An In-Depth Comparison of s-t Reliability Algorithms over Uncertain Graphs

Xiangyu Ke
NTU Singapore
xiangyu001@e.ntu.edu.sg

Arijit Khan
NTU Singapore
arijit.khan@ntu.edu.sg

Leroy Lim Hong Quan
NTU Singapore
llim031@e.ntu.edu.sg

ABSTRACT

Uncertain, or probabilistic, graphs have been increasingly used to represent noisy linked data in many emerging applications, and have recently attracted the attention of the database research community. A fundamental problem on uncertain graphs is the s-t reliability, which measures the probability that a target node t is reachable from a source node s in a probabilistic (or uncertain) graph, i.e., a graph where every edge is assigned a probability of existence.

Due to the inherent complexity of the s-t reliability estimation problem ($\#\mathbf{P}$-hard), various sampling and indexing based efficient algorithms were proposed in the literature. However, since they have not been thoroughly compared with each other, it is not clear whether the later algorithm outperforms the earlier ones. More importantly, the comparison framework, datasets, and metrics were often not consistent (e.g., different convergence criteria were employed to find the optimal number of samples) across these works. We address this serious concern by re-implementing six state-of-the-art s-t reliability estimation methods in a common system and code base, using several medium and large-scale, real-world graph datasets, identical evaluation metrics, and query workloads.

Through our systematic and in-depth analysis of experimental results, we report surprising findings, such as many follow-up algorithms can actually be several orders of magnitude inefficient, less accurate, and more memory intensive compared to the ones that were proposed earlier. We conclude by discussing our recommendations on the road ahead.

PVLDB Reference Format:
X. Ke, A. Khan, and L. Lim. An In-Depth Comparison of s-t Reliability Algorithms over Uncertain Graphs. *PVLDB*, 12(8): 864-876, 2019.
DOI: https://doi.org/10.14778/3324301.3324304

1. INTRODUCTION

Uncertain graphs, i.e., graphs whose edges are assigned a probability of existence, have attracted a great deal of attention [23, 24], due to their rich expressiveness and given that uncertainty is inherent in the data in a wide range of applications, including noisy measurements [2], inference and prediction models [1], and explicit manipulation, e.g., for privacy purposes [5]. A fundamental problem in uncertain graphs is the so-called *reliability*, which asks to measure the probability that two given nodes are reachable [3]. Reliability has been well-studied in the context of device networks, i.e., networks whose nodes are electronic devices and the (physical) links between such devices have a probability of failure [3]. Recently, the attention has been shifted to other types of networks that can be represented as uncertain graphs, such as social and biological networks [18, 31]. Specific problem formulations in this class ask to measure the probability that a certain reliability event occurs, e.g., what is the probability that two given nodes are connected (*two-terminal reliability* [3]), all nodes in the network are pairwise connected (*all-terminal reliability* [33]), or all nodes in a given subset are pairwise connected (k*-terminal reliability* [16]).

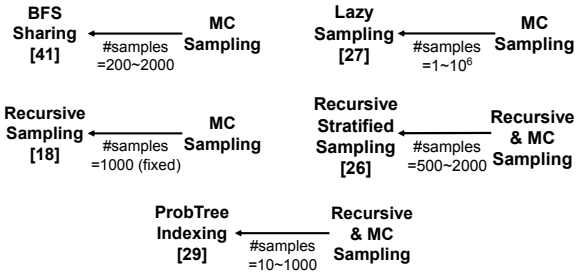


Figure 1: State-of-the-art reliability estimation algorithms in uncertain graphs: A directed arrow depicts reported superiority in prior works. All algorithms have not been thoroughly compared with each other. Moreover, previous works did not employ identical frameworks, datasets, and metrics for comparison. Thus, it is critical to investigate their trade-offs and superiority over each other.

In this work, we shall investigate two-terminal reliability: The probability that a target node t is reachable from a source node s in an uncertain graph, also denoted as the s-t reliability. This s-t reliability estimation has been used in many applications such as measuring the quality of connections between two terminals in a sensor network [15], finding other proteins that are highly probable to be connected with a specific protein in a protein-protein interaction (PPI) network [18], identifying highly reliable peers containing some file to transfer in a peer-to-peer (P2P) network, probabilistic path queries in a road network [17], and evaluating information diffusions in a social influence network [22].

Due to the inherent complexity of the problem ($\#\mathbf{P}$-hard) [4], although the exact reliability detection has received attention in the past [3], the focus nowadays has mainly been on approximate and heuristic solutions over large-scale graphs [23]. The large spectrum of the reliability problem is categorized in Figure 2. In this paper, we shall focus on *sequential algorithms for the fundamental s-t reliability query*. Notice that we would not consider distributed algorithms [9, 42], other simplified versions of the s-t reliability problem [8, 25, 30, 34], neither the reduction of uncertainty

Proceedings of the VLDB Endowment, Vol. 12, No. 8
ISSN 2150-8097.
DOI: https://doi.org/10.14778/3324301.3324304

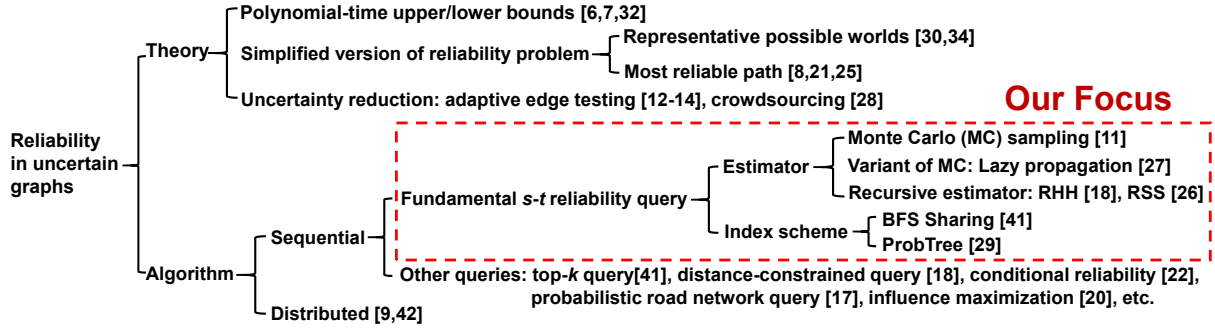


Figure 2: The broad spectrum of reliability problem over uncertain graphs

of a graph (e.g., by crowdsourcing) before s-t reliability estimation [12–14, 28] [1]. *If a method was designed for a specific kind of reliability query (e.g. distance-constrained [18]), it can be generalized to the fundamental s-t reliability query, thus we include the algorithm in our study [18, 41].* In particular, various sampling and indexing-based efficient algorithms were proposed in the literature. Estimation of reliability in uncertain graphs has its beginnings with the usage of Monte Carlo (MC) sampling [11]. Subsequently, more advanced sampling methods were proposed in the form of recursive samplings [18, 26] and shared possible worlds [41], as well as other indexing methods [29]. *With the wide range of algorithms available for estimating the s-t reliability over uncertain graphs, there is an urgent need to realize their trade-offs, and to employ the best algorithm for a given scenario.*

As depicted in Figure 1, we find serious concerns in the existing experimental comparisons of state-of-the-art reliability estimation algorithms over uncertain graphs. **(1)** There is no prior work that compared all state-of-the-art methods with each other. It is, therefore, difficult to draw a general conclusion on the superiority and trade-offs of different methods. **(2)** As shown in Figure 1, except for [27], the experimental studies in [18, 26, 29, 41] either considered a fixed number of samples (e.g., 1 000), or the maximum number of samples was limited by 2 000. However, we observe in our experiments that the number of samples necessary for the convergence of reliability estimation varies a lot depending on the specific algorithm used (e.g., for *Recursive Stratified Sampling* [26], #samples required for convergence is 250∼1 000, while for *Lazy Propagation* [27], it is 500∼1 500), and also on the underlying characteristics of the uncertain graph dataset. Therefore, the running time necessary to achieve convergence (and hence, good-quality results) should be reported differently, as opposed to using the same number of samples in all experiments. **(3)** The metrics used for empirically comparing these techniques were not consistent in the past literature, thereby making them *apple-to-orange* comparisons in the larger context. For example, [26] measured relative variance of different estimators and their running times for the same number of samples (2 000). In both [29, 41], the authors reported accuracy (with respect to baseline MC sampling) and running times of different algorithms using a maximum of 1 000 samples. On the other hand, [18] compared relative variance, accuracy, and running times of various estimators by considering a fixed (1 000) number of samples. In addition, surprisingly none of these studies reported the online memory usage which, according to our experiments, varied a great extent. **(4)** Last but not least, we find certain errors and further optimization scopes in past algorithms (e.g., accuracy of [27], time complexity analysis of [41]), and by correcting (or, updating) them we significantly improve their performance.

Our contribution and roadmap. Our contributions can be summarized as follows.

- We investigate the s-t reliability estimation problem and summarize six state-of-the-art sequential algorithms in Section 2, together with their time complexity and sampling variance.
- We correct certain issues in past algorithms (e.g., accuracy of [27], time complexity analysis of [41]), which significantly improve their performance (Section 2).
- We implemented five state-of-the-art algorithms [11, 18, 26, 27, 41] in C++, and obtained C++ source code of [29] from respective authors. We compare them in a common environment, using same convergence criteria, and present empirical comparisons of six s-t reliability estimation algorithms over six real-world, uncertain graph datasets in Section 3. *Our datasets and source code are available at:* `https://github.com/5555lan/RelComp`
- We report the accuracy, efficiency, and memory usage of six referred methods both at convergence and at #samples= 1 000, summarize their trade-offs, and provide guidelines for researchers and practitioners (Sections 3 and 4).

2. RELIABILITY ESTIMATION METHODS

2.1 s-t Reliability in Uncertain Graphs

An uncertain graph $\mathcal{G}$ is a triple (V, E, P), where V is a set of n nodes, $E \subseteq V \times V$ is a set of m directed edges, and $P : E \to (0, 1]$ is a probability function that assigns a probability of existence to each edge in E.

The bulk of the literature on uncertain graphs and device networks reliability assumes the existence of the edges in the graph independent from one another, and interprets uncertain graphs according to the well-known *possible-world semantics* [6, 7, 16, 18, 31–33, 40]: an uncertain graph $\mathcal{G}$ with m edges yields 2^m possible deterministic graphs, which are derived by sampling independently each edge $e \in E$ with probability $P(e)$. More precisely, a possible graph $G \sqsubseteq \mathcal{G}$ is a pair (V, E_G), where $E_G \subseteq E$, and its sampling probability is:

$$Pr(G) = \prod_{e \in E_G} P(e) \prod_{e \in E \setminus E_G} (1 - P(e)) \qquad (1)$$

For a possible deterministic graph G, we define an indicator function $I_G(s, t)$ to be 1 if there is a path in G from a source node $s \in V$ to a target node $t \in V$, and 0 otherwise. The probability that t is reachable from s in the uncertain graph $\mathcal{G}$, denoted by $R(s, t)$, is computed as:

$$R(s, t) = \sum_{G \sqsubseteq \mathcal{G}} I_G(s, t)\, Pr(G) \qquad (2)$$

The number of possible worlds $G \sqsubseteq \mathcal{G}$ is exponential in the number of edges, which makes the exact computation of $R(s, t)$ infeasible even for modestly-sized networks. In fact, the s-t reliability computation is a prototypical $\#\mathbf{P}$-complete problem [4, 36].

[1] For details about other related work, we refer to our extended version [19].

Due to intrinsic hardness, we tackle the reliability estimation problem from approximation and heuristic viewpoints. In particular, we shall examine six sampling and indexing-based efficient algorithms that were proposed in recent literature as follows: **(1)** Monte Carlo (MC) sampling [11]; **(2)** Indexing via BFS sharing [41]; **(3)** Recursive sampling [18]; **(4)** Recursive stratified sampling [26]; **(5)** Lazy propagation sampling [27]; and **(6)** Indexing via probabilistic trees [29]. [2]

2.2 MC Sampling

In the basic Monte Carlo (MC) sampling, we first sample K possible worlds $G_1, G_2, \ldots, G_K$ of the uncertain graph $\mathcal{G}$ according to independent edge probabilities. We then compute the reachability in each sampled graph G_i, and define $I_{G_i}(s,t) = 1$ if t is reachable from s in G_i, and 0 otherwise. Given this, we have the MC sampling estimator:

$$R(s,t) \approx \hat{R}(s,t) = \frac{1}{K}\sum_{i=1}^{K} I_{G_i}(s,t) \tag{3}$$

This is also known as the hit-and-miss Monte Carlo. The basic sampling estimator $\hat{R}(s,t)$ is an *unbiased estimator* of the s-t reliability, i.e., $E(\hat{R}(s,t)) = R(s,t)$, and its variance can be determined due to Binomial distribution $\sim B(K, R(s,t))$ [11,18].

$$\begin{aligned} Var\left(\hat{R}(s,t)\right) &= \frac{1}{K}\cdot R(s,t)\cdot(1-R(s,t)) \\ &\approx \frac{1}{K}\cdot \hat{R}(s,t)\cdot\left(1-\hat{R}(s,t)\right) \end{aligned} \tag{4}$$

It is possible to derive bounds on the number of MC samples needed to provide a good estimate for the s-t reliability problem. It was shown in [31] by applying the Chernoff bound that with number of samples $K \geq \frac{3}{\epsilon^2 R(s,t)} \ln\left(\frac{2}{\lambda}\right)$, we can ensure the following.

$$Pr\left(\left|\hat{R}(s,t) - R(s,t)\right| \geq \epsilon R(s,t)\right) \leq \lambda \tag{5}$$

The time complexity to generate K possible worlds is $\mathcal{O}(mK)$. In each possible world, the reachability can be determined by performing a breadth-first search (BFS) from the source node. Each BFS requires $\mathcal{O}(m+n)$ time. Therefore, the overall time complexity of MC sampling based reliability estimation is $\mathcal{O}(K(m+n))$. In essence, one may combine MC sampling with BFS from the source node for improved efficiency [18,21]. It means that an edge in the current possible world is sampled only upon request. This avoids sampling of many edges in parts of the graph that are not reached with the current BFS, thus increasing the chance of an early termination.

In practise, MC sampling can be inefficient over large-scale networks due to two reasons.

- For each s-t reliability query, we need to generate K possible worlds via sampling. Based on empirical evidences from state-of-the-art works [18,20,31] as well as according to our own experimental results, K can be in the order of thousands to achieve a reasonable accuracy. However, as correctly pointed out in [30,41], this sampling procedure does not contribute to the reliability estimation process directly. For example, one can pre-compute these K possible worlds in an offline manner to further improve the efficiency of online s-t reliability estimation.
- There could be a significant overlap in structures of different possible worlds [18, 40, 41]. Unfortunately, the reliability estimation via basic MC sampling performs a separate BFS over each possible world, therefore it cannot take advantage of the common substructure across various possible worlds.

[2] Due to lack of space, we provide pseudocode for all algorithms (except Lazy propagation sampling [27]) in our extended version [19]. For the lazy propagation, we present its pseudocode in this paper, as it is essential to fix an error therein.

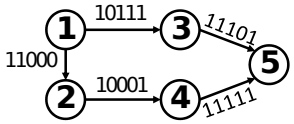


Figure 3: Compact structure with five possible worlds: BFS sharing

2.3 Indexing via BFS Sharing

Zhu et al. [41] developed an *offline sampling* method to generate K possible worlds: $G_1, G_2, \ldots, G_K$. In order to minimize the storage overhead, they proposed a bit-vector based compact structure, as depicted in Figure 3. It essentially stores only one graph $G = (V, E)$ with the same set of nodes and edges as the input uncertain graph $\mathcal{G}$. However, each edge e in G has a bit-vector of size K — its i-th bit represents whether the edge e is present in the sampled graph G_i or not.

Given an s-t reliability query, [41] performs BFS over this compact graph structure, which is equivalent to doing BFS traversals in parallel across the pre-computed possible worlds. We attach an additional bit vector I_v with each node v that keeps track of the possible worlds in which v is reachable from s. Initially, $I_s = [1\,1\,\ldots\,1]$ and $I_v = [0\,0\,\ldots\,0]$ for all $v \neq s$. Let us also denote by U the set of visited nodes based on BFS. Initially $U = \{s\}$. At each step, when we find an unexplored node v that is an out-neighbor of at least one node u in U, we insert v into U. We update the bit vector I_v to include the possible worlds where all such v's are reachable from s. Before proceeding to the next step of BFS, one may note that if v has some out-neighbor w that is already in U, we may need to update I_w. Specifically, let G_i be a possible world that is currently in I_v, but not in I_w. Then, we should also include G_i in I_w. Such changes in I_w may in turn affect I_z, where z is an out-neighbor of w, and z is also in U. In general, we proceed to the next step of BFS only after finishing these cascading updates. Finally, the number of 1's in I_t, divided by K, provides the MC estimation of s-t reliability.

Clearly, BFS sharing has the same variance as the basic MC. However, by generating the K possible worlds in an offline manner, [41] reduces the online s-t reliability estimation time. On the other hand, *unlike the MC sampling approach, in this case no early termination of BFS is possible*, even when the target node t is reached earlier. This is due to performing the required cascading updates in this method. *It often makes BFS Sharing even more time consuming than MC, which is evident in our experimental results*.

Note that the original algorithm in [41] was developed to identify the top-k target nodes having the maximum reliability from a given source node. However, as we discussed above, one can trivially update their technique to estimate the reliability of a given s-t pair.

Our correction in complexity analysis. The offline index building time complexity is $\mathcal{O}(Km)$, and index storage space is $\mathcal{O}(n + Km)$, where $\mathcal{O}(Km)$ is the storage due to edge bit vectors. We load all edge bit vectors in memory during online query processing to improve efficiency. Besides, n node bit vectors of size $\mathcal{O}(Kn)$ are created during online query processing. BFS sharing method has online reliability estimation time complexity $\mathcal{O}(K(m+n))$: Due to cascading updates, each node and edge can be visited at most K times. Notice that in the original paper [41], it was stated that the online query time is independent of K. As we reasoned, however, the running time increases linearly with K due to cascading updates. This is also confirmed by our empirical results (e.g., notice in Tables 8, 9, 10, the running times of BFS sharing increases for larger K over the same dataset).

2.4 Recursive Sampling

Recursive sampling, which was proposed in [18], improves on MC sampling by considering the two following factors.

- When some edges are missing in a possible world, the presence of other edges might no longer be relevant with respect to certain s-t reliability queries. Hence, those edges can be skipped from sampling and query evaluation process.
- Many possible worlds share a significant portion of existing or missing edges. Hence, the reachability checking cost could be shared among them.

The basic approach, which follows a divide-and-conquer technique, is given below. A very similar algorithm, called the Dynamic MC sampling, was developed in [40].

We start from the source node s, and say that s is already *reached*. An edge e is *expandable* if it starts from a *reached* node. We randomly pick an extendable edge e, then sample the existence of e for K iterations. The next step is to divide the samples into two groups: one group with e existing and another group with e not existing. In the first group, we may reach a new node w via e, and in that case, more edges become *expandable*. For both groups, we repeat the process of picking a random *expandable* edge, sampling its existence, and dividing the group into smaller batches.

Formally, assume that $E_1 \subseteq E$ be the set of included edges and $E_2 \subseteq E$ be the set of not-included edges in one group (referred to as a *prefix group* in [18]) at some intermediate stage of our method. Let us denote this group by $\mathcal{G}(E_1, E_2)$, i.e., the set of possible worlds of $\mathcal{G} = (V, E, p)$ which contains all edges in E_1, and no edges in E_2. Clearly, $E_1 \cup E_2 \subseteq E$ and $E_1 \cap E_2 = \phi$. The *generating probability* of the group $\mathcal{G}(E_1, E_2)$ can be defined as:

$$Pr\left(\mathcal{G}(E_1, E_2)\right) = \prod_{e \in E_1} P(e) \prod_{e' \in E_2} (1 - P(e')) \tag{6}$$

The s-t reliability of a group $\mathcal{G}(E_1, E_2)$ is defined as the probability that t is reachable from s conditioned on the existence of the group $\mathcal{G}(E_1, E_2)$, i.e.,

$$R_{\mathcal{G}(E_1,E_2)}(s,t) = \sum_{G \sqsubseteq \mathcal{G}(E_1,E_2)} I_G(s,t) \times \frac{Pr(G)}{Pr(\mathcal{G}(E_1,E_2))} \tag{7}$$

Next, one may verify that the following holds.

$$R(s,t) = R_{\mathcal{G}(\phi,\phi)}(s,t) \tag{8}$$

Also, for any edge $e \in E \setminus (E_1 \cup E_2)$,

$$\begin{aligned} &R_{\mathcal{G}(E_1,E_2)}(s,t) \\ &= P(e) R_{\mathcal{G}(E_1 \cup \{e\},E_2)}(s,t) + (1 - P(e)) R_{\mathcal{G}(E_1,E_2 \cup \{e\})}(s,t) \end{aligned} \tag{9}$$

We terminate aforementioned recursive procedure when either E_1 contains an s-t path with $R_{\mathcal{G}(E_1,E_2)}(s,t) = 1$, or E_2 contains an s-t cut with $R_{\mathcal{G}(E_1,E_2)}(s,t) = 0$.

The efficiency can further be improved by selecting the "best" *expendable* edge (i.e., edge e in Equation 9) at each iteration. In particular, by following the experimentally optimal strategy in [18], we employ depth-first search (DFS) for the next edge expansion. We also find that this strategy works well in our experiments. Starting from the source node s, we start to explore its first neighbor (its next neighbor is explored only if there is no path to t which can be found going through the earlier ones), and then recursively visit the neighbors of this neighbor.

The aforementioned recursive sampling process has the same variance as the basic MC sampling. The variance can be reduced by eliminating the "uncertainty" of the existence of edge e in Equation 9. Let $\pi = R_{\mathcal{G}(E_1,E_2)}(s,t)$, $\pi_1 = R_{\mathcal{G}(E_1 \cup \{e\},E_2)}(s,t)$, and $\pi_2 = R_{\mathcal{G}(E_1,E_2 \cup \{e\})}(s,t)$. Now, instead of directly sampling both the children nodes from the root (as suggested in Equation 9), we consider to estimate both π_1 and π_2 independently, and then combine them together to estimate π. Specifically, for K total samples in the root, we deterministically allocate K_1 of them to the left subtree (prefix group that includes edge e), and K_2 of them to the right subtree (prefix group that excludes edge e). It was shown in [18] that when the sample size allocation is proportional to the edge inclusion probability, i.e., $K_1 = P(e) \cdot K$ and $K_2 = (1 - P(e)) \cdot K$, the variance of the earlier recursive estimator can be reduced. Moreover, when a prefix group size K is below a pre-defined threshold, we use a non-recursive sampling method, such as the basic Monte Carlo or more sophisticated Hansen-Hurwitz estimator to sample the remaining edges [18].

The time complexity of recursive sampling estimator is $\mathcal{O}(na)$, where a is the average recursion depth, and is bounded by the diameter of the graph. Note that in the original paper by Jin et al. [18], recursive sampling was proposed to estimate distance-constrained reliability, that is, the probability that s is reachable to t within an input distance d. In this work, we adapted the proposed approach to compute the s-t reliability without any distance constraint.

Table 1: Stratum design for recursive stratified sampling

Stratum	e_1	e_2	e_3	...	e_r	e_{r+1}	...	e_m	Prob space
Stratum 0	0	0	0	...	0	*	...	*	Ω_0
Stratum 1	1	*	*	...	*	*	...	*	Ω_1
Stratum 2	0	1	*	...	*	*	...	*	Ω_2
...					...				...
Stratum r	0	0	0	...	1	*	...	*	Ω_r

2.5 Recursive Stratified Sampling

Li et al. [26] developed an alternative divide-and-conquer approach to measure reliability, called the recursive stratified sampling (RSS). By using a stratification method that partitions the probability space Ω into $r+1$ non-overlapping subspaces ($\Omega_0, \ldots, \Omega_r$) via selecting r edges, they proceed to fix the states for certain edges in each stratum. For example, as seen in Table 1, in stratum 0, we set the status of r selected edges as 0, while leaving the rest of edges as undetermined. Subsequently, in stratum i ($1 \leq i \leq r$), we set the status of edge i to 1, the status of those before it as 0, and those after it as undetermined.

Let T be the set of selected r edges via BFS from the source node s, and $X_{i,j}$ be the corresponding status vector of the j-th ($1 \leq j \leq r$) selected edge in stratum i. Then, the probability of a possible graph in stratum i is given by:

$$\begin{aligned} \pi_i &= Pr\left[G_P \in \Omega_i\right] \\ &= \prod_{e_j \in T \bigwedge X_{i,j}=1} P(e_j) \prod_{e_j \in T \bigwedge X_{i,j}=0} (1 - P(e_j)) \end{aligned} \tag{10}$$

We then set the sample size of stratum i to $K_i = (\pi_i \cdot K)$, where K being the total sample size. The algorithm recursively applies the sample size to each stratum and simplifies the graph. It terminates when the sample size of a stratum is smaller than a given threshold. Reliability is then calculated by finding the sum of the reliabilities in all subspaces. The pseudocode is presented in our extended version [19].

It was shown in [26] that the time complexity of recursive stratified sampling is same as that of the MC sampling, i.e., $\mathcal{O}(K(m + n))$, while the variance of the estimator is significantly reduced.

2.6 Lazy Propagation Sampling

Li et al. proposed the lazy propagation sampling [27] that aims to bypass MC sampling's requirement of probing a large number of edges. In particular, if an edge has a low probability, it will remain un-activated in most of the samples; and therefore, those probings could be "unnecessary". In contrast, lazy propagation sampling estimates reliability by avoiding unnecessary probing of edges as

Algorithm 1 Lazy Propagation Sampling

Input: source node s, target node t in uncertain graph $\mathcal{G} = (V, E, P)$, K = #samples
Output: reliability $R(s, t)$

```
1:  r ← 0, set all nodes not initialized
2:  for i = 1 to K do
3:    if s == t then
4:      r ← r + 1
5:      goto line 29
6:    end if
7:    h ← φ, set all nodes not visited
8:    h ← {s}        /* set of visited nodes */
9:    mark s visited
10:   while h ≠ ∅ do
11:     v ← h.pop()
12:     if v is not initialized then
13:       c_v ← 0, h_v ← ∅, mark v initialized
14:       for nbr ∈ v's neighbor sets do
15:         X(nbr) ← geometric r.v. instance
16:         h_v ← h_v ∪ {⟨nbr, X(nbr) + c_v⟩}
17:       end for
18:     end if
19:     while h_v.top() == c_v do
20:       ⟨nbr, X(nbr)⟩ ← h_v.pop()
21:       h ← h ∪ {nbr} if nbr is not visited
22:       mark nbr visited
23:       X'(nbr) ← geometric r.v. instance
24:       h_v ← h_v ∪ {⟨nbr, X'(nbr) + c_v+1⟩}   /* our correction in bold */
25:       if nbr == t then
26:         r ← r + 1
27:         goto line 29
28:       end if
29:     end while
30:     c_v ← c_v + 1;
31:   end while
32: end for
33: return r/K
```

much as possible. To achieve this, the algorithm employs a geometric distribution per edge, and probes an edge only when it will be activated. The geometric random instance determines, based on the original edge probability, the number of sampling instances before the edge is activated. This means that the algorithm will predict how many k worlds are sampled before an edge exists. In doing so, the number of times an edge is probed is reduced by a factor of $\frac{1}{p(e)}$ in expectation. It was proved in [27] that there is no statistical difference between using lazy sampling and the classic MC sampling. In other words, lazy sampling has the same variance as that of MC sampling, while improving on the efficiency.

As the original algorithm [27] was developed for personalized social influential tags exploration, we adapt it to measure reliability over uncertain graphs. We show the complete procedure in Algorithm 1. It initializes a heap h_v for every visited node v, and pushes v's each out-neighbor nbr with a geometric random instance, i.e., $\langle nbr, X(nbr)\rangle$ into h_v (lines 12-18). It also maintains a counter c_v to keep track of the number of times v has been visited. Once a random instance in h_v is equal to c_v, the corresponding neighbor will be probed, and then a new random instance is generated to decide the next time to probe the neighbor (lines 19-29).

Our correction in the algorithm. In the original paper [27], it is $h_v \leftarrow h_v \cup \{\langle nbr, X'(nbr) + c_v\rangle\}$ at line 24, which, in fact, incurs error in reliability estimation [3]. This is because node nbr is probed in the current round (since $h_v.top() == c_v$ was true at line 19), and now line 23 aims at assigning a new geometric random instance $X'(nbr)$ with nbr. This new geometric random instance $X'(nbr)$ indicates that after how many times of failure, nbr will be visited again following node v, starting from the next round. Therefore, at line 24, the value of counter c_v shall be that of the next round, i.e., one needs to add $c_v+\mathbf{1}$ with $X'(nbr)$ (as opposed to adding only c_v specified in the original paper [27]).

We shall demonstrate the error in [27] and our correction with the following example.

Figure 4: Toy graph for Example 1

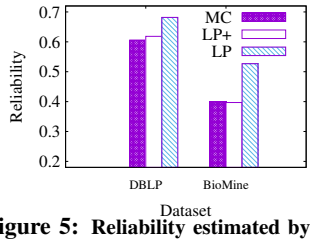


Figure 5: Reliability estimated by MC, Lazy Propagation (LP), and Lazy Propagation+ (LP+) at convergence. Reliability is reported as the average over 100 s-t pairs. For experimental setting, see Sec. 3.

EXAMPLE 1. *Considering the graph in Figure 4: Node* 1 *is the source node, and node* 3 *the target node. In the first round,* $c_1 = 0$ *and we initialize node* 1*, push its neighbor node* 2 *into* h_1*. Suppose that the geometric random instance* $X(2)$ *for* 2 *is generated as 0. Then, node* 2 *exists in the first possible world (i.e., the condition in line 19 holds true for node* 2*), and the algorithm proceeds to line 20-24. Now, if we follow* $h_v \leftarrow h_v \cup \{\langle nbr, X'(nbr) + c_v\rangle\}$ *at line 24 as in the original paper [27], we shall encounter at least one of the two following errors.*

(1) Assume that at line 24, the new geometric random instance $X'(2) > 0$*, e.g.,* $X'(2) = 1$*. It indicates that in the next possible world, edge* $1 \rightarrow 2$ *must not exist. However, when following the original algorithm as in [27],* $\langle Node\ 2, 1\rangle$ *will be stored in* h_1*, since* $c_1 = 0$ *at present. When sampling the next possible world,* $c_1 = 1$ *and node* 2 *will be probed again, which is incorrect. Clearly, this is an overestimation, and it will result in higher estimated reliability from source to target.*

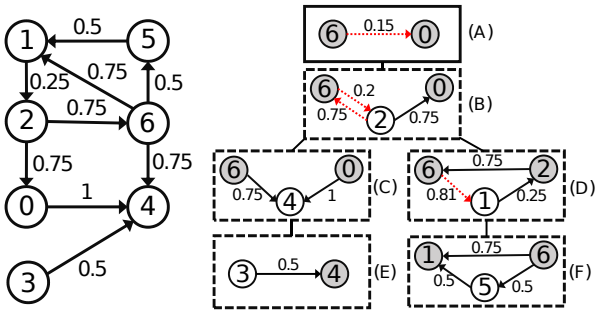
(2) In the other case, when the new geometric random instance $X'(2) = 0$ *at line 24, it means that in the next possible world, edge* $1 \rightarrow 2$ *must exist. By following the original algorithm as in [27],* $\langle Node\ 2, 0\rangle$ *will be pushed in the heap* h_1 *and would be placed at the top. However, when sampling all subsequent possible worlds,* $c_1 \geq 1$ *and* $\langle Node\ 2, 0\rangle$ *at the top of* h_1 *will always stop the algorithm to enter lines 20-24, which makes node* 1 *not expandable anymore, and thus incurs an underestimation error.*

In practice, the first type of error, i.e., overestimation happens in most cases, and the original method in [27] estimates much higher reliability. Thus, we update the algorithm at line 24, as shown in Algorithm 1 to avoid such errors. We denote it as LP+, and the original one as LP. Our experimental results in Figure 5 confirms that LP indeed estimates much higher reliability than Monte Carlo (MC), whereas LP+ estimation is close to that of MC.

2.7 Indexing via Probabilistic Trees

Maniu et al. [29] proposed ProbTree index to improve the efficiency of s-t reliability queries. The method builds a tree index structure, called ProbTree, from the input uncertain graph G. Next, given an s-t reliability query q, an equivalent graph $G(q)$ is created from the ProbTree index structure, and MC sampling is performed

[3] Yuchen Li, first author of [27], acknowledged this issue in email communication.



(a) Input uncertain graph (b) FWD index tree

Figure 6: Example of FWD ProbTree (w=2) construction

on $G(q)$. Clearly, if $G(q)$ can be generated quickly, and if $G(q)$ is smaller than G, then executing the query on $G(q)$ would be faster.

In [29], three index structures were developed. Among them, we employ FWD tree (an abbreviation for fixed-width tree decomposition) because (a) its index building time, space, and query processing time are linear in the input graph size, and (b) the index structure is lossless (i.e., produces good-quality results) for a tree decomposition width $w \leq 2$. In fact, as reported in [29], as well as found in our experiments, FWD tree with w=2 produces high-quality results, and exhibits good efficiency.

FWD tree index building has three phases: fixed-width tree decomposition, building of the FWD index tree, and pre-computation of reliability. The first phase is an adaption of the algorithms in [35, 38], which performs a *relaxed* tree decomposition with a fixed width w. At each step, a node having degree at most w is chosen, marked as *covered*. A bag is created to contain this node and its neighbors, along with the probabilistic edges between them in G. Then, the covered node is removed from G and a clique between its neighbors is added into G. This process repeats until there are no such nodes left. Finally, the rest of the uncovered nodes and the remaining edges are copied in the root of the tree.

The second phase is the creation of the FWD index tree. Each bag is visited in its creation order, and their parents are defined as the bag whose node set contains all uncovered nodes of the visited bag. If no such bag exists, then the parent of the bag will be the root of the tree. In the final phase, we need to compute $R(v_1, v_2)$ for each pair (v_1, v_2) in each bag. It follows a bottom-up manner. For each bag B, it collects the computed reliability from B's children, and combine them with information within current bag to compute the current reliability for all node pairs in B. When $w \leq 2$, it is possible to pre-compute the correct probability distributions between node pairs, thereby making the index structure lossless.

When conducting an s-t query, if the root contains both s and t, the root is the query graph. Otherwise, it searches from the bottom to find the bags containing s or t as the covered node, respectively. Then it propagates them up all the way to the root, and merges them as a combined graph for query answering. By doing so, all irrelevant branches will not be included in the graph returned. An example of index building and query processing is presented below.

EXAMPLE 2. *In Figure 6(b), we show the FWD index tree for the input uncertain graph in Figure 6(a). At first, node 3 has the smallest degree 1 and is chosen. A bag (E) is created to contain it, together with its neighbor, node 4. Then, we put all the unmarked edges between them into this bag and mark them. After the construction of this bag, we remove node 3 (the selected node) from the original graph. Now, no node has degree 1, then we consider degree 2. Due to the removal of node 3, node 4 has degree 2 and can be selected. We treat it in the same way and obtain the second bag (labelled as (C) in Figure 6(b)). Since there is no edge between node 0 and node 6 (neighbors of node 4) in the original graph, a new edge (0,6) is added. We repeat the aforementioned procedure until no more bag can be created. Then, we traverse them in the creation order $(E), (C), (F), (D), (B), (A)$, and decide their parent-child relationships using lines 20-24. For computing reliability in each bag, it follows a bottom-up manner as described before. For example, in bag (D), the reliability from node 6 to node 1 is collected from its child, bag (F), and is computed as: $1 - (1 - 0.75) * (1 - 0.5 * 0.5) = 0.8125$.*

For reliability from node 1 to node 2, we find bags (B) and (D) (which have these nodes as cover), and finally bags (A), (B), and (D) are merged as the query graph. Bags (C) and (E) are irrelevant to this query. Bag (F)'s information about the query node is already contained in bag (D). Thus, it will not be involved as well.

Following [29], the time complexity of FWD probtree index construction is linear in the number of nodes in the graph, and the complexity of pairwise reliability computations in each bag is $\mathcal{O}(w^2 d)$, where w is the tree width ($w = 2$ to ensure loseless index), and d is the diameter of graph. The space complexity of FWD probtree index is $\mathcal{O}(|E|)$. During online s-t query processing, the cost of retrieving s and t from the FWD probtree is linear in the tree depth. The reliability estimating cost is the same as MC sampling, $\mathcal{O}(K(n' + m'))$, but with smaller number of nodes and edges.

Our adaptation in complexity. The original *ProbTree* index is designed to support multiple kinds of s-t queries, e.g., reliability query, shortest path query, etc. Therefore, it pre-computes the distance probability distribution between each node in every bag. However, if only considering s-t reliability query, we can just compute and store the edge probability information, regardless of distance. Since $w = 2$, for each pair of nodes in the current bag, there can exist at most 2 paths in its children. One can easily collect and aggregate them by $1 - (1 - p_1)(1 - p_2)$, where p_1 and p_2 are the probability of these two paths. An illustration can be found in Example 2. The complexity of pairwise reliability computations in each bag can be reduced from $\mathcal{O}(w^2 d)$ to $\mathcal{O}(w^2)$. Empirically, on our largest *BioMine* dataset, we can reduce the index building time from 4062 seconds to 2482 seconds.

2.8 Horizontal Comparison of Algorithms: Strengths and Weaknesses

Estimator variance. The mean squared error (MSE) measures the quality of an estimator. The lower the MSE, the better is an estimator. For unbiased estimator, the MSE is equal to the variance of this estimator. *All estimators compared in this work are proved to be unbiased in original papers. Statistically, BFS Sharing and Lazy Propagation shares same variance as basic MC Sampling. The two recursive methods, RHH and RSS, reduce the estimator variance by dividing the sample size according to either edge probability (e.g, RHH), or stratum probability (e.g., RSS).* We refer to Theorem 2 in [18], and Theorems 4.2 and 4.3 in [26] as the theoretical basis for variance reduction in RHH and RSS, respectively.

Running time per sample. For each sample, MC applies a BFS search from the source node s, each edge encountered exists with the probability on that edge. The BFS search terminates when the target node t is found (early termination), or no new node can be expanded. In contrast, while BFS Sharing follows the same BFS search strategy; no early termination is allowed. The existence of each edge in every sample has been determined offline (via indexing). Therefore, *BFS Sharing tends to sample more edges than MC in each sample, while the time cost of checking the existence of each edge is saved due to indexing*.

Lazy propagation employs geometric distribution to determine the existence of an edge in samples, that is, each time when an edge

Table 2: Properties of datasets

Dataset	#Nodes	#Edges	Edge Prob: Mean, SD, Quartiles
LastFM	6 899	23 696	0.29 ± 0.25, {0.13, 0.20, 0.33}
NetHept_uniform	15 233	62 774	0.04 ± 0.04, {0.001, 0.01, 0.10}
AS_Topology	45 535	172 294	0.23 ± 0.20, {0.08, 0.21, 0.31}
DBLP_0.2	1 291 298	7 123 632	0.33 ± 0.18, {0.18, 0.33, 0.45}
DBLP_0.05	1 291 298	7 123 632	0.11 ± 0.09, {0.05, 0.10, 0.14}
BioMine	1 045 414	6 742 939	0.27 ± 0.21, {0.12, 0.22, 0.36}

is probed during the BFS search in one sample, it decides after how many samples this edge will exist again. Therefore, unnecessary edge probing is avoided.

ProbTree decomposes the graph and pre-computes the reliability information beforehand, thus allowing running online BFS on a smaller graph to improve efficiency. RHH and RSS recursively simplify the graph, and apply MC sampling on the resulted graph when sample size is smaller than a threshold, or the graph is fully simplified. The improvement of the running time per sample depends on how much the graph can be simplified. Additional time cost may arise from graph simplification task and the recursive procedure. In all, *Lazy Propagation and ProbTree can reduce the time cost of investigating a sample when compared with MC, while BFS Sharing, RHH and RSS have both improvements and additional costs*. The detailed experimental study can be found in Section 3.5.

Total running time. To achieve the same estimator variance (e.g., same quality), MC, BFS Sharing, Lazy Propagation, and ProbTree would require same sample size, thus the Lazy Propagation and ProbTree shall improve the efficiency (due to lower running time per sample) compared to MC. For RHH and RSS, less samples are needed, thereby also improving the efficiency compared to MC. *With all these, the final efficiency comparison for Lazy Propagation, ProbTree, RHH, and RSS is unknown. Therefore, our experimental analysis is critical*.

Memory usage. In addition to the original graph, the only memory cost of MC is the current sample, and the count of samples where t is reachable from s. For BFS Sharing, instead of original graph, the whole index shall be kept in main memory. Each edge in BFS Sharing index is a vector of Boolean values, while in the original graph only a single value is stored. Compared with MC, Lazy Propagation additionally requires a global counter for each node, and a geometric random instance heap for its neighbors. The ProbTree index size is linear in the original graph size. Both RHH and RSS store the whole recursive stack, and simplified graph instances in main memory, which makes them more memory intensive.

Indexing cost, re-computation, and adaptability. Both BFS Sharing and ProbTree index sizes are linear in the original graph size. ProbTree index is generally comparable to the original graph size, and is independent of the sample size K. However, BFS Sharing is also about linear in the sample size K. Thus, *BFS Sharing index is usually larger than ProbTree index, and BFS Sharing also requires more loading time into main memory. On the other hand, BFS Sharing index is easier to build than ProbTree index*. Only K boolean values shall be generated based on the edge probability for each edge. The ProbTree index requires decomposing the original graph and pre-computing the reliability information.

To ensure the independence among queries, index shall be updated between two successive queries for BFS Sharing, this is not necessary for ProbTree. Moreover, ProbTree has the adaptability to support estimators other than MC sampling, while BFS Sharing has no such potential.

3. EXPERIMENTAL RESULTS

We conduct experiments to compare six state-of-the-art reliability estimation algorithms, and report the number of samples required for their convergence, estimator accuracy, variance, running time, and memory usage using medium and large-scale, real-world network datasets. We obtain the source code of ProbTree [29] from the authors, which is written in C++, and we further optimize their index building method as discussed in the paragraph "Our improvement in complexity" in Section 2.7. We implement other five algorithms in C++, and perform experiments on a single core of a 100GB, 2.40GHz Intel Xeon E5-4650 v2 server. *Our datasets and source code are available at:* `https://github.com/5555lan/RelComp`.

3.1 Environment Setup

3.1.1 Datasets

We downloaded five real-world networks (Table 2). Many of them have been extensively used in past research on uncertain graphs, including reliability estimation [18,21,26,27,31,40,41]. • *LastFM* (www.last.fm). Last.FM is a musical social network, where users listen to their favorite tracks, and communicate with each other based on their musical preferences. We crawled a local network of Last.FM, and formed a bi-directed graph by connecting two users if they communicated at least once. • *NetHEPT* (www.arXiv.org). This graph was extracted from the "High Energy Physics Theory" section of the e-print arXiv with papers from 1991 to 2003. The nodes are connected by bi-directed edges if they co-authored at least once. • *AS Topology* (http://data.caida.org/datasets/topology/ark/ipv4/). An autonomous system (AS) is a collection of connected Internet Protocol (IP) routing prefixes under the control of one or more network operators on behalf of a single administrative entity, e.g., a university. The AS connections are established with BGP protocol. It may fail due to various reasons, e.g., failure of physical links when one AS updates its connection configuration to ensure stricter security setting, while some of its peers can no longer satisfy it, or some connections are cancelled manually by the AS administrator. We downloaded one network snapshot per month, from January 2008 to December 2017. • *DBLP* (www.informatik. uni-trier.de/ ley/db/). This is a well-known collaboration network. We downloaded it on March 31, 2017. Each node is an author, and bi-directed edges denote their co-authorship relations. • *BioMine* (www.cs. helsinki.fi/group/ biomine/). This is the database of the BIOMINE project [10]. The graph is constructed by integrating cross-references from several biological data -bases. Nodes represent biological concepts such as genes, proteins, etc., and directed edges denote real-world phenomena between two nodes, e.g., a gene "codes" for a protein.

3.1.2 Edge Probability Models

By following bulk of the literature on reliability estimation over uncertain graphs [18, 21, 26, 27, 31, 40, 41], we adopt the following widely-used edge probability models.

• *LastFM*. The probability on any edge corresponds to the inverse of the out-degree of the node from which that arc is outgoing. • *NetHEPT*. Each edge is assigned with a probability, chosen uniformly at random, from (0.1, 0.01, 0.001). • *AS_Topology*. Once an AS connection (i.e., an edge) is observed for the first time, we calculate the ratio of snapshots containing this connection within all follow-up snapshots as the probability of existence for this edge. • *DBLP_0.2* and *DBLP_0.05*. The edge probabilities are derived from an exponential cdf of mean μ to the number of collaborations between two respective authors; hence, if two authors collaborated c times, the corresponding probability is $1 - \exp^{-c/\mu}$. We consider $\mu = 5, 20$ in our experiments, and generate two uncertain networks, *DBLP_0.2* and *DBLP_0.05*, respectively. Clearly, higher values of μ generate smaller edge probabilities. • *BioMine*. Edge probabilities, which quantify the existence of a phenomenon between the two endpoints of that edge (a gene "codes" for a pro-

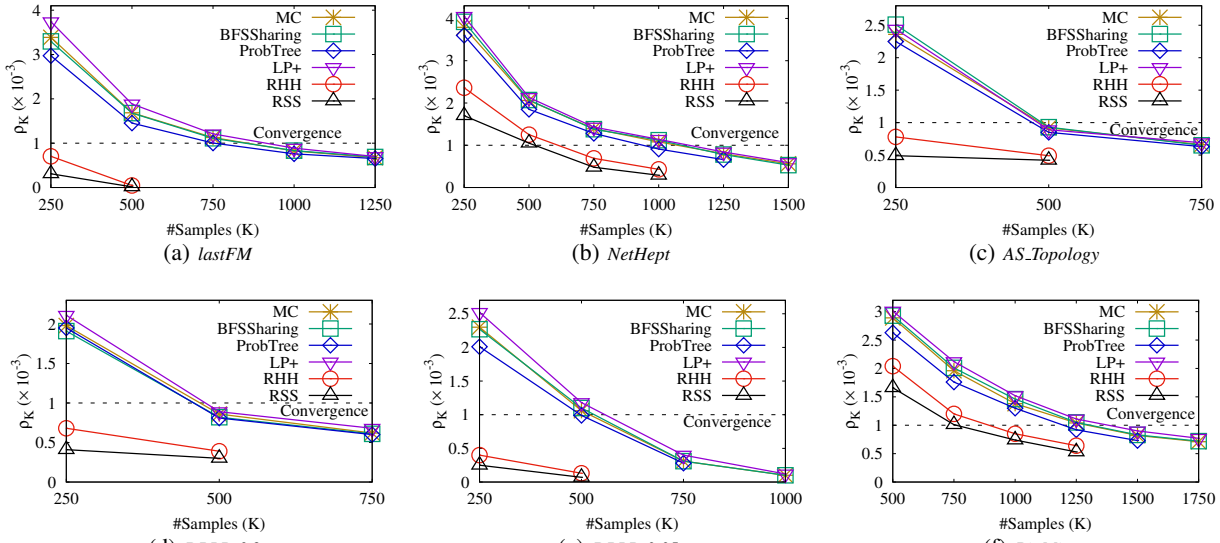


Figure 7: Comparison of estimator variance and convergence. The symbol ρ_K on Y-axis denotes the ratio $\frac{V_K}{R_K} = \frac{\text{average variance at \#samples=}K}{\text{average reliability at \#samples=}K}$.

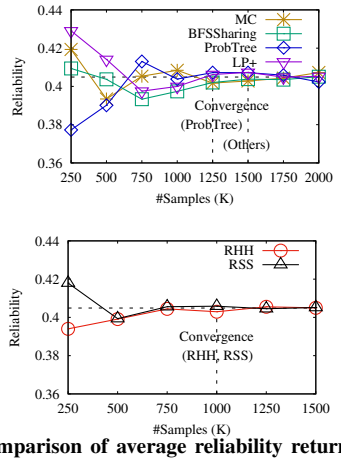


Figure 8: Comparison of average reliability returned by each estimator in regards to average reliability returned by MC sampling at $K = 10\,000$ (shown as horizontal dashed line), *BioMine*.

tein), were determined in [10] as a combination of three criteria: relevance (i.e., relative importance of that relationship type), informativeness (e.g., degrees of the nodes adjacent to that edge), and confidence on the existence of a specific relationship (e.g., conformity with the biological STRING database).

3.1.3 Parameters Setting

For each dataset, 100 distinct s-t pairs are generated as follows. For a specific graph, we first select 100 different source nodes, uniformly at random. For a source node, we next perform BFS up to 2 hops. Among these visited nodes we select one target node that is 2-hop away from s, uniformly at random. These 100 s-t pairs are used consistently for all six competing methods over that dataset. All our results are averaged over these 100 s-t pairs.

Notice that we select s-t pairs with shortest-path distance = 2 hops, because if they are closer, their reliability would usually be higher. On the other hand, if some s-t pairs are far apart, their reliability would be naturally small. Nevertheless, we also demonstrate experimental results by varying the shortest-path distance between s-t pairs in our extended version [19].

The initial K, i.e., #samples considered in all algorithms is 250. We then increase K by a step of 250 till convergence is reached. (we refer to Section 3.1.4 for discussion on convergence).

For recursive estimators [18, 26], a few additional parameters need to be defined. For recursive sampling, we set the prefix group size threshold to be 5 as per [18]. For recursive stratified sampling, we set r = 50 (i.e., #selected edges in each stratum) as recommended in [26]. We find that these parameter values also work well in our experimental setting (For details, we refer to our sensitivity analysis results in the extended version [19]). Following [18, 26], we refer to recursive sampling and recursive stratified sampling as RHH and RSS, respectively.

3.1.4 Performance Metrics

Variance: Unlike MC-based estimators [11, 27, 29, 41], which report reachability from s to t as 1 or 0 in each sample; both recursive sampling algorithms [18, 26] estimate reliability in a holistic manner by considering *all* K samples. Thus, following [18, 26], we compute variance of an estimator by repeating experiments with the current number of samples (K). Given an estimator and K, we repeat querying each s-t pair, e.g., s_i-t_i for T times; and we obtain T estimation results for each pair s_i-t_i: $R_1(s_i, t_i, K)$, $R_2(s_i, t_i, K)$, ..., $R_T(s_i, t_i, K)$. We calculate the variance as follows.

$$V(s_i, t_i, K) = \frac{1}{T-1} \sum_{j=1}^{T} (R_j(s_i, t_i, K) - \overline{R}(s_i, t_i, K))^2 \quad (11)$$

Here, $\overline{R}(s_i, t_i, K)$ is the average value of these T estimation results, for a fixed K. Following [18], we set $T = 100$. We compute this variance for all 100 s-t pairs, and the average variance for this estimator, with given K, is:

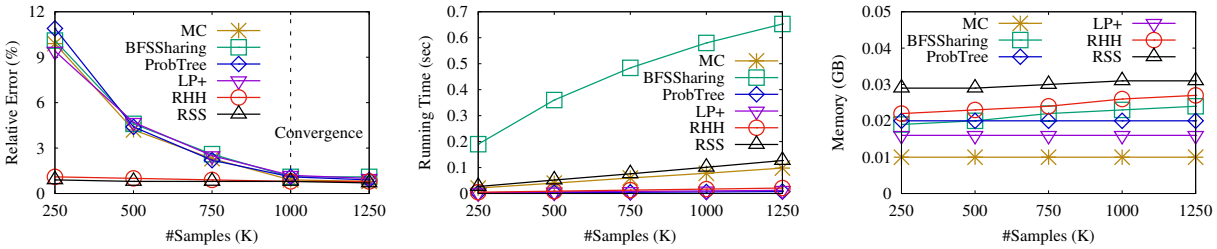
$$V_K = \frac{1}{100} \sum_{i=1}^{100} V(s_i, t_i, K) \quad (12)$$

Analogously, we define average reliability for an estimator, with given K, as:

$$R_K = \frac{1}{100} \sum_{i=1}^{100} \overline{R}(s_i, t_i, K) \quad (13)$$

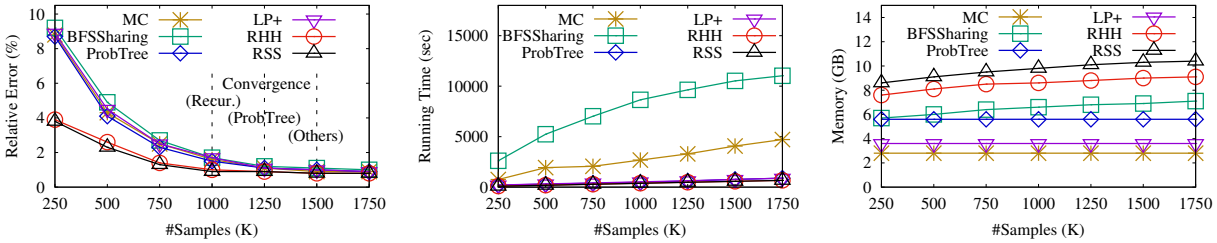
Based on our experiments, we find that as one increases K, the average variance V_K monotonically decreases. However, we also notice that, with same K, V_K varies for different datasets and estimators. This is primarily because the average reliability R_K is different for different datasets and estimators, even for same K. Thus, it is *difficult* to fix a uniform threshold on the average variance V_K which could define convergence in all cases.

Instead, we systematically consider the ratio $\rho_K = \frac{V_K}{R_K}$ to decide convergence of an estimator over a given dataset. The ratio of variance to mean is also known as the **index of dispersion**, which



(a) *Relative Error* (b) *Running Time* (c) *Memory Usage*

Figure 9: The trade-off between relative error and running time/memory usage, *lastFM*.



(a) *Relative Error* (b) *Running Time* (c) *Memory Usage*

Figure 10: The trade-off between relative error and running time/memory usage, *BioMine*.

is a *normalized* measure of the dispersion of a dataset. Dispersion is close to zero if all the data are similar, and increases as the data become more diverse. Given a dataset and an estimator, we keep increasing K (in steps of 250), and when $\rho_K = \frac{V_K}{R_K} < 0.001$, we say that the convergence has been reached for that estimator and over that dataset. For that estimator and dataset, we report average reliability at that specific value of K, since this estimation is robust.

Relative error: By following [18, 29, 41], we report the relative error (RE) of reliability estimation for an estimator with respect to MC sampling. This is computed as follows.

$$RE_K = \frac{1}{100}\sum_{i=1}^{100}\frac{|\overline{R}(s_i,t_i,K) - \overline{R}_{MC}(s_i,t_i,Convergence)|}{\overline{R}_{MC}(s_i,t_i,Convergence)} \tag{14}$$

Here, $\overline{R}_{MC}(s_i,t_i,Convergence)$ denotes the reliability of the pair s_i-t_i, as returned by MC sampling at convergence. On the other hand, $\overline{R}(s_i,t_i,K)$ denotes the reliability of the same pair, returned by the specific estimator, at #samples=K.

Online and offline efficiency and memory usage: In regards to online querying, we report (a) total time cost to answer an s-t reliability query, (b) average time cost per sample (i.e., total time cost/#samples), and (c) memory usage for all algorithms. In addition, for index-based methods, i.e., BFS sharing [41] and ProbTree [29], we report their (i) time costs for creating index, (ii) time costs for loading index in main memory, and (iii) index sizes.

3.2 Estimator Variance and Convergence

We report our empirical findings on estimator variance and convergence in Figure 7 and Figure 8, which are summarized below.

(1) Figure 7 depicts that among six competing estimators, four MC-based estimators: Monte Carlo (MC) sampling, BFS Sharing, ProbTree, and Lazy Propagation (LP+) exhibit nearly similar characteristics in estimator variance. The other two recursive estimators: Recursive Sampling (RHH) and Recursive Stratified Sampling (RSS) share similar estimator variance between them.

In particular, BFS Sharing performs the sampling of MC offline by building indexes, and this does not change the estimator variance compared to MC. Another index-based method, ProbTree decomposes the original graph into "bags" and re-organizes them in a tree structure. When processing an online query, it generates a smaller but equivalent graph from the tree index, by ignoring irrelevant branches. We apply MC (as the original paper [29] did) to estimate the s-t reliability on this smaller but equivalent graph. Hence, the estimator variance remains same. For LP+, it utilizes the geometric distribution to decide the existence of an edge in samples, which is statistically equivalent [27] to MC; therefore results in similar variance. Overall, our empirical results confirm them.

For recursive estimators, [26] discussed that RHH is a special case of RSS when $r = 1$ (r denotes the number of selected edges in RSS). In Figure 7, we observe that the curves of RSS and RHH are close, but RSS always has lower variance than RHH.

(2) Our experimental results in Figure 7 confirm that *recursive estimators have lower variance than MC-based estimators*. By dividing the sample size according to either edge probability (e.g., RHH), or stratum probability (e.g., RSS), the uncertainty during sampling is reduced, which results in lower variance. It can be noted that *some later proposed methods, e.g., ProbTree and LP+ do not outperform RHH and RSS in estimator variance*.

(3) As discussed in Section 2.2, larger K ensures more accurate result by MC sampling. In Figure 8, we compare the reliability returned by each estimator with respect to that of MC sampling with very large K, e.g., $K = 10000$ (shown as horizontal dashed line) [37, 39]. *Figure 8 clearly presents that the reliability estimated at variance convergence is very close to that at very large K, which indicates that estimator variance convergence can help find a sample size for high quality estimation.*

(4) Past works on uncertain graphs generally employed some standard K (i.e., #samples) in their experiments, such as K=500 [31], 1 000 [18, 29], 2 000 [26, 41], or even 10 000 [20], while mentioning that they observed convergence for those specific K in their experimental setting. For future researchers and practitioners, we would like to emphasize that *there is no single K such that all estimators achieve convergence across all datasets*, which can be confirmed from our empirical findings. The K at convergence for every estimator on each dataset is listed in Tables 3-6 (observed from Figure 7). Interestingly for MC-based methods, K values at convergence are almost different over various datasets. Recursive estimators reach convergence with smaller K, e.g., K=250 on three out of five datasets. However, they require around 750 and 1000 samples on *NetHept* and *BioMine* datasets, respectively, for convergence. *Recursive methods generally converge with about 500 less samples than MC-based methods on the same dataset.*

(5) With a closer look, we find that *ProbTree has a slight improvement in estimator variance compared to three other MC-based*

Table 3: Comparison of relative error (RE): *LastFM*

Estimator	At Convergence			At K=1000	
	K	R_K	RE (%)	R_K	RE (%)
MC	1000	0.1025	0.00	0.1025	0.00
BFS Sharing	1000	0.1030	**0.97**	0.1030	**0.97**
ProbTree	1000	0.1007	1.77	0.1007	1.77
Lazy Propagation	1000	0.1052	1.84	0.1052	1.84
Recursive (RHH)	250	0.1041	1.91	0.1040	1.88
Recur. Stratified (RSS)	250	0.1018	1.06	0.1020	1.07
Pairwise Deviation			0.53		0.52

Table 4: Comparison of relative error (RE): *NetHept*

At Convergence			At K=1000	
K	R_K	RE (%)	R_K	RE (%)
1250	0.00190	0.00	0.00183	2.18
1250	0.00194	1.83	0.00187	2.01
1000	0.00187	**1.47**	0.00180	**1.47**
1250	0.00196	1.93	0.00180	2.41
750	0.00196	1.73	0.00196	1.78
750	0.00192	**1.47**	0.00192	1.49
		0.26		0.48

Table 5: Comparison of relative error (RE): *DBLP_0.05*

Estimator	At Convergence			At K=1000	
	K	R_K	RE (%)	R_K	RE (%)
MC	750	0.2128	0.00	0.2136	**0.17**
BFS Sharing	750	0.2133	**1.26**	0.2134	1.15
ProbTree	750	0.2156	1.37	0.2162	1.40
Lazy Propagation	750	0.2154	1.52	0.2153	1.49
Recursive (RHH)	250	0.2114	1.39	0.2108	1.32
Recur. Stratified (RSS)	250	0.2124	1.36	0.2131	1.37
Pairwise Deviation			0.11		0.15

Table 6: Comparison of relative error (RE): *BioMine*

At Convergence			At K=1000	
K	R_K	RE (%)	R_K	RE (%)
1500	0.4019	0.00	0.4038	1.39
1500	0.4040	0.85	0.4041	1.62
1250	0.4050	**0.79**	0.4077	1.43
1500	0.4013	1.09	0.4068	2.47
1000	0.4052	1.15	0.4052	1.15
1000	0.4047	1.08	0.4047	**1.08**
		0.19		0.65

methods. This is because the ProbTree index pre-computes the reliability information contained in a bag's children subtrees, and stores it in the current bag. By directly applying such pre-computed probabilities, one can reduce the uncertainty of sampling, and lower the estimator variance. In our experimental results on *NetHept* and *Biomine*, ProbTree requires around 250 less samples for convergence, when compared with other MC-based estimators.

3.3 Trade-off among Relative Error, Running Time, and Memory Usage

Figures 9 and 10 demonstrate the trade-off among estimation error, running time, and memory usage. The estimation error is provided as the relative error with respect to the reliability returned by MC sampling at variance convergence (as discussed in Section 3.1.4).

As shown in each subfigure (a) of Figures 9 and 10, when reaching the estimator variance convergence, the relative error rates of all six methods are **(1)** very close to each other; **(2)** below 2%; and **(3)** also converge. The estimator accuracy can benefit little by further increasing the sample size K. However, *the running time of all estimators grows about linearly with the sample size K. The memory usages of estimators are not very sensitive to the sample size.* Memory usage of MC, ProbTree, and LP+ nearly remains the same all the way. With larger number of samples, more indexes are required to be loaded into main memory by BFS Sharing, thus slightly increasing its memory cost. For recursive methods, larger K can allow larger recursion depths, thus more memory is consumed. *In summary, the estimator variance convergence can help us to find the sweet point which balances the estimator error and running time/memory consumption.* More detailed accuracy and efficiency comparison can be found in Sections 3.4 and 3.5.

3.4 Estimator Accuracy

We compare relative errors (with respect to MC Sampling at variance convergence, Section 3.1.4) of all algorithms. Relative errors are reported (a) at convergence for that estimator, and (b) at K=1000. We report relative errors also at K=1000, since many prior works [18, 26, 29, 41] did the same, irrespective of whether an estimator has converged or not. Our results are given in Tables 3-6.

(1) *When the value of K at convergence is larger than 1000 for a method, its relative error at convergence is smaller than that at K=1000* (e.g., see the results over *NetHept* and *BioMine*). In contrast, *if the value of K at convergence is smaller than 1000 for a method, the relative error nearly remains the same when increasing K to 1000* (e.g., see the results over *lastFM*, *AS_Topology*, *DBLP_0.2*, and *DBLP_0.05* datasets). Therefore, K=1000 is not a fair setting to compare the estimator accuracy across all estimators and datasets. Rather, K at convergence for that estimator ensures higher accuracy. Similarly, if the estimator has already converged at some K <1000, the relative error would not reduce further, instead only the running time will increase for larger K.

(2) *At convergence, relative errors for all six methods are low (< 2%) and comparable (no common winner exists), which indicates that our approach of finding convergence (based on* **index of dispersion***) ensures high accuracy*. The best relative error rate on each dataset is below 1.5%.

Previous work [18] compared the relative error at K=1000, and concluded that RHH had better accuracy over MC-based methods. However, this does not hold when we consider K at convergence for respective methods. Empirically we find that K=1000 is sufficient for RHH and RSS to achieve convergence, while other methods may still require more samples (e.g., on *NetHept* and *BioMine*) for their convergence. Only considering K=1000 is unfair to them. We notice that at convergence, there does not exist a common winner in regards to relative error among these six estimators.

We further calculate the pairwise deviation (D) of relative errors (RE) across different estimators on each dataset.

$$D = \frac{1}{5 * 6} \sum_{i=1}^{6} \sum_{j=1}^{6} |RE(i) - RE(j)| \tag{15}$$

Here, $RE(i)$ denotes the relative error of method i. One can observe that if a dataset requires more than 1000 samples for convergence (e.g., *NetHept* in Table 4 and *BioMine* in Table 6), the pairwise deviation of relative errors among estimators over that dataset significantly decreases when increasing K from 1000 to that at convergence of respective estimators. These results further demonstrate that *at convergence, relative errors for all six methods are low and comparable; and measuring relative error at a specific value of K (e.g., K=1000) across all methods can be unfair to certain estimators*.

3.5 Estimator Efficiency

We present online running times for all methods both at convergence and at K=1000 in Tables 7-10. The time cost per sample is reported in milliseconds at convergence.

(1) *At convergence, RSS and RHH are the faster estimators*. RSS is similar or even faster than RHH on large datasets. However, it is slower than RHH on smaller datasets. *ProbTree and LP+ are in the middle range*, and are often comparable to RHH. *MC and*

Table 7: Comparison of running time: *LastFM*

Estimator	At Convergence		At K=1000	Time Per
	K	Time (sec)	Time (sec)	Sample (ms)
MC	1000	0.078	0.078	0.078
BFS Sharing	1000	0.593	0.593	0.593
ProbTree	1000	0.006	**0.006**	**0.006**
Lazy Propagation	1000	0.010	0.010	0.010
Recursive (RHH)	250	**0.004**	0.017	0.016
Recur. Stratified (RSS)	250	0.026	0.101	0.104

Table 8: Comparison of running time: *AS_Topology*

At Convergence		At K=1000	Time Per
K	Time (sec)	Time (sec)	Sample (ms)
500	166	327	332
500	641	1235	1282
500	19	**38**	**38**
500	20	39	40
250	**12**	45	48
250	14	55	56

Table 9: Comparison of running time: *DBLP_0.05*

Estimator	At Convergence		At K=1000	Time Per
	K	Time (sec)	Time (sec)	Sample (ms)
MC	750	670	899	893
BFS Sharing	750	2503	3094	3337
ProbTree	750	105	134	140
Lazy Propagation	750	81	**106**	**108**
Recursive (RHH)	250	**38**	150	152
Recur. Stratified (RSS)	250	45	150	179

Table 10: Comparison of running time: *BioMine*

At Convergence		At K=1000	Time Per
K	Time (sec)	Time (sec)	Sample (ms)
1500	4070	2678	2660
1500	12723	8644	8482
1250	600	482	480
1500	770	520	513
1000	389	389	389
1000	**375**	**375**	**375**

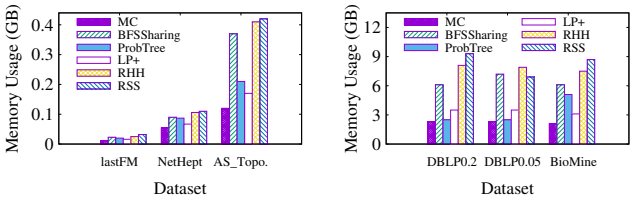


Figure 11: Online memory usage comparison

BFS Sharing are much slower. BFS Sharing consumes about $4\times$ running time when compared with MC.

High efficiency of RHH and RSS (one s-t query finishes within 400 seconds for all datasets) is due to the following reasons. (i) When finally conducting sampling in RHH and RSS, the graph has been simplified. This is because certain parts of the graph will no longer be connected to the source node with the absence of some selected edges. Moreover, these estimators can avoid sampling if there exists a path connecting s and t with selected edges. (ii) Due to faster convergence, the sample size K required in *RHH* and *RSS* is smaller than that for other methods.

ProbTree and LP+ reduces about 80% of the running time of MC. ProbTree simplifies the graph with pre-computed indexes. LP+ speeds up the sampling procedure by avoiding unnecessary probing of edges, which is implemented with geometric distribution. Unlike RSS and RHH, they still require larger number of samples; and hence, their running times are in the middle range.

MC can be terminated early when the target node is visited. But for BFS Sharing, no early termination is possible (see our discussion in Section 2.3). In summary, *though BFS Sharing was proposed after MC, it can be $4\times$ slower than MC in regards to s-t reliability estimation. Moreover, ProbTree and LP+ were developed after recursive methods, but both recursive methods are generally more efficient due to their faster convergence*.

(2) When K=1000, there is no common winner in running time. RHH, RSS, ProbTree, and LP+ are comparable, and each of them wins once or twice out of all our datasets. This is because the advantage of requiring less samples (to reach convergence) does not hold here for RHH and RSS. Therefore, *comparing these methods at a fixed K (e.g., K=1000 as it is done in [29]) is unfair to both RHH and RSS*. We notice that even at a fixed K=1000, MC and BFS Sharing are much slower than other estimators.

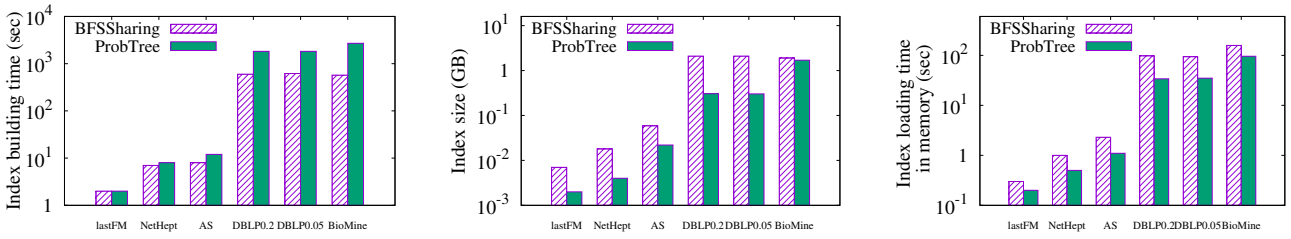
(3) We observe that *except for BFS Sharing, the time cost per sample is about the same at convergence and at K=1000, which indicates that their running time is linear in the sample size K*. For BFS Sharing, a small decrease in running time per sample can be viewed with increasing K, which implies that sharing BFS among samples reduces the impact of sample size K. However, its online running time is still not independent of K. *With larger K, the total running time of BFS Sharing still increases (which is evident in Tables 8-10). Therefore, we do not agree with the claim in [41] that their online running time is independent of K*. In fact, as we already discussed, its theoretical time complexity is still $\mathcal{O}(k(n+m))$, which is due to cascading updates (see the paragraph "Our correction in complexity analysis" in Section 2.3).

3.6 Estimator Memory Usage

Figure 11 presents the online memory usage for each algorithm. Since the memory usage to reach convergence is similar to that at K=1000, we only report the memory cost at convergence for every estimator. *The general increasing order of memory usage is: MC $<$ LP+ $<$ ProbTree $<$ BFS Sharing $<$ RHH $\approx$ RSS*. The memory usage of both RSS and RHH is about $4\times$ to that of MC, and reaches up to 10GB over our larger datasets. RHH and RSS consume space for recursion stack, selected edge sets and their existence statuses, and for the simplified graph instance. MC only stores the graph and BFS status variables. Compared to MC, LP+ requires a global counter for each node and a geometric random instance heap for its neighbors. Both ProbTree and BFS Sharing build indexes, and due to efficiency reasons, we load their indexes into memory. The index size of BFS Sharing is larger than that of ProbTree, and BFS Sharing additionally maintains a state vector for each node online. In spite of that, BFS Sharing and ProbTree require less memory than recursive estimators: RSS and RHH.

3.7 Indexing Time and Space

Since ProbTree and BFS Sharing rely on graph indexes, we evaluate offline cost for building, storing, and loading of their indexes (Figure 12) **(1)** ProbTree index is independent of sample size K. It decomposes the graph into "bags" and stores them in a tree structure. Its index size and building time depends on how many bags can be decomposed from the graph, and the depth of index tree. The maximum index size is 1.8GB over our largest *BioMine* dataset, and the corresponding maximum index loading time is 98 seconds, which is trivial when comparing with the time cost of s-t reliability estimation. However, index building requires about one hour over *BioMine*. **(2)** The index size of BFS Sharing is linear in the sample size K. As K at convergence is not known apriori, a length-L binary vector is attached to each edge in BFS Sharing index to represent the existence of this edge across L samples. In our experiments, we set L=1500 as a safe bound. We find that *the index building time of BFS Sharing is smaller than that of ProbTree*, since former just simply samples each edge L times. *However, its index size can be larger than that of ProbTree, and as a result the index loading time is also higher than that of ProbTree*. Still in-



(a) *Index building time* (b) *Index size* (c) *Index loading Time*

Figure 12: Offline index cost comparison for *ProbTree* and *BFS Sharing*

Table 11: Additional time of BFS Sharing index update (per query) while answering 1000 successive queries

Dataset	Time Cost (sec)
lastFM	0.02
NetHept	0.05
AS_Topology	0.11
DBLP_0.2	6.14
DBLP_0.05	5.64
BioMine	6.98

Table 12: ProbTree with efficient estimators: running time at convergence

Method	Running Time (sec)	
	AS_Top.	*BioMine*
LP+	20	770
ProbTree+LP+	**16**	**663**
RHH	12	389
ProbTree+RHH	**10**	**356**
RSS	14	375
ProbTree+RSS	**10**	**321**

Table 13: Summary and recommendation

Online Query Processing				
Method	Variance	Accuracy	Running Time	Memory
MC	★	★★★	★★	★★★★
BFS Sharing	★	★★★	★	★★
ProbTree	★	★★★	★★★	★★★
LP+	★	★★★	★★★	★★★★
RHH	★★★★	★★★★	★★★★	★
RSS	★★★★	★★★★	★★★★	★

Index-related				
Method	Time (build)	Time (load)	Time (update)	Size
BFS Sharing	★★★★	★★★	★	★★★
ProbTree	★★★	★★★★	★★★★	★★★★

dex loading time for BFS Sharing is within 200 seconds in most cases. *Unlike ProbTree, those edge indexes used by BFS Sharing to answer a query need to be re-sampled before processing the next query, in order to maintain inter-query independence*. We conduct 1000 successive queries with BFS Sharing and present the additional time cost (per query) for index updating in Table 11.

3.8 ProbTree with Efficient Estimators

ProbTree index decomposes the graph and pre-compute the reliability information during index building. When answering online queries, a smaller but equivalent graph is generated from the index. The sampling procedure is conducted on this simplified graph, thus the efficiency is improved. In previous sections, the ProbTree index is analysed only with MC sampling (as the original paper [29] did). As presented in Table 12, ProbTree is able to support other estimators and even improve the efficiency by 10-30%.

4. DISCUSSION AND CONCLUSION

Summary. In this work we investigated six state-of-the-art sequential algorithms for s-t reliability estimation, corrected certain issues in these algorithms to further improve their performance, and conducted a thorough experimental evaluation.

For estimator variance, both recursive methods: RHH and RSS exhibit significantly better performance than other four MC-based approaches: MC, BFSSharing, ProbTree, and LP+. Methods in the same category share very similar variance. In general, RSS is the best regarding variance, and achieves fastest convergence. To achieve convergence, there is no single sample size K that can

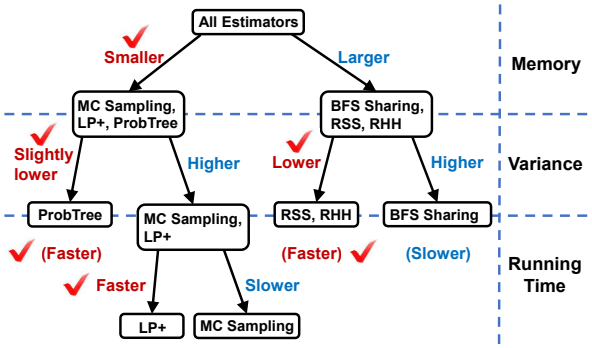


Figure 13: The decision tree for selecting a proper reliability estimator under different scenarios

be used across various datasets and estimators. Usually recursive methods require about 500 less samples than MC-based methods on the same dataset.

For accuracy, all methods have similar relative error ($<$1.5%) at convergence. If K is set as 1000 for all estimators, some of them might not reach convergence, thus their relative errors can further be reduced by using larger K until convergence.

For efficiency, RHH and RSS are the fastest when running time is measured at convergence. When K is set as 1000, there is no common winner in terms of running time. Overall, the running times of RHH, RSS, ProbTree, and LP+ are comparable. BFSSharing is $4\times$ slower than MC, since it estimates all nodes' reliability from the source node.

The memory usage ranking (in increasing order of memory) is: MC $<$ LP+ $<$ ProbTree $<$ BFSSharing $<$ RHH $\approx$ RSS.

Recommendation. Table 13 summarizes the recommendation level of each method according to different performance metrics. The scale is from 1 to 4 stars, and larger star number stands for higher ranking. Clearly, there is no single winner. Considering various trade-offs, in conclusion we recommend ProbTree for s-t reliability estimation. It provides good performance in accuracy, online running time, and memory cost. Its index can slightly reduce the variance, compared to other MC-based estimators. Notably, we adopted MC as ProbTree's reliability estimating component (as the original paper [29] did). However, one can replace this with any other estimator (e.g., recursive estimators: RHH and RSS) to further improve ProbTree's efficiency and to reduce its variance (as demonstrated in Section 3.8).

The decision tree shown in Figure 13 demonstrates our recommended strategy for estimator selection under different constraints. Following the branch with red tick, the better estimator(s) under the current condition can be determined. Notice that the path from the root to the leaf of ProbTree consists of all red ticks, indicating its applicability and trade-off capacity considering various factors.

Acknowledgement. This work was supported by NTU M4081678 and MOE Tier-1 RG83/16. Any opinions, findings, and conclusions in this publication are those of the authors, and do not necessarily reflect the views of the funding agencies.

5. REFERENCES

[1] E. Adar and C. Re. Managing Uncertainty in Social Networks. *IEEE Data Eng. Bull.*, 2007.

[2] C. Aggarwal. *Managing and Mining Uncertain Data*. Springer, 2009.

[3] K. K. Aggarwal, K. B. Misra, and J. S. Gupta. Reliability Evaluation A Comparative Study of Different Techniques. *Micro. Rel.*, 14(1), 1975.

[4] M. O. Ball. Computational Complexity of Network Reliability Analysis: An Overview. *IEEE Tran. Rel.*, 1986.

[5] P. Boldi, F. Bonchi, A. Gionis, and T. Tassa. Injecting Uncertainty in Graphs for Identity Obfuscation. *PVLDB*, 5(11):1376–1387, 2012.

[6] T. B. Brecht and C. J. Colbourn. Lower Bounds on Two-Terminal Network Reliability. *Disc. App. Math.*, 1988.

[7] D. Bulka and J. B. Dugan. Network s-t Reliability Bounds using a 2-Dimensional Reliability Polynomial. *IEEE Tran. Rel.*, 1994.

[8] W. Chen, C. Wang, and Y. Wang. Scalable Influence Maximization for Prevalent Viral Marketing in Large-Scale Social Networks. In *KDD*, 2010.

[9] Y. Cheng, Y. Yuan, L. Chen, and G. Wang. The Reachability Query over Distributed Uncertain Graphs. In *ICDCS*, 2015.

[10] L. Eronen and H. Toivonen. Biomine: Predicting Links between Biological Entities using Network Models of Heterogeneous Databases. *BMC Bioinformatics*, 13(1), 2012.

[11] G. S. Fishman. A Comparison of Four Monte Carlo Methods for Estimating the Probability of s-t Connectedness. *IEEE Tran. Rel.*, 1986.

[12] L. Fu, X. Fu, Z. Xu, Q. Peng, X. Wang, and S. Lu. Determining Source-Destination Connectivity in Uncertain Networks: Modeling and Solutions. *IEEE/ACM Trans. Netw.*, 25(6):3237–3252, 2017.

[13] L. Fu, X. Wang, and P. R. Kumar. Optimal Determination of Source-destination Connectivity in Random Graphs. In *MobiHoc*, 2014.

[14] X. Fu, Z. Xu, Q. Peng, L. Fu, and X. Wang. Complexity vs. optimality: Unraveling Source-Destination Connection in Uncertain Graphs. In *INFOCOM*, 2017.

[15] 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 *INFOCOM*, 2007.

[16] G. Hardy, C. Lucet, and N. Limnios. K-Terminal Network Reliability Measures With Binary Decision Diagrams. *IEEE Tran. Rel.*, 2007.

[17] M. Hua and J. Pei. Probabilistic Path Queries in Road Networks: Traffic Uncertainty aware Path Selection. In *EDBT*, 2010.

[18] R. Jin, L. Liu, B. Ding, and H. Wang. Distance-Constraint Reachability Computation in Uncertain Graphs. *PVLDB*, 4(9):551–562, 2011.

[19] X. Ke, A. Khan, and L. Lim. An In-Depth Comparison of s-t Reliability Algorithms over Uncertain Graphs (Extended Version). `https://arxiv.org/abs/1904.05300`, 2019.

[20] D. Kempe, J. M. Kleinberg, and E. Tardos. Maximizing the Spread of Influence through a Social Network. In *KDD*, 2003.

[21] A. Khan, F. Bonchi, A. Gionis, and F. Gullo. Fast Reliability Search in Uncertain Graphs. In *EDBT*, 2014.

[22] A. Khan, F. Bonchi, F. Gullo, and A. Nufer. Conditional Reliability in Uncertain Graphs. *IEEE TKDE*, 2018.

[23] A. Khan and L. Chen. On Uncertain Graphs Modeling and Queries. *PVLDB*, 8(12):2042–2043, 2015.

[24] A. Khan, Y. Ye, and L. Chen. *On Uncertain Graphs*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2018.

[25] M. Kimura and K. Saito. Tractable Models for Information Diffusion in Social Networks. In *PKDD*, 2006.

[26] R. Li, J. X. Yu, R. Mao, and T. Jin. Recursive Stratified Sampling: A New Framework for Query Evaluation on Uncertain Graphs. *IEEE TKDE*, 28(2):468–482, 2016.

[27] Y. Li, J. Fan, D. Zhang, and K.-L. Tan. Discovering Your Selling Points: Personalized Social Influential Tags Exploration. In *SIGMOD*, 2017.

[28] X. Lin, Y. Peng, B. Choi, and J. Xu. Human-Powered Data Cleaning for Probabilistic Reachability Queries on Uncertain Graphs. *IEEE TKDE*, 29(7):1452–1465, 2017.

[29] S. Maniu, R. Cheng, and P. Senellart. An Indexing Framework for Queries on Probabilistic Graphs. *ACM Trans. Database Syst.*, 42(2):13:1–13:34, 2017.

[30] P. Parchas, F. Gullo, D. Papadias, and F. Bonchi. The Pursuit of a Good Possible World: Extracting Representative Instances of Uncertain Graphs. In *SIGMOD*, 2014.

[31] M. Potamias, F. Bonchi, A. Gionis, and G. Kollios. k-Nearest Neighbors in Uncertain Graphs. *PVLDB*, 3(1-2):997–1008, 2010.

[32] J. S. Provan and M. O. Ball. Computing Network Reliability in Time Polynomial in the Number of Cuts. *Operations Research*, 1984.

[33] A. Sharafat and O. Ma'rouzi. All-Terminal Network Reliability Using Recursive Truncation Algorithm. *IEEE Tran. Rel.*, 2009.

[34] S. Song, Z. Zou, and K. Liu. Triangle-Based Representative Possible Worlds of Uncertain Graphs. In *DASFAA*, 2016.

[35] K. K. T. Akiba, C. Sommer. Shortest-path queries for complex networks: Exploiting low tree-width outside the core. In *EDBT*, 2012.

[36] L. G. Valiant. The Complexity of Enumeration and Reliability Problems. *SIAM J. on Computing*, 1979.

[37] Z. Wang, E. Chen, Q. Liu, Y. Yang, Y. Ge, and B. Chang. Maximizing the Coverage of Information Propagation in Social Networks. In *IJCAI*, 2015.

[38] F. Wei. TEDI: Efficient Shortest path query answering on graphs. In *SIGMOD*, 2010.

[39] Y. Yang, X. Mao, J. Pei, and X. He. Continuous Influence Maximization: What Discounts Should We Offer to Social Network Users? In *SIGMOD*, 2016.

[40] K. Zhu, W. Zhang, G. Zhu, Y. Zhang, and X. Lin. BMC: An Efficient Method to Evaluate Probabilistic Reachability Queries. In *DASFAA*, 2011.

[41] R. Zhu, Z. Zou, and J. Li. Top-k Reliability Search on Uncertain Graphs. In *ICDM*, 2015.

[42] Z. Zou, F. Li, J. Li, and Y. Li. Scalable Processing of Massive Uncertain Graph Data: A Simultaneous Processing Approach. In *ICDE*, 2017.