds used is less than than the number of worlds used by previous edges that were passed to reach this node then the smaller number of worlds used at this node masks the large number of worlds used previously. This causes a problem because then it would appear that these previous edges used fewer worlds than they actually did. This breaks the consistency property. However, because nodes are processed in order of increasing number of worlds and r is maximum thus far this violation cannot happen.

4.7 Experiments

The aim of our experiments is to assess the effectiveness of our proposed algorithms against a variety of approaches and state-of-the-art algorithms for reachability probability estimation. We perform tests on datasets and associated edge probabilities that have been previously used to benchmark the performance of existing algorithms for this problem [54].

We expect our algorithms to attain more *consistent relative error* compared to existing approaches. They are designed with the goal of doing so without incurring additional running time costs. We will empirically assess whether this is the case. To do so we will in Section 4.7.2 compare the algorithms on their estimates' aver-

Dataset	type	nodes	edges
LastFM	directed	6,899	23,696
Nethept	directed	15,233	62,774
DBLP	undirected	1,291,298	7,123,632
BioMine	mixed	1,508,587	32,761,889
LiveJournal	directed	4,847,571	68,993,773

Table 4.1: Datasets

age relative error with respect to the amount of running time they take. While this provides a coarse-grained assessment of the algorithms, taking the average relative error can mask an algorithm's poor performance on a small number of source target pairs. To address this we will also assess the *distribution* of the relative errors over source target pairs in Section 4.7.3. This will also enable us to empirically evaluate whether our algorithms achieve more consistent relative error over all source target pairs. Finally, in Section 4.7.4 we will assess the relationship between the relative error of the estimates produced and the value of the probability of the source reaching the target that is to be estimated.

4.7.1 Experimental Configuration

Datasets. We perform experiments on five real network datasets (see Table 4.1). As the algorithms considered operate on directed graphs, each undirected edge in the datasets is converted into two directed edges. The datasets for Nethept, LastFM and DBLP were obtained from the authors of [54]. The BioMine dataset (specifically biomine_3_oct_2018) was obtained from https://biomine.ijs.si/downloads/ on 2021-09-29. The edge probabilities for Nethept, LastFM and DBLP datasets are as set in [54]. They considered two different edge probability sets for DBLP. Here we use the setting referred to as DBLP_0.05. The BioMine dataset already has edge probabilities associated with each edge which are what we use. The dataset LiveJournal, obtained from [60], is considered with two different settings: one-over in-degree, which is often considered in the literature on influence maximization, and drawn uniformly at random form the range 0 to 0.01. These networks and associated edge weights provide test cases that cover a wide range of possible situations that correspond to whether there are s-t pairs that have high, medium or low

probability of the source reaching the target. Specifically, LastFM and BioMine have high probabilities, NetHept and DBLP have a range of medium probabilities and the two LiveJournal case have low probabilities. This can be seen from the distribution of s-t pair reach probabilities reported in Figure 4.12.

Algorithms. Our proposed algorithms will be referred to as WIR-G and WIR-E. WIR-G uses geometric sampling while WIR-E uses expectation matching sampling. The recursive sampling algorithms (RHH and RHT) proposed in [48] will be compared using their implementation that was made publicly available by the authors. As noted in Section 4.2 their code is slightly different than described in the pseudo-code of [48] where the actual code corrects for a source of error that is present in the pseudo-code. Ke et al. [54] also reimplemented RHH for comparison in their survey. We will refer to their implementation of RHH as RHHs. We also use the implementations of [54] for the following approaches referred to as RSS, MC and LP. Recursive stratified sampling, RSS, [62] is a generalization of RHH were multiple edges are split on simultaneously. The hyper-parameter controlling the number of edges split on is set to 4 as done by the code obtained from [54]. LP [65] is the approach that makes use of geometric sampling to make edge sampling more efficient. The implementation by Ke et al. [54] that we use corrects for an error present in the pseudo-code of [65]. MC refers to Monte Carlo simulation which makes use of breadth-first search that only draws edges as needed to determine if the target is reachable from the source in sampled possible worlds. ARS, which refers to *annealed rejection sampling*, is an additional algorithm that we have created in an attempt to adapt techniques from the literature on rare event estimation, for comparison purposes.

Despite it being a well established field, we are not aware of any existing works that have applied techniques developed for rare event estimation to reachability estimation. What we consider here attempts to adapt the principle behind annealed importance sampling [69] to reachability estimation. At a high level, the key idea is to decompose a rare event that is difficult to estimate into a conjunction of more probable events, and doing so makes them individually easier to estimate. The way we map this to reachability estimation is that each edge in the graph is decomposed into sub-edges such that the product of the probabilities of these sub-edges is equal to the original edge. These sub-edges are then separated out into their own graph

copies. Observe for a path to be formed in the original graph it must be present in all of the graph copies. As such the reachability problem is decomposed into identifying if a common path is present in all the graph copies. This decomposition is the annealing step after which rejection sampling, which is essentially just MC simulation, is applied to estimate the probability that a common path connects the the source to the target in all the copies. Note that this approach is specific to the case that reachability is a rare event and is not expected to be efficient otherwise.

Evaluation metric. Assessing the error of the estimates produced by reachability estimation algorithms is challenging as we are fundamentally unable to compute the true reachability probability on large graphs due to the known intractability: *s-t* reachability is #P-hard [86]. Considering only tiny graphs on which the exact reachability calculation can be performed is not an option as such graphs would be insufficient to stress test the effectiveness of the reachability estimation algorithms.

One approach to evaluation that has been used in [54] is to focus on the variance of the estimators. All of the estimation algorithms that we consider are unbiased and as such their error is entirely due to their variance. Unfortunately we have found that unless very many repetitions are performed variance is an unreliable quality metric. To see why, consider MC simulation using far too few possible world samples, attempting to estimate a very small reachability probability. Almost every MC simulation run will come back with a reachability estimate of zero. Consequently, unless very many repetitions are performed the variance will be found to be zero (as all repetitions come back with an estimate of 0)! A variance of zero would suggest the estimates are very accurate, however their relative error to the true reachability probability is 1, which is the relative error of estimating 0 for any non-zero probability, i.e., *they are in fact extremely poor estimates*. Because of this limitation and the fact that performing sufficiently many repetitions is impractical due to the amount of time such experiments would take, we instead return to attempting to use relative error as the evaluation metric. However, to do so we must have a *pseudo ground truth* of the reachability probability, which should be more accurate than any of the estimators that we are attempting to evaluate.

To produce such a pseudo ground truth we will combine together the reachability estimates produced by all of the algorithms evaluated. This will be done using a weighted average of the best estimates (i.e. highest number of samples that

completed) from all algorithms. To do so we will run each algorithm considered multiple times (100 times on the smaller datasets, LastFM and NetHept, where this is practical and 5 times on the larger datasets, DBLP and BioMine). Initially all algorithms' estimates are given equal weight to compute the average. Following that, on each iteration each algorithm's estimates are weighted according to the algorithm's estimates' average squared error to the previously computed average. This is continued until convergence, which we measure by the average changing by $< 0.1\%$. This is done separately for each source-target pair as some algorithms may produce estimates that are accurate for some source-target pairs but not for others. This weighting of the estimates to compute the average is in accordance with *maximum-ratio-combining*. Maximum-ratio combining is known to be the optimum combiner for independent additive white Gaussian noise [51]. All of the algorithms considered produce unbiased estimates. As such, their errors are entirely due to their variance which is comparable to white Gaussian noise. It follows that they may be combined to produce an estimate that is more accurate than any of them individually. This is what we will use for our pseudo ground truth with respect to which we compute the relative error.

To assess the performance of the algorithms over a variety of sources and targets we will follow the approach taken in [54]. Specifically, for each dataset, we will randomly select 100 source-target (s-t) pairs that have a shortest path of 2 between them for the smaller datasets (LastFM and NetHept), 4 for the larger datasets (DBLP and BioMine) and 2 for the dataset LiveJournal. The choice of a shortest path of 2 on LastFM and NetHept is consistent with what was used in [54]. We used 4 on DBLP and BioMine because the combination of their higher connectivity and higher probability edge weights results in high s-t reach probabilities even at a longer distance. This was not done on LiveJournal because the lower edge probabilities made is such that even with a shortest distance of 2 the s-t reach probabilities were found to already be very low. The resulting s-t reach probabilities are reported in Figure 4.12.

Selecting s-t pairs in this manner has the effect of ensuring no abnormally easy s-t pairs are present: purely randomly selected s-t pair nodes may happen to be neighbors. This cannot happen when the shortest path between the source and the target is required to be a value greater than one. It also makes the s-t

pairs considered to be in a sense similar in difficulty. Intuitively one would expect reachability estimation to become harder the further apart the source and target are and the reachability probability will also tend to decrease. Similar difficulty s-t pairs should make computing the average relative error over them not as misleading as it would otherwise be. Despite this, we find the distributions of relative errors to be quite wide when we drill down to the relative error distributions over *s-t* pairs in Section 4.7.3. In general, having *s-t* pairs that are of similar difficulty should favor the existing algorithms that are adversely affected when the reachability probability varies dramatically between *s-t* pairs of interest.

Test Environment. All experiments were run on an Intel(R) Xeon(R) CPU X5670 @ 2.93GHz server with 96GB of RAM running openSUSE Leap 15.3. All algorithms are implemented in C++ and compiled using gcc version 7.5.0 (SUSE Linux) at optimization level -O3.

4.7.2 Average Performance

Before we compare our algorithms against the competing algorithms we will first compare our algorithms against themselves with different settings of the hyper-parameter, ℓ, which controls the desired number of success indices. Following that we will compare all of the algorithms considered on the two smaller datasets, LastFM and NetHept, with respect to how their average relative error depends on the amount of running time they use. After that will consider only the more effective algorithms on the larger datasets, DBLP and BioMine.

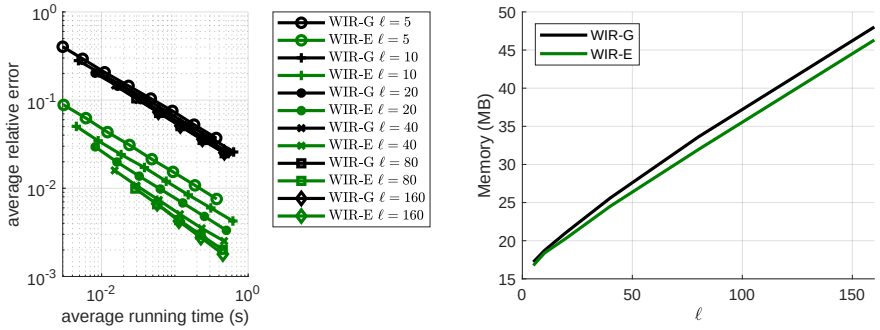


Figure 4.5: Impact of varying the desired number of success indices hyper-parameter, ℓ, on the NetHept dataset.

Hyper Parameter Configuration. Shown in Figure 4.5 is the performance of our two algorithms, WIR-G and WIR-E, as evaluated on the 100 s-t pairs considered on the NetHept dataset as the number of success indices hyper-parameter, ℓ, is varied. For each setting of ℓ the number of repetitions, K, is varied. It can be observed that WIR-G does not appear to significantly benefit from increasing the number of success indices. Increasing the number repetitions, K, of a lower number of success indices is nearly as effective in terms of the reduction in the relative error per unit running time as increasing the number of success indices. In contrast, WIR-E at the highest value of ℓ achieves the relative error attained by WIR-E at the lowest value of ℓ nearly an order of magnitude of running time faster. However, increasing ℓ comes at the cost of substantially increased memory usage (see Figure 4.5 (right)). While the absolute amount of memory involved in this experiment is tiny it is the relative amount of memory needed that is significant. Specifically, it may be projected that $\ell = 160$ will require roughly $3\times$ the memory of $\ell = 5$. For this reason in the following experiments we will use a conservative value of $\ell = 10$ for both of our algorithms.

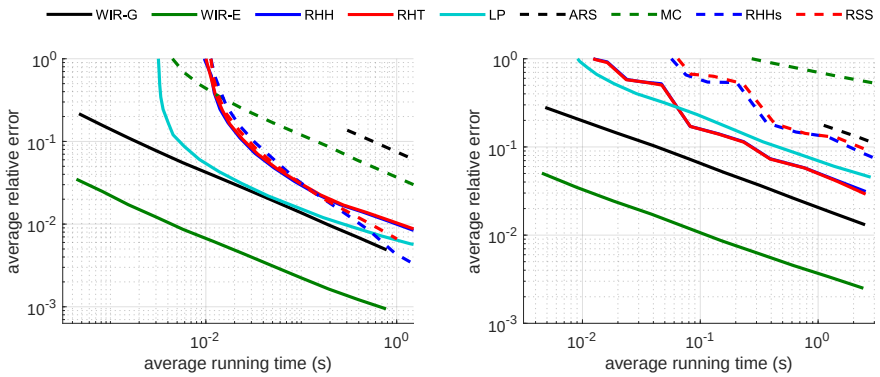


Figure 4.6: Average relative error over all s-t pairs with respect to running time on LastFM (left) and NetHept (right).

Performance on LastFM and NetHept. Here we compare all of of the algorithms on the smaller datasets, LastFM and NetHept, in terms of their average relative error over all s-t pairs with respect to the running time taken. Running time is used for this comparison as the computational cost of one 'sample' differs between algorithms making the number of samples used not useful for this comparison. The

number of samples/repetitions used is abstracted out by running each algorithm for a range of different numbers of samples/repetitions, K, and reporting the average running time taken and the associated average relative error attained. These results are reported in Figure 4.6 for LastFM (left) and NetHept (right). It can be observed that WIR-G performs at least as well as the best competing approach on both datasets while WIR-E attains roughly an order of magnitude less average relative error. Recall that the difference between WIR-G and WIR-E is how they construct the initial samples that are then recycled. WIR-G uses geometric sampling while WIR-E performs the sampling using a technique that we refer to as expectation matching sampling. It can be clearly seen that WIR-E attains consistently lower relative error than WIR-G. This is expected as WIR-E is intended to remove an unnecessary source of variance by setting number of success indices to their expectation and only randomizing on which worlds these success occur on.

The average relative error of all approaches asymptotically reduces as $1/\sqrt{t}$ where t is the running time taken. The is inline with the known error reduction of averaging independent samples/repetitions. For all algorithms being allowed to run longer translates into being able to perform more samples/repetitions. Where the algorithms differ is the constant offset which captures how efficiently each algorithm reduces the relative error. Observe that on both LastFM and NetHept the best competing approaches require nearly $100\times$ the running time to match the relative error attained by WIR-E.

Reported in Table 4.2 is the maximum memory used by the compared approaches. We do not report the memory usage with respect to the number of samples/repetitions used as this does not increase the memory required for many of the algorithms considered. This is obviously the case for our algorithms (WIR-G and WIR-E) as well as all of the other approaches that also construct their samples/repetitions independently (LP, MC and ARS). The recursive sampling algorithms can use more memory at higher numbers of samples but we observe that this is small for the number of samples considered here compared to the consistent amount of memory required regardless of the number of samples. It can be observed that LP and MC require the least memory. This shouldn't be surprising since they use no additional data other than the probabilistic graph and the bookkeeping needed to perform a breadth-first search traversal. Our algorithms in general use more

memory however our choice of the hyper parameter ℓ has kept the memory usage comparable to that of the other algorithms. Recall that it has been observed when assessing the effect of ℓ that WIR-E has the option of substantially reducing the running time needed to attain a desired relative error at the cost of increased memory usage.

Dataset	WIR-G	WIR-E	RHH	RHT	LP	ARS	MC	RHHs	RSS
LastFM	9.53	9.38	9.09	9.15	9.04	10.1	7.92	10.6	10.5
Nethept	19.1	18.7	14.1	14.1	13.9	15.2	13.9	19.8	19.8

Table 4.2: Maximum memory usage in MB

Returning to Figure 4.6, of the competing approaches RHH, RHT and LP are found to be the most effective. RHHs (the alternative implementation of RHH by [54]) and RSS are found to perform the best on the smallest dataset, LastFM, but then perform worse on the larger dataset of NetHept. We suspect that this may be caused by the different implementations using slightly different techniques to choose which edges to split on and the effectiveness of which may vary between datasets. The margin by which MC is outperformed by the other approaches grows substantially when considering NetHept compared to the smallest dataset LastFM. ARS performs very poorly on LastFM, even being outperformed by MC, but becomes more competitive on NetHept. This is not surprising as ARS is designed for rare event estimation which can be seen to be largely unnecessary on the smallest dataset and the introduced overhead is detrimental. In line with these observations, we focus our experiments on the algorithms RHH, RHT, LP and ARS in addition to our own (WIR-G and WIR-E) which are expected to be the most competitive on the larger datasets, DBLP and BioMine, which we consider next.

Performance on DBLP, BioMine and LiveJournal. Shown in Figure 4.7 is the performance as measured by average relative error compared to average running time spent on the larger datasets, DBLP and BioMine. On these datasets for WIR-G and WIR-E we first vary ℓ from 1 to 10 with 1 repetition before increasing the number of repetitions. On both datasets the performance of WIR-G becomes similar to LP. On DBLP WIR-E remains appreciably better than the best competing approach. However, on BioMine ARS performs similarly to WIR-E and RHH and RHT are slightly better. The reason that our algorithms WIR-E and WIR-G do not

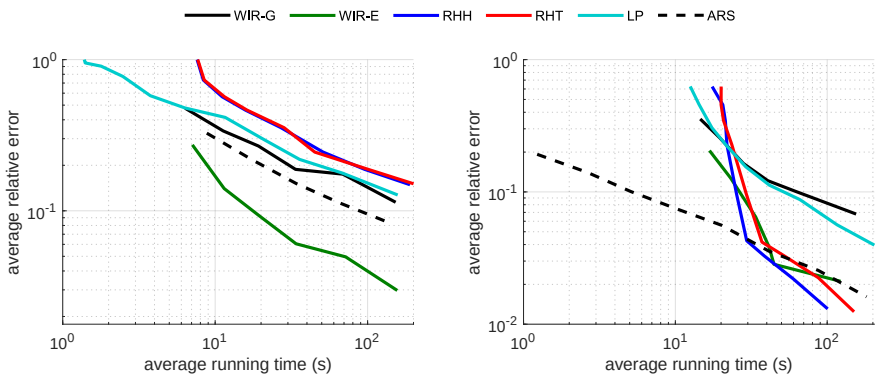


Figure 4.7: Average relative error over all s-t pairs with respect to running time on DBLP (left) and BioMine (right).

have result at lower amounts of running time is that performing one repetition with 10 success indices already takes the smallest time reported. As such, if one wants an estimate with a low relative error it can be seen that WIR-E is the best of the algorithms considered on DBLP. On BioMine at low relative error RHH and RHT perform the best while at higher relative error ARS is most effective. While RHH and RHT perform well on BioMine they perform the worst on DBLP. It would appear their effectiveness is highly dataset dependent. ARS, which performed very poorly on the smaller datasets, becomes one of the strongest contenders on the larger datasets, performing similar to RHH, RHT and WIR-E on BioMine and only worse than WIR-E on DBLP.

We conjecture that the reason that our algorithms are less effective on BioMine than the other datasets considered is because it is a considerably more dense graph, especially after accounting for many edges being undirected. Notice that BioMine contains edges that are on average quite high probability (average edge probability in BioMine is 0.14). The reason this decreases the efficiency of our algorithms is that many nodes end up with more success indices than desired, despite our attempts to compensate for this in our algorithm's design. As shown in Table 4.3 when our algorithms are unable to keep the number of success indices at the desired number, the memory usage is also increased. In general, we expect that our algorithms will perform well on any uncertain graph whose largest eigenvalue is close to 1 and that the efficiency may degrade the more that this is exceeded. The

largest eigenvalues for these datasets are: LastFM 1.0, NetHept 1.2, DBLP 51, and BioMine 260. On LiveJournal the consider edge probabilities result in relatively low eigenvalues compared to DBLM and BioMine. For edge weights chosen uniformly at random in the $(0, 0.01)$ range, the largest eigenvalue is 1.761 and for one-over in-degree the largest eigenvalue is 1. For a graph to have a large eigenvalue it must be highly connected with high probability. A largest eigenvalue near 1 is a sufficient condition for the number of success indices propagated through the graph by our algorithms to not greatly exceed the desired number of success indices on many nodes.

On LiveJournal with edge weights drawn uniformly from $(0, 0.01)$, the reachability probabilities are found to be very low despite considering s-t pairs that have a minimum distance of 2. On this dataset we we do not consider RHT. Previous results show that RHH and RHT perform almost identically. Only our approaches WIR-G / WIR-E and LP are found to perform well, as can be seen in Figure 4.8. For the uniform random edge weights case RHH is found to take nearly the maximum about of time given even with the number of samples set to 1. Furthermore, the time taken continues to increase substantially when the number samples is set to 2. As such, it is not the case that a pre-processing step is dominating the running time. Unlike our approach where one repetition already can provide a reasonable reachability estimate RHH generally needs a substantial number of samples for its estimate to become accurate. Consequently, it is not surprising that its relative error is very high. While ARS performed well on DBLP and BioMine it does not on LiveJournal. While ARS is supposed to handle cases where the probability of reaching the target is small this comes at the cost of greatly increased time need to construct samples. Furthermore, ARS was found to frequently fail to construct a valid samples resulting in the work spent on these to be wasted. It may be possible for ARS to perform better if the number of stages it used were to be tuned to the reachability probability, specifically, increasing the number of stages when the reachability probability is small. However, as ARS isn't our focus in this work and since one doesn't know the reachability probability beforehand, we do not explore this further.

For the one-over in-degree edge weights case we allow all algorithms to run substantially longer. We are not able to do this for all of the datasets and edge

weight configurations as the experiments for just this case took approximately 1 week to run. Here it can be seen that RHH begins to reduce its relative error as it is given more running time but it remains far less efficient that WIR-G, WIR-E and LP.

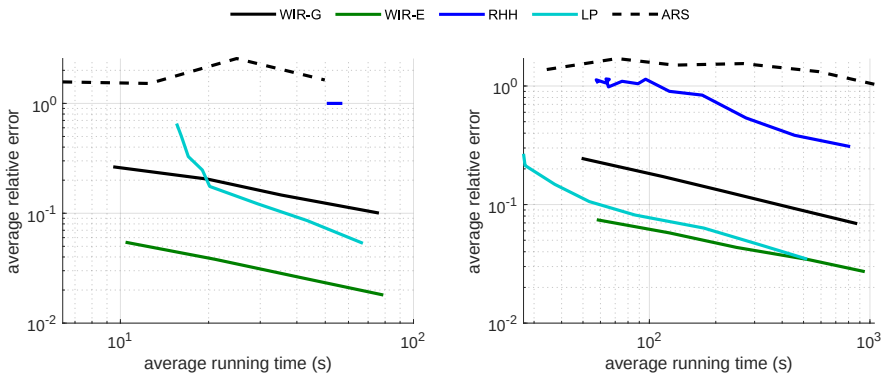


Figure 4.8: Average relative error over all s-t pairs with respect to running time on LiveJournal with uniform 0 to 0.01 edge weights (left) and one-over in-degree edge weights (right).

Dataset	WIR-G	WIR-E	RHH	RHT	LP	ARS
DBLP	9.89	8.37	1.28	1.28	1.14	1.39
BioMine	11.4	10.3	8.57	8.57	8.53	9.39
LiveJournal (Uniform)	10.1	10.1	11.2	-	10.1	10.1
LiveJournal (1/in-degree)	27.7	28.4	11.2	-	10.1	11.0

Table 4.3: Maximum memory usage in GB

4.7.3 Relative Error Distribution

In this section we will investigate the distribution of relative errors over the s-t pairs that results from the various algorithms. Reported in Figure 4.9 and Figure 4.10 is cumulative distribution of the relative error on the s-t pairs on the smaller datasets, LastFM (left) and NetHept (right) and larger datasets, DBLP (left) and BioMine (right), respectively. The reported relative errors are for the lowest relative error run attained by each algorithm when allowed a common amount of time to complete on all s-t pairs. The number of samples/repetitions used by each algorithm were

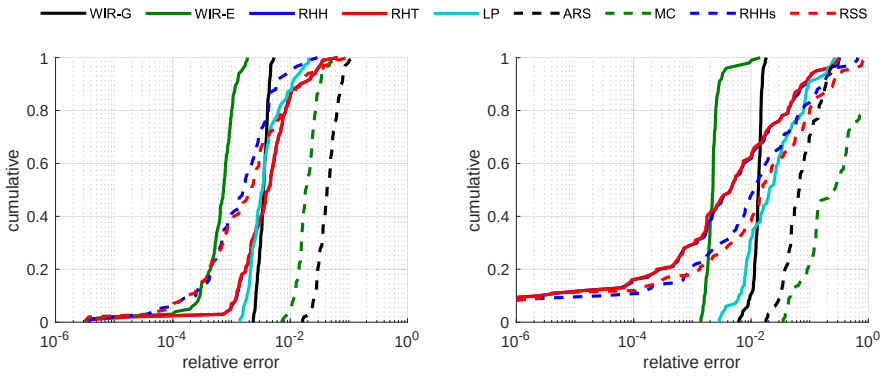


Figure 4.9: Relative error cumulative distribution over s-t pairs on LastFM (left) and NetHept (right).

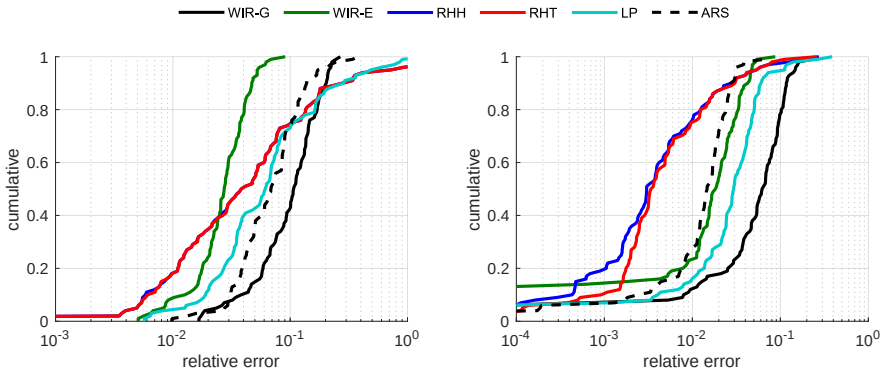


Figure 4.10: Relative error cumulative distribution over s-t pairs on DBLP (left) and BioMine (right).

increased as powers of 2. As such the actual time used is similar but doesn't match exactly. To account for this, we report the actual average time spent per s-t pair by each algorithm in Table 4.4.

From Figure 4.9 it can be seen that both of our algorithms (WIR-G and WIR-E) have a much narrow spread of relative errors. In particular, the maximum relative error of any s-t pair is less than 10^{-2} on LastFM and close to this on NetHept. In comparison the maximum relative error of the best competing approach is nearly an order of magnitude higher. Even on the BioMine dataset, which is the most challenging for our algorithms, it can be seen in Figure 4.10 (right) that our algorithm

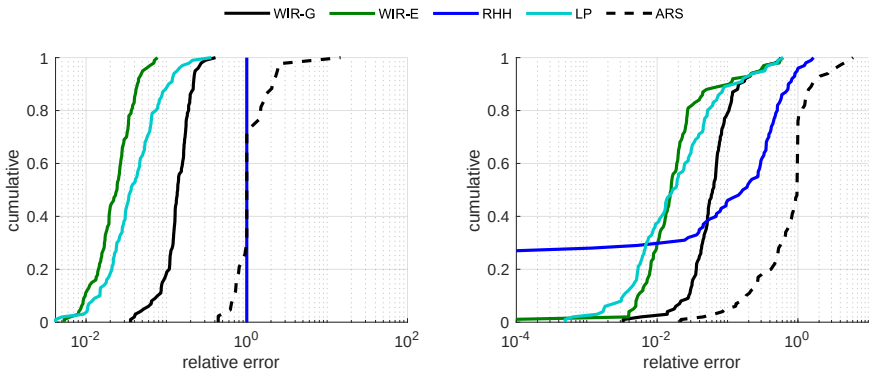


Figure 4.11: Relative error cumulative distribution over s-t pairs on LiveJournal Uniform(0,0.01) edge weights (left) and 1/in-degree edge weights (right).

WIR-E has a lower maximum (i.e., worst case) error than RHH and RHT, as seen by the cumulative distribution topping out sooner. This is despite the fact that RHH and RHT have lower average relative error, as found in Figure 4.7 (right). Only ARS achieves similar maximum relative error as WIR-E. Producing *estimates with more consistent relative error* was the main goal that we set out to achieve when designing our algorithms. These results provide empirical evidence that we have succeeded in doing so.

In contrast to our approach, the other compared approaches have much wider relative error distributions. While for some s-t pairs they can provide very accurate estimates for others the relative error remains high. This is expected as it is known that the number of samples needed by MC simulation based approaches in order to attain a desired relative error depends on the reachability probability being estimated. The basic MC simulation approach (MC) may seem to have a narrow relative error distribution but this is simply an artifact of all of the errors being high. This is a consequence of it not being able to perform enough samples to effectively reduce the relative error in the given running time. Recall that LP is equivalent to MC except that it is more efficient. This enables it to generate more possible world samples in the same amount of time. From the results produced by LP it becomes apparent that it is much more effective for some s-t pairs than others. Note that if MC were given sufficient time to perform as many samples as

LP is able to perform it would exhibit the exact same behaviour as LP; being much more effective for some s-t pairs than others. The recursive sampling algorithms, RHH, RHT, RHHs and RSS are observed to have this effect even further exaggerated. Their primary improvement over LP is to further reduce the relative error of s-t pairs that already had low relative error. We suspect that is a consequence of the recursive sampling's analytical component being able to exactly account for some of the paths which makes it very accurate on easier s-t pairs but provides limited benefit on those that are harder (i.e., involve many paths or have paths that have low probability).

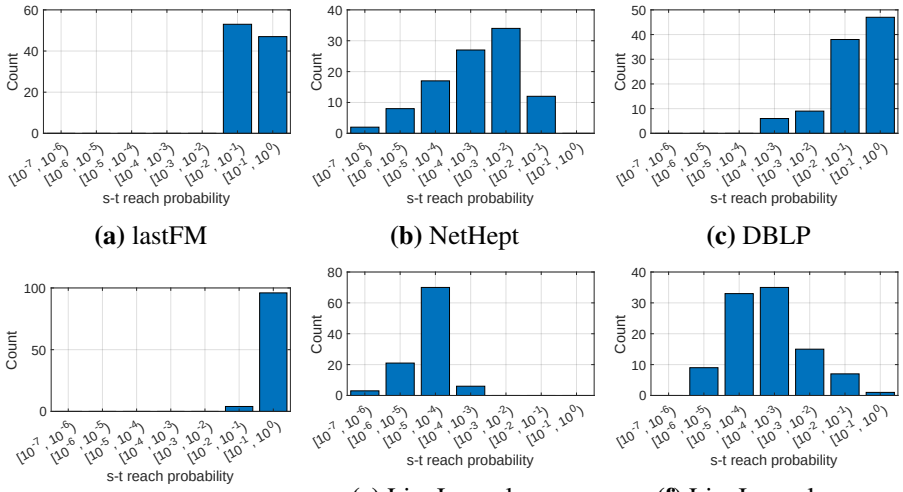
Dataset	WIR-G	WIR-E	RHH	RHT	LP	ARS	MC	RHHs	RSS
LastFM	1.56	1.55	2.70	2.99	2.76	2.46	2.99	1.80	1.01
Nethept	2.50	2.42	2.53	2.53	2.79	4.45	2.91	4.41	2.83
DBLP	153	156	189	207	157	140	-	-	-
BioMine	155	143	101	150	204	182	-	-	-
LiveJournal (Uniform)	35.4	41.7	57.2	-	67.2	49.8	-	-	-
LiveJournal (1/in-degree)	875.6	947.7	813.4	-	519.5	1047	-	-	-

Table 4.4: Average running time in seconds per s-t pair for each algorithm's relative errors reported in Figure 4.6. The selection of the number of samples/repetitions used ensures the running times of all algorithms are within one doubling of the time taken by WIR-G/WIR-E.

4.7.4 Impact of s-t Reach Probability on Relative Error

We will now assess the impact of the probability of the source reaching the target on the relative error of the estimates of this probability. Shown in Figure 4.12 is the distribution of the probabilities of the source reaching the target for the 100 s-t pairs considered on each dataset. It can be seen that distribution varies substantially between datasets. Some datasets contain a wide range of s-t reach probabilities while others have primarily high reachability probabilities (i.e. larger than 0.1). As discussed previously, we expect existing approaches to struggle to produce estimates with low relative error when the probability to be estimated is small.

Shown in Figures 4.13, 4.14 and 4.15 is the impact of the probability of the source reaching the target on the relative error of the estimates produced by the



(a) lastFM **(b)** NetHept **(c)** DBLP

(d) BioMine **(e)** LiveJournal Uniform(0,0.01) **(f)** LiveJournal 1/in-degree

Figure 4.12: s-t reach probability distribution of the s-t pairs considered on each dataset.

algorithms that have been found to be most effective in our previous experiments: WIR-G, WIR-E, RHH, LP, and ARS. RHT is not included due to its performance being nearly identical to that of RHH. To reduce clutter on the plots, instead of reporting a scatter plot of all of the results of the s-t pairs we have grouped the s-t pairs into groups of 10 according to their s-t reach probability and then report the median of the group. In addition, a linear fit in log-space is reported to make the overall trend for each algorithm's data points easier to discern. Note that in log-space a linear fit corresponds to the equation, $y(x) = \beta x^{\alpha}$, where the slope is α and the offset is β. The running time of all algorithms is set to be be similar in the same manner as described in Section 4.7.3.

The main takeaway from these results is that our algorithms, WIR-G and WIR-E, exhibit a much flatter trend than the existing algorithms. A flat line indicates $\alpha = 0$ which corresponds to the algorithm's relative error having no dependence on s-t reach probability. We consider this to be the ideal case as this means one can attain consistently accurate estimates regardless of the value of the probability being estimated. Notice that a wide range of s-t reach probabilities is needed to establish the relationship between the s-t reach probability and the relative error.

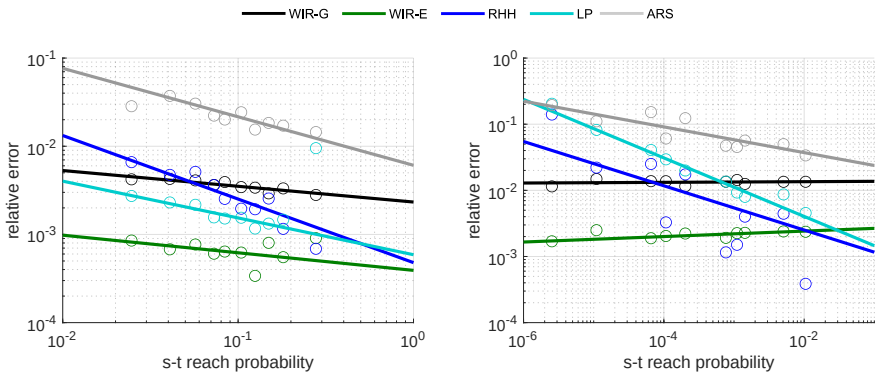


Figure 4.13: Relative error vs. s-t reach probability on LastFM (left) and NetHept (right).

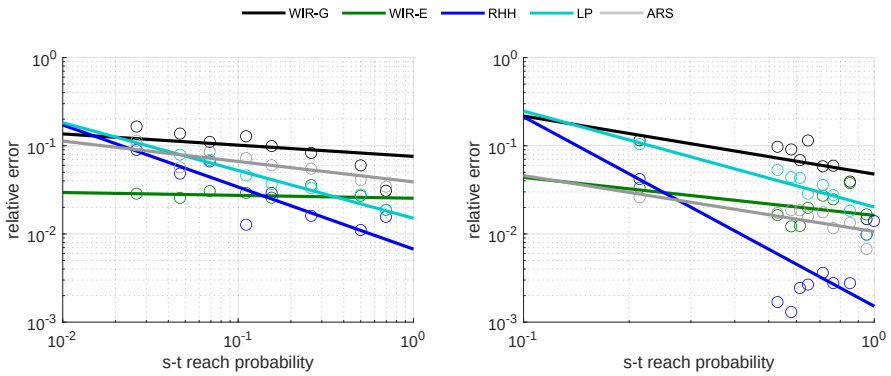


Figure 4.14: Relative error vs. s-t reach probability on DBLP (left) and BioMine (right).

As a result, those datasets where the s-t reach probabilities vary over a wide range (i.e., NetHept and LiveJournal) are more reliable indicators of this relationship than datasets (e.g., BioMine) where most s-t reach probabilities are concentrated in the high range. WIR-G and WIR-E still exhibiting a slight slope (e.g. in Figure 4.15) may be attributed to the dependence of the relative error on the path length. If the s-t reach probability is low it is more likely the case that the short paths from the source to the target were low probability making longer paths comparatively more important. The presence of a slight positive slope in in Figure 4.13 (right) is most likely due to random variation and we expect it to disappear if a larger

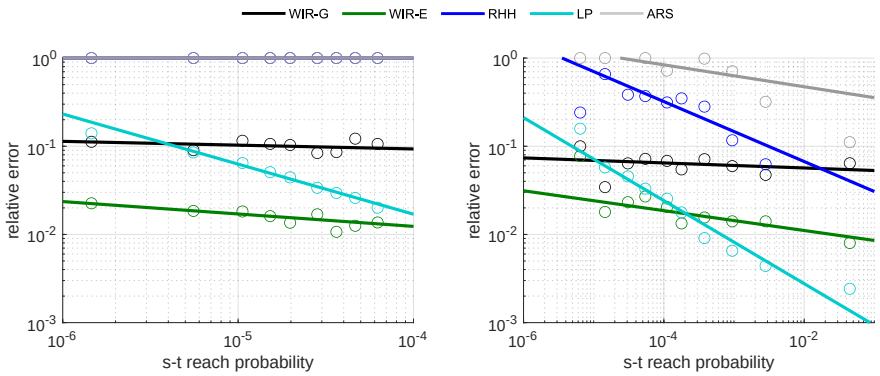


Figure 4.15: Relative error vs. s-t reach probability on LiveJournal Uniform(0,0.01) edge weights (left) and 1/in-degree edge weights (right).

number of s-t pairs were considered. The existing algorithms of LP and RHH are expected to exhibit a slope of approximately $\alpha = -0.5$. This follows from the known dependence of relative error of the estimate produced by MC simulation on the probability being estimated. LP and RHH having a non-negligible slope is apparent in all of the plots. An exception is Figure 4.15 (left) where RHH exhibits a constant relative error of 1. This is caused by all of the s-t reach probabilities being very small resulting in RHH always estimating the probability of the source reaching the target to be 0. Doing so will always incur a relative error of 1 when the true probability is non-zero. ARS in some cases exhibits less dependence on the s-t reach probability than RHH and LP as seen in Figure 4.13 (right) and Figure 4.14. However, ARS was unable to handle the very small probabilities present in the case of LiveJournal with edge weights drawn uniformly at random from [0,0.01] as seen by the relative error being consistently 1 (see Figure 4.15 (left)).

As expected, our algorithms, WIR-G and WIR-E, perform best compared to the existing algorithms when the probability of the source reaching the target is low. As such, if one were to know beforehand that the s-t reach probability to be estimated is small, then our algorithms are expected to be very useful. Conversely, if one is only interested in s-t reach probability that are large then existing algorithms are likely sufficient, as they only begin to suffer when the probability to be estimated is small. While the existing algorithm may be more efficient when the probability to be estimated is large, in the absence of knowledge of the probability to be estimated

being large or small, our approach has the advantage of performing much more consistently than existing approaches.

4.8 Discussion and Conclusions

In this chapter, we have presented a technique that we refer to as *edge draw recycling*, which has enabled us to develop algorithms that achieve consistent relative error regardless of the reachability probability being estimated. This is in contrast with the algorithms from existing works, which have their relative error become much worse when the reachability probability to be estimated is small. From our inspection of the distribution of relative errors we find that existing approaches primarily only reduce the relative error of the 'easier' source-target (s-t) pairs (i.e. those that Monte Carlo (MC) simulation is already effective on). Our algorithms overcome this limitation by greatly reducing the relative error of those s-t pairs that are estimated with the least accuracy by MC simulation (as well as by the existing approaches built on MC simulation). In addition to having more consistent relative error our algorithms are found empirically to be substantially more efficient than competing algorithms in terms of the average relative error of the estimates produced when spending the same amount of running time on many of the datasets considered.

We have proven that the estimates produced when using edge draw recycling remain unbiased. As such the only source of error is the variance of the estimator. While we have analyzed the variance that results from edge draw recycling when estimating the reachability probability of a path and presented intuitions as to how this may be extended to the general case, a formal analysis of variance for the general case where there may be multiple paths between a *s-t* pair is currently open. Furthermore, the technique of edge draw recycling may have application beyond the context considered here, as it is not specific to the problem of reachability estimation.

In our experiments, we have found that the version of our algorithm that uses *expectation matching sampling* can have its relative error reduction rate further improved by increasing the number of success indices used. However, doing so comes at the cost of increasing the memory used. Our analysis of the variance of

edge draw recycling on a path suggests that the number of success indices should be at least as high as the length of the paths one needs to estimate the probability of. As such, on a graph a sensible heuristic may be to set the number of success indices according to the diameter of the graph. It is possible that with more careful pruning of unneeded success indices the memory usage of this algorithm could be reduced. This would enable using a larger number of success indices than the rather conservative number we have used in the majority of our experiments.

Chapter 5

Summary and Future Research

5.1 Summary

In this dissertation we have identified and developed new approaches to address key computational inefficiencies (in terms of both running time and memory) that prevent existing approaches from attaining high quality solutions for the important problems of coverage maximization, known as Influence Maximization (IM), and reachability estimation, also known as reliability, on uncertain graphs.

In Chapter 2 we developed a general approach for efficiently performing edge sampling on uncertain graphs by efficiently testing groups of edges together to determine if any edge in the group succeeds. We assemble these group tests into a hierarchy to efficiently identify the edges that succeed. Our approach has time complexity that scales linearly with the number of edges that succeed but only logarithmically with the number of edges that fail. This is in contrast with the standard approach of testing each edge independently which results in a time complexity that scales linearly with the number of edges that fail. We have applied our approach to accelerate the generation of RIS samples used by state-of-the-art approaches to solve the IM problem. Our experiments show that our approach achieves nearly an order of magnitude speedup in RIS sample generation on the larger networks considered.

In Chapter 3 we presented a novel algorithm based on optimization of a pair of fractional objectives that enables us to process the RIS samples, used for solving

the problem of influence maximization, as a stream. This enabled us to completely avoid the need to store them, which is the reason the existing approaches require a prohibitive amount of memory. We established both a lower bound on the coverage of the set of nodes our algorithm returns as well as an instance specific upper bound on the coverage of the optimal solution. This upper bound enables us to provide instance specific guarantees on the quality of the returned solution compared to the optimal solution. In our experiments performed on a variety of real data sets we show that not only does our approach completely eliminate the large memory cost associated with storing the large RIS collection but it does so in a manner that its running time and attained coverage remains competitive with existing state-of-the-art algorithms. In addition, the instance specific guarantees that we are able to provide are found to be superior to that which the best existing approach achieves.

In Chapter 4 we created a novel algorithm that is designed to have the key property that both it's relative error and running time is not impacted by the probability that is being estimated. To do so we developed a technique that we refer to as *random draw recycling* which enables the efficient generation of unbiased samples of the *indices* of the possible worlds in which the source node reaches the target node. We perform experiments on a variety of real datasets and demonstrate the effectiveness of our approach. In particular, we investigate not only the average relative error, as done by prior works, but also the *distribution* of relative errors over a variety of randomly selected source-target (s-t) pairs. This highlights the ability of our approach to not only achieve low average relative error but to achieve consistently low relative error over all s-t pairs, in contrast with existing approaches.

At a high level, Chapters 2 and 3 focus on developing efficient algorithms for influence maximization which is fundamentally a source placement problem. The canonical application being identification of users that would be effective for a viral marketing campaign. Chapter 4, which focuses on developing efficient algorithms for reachability estimation, may be viewed as being useful for addressing the complimentary problem of source identification. For instance if information is observed to have reached a particular node we may wish to estimate how likely it is to have originated from a candidate source. This is the problem that reachability estimation addresses.

5.2 Limitations

An underlying requirement of the techniques that we consider and propose in this thesis is that diffusion occurs independently for each edge in the graph. This setting has received a great deal of focus due to it striking a balance between model expressiveness and computational tractability that is sufficient for modeling a number of practical phenomena but already presents a variety of significant computational challenges, as have been explored. However, the independence assumption may be a poor fit for applications where edges are in fact correlated. For instance, in protein-protein interaction (PPI) networks an interaction between a pair of proteins may not be always independent of other interactions. Some proteins (e.g., enzymes) may act as a catalyst in a PPI network. That is, an interaction happens only when another protein playing the catalyst role interacts with one of the interacting proteins. Such co-interactions cannot be represented simply by independent edges between nodes representing the proteins. Furthermore, in a social network it may not be the case that a user has an independent probability of activating each of their neighbors. This may occur if there is in fact a common cause such as a shared post that is likely to either activate many or no neighbors depending on how well the post was received.

Interestingly, there appears to be the opportunity to model such situations while retaining the edge independence property. The main idea is to introduce pseudo-nodes that may be used to represent a common cause. Such a construction has the capability of representing positive correlation of the edges from a node to its neighbors. As the end result is an extended graph on which the edge correlations have been processed out the techniques considered in this thesis would still apply. However, the number of intermediate nodes required to represent complex correlations may become prohibitive. In addition, such an approach would unlikely be applicable if one were to need to represent correlations among the edges out of multiple nodes (e.g. the success of the out edges of one node increasing the chance that the edges out of other nodes also succeed).

At this point we have not explored how common it is that edge correlations that occur in real applications take on a form that admits them to be represented in this manner. In general, specifying the joint probability of edges grows exponentially.

As such, removing the independence requirement and exploring a more general model must be tied to how the edges' correlations may be compactly represented.

5.3 Future Research

There are a variety of open questions and possible research directions that were encountered that are worth further consideration:

1. While the technique of group edge testing was applied only for the acceleration of RIS sample generation it may be more generally useful for other applications that operate on uncertain graphs. The setting where it is expected to be most effective is when individual edges have only a low probability of succeeding, however the presence of many such edges results in at least some succeeding, and these few successful edges must be identified from the vast majority that do not.

2. The use of fractional relaxations to tackle the IM problem and related problems likely deserves further attention. There are many variants of the IM problem (competitive variants, revenue maximization, welfare maximization, etc) to which the application of fractional relaxation optimization could be considered in the place of the Greedy algorithm. Far too much of the existing literature relies on the Greedy algorithm which despite its desirable qualities of being simple and having a known approximation guarantee also has the undesirable property of sequential selection that prohibits removal of previously selected elements. It was only by breaking away from using the Greedy algorithm were we able to develop our algorithm that can process the a stream of RIS samples.

   Whether the approach of optimizing fractional objectives, as done in Chapter 3, may be extended to these settings depends on the following: 1) Can the diffusion process be approximated using RIS sampling? This is required to establish the connection to max-k cover which the fractional objectives are based on. RIS rely on the objective being representable as a coverage function. In particular, it must be monotone and submodular. 2) Does the constraint admit an efficient solution when the objective is modular? Specif-

ically, our approach involves solving the linear overestimate to the objective function. The solution to which we need to update efficiently as RIS are generated. As we have shown this can be done efficiently for a cardinality constraint. More generally, the same approach would also work for a partition matroid constraint by applying the same optimization technique to each partition. An example application of this is to restrict the number of seed users that may be selected from demographic groups (e.g. divided by gender, age or ethnicity) to enforce a form of fairness.

3. Despite the empirical effectiveness of our algorithm the theoretical time complexity remains open. Unfortunately characterizing it requires determining its convergence rate which appears to be closely related to the convergence rate of Fictitious Play. The Fictitious Play algorithm has been conjectured (over 60 years ago!) to have a convergence rate of $O(t^{-1/2})$. To our knowledge this conjecture remains open with only a much slower rate having been proven. This is despite the observe empirical convergence being in agreement with the conjecture.

4. The edge draw recycling technique that we propose may have applications to other problems that operate on uncertain graphs. In fact, the idea isn't even necessarily limited to uncertain graphs as the general technique can in principle be applied to any logic function composed of conjunction and disjunction on a collection of independent Boolean random variables.

5. There is substantial potential for our algorithm which uses expectation matching sampling to be improved further by more precise pruning of success indices that are not needed. Doing so would enable using a larger number of success indices without it requiring an excessive amount of memory. In our experiments it was found that our algorithm could have a substantially improve rate at which it can reduce its estimate's relative error if more success indices are used. However with our current implementation it results in a large amount of memory being needed.

Bibliography

[1] A. Ageev and M. Sviridenko. Pipage rounding: A new method of constructing algorithms with proven performance guarantee. *J. of Combinatorial Optimization*, 8(3):307–328, Sep 2004. → page 44

[2] A. Arora, S. Galhotra, and S. Ranu. Debunking the myths of influence maximization: An in-depth benchmarking study. In *SIGMOD*, pages 651–666, 2017. → pages 36, 86

[3] A. Badanidiyuru, B. Mirzasoleiman, A. Karbasi, and A. Krause. Streaming submodular maximization: Massive data summarization on the fly. KDD, page 671–680, 2014. → page 42

[4] M. O. Ball. Computational complexity of network reliability analysis: An overview. *IEEE Transactions on Reliability*, 35(3):230–239, 1986. doi:10.1109/TR.1986.4335422. → page 4

[5] M. Bateni, H. Esfandiari, and V. Mirrokni. Almost optimal streaming algorithms for coverage problems. In *SPAA*, pages 13–23, 2017. → page 42

[6] G. Bevilacqua and L. Lakshmanan. A fractional memory-efficient approach for online continuous-time influence maximization. *The VLDB Journal*, 06 2021. doi:10.1007/s00778-021-00679-0. → page v

[7] C. Borgs, M. Brautbar, J. Chayes, and B. Lucier. Maximizing social influence in nearly optimal time. In *SODA*, pages 946–957, 2014. → pages 2, 4, 8, 18, 35, 36, 85

[8] K. V. Bury. *Statistical Models in Applied Science*. J Wiley & Sons, London, 1975. ISBN 0471125903. → page 23

[9] G. Calinescu, C. Chekuri, M. Pál, and J. Vondrák. Maximizing a submodular set function subject to a matroid constraint (extended abstract). In *IPCO*, pages 182–196, 2007. → page 43

[10] A. Chakrabarti and A. Wirth. *Incidence Geometries and the Pass Complexity of Semi-Streaming Set Cover*, pages 1365–1373. 2016. → page 42

[11] L. Chen, H. Hassani, and A. Karbasi. Online continuous submodular maximization. In *AISTATS*, volume 84, pages 1896–1905, 2018. → page 85

[12] W. Chen. An issue in the martingale analysis of the influence maximization algorithm imm. In *Computational Data and Social Networks*, pages 286–297, 2018. → page 68

[13] W. Chen, Y. Wang, and S. Yang. Efficient influence maximization in social networks. In *KDD*, pages 199–208, 2009. → pages 35, 68, 85

[14] W. Chen, C. Wang, and Y. Wang. Scalable influence maximization for prevalent viral marketing in large-scale social networks. In *KDD*, pages 1029–1038, 2010. → page 7

[15] W. Chen, Y. Yuan, and L. Zhang. Scalable influence maximization in social networks under the linear threshold model. In *ICDM*, pages 88–97, 2010. → pages 2, 6, 7, 10, 17, 35, 36, 41, 85

[16] S. Cheng, H. Shen, J. Huang, W. Chen, and X. Cheng. Imrank: Influence maximization via finding self-consistent ranking. SIGIR, page 475–484, 2014. → page 68

[17] E. Cohen, D. Delling, T. Pajor, and R. F. Werneck. Sketch-based influence maximization and computation: Scaling up with guarantees. In *CIKM*, pages 629–638, 2014. → pages 35, 85

[18] P. Dagum, R. Karp, M. Luby, and S. Ross. An optimal algorithm for monte carlo estimation. *SIAM J. on Computing*, 29(5):1484–1496, 2000. → pages 55, 69

[19] E. D. Demaine, P. Indyk, S. Mahabadi, and A. Vakilian. On streaming and communication complexity of the set cover problem. In *Dist. Comput.*, pages 484–498, 2014. → page 42

[20] P. Domingos and M. Richardson. Mining the network value of customers. In *KDD*, pages 57–66, 2001. → pages 2, 35

[21] N. Du, L. Song, M. Yuan, and A. J. Smola. Learning networks of heterogeneous influence. In *NeurIPS*, pages 2780–2788. Curran Associates, Inc., 2012. → pages 3, 4, 35, 41

[22] N. Du, L. Song, M. Gomez Rodriguez, and H. Zha. Scalable influence estimation in continuous-time diffusion networks. In *NeurIPS*, pages 3147–3155. Curran Associates, Inc., 2013. → pages 22, 30, 35, 36, 85

[23] J. C. Duchi, P. L. Bartlett, and M. J. Wainwright. Randomized smoothing for stochastic optimization. *SIAM J. on Optimization*, 22(2):674–701, 2012. → page 85

[24] J. F. Lawless. *Statistical Models and Methods for Lifetime Data*. Wiley-Interscience, 2002. → page 22

[25] M. Farajtabar, N. Du, M. Gomez-Rodriguez, I. Valera, H. Zha, and L. Song. Shaping social activity by incentivizing users. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, page 2474–2482, Cambridge, MA, USA, 2014. MIT Press. → page 6

[26] U. Feige. A threshold of ln n for approximating set cover. *J. ACM*, 45(4): 634–652, July 1998. ISSN 0004-5411. → pages 11, 42

[27] G. S. Fishman. A comparison of four monte carlo methods for estimating the probability of s-t connectedness. *IEEE Transactions on Reliability*, 35 (2):145–155, 1986. doi:10.1109/TR.1986.4335388. → page 7

[28] M. Frank and P. Wolfe. An algorithm for quadratic programming. *Naval Research Logistics Quarterly*, 3:95–110, 1956. → page 85

[29] S. Galhotra, A. Arora, and S. Roy. Holistic influence maximization: Combining scalability and efficiency with opinion-aware models. SIGMOD, page 743–758, 2016. → page 68

[30] J. Ghosh, H. Q. Ngo, S. Yoon, and C. Qiao. On a routing problem within probabilistic graphs and its application to intermittently connected networks. In *IEEE INFOCOM 2007 - 26th IEEE International Conference on Computer Communications*, pages 1721–1729, 2007. doi:10.1109/INFCOM.2007.201. → pages 1, 89

[31] D. L. Gibbs and I. Shmulevich. Solving the influence maximization problem reveals regulatory organization of the yeast cell cycle. *PLOS Computational Biology*, 2017. → pages 2, 35

[32] M. X. Goemans and D. P. Williamson. New 3/4-approximation algorithms for max sat. *SIAM Journal on Discrete Mathematics*, 7:313–321, 1994. → page 44

[33] J. Goldenberg, B. Libai, and Muller. Using complex systems analysis to advance marketing theory development. *Academy of Marketing Science Review*, 2001. → pages 4, 35

[34] J. Goldenberg, B. Libai, and E. Muller. Talk of the network: A complex systems look at the underlying process of word-of-mouth. *Marketing Letters*, 12(3):211–223, 2001. → pages 4, 35

[35] M. Gomez-Rodriguez, D. Balduzzi, and B. Schölkopf. Uncovering the temporal dynamics of diffusion networks. In *ICML*, pages 561–568, 2011. → pages 3, 4, 5, 6, 30, 35, 41, 68

[36] M. Gomez-Rodriguez, J. Leskovec, and B. Schölkopf. Modeling information propagation with survival theory. In *ICML*, pages III–666–III–674, 2013.

[37] M. Gomez Rodriguez, J. Leskovec, and B. Schölkopf. Structure and dynamics of information pathways in online media. In *WSDM*, pages 23–32, 2013. → pages 3, 4, 22, 35, 41

[38] A. Goyal, W. Lu, and L. V. S. Lakshmanan. Simpath: An efficient algorithm for influence maximization under the linear threshold model. In *ICDM*, pages 211–220, 2011. → pages 35, 68, 85

[39] M. Granovetter. Threshold models of collective behavior. *American Journal of Sociology*, 83(6):1420–1443, 1978. → page 35

[40] Q. Guo, S. Wang, Z. Wei, and M. Chen. Influence maximization revisited: Efficient reverse reachable set generation with bound tightened. In *SIGMOD*, page 2167–2181, 2020. → page 85

[41] Q. Guo, S. Wang, Z. Wei, and M. Chen. Influence maximization revisited: Efficient reverse reachable set generation with bound tightened. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, SIGMOD '20, page 2167–2181, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450367356. doi:10.1145/3318464.3389740. URL https://doi.org/10.1145/3318464.3389740. → pages 13, 20

[42] S. Har-Peled, P. Indyk, S. Mahabadi, and A. Vakilian. Towards tight bounds for the streaming set cover problem. PODS, page 371–383, 2016. → page 42

[43] D. Hochbaum. *Approximation Algorithms for NP-Hard Problems*. PWS Publishing Company, 1997. → page 11

[44] M. Hua and J. Pei. Probabilistic path queries in road networks: Traffic uncertainty aware path selection. EDBT '10, page 347–358, New York, NY, USA, 2010. Association for Computing Machinery. ISBN 9781605589459. doi:10.1145/1739041.1739084. URL https://doi.org/10.1145/1739041.1739084. → page 1

[45] K. Huang, S. Wang, G. S. Bevilacqua, X. Xiao, and L. V. S. Lakshmanan. Revisiting the stop-and-stare algorithms for influence maximization. *PVLDB*, 10(9):913–924, 2017. → pages 2, 10, 14, 18, 35, 36, 85

[46] D. Ienco, F. Bonchi, and C. Castillo. The meme ranking problem: Maximizing microblogging virality. In *ICDMW*, pages 328–335, 2010. → pages 2, 35

[47] M. Jaggi. Revisiting Frank-Wolfe: Projection-free sparse convex optimization. In *ICML*, volume 28, pages 427–435, 2013. → page 85

[48] R. Jin, L. Liu, B. Ding, and H. Wang. Distance-constraint reachability computation in uncertain graphs. *Proc. VLDB Endow.*, 4(9):551–562, jun 2011. ISSN 2150-8097. doi:10.14778/2002938.2002941. URL https://doi.org/10.14778/2002938.2002941. → pages 1, 4, 6, 15, 89, 90, 92, 93, 122

[49] K. Jung, W. Heo, and W. Chen. IRIE: scalable and robust influence maximization in social networks. In *ICDM*, pages 918–923, 2012. → pages 35, 68, 85

[50] S. K. Thompson. *Sampling, Second Edition*. Wiley, New York, 2002. → page 92

[51] L. Kahn. Ratio squarer. *Proc. IRE (Corresp.).*, 42(11):1704, November 1954. doi:10.1109/JRPROC.1954.274666. → page 124

[52] M. Karimi, M. Lucic, H. Hassani, and A. Krause. Stochastic submodular maximization: The case of coverage functions. In *NeurIPS*, pages 6853–6863. Curran Associates, Inc., 2017. → page 85

[53] S. Karlin. *Mathematical Methods and Theory in Games, Programming, and Economics*. Addison-Wesley, Reading, Mass., 1959. → page 61

[54] X. Ke, A. Khan, and L. L. H. Quan. An in-depth comparison of s-t reliability algorithms over uncertain graphs. *Proc. VLDB Endow.*, 12(8):864–876, apr 2019. ISSN 2150-8097. doi:10.14778/3324301.3324304. URL

https://doi.org/10.14778/3324301.3324304. → pages 4, 6, 15, 93, 120, 121, 122, 123, 124, 128

[55] D. Kempe, J. Kleinberg, and E. Tardos. Maximizing the spread of influence through a social network. In *KDD*, pages 137–146, 2003. → pages 2, 3, 4, 7, 8, 9, 11, 12, 17, 18, 35, 36, 40

[56] A. Khan, F. Bonchi, F. Gullo, and A. Nufer. Conditional reliability in uncertain graphs. *IEEE Transactions on Knowledge and Data Engineering*, 30(11):2078–2092, 2018. doi:10.1109/TKDE.2018.2816653. → pages 1, 89

[57] H. Kumamoto, K. Tanaka, K. Inoue, and E. J. Henley. Dagger-sampling monte carlo for system unavailability evaluation. *IEEE Transactions on Reliability*, R-29(2):122–125, 1980. doi:10.1109/TR.1980.5220749. → page 7

[58] H. Kwak, C. Lee, H. Park, and S. Moon. What is twitter, a social network or a news media? http://an.kaist.ac.kr/traces/WWW2010.html, 2010. → page 29

[59] D. Lee, K. Hosanagar, and H. Nair. Advertising content and consumer engagement on social media: Evidence from facebook. *Management Science*, 64, 01 2018. → page 37

[60] J. Leskovec and A. Krevl. SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data, June 2014. → pages 29, 121

[61] J. Leskovec, A. Krause, C. Guestrin, C. Faloutsos, C. Faloutsos, J. VanBriesen, and N. Glance. Cost-effective outbreak detection in networks. In *KDD*, pages 420–429, 2007. → pages 2, 8, 10, 35, 37, 84, 86, 87

[62] R.-H. Li, J. X. Yu, R. Mao, and T. Jin. Recursive stratified sampling: A new framework for query evaluation on uncertain graphs. *IEEE Transactions on Knowledge and Data Engineering*, 28(2):468–482, 2016. doi:10.1109/TKDE.2015.2485212. → pages 15, 90, 92, 122

[63] X. Li, J. D. Smith, T. N. Dinh, and M. T. Thai. Why approximate when you can get the exact? optimal targeted viral marketing at scale. In *INFOCOM*, pages 1–9, May 2017. → pages 35, 42, 87

[64] Y. Li, J. Fan, D. Zhang, and K.-L. Tan. Discovering your selling points: Personalized social influential tags exploration. In *SIGMOD*, page 619–634, 2017. → page 85

[65] Y. Li, J. Fan, D. Zhang, and K.-L. Tan. Discovering your selling points: Personalized social influential tags exploration. In *Proceedings of the 2017 ACM International Conference on Management of Data*, SIGMOD '17, page 619–634, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450341974. doi:10.1145/3035918.3035952. URL https://doi.org/10.1145/3035918.3035952. → pages 13, 15, 20, 90, 93, 122

[66] C. McDiarmid. Concentration. In M. Habib, C. McDiarmid, J. Ramirez-Alfonsin, and B. Reed, editors, *Probabilistic Methods for Algorithmic Discrete Mathematics*. Springer, 1998. → page 49

[67] A. Mokhtari, H. Hassani, and A. Karbasi. Conditional gradient method for stochastic submodular maximization: Closing the gap. In *AISTATS*, volume 84, pages 1886–1895, 2018. → page 85

[68] S. Muthukrishnan. Data streams: Algorithms and applications. *Found. Trends Theor. Comput. Sci.*, 1(2):117–236, Aug. 2005. → page 37

[69] R. M. Neal. Annealed importance sampling. *Statistics and Computing*, 11 (2):1573–1375, 2001. doi:10.1023/A:1008923215028. → page 122

[70] H. Nguyen, T. Nguyen, N. H. Phan, and T. Dinh. Importance sketching of influence dynamics in billion-scale networks. In *ICDM*, pages 337–346, 11 2017. → pages 38, 86

[71] H. T. Nguyen, M. T. Thai, and T. N. Dinh. Stop-and-stare: Optimal sampling algorithms for viral marketing in billion-scale networks. In *SIGMOD*, pages 695–710, 2016. → pages 2, 4, 8, 10, 14, 18, 35, 36, 37, 41, 42, 55, 68, 77, 85

[72] N. Ohsaka. The solution distribution of influence maximization: A high-level experimental study on three algorithmic approaches. In *SIGMOD*, page 2151–2166, 2020. → pages 69, 85, 86

[73] N. Ohsaka, T. Akiba, Y. Yoshida, and K.-I. Kawarabayashi. Fast and accurate influence maximization on large networks with pruned monte-carlo simulations. In *AAAI*, page 138–144, 2014. → pages 8, 35, 85

[74] N. Ohsaka, T. Sonobe, S. Fujita, and K.-i. Kawarabayashi. Coarsening massive influence networks for scalable diffusion analysis. In *SIGMOD*, pages 635–650, 2017. → pages 36, 38, 86

[75] D. Popova, N. Ohsaka, K.-i. Kawarabayashi, and A. Thomo. Nosingles: A space-efficient algorithm for influence maximization. In *SSDBM*, pages 18:1–18:12, 2018. → pages 38, 86

[76] M. Potamias, F. Bonchi, A. Gionis, and G. Kollios. K-nearest neighbors in uncertain graphs. *Proc. VLDB Endow.*, 3(1–2):997–1008, sep 2010. ISSN 2150-8097. doi:10.14778/1920841.1920967. URL https://doi.org/10.14778/1920841.1920967. → page 7

[77] M. Richardson and P. Domingos. Mining knowledge-sharing sites for viral marketing. In *KDD*, pages 61–70, 2002. → pages 2, 35

[78] M. G. Rodriguez and B. Scholkopf. Influence maximization in continuous time diffusion networks. In *ICML 2012*, 2012. → pages 5, 6

[79] B. Saha and L. Getoor. On Maximum Coverage in the Streaming Model & Application to Multi-topic Blog-Watch. *SDM*, pages 697–708, 01 2009. → page 42

[80] H. N. Shapiro. Note on a computation method in the theory of games. *Communications on Pure and Applied Mathematics*, 11(4):587–593, 1958. → page 61

[81] D. Shewan. The comprehensive guide to online advertising costs. https://www.wordstream.com/blog/ws/2017/07/05/online-advertising-costs, 2020. → page 37

[82] X. Song, Y. Chi, K. Hino, and B. L. Tseng. Information flow modeling based on diffusion rate for prediction and ranking. In *WWW*, pages 191–200, 2007. → pages 2, 35

[83] J. Tang, X. Tang, X. Xiao, and J. Yuan. Online processing algorithms for influence maximization. In *SIGMOD*, pages 991–1005, 2018. → pages 2, 4, 10, 14, 18, 35, 36, 37, 41, 42, 68, 77, 86, 87

[84] Y. Tang, X. Xiao, and Y. Shi. Influence maximization: Near-optimal time complexity meets practical efficiency. In *SIGMOD*, pages 75–86, 2014. → pages 2, 7, 8, 9, 18, 36, 41, 56, 85

[85] Y. Tang, Y. Shi, and X. Xiao. Influence maximization in near-linear time: A martingale approach. In *SIGMOD*, pages 1539–1554, 2015. → pages 2, 4, 9, 10, 14, 18, 22, 30, 35, 36, 37, 41, 42, 48, 55, 68, 78, 85

[86] L. G. Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, 8(3):410–421, 1979. doi:10.1137/0208032. → pages 1, 3, 4, 6, 17, 90, 123

[87] C. Wang, W. Chen, and Y. Wang. Scalable influence maximization for independent cascade model in large-scale social networks. *Data Mining and Knowledge Discovery*, 25(3):545–576, 2012. → pages 2, 4, 6, 7, 10, 17, 35, 36, 41, 85

[88] K. Zhang, S. Bhattacharyya, and S. Ram. Large-scale network analysis for online social brand advertising. *MIS Quarterly*, 40:849–868, 04 2016. → page 37

[89] P. P. Zubcsek and M. Sarvary. Advertising to a social network. *Quantitative Marketing and Economics*, 9:71–107, 2011. → page 37

Appendix A

Supporting Materials

A.1 Additional Proofs

Proof of Lemma 6. Starting from $\Pr[X \geq (1+\varepsilon)\chi] < \exp\left(-\frac{\varepsilon^2\chi}{2(1+\varepsilon/3)}\right)$, where $\chi :- \mu t$, a concentration bound violation probability of at most δ is needed (i.e. $\delta = \exp(-\frac{\varepsilon^2\chi}{2(1+\varepsilon/3)})$). Rearranging into quadratic form in ε,

$$\begin{aligned} 2(1+\varepsilon/3)\log(\delta) &= -\varepsilon^2\chi \\ \chi\varepsilon^2 + 2/3\log(\delta)\varepsilon + 2\log(\delta) &= 0 \end{aligned}$$

By solving the quadratic it can be determined that this is ensured if,

$$\varepsilon = \left(-2/3\log(\delta) + \sqrt{4/9\log(\delta)^2 - 8\chi\log(\delta)}\right)/(2\chi) \tag{A.1}$$

$$= \left(-\log(\delta) + \sqrt{-\log(\delta)(18\chi - \log(\delta))}\right)/(3\chi) \tag{A.2}$$

Hence,

$$\Pr[X \geq \chi + \left(-\log(\delta) + \sqrt{-\log(\delta)(18\chi - \log(\delta))}\right)/3] \leq \delta \tag{A.3}$$

From the inner constraint we have,

$$X \geq \chi + \left(-\log(\delta) + \sqrt{-\log(\delta)(18\chi - \log(\delta))}\right)/3 \tag{A.4}$$

$$3X + \log(\delta) - 3\chi \geq \sqrt{-\log(\delta)(18\chi - \log(\delta))} \tag{A.5}$$

Squaring both sides and rearranging.

$$0 \leq \log(\delta)(18\chi - \log(\delta)) + (3X + \log(\delta) - 3\chi)^2 \tag{A.6}$$

$$0 \leq 3\chi^2 + (4\log(\delta) - 6X)\chi + 2X\log(\delta) + 3X^2 \tag{A.7}$$

Solving for χ via the quadratic formula and simplifying.

$$\chi \leq X - 2/3\log(\delta) - \sqrt{-2/9\log(\delta)(9X - 2\log(\delta))} \tag{A.8}$$

As such we have,

$$\Pr[\chi \leq X - \tfrac{2}{3}\log(\delta) - \sqrt{-\tfrac{2}{9}\log(\delta)(9X - 2\log(\delta))}] \leq \delta \tag{A.9}$$

$$\Pr[\chi > X + \tfrac{2}{3}\log(\tfrac{1}{\delta}) - \sqrt{\tfrac{2}{9}\log(\tfrac{1}{\delta})(9X + 2\log(\tfrac{1}{\delta}))}] \geq 1 - \delta \tag{A.10}$$

Let $\mathrm{LB}(X, \delta) \coloneqq X + \frac{2}{3}\log(\frac{1}{\delta}) - \sqrt{\frac{2}{9}\log(\frac{1}{\delta})(9X + 2\log(\frac{1}{\delta})}$ then,
$\Pr[\mu > \mathrm{LB}(X, \delta)/t] \geq 1 - \delta$ □

Proof of Lemma 7. Starting from $\Pr[X \leq (1-\varepsilon)\chi] \leq \exp\left(-\frac{1}{2}\varepsilon^2\chi\right)$, where $\chi \coloneqq \mu t$, a concentration bound violation probability of at most δ is needed (i.e. $\delta = \exp(-\frac{1}{2}\varepsilon^2\chi)$). This is ensured if $\varepsilon = \sqrt{-2\log(\delta)/\chi}$. Hence,

$$\Pr[X \leq (1 - \sqrt{-2\log(\delta)/\chi})\chi] \leq \delta \tag{A.11}$$

From the inner constraint we have, $0 \leq \chi - \sqrt{-2\log(\delta)\chi} - X$
Solving for $\sqrt{\chi}$ via the quadratic formula gives,
$\sqrt{\chi} \geq \left(\sqrt{-2\log(\delta)} + \sqrt{4X - 2\log(\delta)}\right)/2$.

As such we have,

$$\Pr[\chi \geq \left(\sqrt{-2\log(\delta)} + \sqrt{4X - 2\log(\delta)}\right)^2/4] \leq \delta \tag{A.12}$$

$$\Pr[\chi < \left(\sqrt{-2\log(\delta)} + \sqrt{4X - 2\log(\delta)}\right)^2/4] \geq 1 - \delta \tag{A.13}$$

$$\Pr[\chi < X - \log(\delta) + \sqrt{-\log(\delta)(2X - \log(\delta))}] \geq 1 - \delta \tag{A.14}$$

Let $\mathrm{UB}(X,\delta) \coloneqq X + \log(\frac{1}{\delta}) + \sqrt{\log(\frac{1}{\delta})(2X + \log(\frac{1}{\delta}))}$ then,
$\Pr[\mu < \mathrm{UB}(X,\delta)/t] \geq 1 - \delta$ □