Distance-Constraint Reachability Computation in Uncertain Graphs [*]

Ruoming Jin[†] Lin Liu[†] Bolin Ding[‡] Haixun Wang[§]

[†] Kent State University
Kent, OH, USA
{jin, lliu}@cs.kent.edu

[‡] UIUC
Urbana, IL, USA
bding3@uiuc.edu

[§] Microsoft Research Asia
Beijing, China
haixunw@microsoft.com

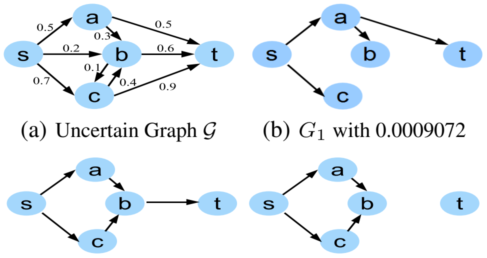
ABSTRACT

Driven by the emerging network applications, querying and mining uncertain graphs has become increasingly important. In this paper, we investigate a fundamental problem concerning uncertain graphs, which we call the *distance-constraint reachability (DCR)* problem: *Given two vertices s and t, what is the probability that the distance from s to t is less than or equal to a user-defined threshold d in the uncertain graph*? Since this problem is #P-Complete, we focus on efficiently and accurately approximating DCR online. Our main results include two new estimators for the probabilistic reachability. One is a *Horvitz-Thomson* type estimator based on the unequal probabilistic sampling scheme, and the other is a novel *recursive sampling* estimator, which effectively combines a deterministic recursive computational procedure with a sampling process to boost the estimation accuracy. Both estimators can produce much smaller variance than the direct sampling estimator, which considers each trial to be either 1 or 0. We also present methods to make these estimators more computationally efficient. The comprehensive experiment evaluation on both real and synthetic datasets demonstrates the efficiency and accuracy of our new estimators.

1. INTRODUCTION

Querying and mining uncertain graphs has become an increasingly important research topic [13, 24, 25]. In the most common uncertain graph model, edges are independent of one another, and each edge is associated with a probability that indicates the likelihood of its existence [13, 24]. This gives rise to using the *possible world* semantics to model uncertain graphs [13, 1]. A possible graph of an uncertain graph $\mathcal{G}$ is a possible *instance* of $\mathcal{G}$. A possible graph contains a subset of edges of $\mathcal{G}$, and it has a weight which is the product of the probabilities of all the edges it has. For example, Figure 1 illustrates an uncertain graph $\mathcal{G}$, and three of its possible graphs G_1, G_2 and G_3, each with a weight.

A fundamental question for uncertain graphs is to categorize and compute reachability between any two vertices. In a deterministic directed graph, the reachability query, which asks whether one vertex can reach another one, is the basis for a variety of databases (XML/RDF) and network applications (e.g., social and biological networks) [8, 22]. For uncertain graphs, reachability is not a simple Yes/No question, but instead, a probabilistic one. Specifically, reachability from vertex s to vertex t is expressed as the overall probability of those possible graphs of $\mathcal{G}$ in which s can reach t. For uncertain graph $\mathcal{G}$ in Figure 1, we can see that s can reach t in its possible graphs G_1 and G_2 but not in G_3; if we enumerate all the possible graphs of $\mathcal{G}$ and add up the weights of those possible graphs where s can reach t, we get s can reach t with probability 0.5104. The simple reachability in uncertain graphs has been widely studied in the context of network reliability and system engineering [5].



(a) Uncertain Graph $\mathcal{G}$ (b) G_1 with 0.0009072

(c) G_2 with 0.0009072 (d) G_3 with 0.0006048

Figure 1: Running Example

In this paper, we investigate a more generalized and informative **distance-constraint reachability (DCR)** query problem, that is: *Given two vertices s and t in an uncertain graph $\mathcal{G}$, what is the probability that the distance from s to t is less than or equal to a user-defined threshold d?* Basically, the distance-constraint reachability (DCR) between two vertices requires them not only to be connected in the possible graphs, but also to be close enough. For the example in Figure 1, if the threshold d is selected to be 2, then, t is considered to be unreachable from 2 in G_2 (under this distance constraint). Clearly, DCR query enables a more informative categorization and interrogation of the reachability between any two vertices. At the same time, the simple reachability also becomes a special case of the distance-constraint reachability (considering the case where the threshold d is larger than the length of the longest path, or simply the sum of all edge weights in $\mathcal{G}$).

Distance-constraint reachability plays an important and even critical role in a wide range of applications. In a variety of real-world emerging communication networks, DCR is essential for analyzing their reliability and communication quality. For instance, in peer-to-peer (P2P) networks, such as Freenet and Gnutella [4, 11], the communication between two nodes is only allowed if they are separated by a small number of intermediate hops (to avoid congestion). In such situation, as the uncertain graph naturally models the link failure probability, the DCR query serves as the basic tool to in-

[*] R. Jin and L. Liu were partially supported by the National Science Foundation, under CAREER grant IIS-0953950.

Proceedings of the VLDB Endowment, Vol. 4, No. 7

terrogate the probability whether one node can communicate with another, and to study the network reliability in general. Indeed, such diameter-constrained (or hop-constrained) reliability has been proposed in the context of communication network reliability [12] though its computation remains difficult.

Recently, there have been efforts to model the road network as an uncertain graph due to the unexpected traffic jam [7]. Here, each link in the road network can be weighted using the distance or the travel time between them. In addition, a probability can be assigned to model the likelihood of a traffic jam. Given this, one of the basic problems is to determine the probability whether the travel distance (or travel time) from one point to another is less than or equal to a threshold considering the uncertainty issue. Clearly, this directly corresponds to a DCR query.

The DCR query can also be applied to trust analysis in social networks. Specifically, in a trust network, one person can trust another with a probabilistic trust score. When two persons are not directly connected, the trust between them can be modeled by their distance (the number of hops between them). In particular, a real world study [17] finds that people tend to trust others if they are connected through some trust relationship; however, the number of hops between them must be small. Thus, given a trust radius, which constrains how far away can a trusted person be, the likelihood that the distance between one person to another is within this radius can be used to evaluate the trust between those non-adjacent pairs. In the uncertain graph terminology, this is a DCR query.

1.1 Problem Statement

Uncertain Graph Model: Consider an *uncertain directed graph* $\mathcal{G} = (V, E, p, w)$, where V is the set of vertices, E is the set of edges, $p : E \rightarrow (0, 1]$ is a function that assigns each edge e a probability that indicates the likelihood of e's existance, and $w : E \rightarrow (0, \infty)$ associates each edge a weight (length). Note that we assume the existence of an edge e is independent of any other edges.

In our example (Figure 1), we assume each edge has unit-length (unit-weight). Let $G = (V_G, E_G)$ be the *possible graph* which is realized by sampling each edge in $\mathcal{G}$ according to the probability $p(e)$ (denoted as $G \sqsubseteq \mathcal{G}$). Clearly, we have $E_G \subseteq E$ and the possible graph G has $\mathbf{Pr}[G]$ *sampling probability*:

$$\mathbf{Pr}[G] = \prod_{e \in E_G} p(e) \prod_{e \in E \setminus E_G} (1 - p(e)).$$

There are a total of 2^m possible graphs (for each edge e, there are two cases: e exists in $\widehat{G}$ or not). In our example (Figure 1), graph $\mathcal{G}$ has 2^9 possible graphs, and as an example for the graph sampling probability, we have

$$\mathbf{Pr}[G_1] = p(s,a)p(a,b)p(a,t)p(s,c)(1-p(s,b))(1-p(b,t)) \times (1-p(s,c))(1-p(b,c))(1-p(c,b)) = 0.0009072$$

Distance-Constraint Reachability: A path from vertex v_0 to vertex v_p in G is a vertex (or edge) sequence $(v_0, v_1, \cdots, v_p)$, such that (v_i, v_{i+1}) is an edge in E_G $(0 \leq i \leq p-1)$. A path is *simple* if no vertex appears more than once in the sequence. To study distance-constraint reachability in uncertain graph, only simple paths need to be considered. Given two vertices s and t in G, a path starting from s and ending at t is referred to as an *s-t-path*. We say vertex t is reachable from vertex s in G if there is an s-t-path in G. The *distance* or *length* of an s-t-path is the sum of the lengths of all the edges on the path. The *distance* from s to t in G, denoted as $dis(s, t|G)$, is the distance or length of the shortest path from s to t, i.e., *minimal length* of all s-t-paths. Given *distance-constraint* d, we say vertex t is *d-reachable* from s if the distance from s to t in G is less than or equal to d.

DEFINITION 1. *(s-t **distance-constraint reachability**)* Computing s-t distance-constraint reachability *in an uncertain graph $\mathcal{G}$ is to compute the probability of the possible graphs G, in which vertex t is d-reachable from s, where d is the distance constraint. Specifically, let*

$$\mathbf{I}^d_{s,t}(G) = \begin{cases} 1, & \textit{if } dis(s,t|G) \leq d \\ 0, & \textit{otherwise} \end{cases}$$

Then, the s-t distance-constraint reachability in uncertain graph $\mathcal{G}$ with respect to parameter d is defined as

$$\mathbf{R}^d_{s,t}(\mathcal{G}) = \sum_{G \sqsubseteq \mathcal{G}} \mathbf{I}^d_{s,t}(G) \cdot \mathbf{Pr}[G]. \quad (1)$$

Note that the problem of computing s-t distance-constraint reachability is a generalization of computing s-t reachability without the distance-constraint, which is often referred to as the *two-point reliability problem* [14]. Simply speaking, it computes *the total sampling probability of possible graphs $G \sqsubseteq \mathcal{G}$, in which vertex t is reachable from vertex s*. Using the aforementioned distance-constraint reachability notation, we may simply choose an upper bound such as $W = \sum_{e \in E} w(e)$ (the total weight of the graph as an example), and then $\mathbf{R}^W_{s,t}(\mathcal{G})$ becomes simple s-t reachability.

Computational Complexity and Estimation Criteria The simple s-t reachability problem is known to be #P-Complete [21, 2], even for special cases, *e.g.*, planar graphs and DAGs, and so is its generalization, s-t distance-constraint reachability. Thus, we cannot expect the existence of a polynomial-time algorithm to find the exact value of $\mathbf{R}^d_{s,t}(G)$ unless P=NP. The distance-constraint reachability problem is much harder than the simple s-t reachability problem as we have to consider the shortest path distance between s and t in all possible graphs. Indeed, the existing s-t reachability computing approaches have mainly focused on the small graphs (in the order of tens of vertices) and cannot be directly extended to our problem (Appendix 5). Given this, the key problem this paper addresses is how to *efficiently and accurately approximate the s-t distance-constraint reachability online*.

Now, let us look at the key criteria for evaluating the quality of an approximate approach (or the quality of an estimator). Let $\widehat{R}$ be a general estimator for $\mathbf{R}^d_{s,t}(\mathcal{G})$. Intuitively, $\widehat{R}$ should be as close to $\mathbf{R}^d_{s,t}(\mathcal{G})$ as possible. Mathematically, this property can be captured by the *mean squared error* (MSE), $E(\widehat{R} - \mathbf{R}^d_{s,t}(\mathcal{G}))^2$, which measures the expected difference between an estimator and the true value. It can also be decomposed into two parts:

$$\begin{aligned} E(\widehat{R} - \mathbf{R}^d_{s,t}(\mathcal{G}))^2 &= Var(\widehat{R}) + (E(\widehat{R}) - \mathbf{R}^d_{s,t}(\mathcal{G}))^2 \\ &= Var(\widehat{R}) + (Bias\widehat{R})^2 \end{aligned}$$

An estimator is *unbiased* if the expectation of the estimator is equal to the true value ($Bias\widehat{R} = 0$), i.e., $E(\widehat{R}) = \mathbf{R}^d_{s,t}(\overline{G})$ (for our problem). The *variance* of estimator $Var(\widehat{R})$ measures the average deviation from its expectation. For an unbiased estimator, the variance is simply the MSE. In other words, the variance of an unbiased estimator is the indicator for measuring its accuracy. In addition, the variance is also frequently used for constructing the confidence interval of an estimate for approximation and the smaller the variance, the more accurate confidence interval estimate we have [18]. All estimators studied in this paper will be proven to be the unbiased estimators of $\mathbf{R}^d_{s,t}(\mathcal{G})$. Thus, the key criterion to discriminate them is their variance [18, 6].

Besides the accuracy of the estimator, the *computational efficiency* of the estimator is also important. This is especially important for online answering s-t distance-constraint reachability query. In sum, in this paper, **our goal is to develop an *unbiased estimator* of $\mathbf{R}^d_{s,t}(\mathcal{G})$ with *minimal variance* and *low computational cost*.**
Minimal DCR Equivalent Subgraph: Before we proceed, we note that given vertices s and t, only subsets of vertices and edges in $\mathcal{G}$ are needed to compute the s-t distance-constraint reachability. Specifically, given vertices s and t, *the minimal equivalent DCR subgraph* $\mathcal{G}_s = (V_s, E_s, p, w) \subseteq \mathcal{G}$ where

$$\begin{aligned} V_s &= \{v \in V | dis(s, v|\overline{G}) + dis(v, t|\overline{G}) \leq d\}, \\ E_s &= \{e = (u, v) \in E | dist(s, u|\overline{G}) + w(e) + dis(v, t|\overline{G}) \leq d\}. \end{aligned}$$

Here, $\overline{G}$ is the **complete possible graph** with respect to $\mathcal{G}$ which includes all the nodes and edges in $\mathcal{G}$. Basically, V_s and E_s contain those vertices and edges that appear on some s-t paths whose distance is less than or equal to d. Clearly, we have $\mathbf{R}^d_{s,t}(\mathcal{G}_s) = \mathbf{R}^d_{s,t}(\mathcal{G})$. A fast linear method can help extract the minimal equivalent DCR subgraph (See Appendix A). Since we only need to work on $\mathcal{G}_s$, in the remainder of the paper, we simply use $\mathcal{G}$ for $\mathcal{G}_s$ when no confusion can arise.

2. BASIC MONTE-CARLO METHODS

In this section, we will introduce two basic Monte-Carlo methods for estimating $\mathbf{R}^d_{s,t}(\mathcal{G})$, the s-t distance-constraint reachability.

2.1 Direct Sampling Approach

A basic approach to approximate the s-t distance-constraint reachability is using sampling: 1) we first sample n possible graphs, $G_1, G_2, \cdots, G_n$ of $\mathcal{G}$ according to edge probability p; and 2) we then compute the shortest path distance in each sample graph G_i, and thus $\mathbf{I}^d_{s,t}(G_i)$. Given this, the basic sampling estimator ($\widehat{\mathbf{R}}_B$) is:

$$\mathbf{R}^d_{s,t}(\mathcal{G}) \approx \widehat{\mathbf{R}}_B = \frac{\sum_{i=1}^n \mathbf{I}^d_{s,t}(G_i)}{n}$$

The basic sampling estimator $\widehat{\mathbf{R}}_B$ is an *unbiased* estimator of the s-t distance-constraint reachability, i.e., $E(\widehat{\mathbf{R}}_B) = \mathbf{R}^d_{s,t}(\mathcal{G})$. Its variance can be simply written as [6]

$$Var(\widehat{\mathbf{R}}_B) = \frac{1}{n}\mathbf{R}^d_{s,t}(\mathcal{G})(1 - \mathbf{R}^d_{s,t}(\mathcal{G})) \approx \frac{1}{n}\widehat{\mathbf{R}}_B(1 - \widehat{\mathbf{R}}_B)$$

The basic sampling method can be rather computationally expensive. Even when we only need to work on the minimal DCR equivalent subgraph $\mathcal{G}_s$, its size can be still large, and in order to generate a possible graph G, we have to toss the coin for each edge in E_s. In addition, in each sampled graph G, we have to invoke the shortest path distance computation to compute $\mathbf{I}^d_{s,t}(G)$, which again is costly.

We may speedup the basic sampling methods by extending the shortest-path distance method, like Dijkstra's or A^* [15] algorithm for sampling estimation. Recall that in both algorithms, when a new vertex v is visited, we have to immediately visit all its neighbors (corresponding to visiting all outgoing edges in v) in order to maintain their corresponding estimated shortest-path distance from the source vertex s. Given this, we may not need to sample all edges at the beginning, but instead, only sample an edge when it will be used in the computational procedure. *Specifically, only when a vertex is just visited, we will sample all its adjacent (outgoing) edges; then, we perform the distance update operations for the end vertices of those sampled edges in the graph; we will stop this process either when the targeted vertex t is reached or when the minimal shortest-distance for any unvisited vertex is more than threshold d.* A similar procedure on Dijkstra's algorithm is applied in [13] for discovering the K nearest neighbors in an uncertain graph.

2.2 Path-Based Approach

In this subsection, we introduce the paths (or cuts) based approach for estimating $\mathbf{R}^d_{s,t}(\mathcal{G})$. To facilitate our discussion, we first formally introduce d-*path* from s to t. A s-t path in G with length less than or equal to distance constraint d is referred to as d-path between s and t. The d-path is closely related to the s-t distance-constraint reachability: *If vertex t is d-reachable from vertex s in a graph G, i.e., $dis(s, t|G) \leq d$, then, there is a d-path between s and t. If not, i.e., $dis(s, t|G) > d$, then, for any s-t path in G, its length is higher than d (there is no d-path).*

Given this, the *complete set of all d-paths* in $\overline{G}$ (the complete possible graph with respect to $\mathcal{G}$ which includes all the edges in $\mathcal{G}$), denoted as $\mathcal{P} = \{P_1, P_2, \cdots, P_L\}$, can be used for computing the s-t distance-constraint reachability:

$$\mathbf{R}^d_{s,t}(\mathcal{G}) = \mathbf{Pr}[P_1 \vee P_2 \cdots \vee P_L] = \sum \mathbf{Pr}[P_i]$$
$$- \sum_{i \neq j} \mathbf{Pr}[P_i \cap P_j] + \cdots + (-1)^L \pi \mathbf{Pr}[P_1 \cap P_2 \cdots \cap P_L]$$

Given this, we can apply the Monte-Carlo algorithm proposed in [9] to estimating $\mathbf{Pr}[P_1 \vee P_2 \cdots \vee P_L]$ within absolute error ϵ with probability at least $1 - \delta$. In sum, the path-based estimation approach contains two steps:
1) *Enumerating all d-paths from s to t in $\overline{G}$ (See Subsection E.1);*
2) *Estimating $\mathbf{Pr}[P_1 \vee P_2 \cdots \vee P_L]$ using the Monte-Carlo algorithm [9].*

This estimator, denoted as $\widehat{\mathbf{R}}_P$, which is an unbiased estimator of $\mathbf{R}^d_{s,t}(\mathcal{G})$ [9], has the following variance as [6]:

$$Var(\widehat{\mathbf{R}}_P) = \frac{1}{n}\mathbf{R}^d_{s,t}(\mathcal{G})(\sum_{i=1}^{L} \mathbf{Pr}[P_i] - \mathbf{R}^d_{s,t}(\mathcal{G}))$$

Thus, depending on whether $\sum_{i=1}^L \mathbf{Pr}[P_i]$ is bigger than or less than 1, the variance of $\widehat{\mathbf{R}}_P$ can be bigger or smaller than that of $\widehat{\mathbf{R}}_B$. The key issue of this approach is the computational requirement to enumerate and store all d-paths between s and t. This can be both computationally and memory expensive (the number of d-paths can be exponential).

Can we derive a faster and more accurate estimator for $\mathbf{R}^d_{s,t}(\mathcal{G})$ than these two estimators, $\widehat{\mathbf{R}}_B$ and $\widehat{\mathbf{R}}_P$? In the next section, we provide a positive answer to this question.

3. NEW SAMPLING ESTIMATORS

In this section, we will introduce new estimators based on *unequal probability sampling* (UPS) and an optimal *recursive sampling estimator*. To achieve that, we will first introduce a divide-and-conquer strategy which serves as the basis of the fast computation of s-t distance constraint reachability (Subsection 3.1).

3.1 A Divide-and-Conquer Exact Algorithm

Computing the exact s-t distance-constraint reachability ($\mathbf{R}^d_{s,t}(\mathcal{G})$) is the basis to fast and accurately approximate it. The naive algorithm to compute $\mathbf{R}^d_{s,t}(\mathcal{G})$ is to enumerate $G \sqsubseteq \mathcal{G}$, and in each G, compute shortest path distance between s and t to test whether $d(s, t|G) \leq d$. The total running time of this algorithm is $\mathbf{O}\left(2^{|E|}(|E| + |V| \log |V|)\right)$ assuming Dijkstra algorithm is used for distance computation[1]. Here, we introduce a much faster exact algorithm to compute $\mathbf{R}^d_{s,t}(\mathcal{G})$. Though this algorithm still has the exponential computational complexity, it significantly reduces our search space by avoiding enumerating 2^m possible graphs of $\mathcal{G}$. The basic idea is to recursively partition all (2^m) possible graphs

[1] Here, $\mathcal{G}$ is actually the minimal DCR equivalent subgraph $\mathcal{G}_s$.

of $\mathcal{G}$ into the groups so that the reachability of these groups can be computed easily. To specify the grouping of possible graphs, we introduce the following notation:

DEFINITION 2. **((E_1, E_2)-*prefix group*)** *The* (E_1, E_2)-prefix group *of possible graphs from uncertain graph $\mathcal{G}$, which is denoted as $\mathcal{G}(E_1, E_2)$, includes all the possible graphs of $\mathcal{G}$ which contains all edges in edge set $E_1 \subseteq E$ and does not contain any edge in edge set $E_2 \subseteq E$, i.e., $\mathcal{G}(E_1, E_2) = \{G \sqsubseteq \mathcal{G} | E_1 \subseteq E_G \wedge E_2 \cap E_G = \emptyset\}$.*
We refer to E_1 and E_2 as the inclusion edge set *and the* exclusion edge set*, respectively.*

Note that for a nonempty prefix group, the inclusion edge set E_1 and the exclusion edge set E_2 are disjoint ($E_1 \cap E_2 = \emptyset$). In Figure 1, if we want to specify those possible graphs which all include edge (s, a) and do not contain edges (s, b) and (b, t), then, we may refer those graphs as $(\{(s, a)\}, \{(s, b), (b, t)\})$-prefix group. To facilitate our discussion, we introduce the *generating probability of the prefix group $\mathcal{G}(E_1, E_2)$* as:

$$\mathbf{Pr}[\mathcal{G}(E_1, E_2)] = \prod_{e \in E_1} p(e) \prod_{e \in E_2} (1 - p(e))$$

This indicates the overall sampling probability of any possible graph in the prefix group.

Given this, *the s-t distance-constraint reachability of a* (E_1, E_2)-*prefix group* is defined as

$$\mathbf{R}^d_{s,t}(\mathcal{G}(E_1, E_2)) = \sum_{G \in (\mathcal{G}(E_1, E_2))} \mathbf{I}^d_{s,t}(G) \cdot \frac{\mathbf{Pr}[G]}{\mathbf{Pr}[\mathcal{G}(E_1, E_2)]} \quad (2)$$

Basically, it is the overall likelihood that t is d-reachable from s *conditional on the fixed prefix* $\mathcal{G}(E_1, E_2)$. It is easily derived that $\mathbf{R}^d_{s,t}(\mathcal{G}) = \mathbf{R}^d_{s,t}(\mathcal{G}(\emptyset, \emptyset))$.

The following lemma characterizes the s-t distance-constraint reachability of (E_1, E_2)-groups and forms the basis for its efficient computation. Its proof is omitted for simplicity.

LEMMA 1. *(**Factorization Lemma**) For any (E_1, E_2)-prefix group of uncertain $\mathcal{G}$ and any uncertain edge $e \in E \backslash (E_1 \cup E_2)$,*

$$\begin{aligned} \mathbf{R}^d_{s,t}(\mathcal{G}(E_1, E_2)) &= p(e)\mathbf{R}^d_{s,t}(\mathcal{G}(E_1 \cup \{e\}, E_2)) \\ &+ (1 - p(e))\mathbf{R}^d_{s,t}(\mathcal{G}(E_1, E_2 \cup \{e\})). \end{aligned}$$

In addition, for any (E_1, E_2)-prefix group of uncertain $\mathcal{G}$, *if E_1 contains a d-path from s to t, then, $\mathbf{R}^d_{s,t}(\mathcal{G}(E_1, E_2)) = 1$; if E_2 contains a d-cut* [2] *between s and t, then, $\mathbf{R}^d_{s,t}(\mathcal{G}(E_1, E_2)) = 0$. Also, E_1 containing a d-path and E_2 containing a d-cut cannot be both true at the same time though both can be false at the same time.*

Algorithm 1 $\mathbf{R}(\mathcal{G}, E_1, E_2)$

```
Parameter: $\mathcal{G}$: Uncertain Graph;
Parameter: $E_1$: Inclusion Edge List;
Parameter: $E_2$: Exclusion Edge List;
1: if $E_1$ contains a $d$-path from $s$ to $t$ then
2:     return 1;
3: else if $E_2$ contains a $d$-cut from $s$ to $t$ then
4:     return 0;
5: end if
6: select an edge $e \in E \backslash (E_1 \cup E_2)$ {Find a remaining uncertain edge}
7: return $p(e)\mathbf{R}(\mathcal{G}, E_1 \cup \{e\}, E_2) + (1 - p(e))\mathbf{R}(\mathcal{G}, E_1, E_2 \cup \{e\})$
```

Algorithm 1 describes the divide-and-conquer computation procedure for $\mathbf{R}^d_{s,t}(\mathcal{G})$ based on Lemmas 1. To compute $\mathbf{R}^d_{s,t}(\mathcal{G})$, we will invoke the procedure $\mathbf{R}(\mathcal{G}, \emptyset, \emptyset)$. Based on the factorization lemma (Lemma 1), this procedure first partitions the entire set of possible graphs of uncertain graph $\mathcal{G}$ into two parts (prefix groups) using any edge e in $\mathcal{G}$:

$$\mathbf{R}^d_{s,t}(\mathcal{G}(\emptyset, \emptyset)) = p(e)\mathbf{R}^d_{s,t}(\mathcal{G}(\{e\}, \emptyset)) + (1 - p(e))\mathbf{R}^d_{s,t}(\mathcal{G}(\emptyset, \{e\})).$$

Then, it applies the same approach to partition each prefix group recursively (Line $6 - 7$) until either E_1 contains a d-path or E_2 contains a d-cut (Line $1 - 5$) in the prefix group $\mathcal{G}(E_1, E_2)$.

The computational process of the recursive procedure $\mathbf{R}$ can be represented in a *full binary enumeration tree* (Figure 2 (a)). In the tree, each node corresponds to a prefix group $\mathcal{G}(E_1, E_2)$ (also an invoke of the procedure $\mathbf{R}$). Each internal node has two children, one corresponding to including an uncertain edge e, another excluding it. In other words, the prefix group is partitioned into two new prefix groups: $\mathcal{G}(E_1 \cup \{e\}, E_2)$ and $\mathcal{G}(E_1, E_2 \cup \{e\})$. Further, we may consider each edge in the tree is weighted with probability $p(e)$ for edge inclusion and $1 - p(e)$ for edge exclusion. In addition, the leaf node can be classified into two categories, $\mathcal{L}$ which contains all the leaf nodes with E_1 containing a d-path, and $\overline{\mathcal{L}}$ which contains the remaining leaf nodes, i.e., all those leaf nodes with E_2 include a d-cut.

Note that any uncertain edge e can be selected for each prefix group (in Line 6) without affecting the correctness of the recursive procedure. However, it does affect its computational complexity, which is determined by average recursive depth (average prefix-length), i.e., the average number of edges $|E_1 \cup E_2|$ we have to select in order to determine whether t is d-reachable from s for all the possible graphs in the prefix group. If the average recursive depth is a, then, a total of $O(2^a)$ prefix groups need to be enumerated, which can be significantly smaller than the complete $O(2^m)$ possible graphs of $\mathcal{G}$. An uncertain edge selection approach in Section E is provided to minimize the average recursive depth.

3.2 Unequal Probability Sampling Framework

Now, we study an estimation framework of $\mathbf{R}^d_{s,t}(\mathcal{G})$ using the *unequal probability sampling* scheme [18] based on Algorithm 1.
Unequal Probability Sampling (UPS) Framework: To estimate $\mathbf{R}^d_{s,t}(G)$, we apply the *unequal sampling scheme*: 1) each leaf node in the enumeration tree (Figure 2 (a)) is associated with a **leaf weight**: *the generating probability of the corresponding prefix group*, $\mathbf{Pr}[\mathcal{G}(E_1, E_2)]$; and 2) each leaf node $\mathcal{G}(E_1, E_2)$ in the enumeration tree is sampled with a **leaf sampling probability** $q(\mathcal{G}(E_1, E_2))$, where the sum of all leaf sampling probability ($q(\mathcal{G}(E_1, E_2))$ is 1. Note that in the UPS framework, *the* **leaf sampling probability q** *can be different from the* **leaf weight**.

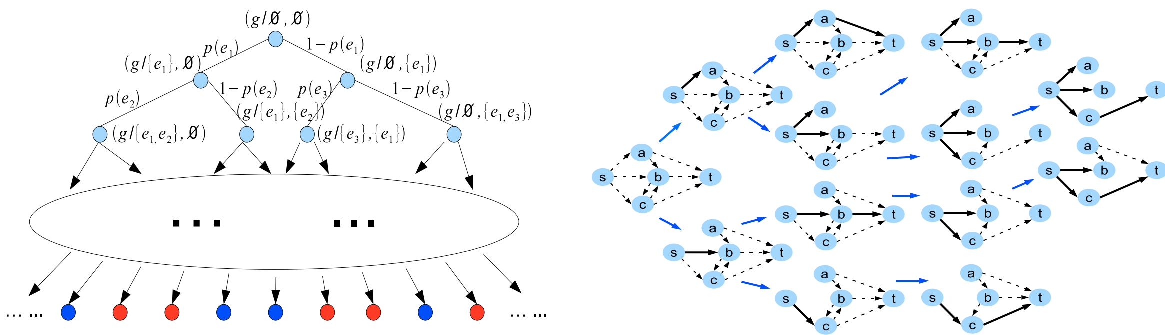
Given this, we now study the well-known unequal sampling estimator, the **Hansen-Hurwitz estimator** [18]: assuming we sampled n leaf nodes, $1, 2, \cdots, n$, in the enumeration tree, and let Pr_i be the weight associated with the i-th sampled leaf node and let q_i be the leaf sampling probability, then the Hansen-Hurwitz estimator (denoted as $\widehat{\mathbf{R}}_{HH}$) for $\mathbf{R}^d_{s,t}(\mathcal{G})$ is:

$$\widehat{\mathbf{R}}_{HH} = \frac{1}{n} \sum_{i=1}^{n} \frac{Pr_i \mathbf{I}^d_{s,t}(\mathcal{G})}{q_i} \quad (3)$$

In other words, we may consider each leaf node in $\mathcal{L}$ contributes Pr_i and each leaf node in $\overline{\mathcal{L}}$ contributes 0 to the estimation. It is easy to show the Hansen-Hurwitz estimator ($\widehat{\mathbf{R}}_{HH}$) is an *unbiased* estimator for $\mathbf{R}^d_{s,t}(\mathcal{G})$, and its variance can be derived as

$$Var(\widehat{\mathbf{R}}_{HH}) = \frac{1}{n}(\sum_{i \in \mathcal{L}} q_i (\frac{Pr_i}{q_i} - \mathbf{R}^d_{s,t}(\mathcal{G}))^2 + \sum_{i \in \overline{\mathcal{L}}} q_i \mathbf{R}^d_{s,t}(\mathcal{G})^2)$$

[2] An edge set C_d of $\mathcal{G}$ is a d-cut between s and t if $\overline{G} \backslash C_d$ has a distance greater than d, i.e., $dis(s, t|\overline{G} \backslash C_d) > d$.



(a) Enumeration Tree of Recursive Computation of $\mathbf{R}^d_{s,t}(\mathcal{G})$

(b) Divide and Conquer

Figure 2: Divide-and-Conquer method

Applying the Lagrange method, we can easily find that *the optimal sampling probability for minimal variance $Var(\widehat{\mathbf{R}}_{HH})$ is achieved when $q_i = Pr_i$, and the minimal variance is $Var(\widehat{\mathbf{R}}_{HH}) = \frac{1}{n}\mathbf{R}^d_{s,t}(\mathcal{G})(1-\mathbf{R}^d_{s,t}(\mathcal{G}))$.* In other words, *the best leaf sampling probability $\mathbf{q}$ to minimize the variance of $\widehat{\mathbf{R}}_{HH}$ is the one equal to the leaf weight in $\mathcal{L}$*! Note that this is consistent with the general UPS theory which suggests the sampling probability should be proportional to the corresponding sampling weight [18].

Given this, we can sample a leaf node in the enumeration tree as follows: *Simply tossing a coin at each internal node in the enumeration tree to determine whether the uncertain edge e (also corresponding to Line 6 in Algorithm 1) should be included (in E_1) with probability $q(e) = p(e)$ or excluded (in E_2) with probability $1-p(e)$; continuing this process until a leaf node is reached.* Basically, we perform a **random walk** starting from the root node and stopping at the leaf node in the enumeration tree (Figure 2 (a)), and at each internal node, we randomly goes to its left (selecting the edge e) with probability $p(e)$ or goes to its right (excluding edge e) with probability $1-p(e)$. Such random walk sampling can guarantee that *the leaf sampling probability is equal to the corresponding leaf weight*! Due to space limitation, the pseudocode of the random walk sampling scheme for $\widehat{\mathbf{R}}_{HH}$ is sketched in Appendix B.

Interestingly, we note this UPS estimator is equivalent to the direct sampling estimator, as each leaf node is counted as either 1 or 0 (like Bernoulli trial): $\widehat{\mathbf{R}}_{HH} = \widehat{\mathbf{R}}_B$. In other words, the direct sampling scheme is simply a special (and optimal) case of the Hassen-Hurvitz estimator! This leads to the following observation: *for any optimal Hassen-Hurvitz estimator ($\widehat{\mathbf{R}}_{HH}$) or direct sampling estimator ($\widehat{\mathbf{R}}_B$), their variance is only determined by n and has no relationship to the enumeration tree size.* This seems to be rather counter-intuitive as the smaller the tree-size (or the smaller number of the leaf nodes), the better chance (information) we have for estimating $\mathbf{R}^d_{s,t}(\mathcal{G})$.

A Better UPS Estimator: Now, we introduce another UPS estimator, the **Horvitz-Thomson** estimator ($\widehat{\mathbf{R}}_{HT}$), which can provide smaller variance than the Hansen-Hurvitz estimator $\widehat{\mathbf{R}}_{HH}$ and the direct sampling estimator $\widehat{\mathbf{R}}_B$ under mild conditions. Assuming we sampled n leaf nodes in the enumeration tree and among them there are l *distinctive* ones $1, 2, \cdots, l$ (l is also referred to as the effective sample size), let the **leaf inclusion probability** π_i be probability to include leaf i in the sample, which is define as $\pi_i = 1-(1-q_i)^n$ where q_i is the leaf sampling probability. The Horvitz-Thomson estimator for $\mathbf{R}^d_{s,t}(\mathcal{G})$ is:

$$\hat{\mathbf{R}}_{HT} = \sum_{i=1}^{l} \frac{Pr_i \mathbf{I}^d_{s,t}(\mathcal{G})}{\pi_i}.$$

Note that if q_i is very small, then $\pi_i \approx nq_i$. The Horvitz-Thomson estimator ($\widehat{\mathbf{R}}_{HT}$) is an *unbiased* estimator for the population total ($\mathbf{R}^d_{s,t}(\mathcal{G})$). Its variance can be derived as follows [18], where π_{ij} is the probability that both leafs i and j are included in the sample: $\pi_{ij} = 1-(1-q_i)^n - (1-q_j)^n + (1-q_i-q_j)^n$:

$$\begin{aligned} Var(\widehat{\mathbf{R}}_{HT}) &= \sum_{i\in\mathcal{L}} \left(\frac{1-\pi_i}{\pi_i}\right) Pr_i^2 \\ &+ \sum_{i,j\in\mathcal{L}, i\neq j} \left(\frac{\pi_{ij}-\pi_i\pi_j}{\pi_i\pi_j}\right) Pr_i Pr_j. \end{aligned}$$

Using Taylor expansions and Lagrange method, we can find *the minimal variance can be approximated when $q_i = Pr_i$*. This basically suggests the similar leaf sampling strategy (the random walk from the root to the leaf) for the Hansen-Hurwitz estimator can be applied to the Horvitz-Thomson estimator as well. However, different from the Hansen-Hurwitz estimator, the Horvitz-Thomson estimator utilizes each distinctive leaf once. Though in general the variances between the Hansen-Hurwitz estimator and the Horvitz-Thomson estimator are not analytically comparable, in our tree-based sampling framework and under reasonable approximation, we are able to prove the latter one has smaller variance.

THEOREM 1. ($Var(\widehat{\mathbf{R}}_{HT}) \leq Var(\widehat{\mathbf{R}}_{HH})$) *When for any sample leaf node i, $nPr_i \ll 1$, $Var(\widehat{\mathbf{R}}_{HH}) - Var(\widehat{\mathbf{R}}_{HT}) = \Omega(\sum_{i\in\mathcal{L}} Pr_i^2)$.*

The proof of this theorem can be found in Appendix. This result suggests that for small sample size n and/or when the generating probability of the leaf node is very small, then the Horvitz-Thomson estimator is guaranteed to have smaller variance. In Section 4, the experimental results will further demonstrate the effectiveness of this estimator. A reason for this estimator to be effective is that it directly works on the distinctive leaf nodes which partly reflect the tree structure. In the next subsection, we will introduce a novel recursive estimator which more aggressively utilizes the tree structure to minimize the variance.

3.3 Optimal Recursive Sampling Estimator

In this subsection, we explore how to reduce the variance based on the factorization lemma (Lemma 1). Then, we will describe a novel recursive approximation procedure which combines the deterministic procedure with the sampling process to minimize the estimator variance.

Variance Reduction: Recall that for the root node in the enumeration tree, we have the following results based on the the factorization lemma (Lemma 1):

$$\mathbf{R}^d_{s,t}(\mathcal{G}) = p(e)\mathbf{R}^d_{s,t}(G(\{e\},\emptyset)) + (1-p(e))\mathbf{R}^d_{s,t}(G(\emptyset,\{e\}))$$

To facilitate our discussion, let $\tau = \mathbf{R}^d_{s,t}(\mathcal{G})$, $\tau_1 = \mathbf{R}^d_{s,t}(\mathcal{G}(\{e\},\emptyset))$ and $\tau_2 = \mathbf{R}^d_{s,t}(\mathcal{G}(\emptyset,\{e\}))$.

Now, instead of directly sampling all the leaf nodes from the root (like suggested in last subsection), we consider to estimate both τ_1

and τ_2 independently, and then combine them together to estimate τ. Specifically, for n total leaf samples, we deterministically allocate n_1 of them to the left subtree (including edge e, τ_1), and n_2 of them to the right subtree (excluding edge e, τ_2); then, we can apply the aforementioned sampling estimators, such as $\widehat{\mathbf{R}}_{HH}$, or equivalently $\widehat{\mathbf{R}}_B$), to both subtrees. Let $\widehat{\mathbf{R}}_1$ and $\widehat{\mathbf{R}}_2$ be the estimators for τ_1 (left subtree) and τ_2 (right subtree), respectively. Thus, the combined estimator for $\mathbf{R}^d_{s,t}(\mathcal{G})$ is

$$\widehat{\mathbf{R}} = p(e)\widehat{\mathbf{R}}_1 + (1-p(e))\widehat{\mathbf{R}}_2 \qquad (4)$$

Clearly, this combined estimator is *unbiased* as both $\widehat{\mathbf{R}}_1$ and $\widehat{\mathbf{R}}_2$ are *unbiased* estimators for τ_1 and τ_2, respectively. Why this might be a better way to estimate $\mathbf{R}^d_{s,t}(\mathcal{G})$? Intuitively, this is because we eliminate the "uncertainty" of edge e from the estimation equation. Of course, the important question is how such elimination can benefit us, and to answer this, we need address this problem: *what are the optimal sample allocation strategy to minimize the overall estimator variance?*

The variance of the combined estimator depends on the variance of the two individual estimators (they are independent and their covariance is 0):

$$\begin{aligned} Var(\widehat{\mathbf{R}}) &= p(e)^2 Var(\widehat{\mathbf{R}}_1) + (1-p(e))^2 Var(\widehat{\mathbf{R}}_2) \\ &= p(e)^2 \frac{\tau_1(1-\tau_1)}{n_1} + (1-p(e))^2 \frac{\tau_2(1-\tau_2)}{n_2} \end{aligned}$$

When τ_1 and τ_2 are known, we clearly can find the optimal sample allocation (n_1 and n_2 are functions of τ_1 and τ_2) for minimizing $Var(\widehat{\mathbf{R}})$. However, in this problem, such prior knowledge is clearly unavailable. Given this, *can we still allocate samples to reduce the variance?* An interesting discovery we made is when the sample size allocation is proportional to the edge inclusion probability, i.e., $n_1 = p(e)n$ and $n_2 = (1-p(e))n$, the variance of the original optimal Hassen-Hurvitz estimator $Var(\widehat{\mathbf{R}}_{HH}) = \frac{\tau(1-\tau)}{n}$ can be reduced!

THEOREM 2. **(Variance Reduction)** *When, $n_1 = p(e)n$ and $n_2 = (1-p(e))n$, $Var(\widehat{\mathbf{R}}) \leq Var(\widehat{\mathbf{R}}_{HH})$, and more specifically, the variance is reduced by*

$$Var(\widehat{\mathbf{R}}_{HH}) - Var(\widehat{\mathbf{R}}) = \frac{p(e)(1-p(e))(\tau_1-\tau_2)^2}{n}.$$

For its simplicity, we omit the proof for the above theorem.

Recall τ_1 is the overall probability of those leaf nodes in the left subtree ($G(\{e\}, \emptyset)$) and in $\mathcal{L}$, i.e., when edge e is included and t is d-reachable from s and τ_2 is the overall probability of those possible graphs where e is excluded. Clearly, when edge e is included, the probability for t is d-reachable from s is greater. Especially, this theorem suggests the bigger the impact for edge e being included or excluded, the greater the variance reduction effect (directly proportional to $(\tau_1-\tau_2)^2$). In addition, this sample size allocation method can be generalized and applied at the root node of any subtree in the enumeration tree for reducing the variance.

Recursive Sampling Estimator: Given this, we introduce our *recursive sampling estimator* $\widehat{\mathbf{R}}_R$, which is outlined in Algorithm 2. Basically, it follows the exact computational recursive procedure (Algorithm 1) and recursively split the sample size n to $\lfloor np(e) \rfloor$ and $n - \lfloor np(e) \rfloor$ for estimating $\mathbf{R}^d_{s,t}(\mathcal{G}(E_1 \cup \{e\}, E_2))$ and $\mathbf{R}^d_{s,t}(\mathcal{G}(E_1, E_2 \cup \{e\}))$, respectively (Line 10). In addition, when the sample size n is smaller than the threshold (typically the threshold is very small, less than 5), we can avoid the recursive allocation by perform the direct sampling (Line 1 and 2). Note that when the sample size is very small, all the non-recursive sampling estimators, including $\widehat{\mathbf{R}}_{HT}$, $\widehat{\mathbf{R}}_{HH}$, and $\widehat{\mathbf{R}}_B$, all become equivalent.

Algorithm 2 OptEstR($\mathcal{G}, E_1, E_2, n$)

```
Parameter: $E_1$: Inclusion Edge List;
Parameter: $E_2$: Exclusion Edge List;
Parameter: $n$: sample size;
 1: if $n \leq threshold$ {Stop recursive sample allocation} then
 2:     return $\widehat{\mathbf{R}}_{HH}(\mathcal{G}, E_1, E_2, n)$; {apply non-recursive sampling estimator}
 3: end if
 4: if $E_1$ contains a $d$-path from $s$ to $t$ then
 5:     return 1;
 6: else if $E_2$ contains a $d$-cut from $s$ to $t$ then
 7:     return 0;
 8: end if
 9: select an edge $e \in E \backslash (E_1 \cup E_2)$ {Find a remaining uncertain edge}
10: return $p(e)$OptEstR$(\mathcal{G}, E_1 \cup \{e\}, E_2, \lfloor np(e) \rfloor)$+
        $(1-p(e))$OptEstR$(\mathcal{G}, E_1, E_2 \cup \{e\}, n - \lfloor np(e) \rfloor)$;
```

Table 1: Relative Error (in %)

	$\widehat{\mathbf{R}}_B$	$\widehat{\mathbf{R}}^D_B$	$\widehat{\mathbf{R}}_P$	$\widehat{\mathbf{R}}_{HT}$	$\widehat{\mathbf{R}}_{RHH}$	$\widehat{\mathbf{R}}_{RHT}$
15-25	3.42	3.00	0.98	0.90	0.96	0.71
26-35	2.52	2.80	1.50	1.08	0.90	0.72
36-45	2.30	1.75	1.17	1.77	1.36	1.33
46-55	1.79	1.42	1.59	1.39	1.33	1.30

Table 2: Relative Variance Efficiency

	$\widehat{\mathbf{R}}_B$	$\widehat{\mathbf{R}}^D_B$	$\widehat{\mathbf{R}}_P$	$\widehat{\mathbf{R}}_{HT}$	$\widehat{\mathbf{R}}_{RHH}$	$\widehat{\mathbf{R}}_{RHT}$
15-25	1.00	0.81	0.15	0.12	0.12	0.08
26-35	1.00	0.77	0.44	0.23	0.27	0.17
35-45	1.00	0.58	0.58	0.45	0.23	0.20
45-55	1.00	0.73	0.82	0.80	0.44	0.43

Table 3: Query Time (in ms)

	$\mathbf{R}^*$	$\widehat{\mathbf{R}}_B$	$\widehat{\mathbf{R}}^D_B$	$\widehat{\mathbf{R}}_P$	$\widehat{\mathbf{R}}_{HT}$	$\widehat{\mathbf{R}}_{RHH}$	$\widehat{\mathbf{R}}_{RHT}$
15-25	2	314	314	532	193	11	15
26-35	53	358	345	564	233	20	34
35-45	1828	314	313	535	234	23	42
45-55	91748	344	345	565	251	23	45

The computational complexity of this recursive sampling estimator is $O(na)$, where a is the average recursive depth or the average length from the root node to the leaf node in the enumeration tree. But it tends to be more computationally efficient than the UPS estimators $\widehat{\mathbf{R}}_{HH}$ and $\widehat{\mathbf{R}}_{HT}$. This is because the recursively sampling estimator visits the upper-part of the enumeration tree (for the recursive sample size allocation) only once and does not need perform any coin-toss for the each node at this part of the tree. However, the UPS estimators have to perform coin-toss for each node and may repetitively revisit the same node in the upper-level of the tree, where this new estimator visits each node only once. Finally, we note that the analytically comparison between the variance of the Horvitz-Thomson estimator $\widehat{\mathbf{R}}_{HT}$ and the recursively estimator $\widehat{\mathbf{R}}$ is not conclusive (See further analysis in Appendix D), though the experimental evaluation demonstrates the superiority of the new sampling estimator (Section 4).

4. EXPERIMENTAL EVALUATION

In the experimental study, we will focus on studying the accuracy and computational efficiency of different sampling estimators on both synthetic and real datasets. Specifically, the sampling estimators include: 1) $\widehat{\mathbf{R}}_B$: this is the direct sampling estimator using A^* algorithm for searching shortest path distance [15]. The sampling process is also combined with the search process to maximize its computational efficiency; 2) $\widehat{\mathbf{R}}^D_B$: we apply a state-of-the-art variance reduction method, *Dagger Sampling* [10, 6], on top of $\widehat{\mathbf{R}}_B$ to boost the estimation accuracy; 3) $\widehat{\mathbf{R}}_P$: this is the path-based estimator based on [9] which needs to enumerate all the d-paths from s to t; 4) $\widehat{\mathbf{R}}_{HT}$: this is the Horvitz-Thomson estimator based on the unequal probabilistic sampling framework; 5) $\widehat{\mathbf{R}}_{RHH}$: this is the optimal recursive sampling estimator *OptEstR* and when the number of samples in the recursive sampling process is less than the *threshold* (set to be 5), the non-recursive sampling estimator $\widehat{\mathbf{R}}_{HH}$ (Hansen-Hurwitz estimator) is used; 6) $\widehat{\mathbf{R}}_{RHT}$: this is the optimal recursive sampling estimator *OptEstR* and when the number of samples in the recursive sampling process is less than the *threshold* (5), the non-recursive sampling estimator $\widehat{\mathbf{R}}_{HT}$ (Horvitz-Thomson estimator) is used. For all the last three estimators, they all utilize the *FindDPath* procedure to select the next edge in the recursive computation procedure. In addition, we omit the results for $\widehat{\mathbf{R}}_{HH}$ (Hansen-Hurwitz estimator) because it is equivalent to the direct sampling estimator $\widehat{\mathbf{R}}_B$.

To compare the accuracy of these different sampling estimators, we utilize two criteria: the *relative error* and the *estimation variance*. For the relative error, we apply $\mathbf{R}^*$ procedure to compute the exact distance-constraint reachability. Let R be the exact result and $\widehat{R}$ be the estimation result. Then, the relative error ϵ is computed as $\epsilon = \frac{|\widehat{R}-R|}{R}$. For the estimation variance, for each query, we will run each estimator K times, and thus, we have 100 different estimating results: $\widehat{R}_1$,$\widehat{R}_2$, $\cdots$, $\widehat{R}_K$ (in this work, we set $K = 100$). The estimation variance σ is estimated as: $\sigma = \frac{\sum_{i=1}^{K}(\widehat{R}_i - \overline{R})^2}{K-1}$. Here $\overline{R}$ is the estimation average $(\sum_{i=1}^{K} \widehat{R}_i)/K$. The computational efficiency is evaluated by the running time of each estimator.

All algorithms are implemented by using C++ and the Standard Template Libaray (STL) and were conducted on a 2.0GHz Dual Core AMD Opteron CUP with 4.0GB RAM running Linux.

4.1 Experimental Results on Synthetic Datasets

We first report the experimental results on synthetic uncertain graphs. Here, the graph topologies are generated by either Erdös-Rényi random graph model or power law graph generator [3]. The edge weight is randomly generated between 1 to 100 according to uniform distribution. The edge probability is randomly generated between 0 to 1 according to uniform distribution.

Random Graph: In this experiment, we generate an Erdös-Rényi random graph with 5000 vertices and edge density 10. We report the relative error, estimation variance, and the query time with respect to the edge number of *minimal DCR equivalent subgraph size* $\mathcal{G}_s$. Recall $\mathcal{G}_s$ is the uncertain subgraph which will be used for the sampling estimator. We partition the queries into four groups $15-25, 26-35, 36-45$ and $46-55$. This is because for any graph with edge number no larger than 15, the exact computation can be done very efficiently; and when the graph size is larger than 55, it becomes too expensive to compute the exact distance-constraint reachability. Since in this experiment, we would like to report the relative error, we limit ourselves to the smaller $\mathcal{G}_s$. For each of the four groups, we generate 1000 random queries. In addition, the sample size is set to be 1000 for each estimator.

Table 1 shows the relative errors of six different estimators. Overall, the two recursive estimators $\widehat{R}_{RHH}$ and $\widehat{R}_{RHT}$ are the clear winners and $\widehat{R}_{RHT}$ is slightly better than $\widehat{R}_{RHH}$. They can cut the relative error of the direct sampling estimator $\widehat{R}_B$ by more than half. The Dagger sampling method can only reduce the relative error of $\widehat{R}_B$ by less than 10%. The path-based sampling estimator $\widehat{R}_P$ and $\widehat{R}_{HT}$ are comparable though the latter is slightly better. They can reduce the relative error of the direct sampling estimator by around 45%. However, as we will see the path-based sampling is much more computationally expensive as it has to enumerate all the d-paths from the source vertex s to the destination vertex t.

Table 2 reports the *relative variance efficiency* of different approaches using the variance of $\sigma_{\widehat{R}_B}$ as the baseline. In the second column under $\widehat{R}_B$, we have the value to be 1, and the second column under $\widehat{R}_B^D$, the values are $\sigma_{\widehat{R}_B^D}/\sigma_{\widehat{R}_B}$. The relative variance efficiency is consistent with the results on relative error. The Dagger sampling estimator $\widehat{R}_B^D$, the path-based estimator $\widehat{R}_P$, the Horvitz-Thomson estimator $\widehat{\mathbf{R}}_{HT}$ on average has only 72%, 50% and 40% of the baseline variance; the recursive sampling operators $\widehat{\mathbf{R}}_{RHH}$ and $\widehat{\mathbf{R}}_{RHT}$ reduces the variance by almost 5 times (with 26% and 22% of the baseline variance)!

Table 4: Relative Error with Real Graphs (in %)

	$\widehat{\mathbf{R}}_B$	$\widehat{\mathbf{R}}_B^D$	$\widehat{\mathbf{R}}_P$	$\widehat{\mathbf{R}}_{HT}$	$\widehat{\mathbf{R}}_{RHH}$	$\widehat{\mathbf{R}}_{RHT}$
DBLP	4.40	3.23	1.66	1.71	1.94	1.74
Yeast PPI	3.85	3.47	1.36	2.22	2.21	1.73
Fly PPI	3.62	3.22	1.40	1.92	2.08	1.64

Table 5: Query Time with Real Graphs (in Seconds)

	$\widehat{\mathbf{R}}_B$	$\widehat{\mathbf{R}}_B^D$	$\widehat{\mathbf{R}}_P$	$\widehat{\mathbf{R}}_{HT}$	$\widehat{\mathbf{R}}_{RHH}$	$\widehat{\mathbf{R}}_{RHT}$
DBLP	51.65	55.50	5915.12	26.08	5.93	10.08
Yeast PPI	1.15	1.97	4959.37	0.50	0.11	0.21
Fly PPI	2.55	4.77	215.98	1.13	0.45	0.67

Table 3 shows the computational time of different sampling operators. First, we can see that when the extracted subgraph $\mathcal{G}_s$ is fairly small (less than 35 edges), the exact recursive algorithm $\mathbf{R}^*$ is quite fast (even faster than most of the sampling approach). However, when the subgraph grows, the exact computational cost grows exponentially. Second, the path-based method is the slowest one as we expected (it is on average 1.65 times slower than the direct sampling approach with A^* search); and the unequal sampling estimator $\widehat{\mathbf{R}}_{HT}$ is around 1.5 times faster than the direct sampling estimator. Finally, very impressively, the two recursively estimators are much faster than other estimators: especially, $\widehat{\mathbf{R}}_{RHH}$ is on an average 20 times faster than the direct sampling estimator and $\widehat{\mathbf{R}}_{RHT}$ is around 10 times faster!

Note that additional experimental results on how sample size affect the estimation accuracy and performance, and on the scalability of different estimators are reported in Appendix F.

4.2 Experimental Results on Real Data

We study different estimators on three real uncertain graphs: DBLP and two Protein-Protein Interaction (PPI) networks. The DBLP is a coauthor graph with $226,000$ vertices and $1,400,000$ edges (provided by authors in [13]). The Yeast PPI network has almost 5499 vertices and 63796 edges and Fly PPI network 7518 vertices and 51660 edges. They are constructed by combining data from BioGrid [20] and MIPS [19]. In this experiment, we ran 1000 random queries with sample size 1000. Table 4 and 5 report the relative error and the running time of the different approaches, respectively. In order to report the relative error, we limit the extracted subgraphs with number of edges from 20 to 50. In Table 4, we can see that the $\widehat{\mathbf{R}}_P$, $\widehat{\mathbf{R}}_{HT}$, $\widehat{\mathbf{R}}_{RHH}$ and $\widehat{\mathbf{R}}_{RHT}$ reduced the relative errors by half of the $\widehat{\mathbf{R}}_B$ and $\widehat{\mathbf{R}}_B^D$ estimators; and $\widehat{\mathbf{R}}_P$ is slightly better than our methods. However, as shown in Table 5, the query time of $\widehat{\mathbf{R}}_P$ is several hundred times slower than the $\widehat{\mathbf{R}}_{HT}$ and $\widehat{\mathbf{R}}_{RHT}$ and even 1000 times slower than $\widehat{\mathbf{R}}_{RHH}$ estimator.

5. RELATED WORK

Managing and mining uncertain graphs has recently attracted much attention in the database and data mining research community [13, 23, 24, 25]. Especially, Potamias *et. al.* recently studied the k-Nearest Neighbors in uncertain graphs [13]. They provide a list of alternative shortest-path distance measures in the uncertain graph in order to discover the k closest vertices to a given vertex. They also combine sampling with Dijkstra's single source shortest-path distance algorithm for estimation. The estimator used in [13] is based on direct sampling.

Our work on distance-constraint reachability query is a generalization of the *two point reliability* problem, or the simple s-t reachability problem [14]. There has been an extensive study on computing the two points reliability exactly and no known exact methods can handle networks with one hundred vertices except for certain special topologies [14]. Note that the recursive method proposed in this paper can be considered as a generalization of [16] for the *two point reliability* problem. Monte-Carlo methods have been studied to estimate the two point reliability on large graphs [6]. From the users' point of view, the quality of a Monte-Carlo method is measured by both its computational efficiency and its accuracy (estimator variance). The basic method is based on directly sampling, just like the $\widehat{\mathbf{R}}_B$ estimator used in this paper. Since its variance is quite high, researchers have developed methods in trying to improve its accuracy. However, most of the methods for variance reduction need the per-computation of path or cut sets [6, 9], which clearly are too expensive for online query. The $\widehat{R}_P$ estimator is an extension of this type of efforts. Though the methods proposed in this paper even target on the general distance-constraint reachability, they can be applied to the simple reachability. To the best of our knowledge, the Horvitz-Thomson estimator $\widehat{\mathbf{R}}_{HT}$ and the recursive estimator $\widehat{\mathbf{R}}_R$ have not been studied or discovered for the simple reachability.

Finally, we note that the approaches developed in this paper can be applied to answer reachability problems with other types of constraints. For instance, in [23], the authors studied to discover the shortest paths in uncertain graph with the condition that each such shortest path has probability no less than certain threshold. Inspired by this, in Appendix G, we describe a simple extension of DCR (Distance-Constraint Reachability) query and discuss how our approaches can be applied to the new problem.

6. CONCLUSIONS

In this paper, we study a novel s-t distance-constraint reachability problem in uncertain graphs. We not only develop an efficient exact computation algorithm, but also present different sampling methods to approximate the reachability. Specifically, we introduce a unified *unequal probabilistic sampling estimation* framework and a novel Monte-Carlo method which effectively combines the deterministic recursive computational procedure and sampling process. Both can significantly reduce the estimation variance. Especially, the recursive sampling estimator is accurate and computationally efficient! It can on average reduce both variance and running time by an order of magnitude comparing with the direct sampling estimators. In the future work, we would like to investigate how the estimation method can be applied into other graph mining and query problem in uncertain graphs.

7. REFERENCES

[1] Charu C. Aggarwal, editor. *Managing and Mining Uncertain Data*. Advances in Database Systems. Springer, 2009.

[2] Michael O. Ball. Computational complexity of network reliability analysis: An overview. *IEEE Transactions on Reliability*, 35:230–239, 1986.

[3] A. L. Barabasi and R. Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999.

[4] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. Freenet: a distributed anonymous information storage and retrieval system. In *International workshop on Designing privacy enhancing technologies: design issues in anonymity and unobservability*, pages 46–66, 2001.

[5] Charles J. Colbourn. *The Combinatorics of Network Reliability*. Oxford University Press, Inc., 1987.

[6] George 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.

[7] Ming Hua and Jian Pei. Probabilistic path queries in road networks: traffic uncertainty aware path selection. In *EDBT*, pages 347–358, 2010.

[8] Ruoming Jin, Yang Xiang, Ning Ruan, and David Fuhry. 3-hop: a high-compression indexing scheme for reachability query. In *SIGMOD Conference*, pages 813–826, 2009.

[9] Richard M. Karp and Michael G. Luby. A new monte-carlo method for estimating the failure probability of an n-component system. Technical report, Berkeley, CA, USA, 1983.

[10] Hiromitsu Kumamoto, Kazuo Tanaka, Inoue Koichi, and Henley Ernest J. Dagger-sampling monte carlo for system unavailability evaluation. *IEEE Transactions on Reliability*, 29(2):122–125.

[11] G. Pandurangan. Building low-diameter p2p networks. In *Proceedings of the 42nd IEEE symposium on Foundations of Computer Science*, FOCS '01, 2001.

[12] Louis Petingi. Combinatorial and computational properties of a diameter constrained network reliability model. In *Proceedings of the WSEAS International Conference on Applied Computing Conference*, ACC'08, 2008.

[13] Michalis Potamias, Francesco Bonchi, Aristides Gionis, and George Kollios. k-nearest neighbors in uncertain graphs. *PVLDB*, 3(1):997–1008, 2010.

[14] Gerardo Rubino. Network reliability evaluation. pages 275–302, 1999.

[15] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2003.

[16] A. Satyanarayana and R. Kevin Wood. A linear-time algorithm for computing k-terminal reliability in series-parallel networks. *SIAM Journal on Computing*, 14(4):818–832, 1985.

[17] Gayatri Swamynathan, Christo Wilson, Bryce Boe, Kevin Almeroth, and Ben Y. Zhao. Do social networks improve e-commerce?: a study on social marketplaces. In *WOSP '08*.

[18] S.K. Thompson. *Sampling, Second Edition*. New York: Wiley., 2002.

[19] http://mips.helmholtz-muenchen.de/genre/proj/mpact/.

[20] http://thebiogrid.org/.

[21] Leslie G. Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, 8(3):410–421, 1979.

[22] Hilmi Yildirim, Vineet Chaoji, and Mohammed Javeed Zaki. Grail: Scalable reachability index for large graphs. *PVLDB*, 3(1):276–284, 2010.

[23] Ye Yuan, Lei Chen, and Guoren Wang. Efficiently answering probability threshold-based shortest path queries over uncertain graphs. In *DASFAA'10*.

[24] Zhaonian Zou, Hong Gao, and Jianzhong Li. Discovering frequent subgraphs over uncertain graph databases under probabilistic semantics. In *KDD'10*.

[25] Zhaonian Zou, Jianzhong Li, Hong Gao, and Shuo Zhang. Finding top-k maximal cliques in an uncertain graph. In *ICDE*, pages 649–652, 2010.

APPENDIX

A. COMPUTING MINIMAL DCR EQUIVALENT SUBGRAPH

Given vertices s and t, only subsets of vertices and edges in $\mathcal{G}$ are needed to compute the s-t distance-constraint reachability. Intuitively, those edges and vertices have to be on some s-t paths whose distance is less than or equal to d. Because if they are not, their existence will not affect the s-t distance-constraint reachability in the possible graph G. Thus, we can simply exclude them from $\mathcal{G}$ when generating the possible graph. Formally, we have the following observations:

LEMMA 2. *Let* $\overline{\mathbf{G}}$ *be the* **complete possible graph** *with respect to* $\mathcal{G}$ *which includes all the nodes and edges in* $\mathcal{G}$*. Given vertices* s *and* t*, let the* **the minimal equivalent DCR subgraph** $\mathcal{G}_s = (V_s, E_s, p, w) \subseteq \mathcal{G}$ *be:*

$$\begin{aligned} V_s &= \{v \in V | dis(s, v|\overline{G}) + dis(v, t|\overline{G}) \leq d\}, \\ E_s &= \{e = (u, v) \in E | dist(s, u|\overline{G}) + w(e) + dis(v, t|\overline{G}) \leq d\}. \end{aligned}$$

Basically, V_s *and* E_s *contain those vertices and edges that appear on some s-t paths whose distance is less than or equal to d. Given this, we have* $\mathcal{G}_s$ *is the minimal subgraph of* $\mathcal{G}$ *such that*

$$\mathbf{R}^d_{s,t}(\mathcal{G}_s) = \mathbf{R}^d_{s,t}(\mathcal{G}) \qquad (*)$$

Here, the minimal subgraph mean that we cannot further remove any vertices or edges in $\mathcal{G}_s$ *to still hold (*).*

The proof is omitted for simplicity. This result suggests that we can directly work on $\mathcal{G}_s$ instead of $\mathcal{G}$ for the s-t distance-constraint reachability computation. The lemma also directly suggests a simple method (Bread-First-Search from s) to extract $\mathcal{G}_s$ from $\mathcal{G}$ if we precompute the distance matrix on $\overline{G}$. If the distance matrix is not available, we can perform an online discovery:

1. Starting from s, run the single-source Dijkstra algorithm within diameter d (discover all vertices from s with distance less than or equal to d);

2. In the induced subgraph computed in the first step, run the reverse single-source Dijkstra algorithm from vertex t within diameter d (discover those vertices in the first step to t with distance no higher than d);

3. Starting from s again, using a BFS to visit those vertices generated in the second step to compute V_s and E_s. Note that the first two steps have already computed $dis(s, u|\overline{G})$ and $dis(u, t|\overline{G})$ for any vertices generated in the second step.

B. SAMPLING ENUMERATION TREE FOR HANSEN-HURWITZ ESTIMATOR ($\widehat{\mathbf{R}}_{HH}$)

C. PROOF OF THEOREM 1.

Proof Sketch: Let $\tau = \mathbf{R}^d_{s,t}(\mathcal{G})$. When $q_i = Pr_i$, we have

$$Var(\widehat{\mathbf{R}}_{HH}) = \frac{1}{n}(\sum_{i \in \mathcal{L}} q_i(1-\tau)^2 + \sum_{i \in \overline{\mathcal{L}}} q_i(\tau)^2) = \frac{\tau(1-\tau)}{n}.$$

The variance of Horvitz-Thomson estimator can be simplified by considering

$$\pi_i \approx nq_i - \frac{n(n-1)}{2}q_i^2; \quad \pi_{ij} \approx n(n-1)q_iq_j.$$

Algorithm 3 **SamplingR**$(\mathcal{G}, E_1, E_2, Pr, q)$

```
Parameter: $\mathcal{G}$: Uncertain Graph;
Parameter: $E_1$: Inclusion Edge List;
Parameter: $E_2$: Exclusion Edge List;
Parameter: $Pr$: leaf weight;
Parameter: $q$: leaf sampling probability;
if $E_1$ contains a $d$-path from $s$ to $t$ then
    return $Pr/q$; {for optimal $\widehat{\mathbf{R}}_{HH}$, $Pr/q = 1$}
else if $E_2$ contains a $d$-cut from $s$ to $t$ then
    return 0;
end if
select an edge $e \in E \backslash (E_1 \cup E_2)$ {Find a remaining uncertain edge}
Random Toss a Coin with Head Probability $q(e)$; for optimal $\widehat{\mathbf{R}}_{HH}$, $q(e) = p(e)$;
if Head {Case I (including $e$):} then
    return SamplingR$(\mathcal{G}, E_1 \cup \{e\}, E_2, p(e)Pr, q(e)q)$;
else if Tail {Case II (excluding $e$):} then
    return R$(\mathcal{G}, E_1, E_2 \cup \{e\}, (1-p(e))Pr, (1-q(e))q)$
end if
```

In addition, when $q_i = Pr_i$ and $nq_i = nPr_i \ll 1$, $Var(\widehat{\mathbf{R}}_{HT})$

$$\begin{aligned} &= \sum_{i \in \mathcal{L}} \frac{1-\pi_i}{\pi_i} Pr_i^2 + \sum_{i \in \mathcal{L}} \sum_{j \in \mathcal{L}, j \neq i} \frac{\pi_{ij} - \pi_i\pi_j}{\pi_i\pi_j} Pr_iPr_j \\ &\approx \sum_{i \in \mathcal{L}} (\frac{1}{nq_i(1-\frac{n-1}{2}q_i)} - 1)Pr_i^2 - \sum_{i \in \mathcal{L}} \sum_{j \in \mathcal{L}, \neq i} \frac{1}{n} Pr_iPr_j \\ &= \sum_{i \in \mathcal{L}} (\frac{1}{nq_i} + \frac{n-1}{2n - n(n-1)q_i} - 1)Pr_i^2 - \sum_{i \in \mathcal{L}} \frac{1}{n} Pr_i(\tau - Pr_i) \\ &\approx \sum_{i \in \mathcal{L}} (\frac{1}{nq_i} + \frac{n-1}{2n} - 1)Pr_i^2 - \frac{1}{n}\tau^2 + \frac{1}{n} \sum_{i \in \mathcal{L}} Pr_i^2 \\ &= \sum_{i \in \mathcal{L}} \frac{Pr_i^2}{nq_i} - \sum_{i \in \mathcal{L}} \frac{(n-1)Pr_i^2}{2n} - \frac{1}{n}\tau^2 \\ &= \frac{\tau - \tau^2}{n} - \sum_{i \in \mathcal{L}} \frac{(n-1)Pr_i^2}{2n} \end{aligned}$$

Thus, $Var(\widehat{\mathbf{R}}_{HH}) - Var(\widehat{\mathbf{R}}_{HT})$=$\Omega(\sum_{i \in \mathcal{L}} Pr_i^2)$, and then we have $Var(\widehat{\mathbf{R}}_{HT}) \leq Var(\widehat{\mathbf{R}}_{HH})$. □

D. COMPARISON BETWEEN $\widehat{\mathbf{R}}_{HT}$ AND $\widehat{\mathbf{R}}$

In the following, we compare the variance (estimation accuracy) of Horvitz-Thomson estimator $\widehat{\mathbf{R}}_{HT}$ to that of recursive estimator $\widehat{\mathbf{R}}$. We utilize our earlier results for variance comparison to the Hansen-Hurwitz estimator. $\widehat{\mathbf{R}}_{HH}$.

From Theorem 1, we have ($nPr_i \ll 1$)

$$Var(\widehat{\mathbf{R}}_{HH}) - Var(\widehat{\mathbf{R}}_{HT}) \approx \sum_{i \in \mathcal{L}} \frac{(n-1)Pr_i^2}{2n} \approx \frac{1}{2} \sum_{i \in \mathcal{L}} Pr_i^2$$

From Theorem 2, we have

$$Var(\widehat{\mathbf{R}}_{HH}) - Var(\widehat{\mathbf{R}}) = \frac{p(e)(1-p(e))(\tau_1 - \tau_2)^2}{n}$$

Putting these together, we have

$$Var(\widehat{\mathbf{R}}_{HT}) - Var(\widehat{\mathbf{R}}) \approx \frac{p(e)(1-p(e))(\tau_1 - \tau_2)^2}{n} - \frac{1}{2} \sum_{i \in \mathcal{L}} Pr_i^2$$

Since $\frac{p(e)(1-p(e))(\tau_1-\tau_2)^2}{n}$ in general is not correlated with the size of minimal DCR equivalent subgraph, the variance difference is mainly determined by $\sum_{i \in \mathcal{L}} Pr_i^2$. When $\sum_{i \in \mathcal{L}} Pr_i^2$ is relatively large, $Var(\widehat{\mathbf{R}}_{HT})$ will be smaller than $Var(\widehat{\mathbf{R}})$. When $\sum_{i \in \mathcal{L}} Pr_i^2$ is relatively small, $Var(\widehat{\mathbf{R}}_{HT})$ will be larger than $Var(\widehat{\mathbf{R}})$. In addition, when the size of the uncertain subgraph with respect to the query is small, Pr_i tends to be relatively large and so does $\sum_{i \in \mathcal{L}} Pr_i^2$. As the subgraph size increases, Pr_i becomes smaller and so does $\sum_{i \in \mathcal{L}} Pr_i^2$.

The experimental results in Tables 1 and 2 (Section 4) also provide evidence to support the above analysis. We can see that when the minimal DCR uncertain subgraph size (the number of edges) is less than 35, the relative error and variance of Horvitz-Thomson estimator $\widehat{\mathbf{R}}_{HT}$ are significantly smaller than those of the Hansen-Hurwitz estimator $\widehat{\mathbf{R}}_{HH}$. Recall in Theorem 1, the variance of the difference between them is approximately on the order of $\sum_{i \in \mathcal{L}} Pr_i^2$. In this case, the relative error and variance of the recursive estimator $\widehat{\mathbf{R}}_{RHH}$ ($\widehat{\mathbf{R}}$ in above discussion) are comparable to or even higher than those of the Horvitz-Thomson estimator $\widehat{\mathbf{R}}_{HT}$. However, as the minimal DCR uncertain subgraph size increases (higher than 35), the advantage of the recursive estimator becomes quite apparent.

E. CONSTRUCTING FAST EXACT AND APPROXIMATION ALGORITHM

In this section, we will discuss a method for edge selection and quickly test the distance-constraint reachability. Then we will combine this method with our aforementioned recursive algorithm to construct both exact and approximate algorithms.

E.1 Recognizing d-path or d-cut

In this subsection, we focus on the following problem: *given a resulting graph G of $\mathcal{G}$, how can we quickly determine whether t is d-reachable from s or not?* Specifically, we would like to visit as few number of edges in G as possible for this task. This is because later we will apply the developed procedure for this task to selecting the next edge in the recursive computation procedure (Algorithm 1 and 2).

A straightforward solution to this problem is to utilize Dijkstra or A^* algorithm to compute the shortest-path distance from s to t in G. However, in these types of algorithms, when we visit a new vertex v in G, we have to immediately visit all its neighbors (corresponding to visiting all outgoing edges in v) in order to maintain the estimated shortest-path distance from s to them so far. This "eager" strategy thus requires us to visit a large number of edges in G and it is also the essential step in the shortest-path distance computation. Fortunately, in our problem, we do not need to compute the exact distance between s and t. Indeed, we only need to determined whether there is a d-path from s to t or not.

Algorithm 4 FindDPath$(G, v, path, plen)$

```
Parameter: $G$: Graph Defined by Selected Existence Edges;
Parameter: $v$: the current vertex;
Parameter: $path$: the current active path;
Parameter: $plen$: the current active path length;
 1: if $v = t$ {Find a d-path} then
 2:     return path;
 3: end if
 4: for each $v' \in N(v)$ {visit $v'$ from closest to farthest} do
 5:     if $(plen + w(v,v') < gdis(s,v'))$ {(a) $gdis(s,v')$ is reduced}
        $\wedge(gdis(v',t) + plen + w(v,v') \leq d)$ {(b) estimated total length
        no larger than $d$} then
 6:         $gdis(s,v') \leftarrow plen + w(v,v')$; {update $gdis(s,v')$ }
 7:         FindDPath($G$,$v'$, $path \cup \{v'\}$, $plen + w(v,v')$);
 8:     end if
 9: end for
10: $gdis(v,t) \leftarrow \min_{v' \in N(v)}\{w(v,v') + g(v',t)\}$; {update
    $gdis(v,t)$}
```

Given this, we design a DFS fashion procedure to discover the d-path from s to t. This DFS procedure is "lazy" compared with the Dijkstra or A^* algorithm. Basically, a new edge is needed to expand only if it should be visited along the depth-first-search process and it is promising to be on a d-path. The DFS procedure is sketched in Algorithm 4. Starting from vertex s, we will start to explore its first neighbor (its next neighbor will be explored only if there is no d-path which can be find going through the earlier ones, Line 5), and then recursively visit the neighbors of this neighbor.

Pruning Search Space: To reduce the number of edges which need to visited, we design a pruning technique which can determine whether an edge (v, v') should be expanded at a given time (Line 5). The condition is based on whether the new edge (v, v') has the potential to be on a d-path. Note that all the vertices in $\overline{G}$ (including all edges in $\mathcal{G}$) satisfy $dis(s, v|\overline{G}) + dis(v, t|\overline{G}) < d$ which suggests that every vertex has the potential to be on a d-path in $\overline{G}$. However, for $G \subseteq \overline{G}$, since some edges are not selected in the resulting graph G, some vertices may not appear in any d-path. To perform the pruning, we maintain two values $gdis(s, v)$ and $gdis(v, t)$ associated with each vertex v, which records **the current shortest path distance from** s **to** v on the partial graph visited by DFS so far ($gdis(s, v)$) and **the lower bound estimate on the shortest path distance from** v **to** t ($gdis(v, t)$).

Initially, $gdis(s, v)$ has an infinite value (∞) for each vertex except vertex s ($gdis(s, s) = 0$), and $gdis(v, t) = dis(v, t|\overline{G})$. The maintenance of $g(s, v')$ is straightforward (Line 6): *if the new path from s to v' has smaller length, we update $g(s, v')$*. The $g(v, t)$ is defined recursively and is updated (at traceback) when we have visited each of its neighbors (Line 10): $g(v, t)$ *is chosen as the minimal one of the weights between v to its neighbors v' plus their estimated shortest distance to t*, i.e., $g(v, t) = \min_{v' \in N(v)} w(v, v')$ $+g(v', t)$.

For the currently visited vertex v, we will check each of its neighbors v' according to the increasing order of the edge weight $w(v, v')$. This order can help minimize the number of times to revisit any given node. If any of the neighbors v' can be visited, i.e., edge (v, v') may be part of a d-path, it has to satisfy two conditions (Line 5): *a) it decreases the $gdis(s, v')$, i.e., the new path from s to v' has smaller length than the earlier ones; and b) the new path from s to v' together with the updated lower bound of the shortest-path distance from v' to t is no higher than d.* Basically these two conditions are the necessary ones for the new edge (v, v') may occur in a d-path. The *FindDPath* algorithm has the following property:

LEMMA 3. *If Algorithm 4 returns a path, it is the d-path from s to t defined in the order of DFS procedure; if it does not return a path, there is no d-path from s to t. Also, if we allow this procedure to continue search after its discovery of the first d-path, this procedure can eventually enumerate all the d-path from s to t in G.*

We note that we can utilize this algorithm to *enumerate all the d-paths in $\overline{G}$*, which is the first step in the path-based estimator for the $\mathbf{R}^d_{s,t}(\mathcal{G})$ (Subsection 2.2). In the next subsection, we will fuse this algorithm with Algorithm 1 for a fast exact computation of $\mathbf{R}^d_{s,t}(\mathcal{G})$.

E.2 The Complete Algorithm

In this subsection, we will combine recursive computation procedures **R**(Algorithm 1) and *FindDPath*(Algorithm 4) together to calculate $\mathbf{R}^d_{s,t}(\mathcal{G})$ efficiently. The combination of **OptEstR** with *FindDPath* is similar and thus is omitted for simplicity. Recall in procedure **R**, the first key problem is how to select an uncertain edge e for any $\mathcal{G}(E_1, E_2)$ prefix group of possible graphs. To solve this problem, we choose *the edge e to be the one which needs to be visited once all edges in E_1 (and E_2) have been visited in the process of identifying the first d-path according to the* FindDPath

Algorithm 5 $\mathbf{R}^*(\mathcal{G}, E_1, E_2, S_v, S_i)$

```
Parameter: $S_v$: Vertex Stack for DFS;
Parameter: $S_i$: Edge Index Stack for DFS;
 1: if $S_v.top() = t$ {Condition 1: $E_1$ contains a $d$-path} then
 2:     return 1;
 3: end if
    {Find next edge $e = (v, v')$ can be explored in DFS:}
 4: $e \leftarrow$ NextEdge($\mathcal{G}, E_1, E_2, S_v, S_i$)
 5: if $e = \emptyset$ {Condition 2: $E_2$ contains a $d$-cut} then
 6:     return 0
 7: end if
 8: return $p(e)\mathrm{R}^*(\mathcal{G}, E_1 \cup \{e\}, E_2, S_v.push(w), S_i.push(1))$
        $+(1 - p(e))\mathrm{R}^*(\mathcal{G}, E_1, E_2 \cup \{e\}, S_v, S_i)$;
Procedure NextEdge($\mathcal{G}, E_1, E_2, S_v, S_i$)
 1: while $!S_v.empty()$ do
 2:     $v \leftarrow S_v.top()$;
 3:     for $i$ from $S_i.top()$ to $|N(v)|$ do
 4:         $v' \leftarrow v[i]$ {v's $i$-th neighbor}; $e = (v, v')$; $S_i.top() + +$;
 5:         if $e \notin E_2$ {not in excluding edge list} $\wedge\, plen(S_v) + w(v, v') <$
            $gdis(s, v') \wedge gdis(v', t) + plen(S_v) + w(v, v') \leq d$ {conditions
            (a) and (b)} then
 6:             if $e \in E_1$ {Determined earlier} then
 7:                 $S_v.push(v')$; $S_i.push(1)$; goto 2;
 8:             else
 9:                 return $e$
10:             end if
11:         end if
12:     end for
13:     $S_v.pop(), S_i.pop()$ {DFS trace back};
14: end while
15: return $\emptyset$
```

procedure. Note that the edges in the exclusion set E_2 are explicitly marked as the "forbidden" edges when they are in the line to be visited for identifying the d-path, i.e., they cannot be utilized during the search process. In other words, we may also consider edge e is the next edge to be visited for the $\mathcal{G}(E_1, E_2)$ prefix group.

A major difficulty to implementing the aforementioned edge selection strategy is that we have to couple two recursive procedures (**R** and *FindDPath*) together. To solve this problem, we use two stacks S_v and S_i to simulate the DFS process for *FindDPath*: stack S_v records the current active vertices (the active path) of the *FindDPath* for the partial group $(\mathcal{G}, E_1, E_2)$, and S_i records the index of the next edge in the line to be visited for the corresponding vertex in stack S_v. To start with the search, we always store vertex s in the bottom of stack S_v and put index 1 in S_i as the first edge of s needs to be visited first.

Using stacks S_v, S_i, the procedure *NextEdge* describes how we can get the next *uncertain* edge to be visited according to the *FindDPath* procedure (Algorithm 5). Basically, we apply the stacks and iterations (Line 1, 3) to simulate the recursive process. Specifically, the top of stack S_v records the current active vertex v (Line 2) and we iterate on each of its remaining neighbors from $S_i.top()$ to $|N(v)|$ to search for the next candidate edge, which has the potential to be a d-path(condition (a) and (b), Line 5 in *FindDPath* and in *NextEdge*). Note that we do not consider those edges which have been determined to be excluded from the resulting graph $e \notin E_2$ (Line 5). However, edge $e = (v, v')$ may be selected more than once and after the first time is being visited, this edge is not *uncertain* any more, i.e., $e \in E_1$ (Line 6). In this case, we will continue the search process by adding v' to stack S_v and planning to visit its first edge (Line 7). For any vertex v, if we exhaust all its outgoing edges (or neighbors), we have to trace back (pop up the vertices in the stack) to find the next edge (Line 13). Finally, when there are no edges that can be selected to further extend the search (S_v is empty, Line 1), empty edge $\emptyset$ is returned.

Algorithm 6 $\mathbf{R}^*(\mathcal{G}, E_1, E_2, S_v, S_i)$

```
Parameter: $S_v$: Vertex Stack for DFS;
Parameter: $S_i$: Edge Index Stack for DFS;
 1: if $S_v.top() = t$ {Condition 1: $E_1$ contains a $d$-path} then
 2:     return 1;
 3: end if
    {Find next edge $e = (v, v')$ can be explored in DFS:}
 4: $(\mathbf{e}, \mathbf{x}, \mathbf{Xlist}, \mathbf{Ylist}) \leftarrow$ NextEdge($\mathcal{G}, \mathbf{E_1}, \mathbf{E_2}, \mathbf{S_v}, \mathbf{S_i}$)    $(*)$
 5: if $e = \emptyset$ {Condition 2: $E_2$ contains a $d$-cut} then
 6:     return 0
 7: end if
 8: store and then update $gdis(s, v')$ with $x$;                        (*)
 9: $R_1 \leftarrow \mathrm{R}^*(\mathcal{G}, E_1 \cup \{e\}, E_2, S_v.push(w), S_i.push(1))$;
10: restore $gis(s, v')$;                                             (*)
11: $R_2 \leftarrow \mathrm{R}^*(\mathcal{G}, E_1, E_2 \cup \{e\}, S_v, S_i)$;
12: restore $gdis(s, v)$ and $gdis(v, t)$ for those vertices in $Xlist$ and
    $Ylist$, respectively;                                             (*)
13: return $p(e)R_1 + (1 - p(e))R_2$;
Procedure NextEdge($\mathcal{G}, E_1, E_2, S_v, S_i$)
 1: while $!S_v.empty()$ do
 2:     $v \leftarrow S_v.top()$;
 3:     $\mathbf{Xlist} \leftarrow \emptyset$; $\mathbf{Ylist} \leftarrow \emptyset$;                              $(*)$
 4:     for $i$ from $S_i.top()$ to $|N(v)|$ do
 5:         $v' \leftarrow v[i]$ {v's $i$-th neighbor}; $e = (v, v')$; $S_i.top() + +$;
 6:         if $e \notin E_2 \wedge plen(S_v) + w(v, v') < gdis(s, v') \wedge gdis(v', t) +$
            $plen(S_v) + w(v, v') \leq d$ {conditions (a) and (b)} then
 7:             if $e \in E_1$ {Determined earlier} then
 8:                 $\mathbf{Xlist} \leftarrow \mathbf{Xlist} \cup \{(\mathbf{v'}, \mathbf{dis}(\mathbf{s}, \mathbf{v'}))\}$ {for $v' \notin Xlist$};
                    $\mathbf{dis}(\mathbf{s}, \mathbf{v'}) \leftarrow \mathbf{plen}(\mathbf{S_v}) + \mathbf{w}(\mathbf{v}, \mathbf{v'})$            $(*)$
 9:                 $S_v.push(v')$; $S_i.push(1)$; goto 2;
10:             else
11:                 return $(\mathbf{e}, \mathbf{plen}(\mathbf{S_v}) + \mathbf{w}(\mathbf{v}, \mathbf{v'}), \mathbf{Xlist}, \mathbf{Ylist})$   $(*)$
12:             end if
13:         end if
14:     end for
15:     $\mathbf{Ylist} \leftarrow \mathbf{Ylist} \cup \{(\mathbf{v}, \mathbf{dis}(\mathbf{v}, \mathbf{t}))\}$ {for $v$ not in $Ylist$};
        $\mathbf{dis}(\mathbf{v}, \mathbf{t}) \leftarrow \min_{(\mathbf{v}, \mathbf{v'}) \in \mathbf{E_1}} \mathbf{w}(\mathbf{v}, \mathbf{v'}) + \mathbf{gdis}(\mathbf{v'}, \mathbf{t})$          $(*)$
16:     $S_v.pop(), S_i.pop()$ {DFS trace back};
17: end while
18: return $\emptyset$
```

The complete algorithm using the *NextEdge* procedure is illustrated in $\mathbf{R}^*$ (Algorithm 5). Here, we not only utilize the *NextEdge* procedure for selecting the next edge e, but also use it to answer whether E_1 contains a d-path or E_2 contains a d-cut for Algorithm 1: *if the top element of stack S_v is vertex t, then we basically find a d-path from s to t using edges in E_1; if the returned edge e is $\emptyset$ which suggests that there is no way to further extend the search, then we can determine there is no d-path from s to t.* Line $1-7$ are based on these two conditions to determine whether the recursion can be stopped.

The enumeration process in Figure 2(b) illustrates the compete algorithm ($\mathbf{R}^*$) which uses the DFS procedure for selecting next edge. The correctness of the $\mathbf{R}^*$ is easily established by Lemma 1 and 3, and

$$\mathbf{R}^d_{s,t}(\mathcal{G}) = \mathbf{R}^*(\mathcal{G}, \emptyset, \emptyset, S_v.push(s), S_i.push(1),$$

where stacks S_v and S_i are empty initially.

The total computational complexity of $\mathbf{R}^*$ can be written as $O(2^a L)$, where a is the average height of enumeration tree generated by $\mathbf{R}^*$ and L is the average number of edges (vertices) visited by *FindDPath* procedure for determining whether there is a d-path in E_1 or a d-cut in E_2. Note that a is the lower bound of L as some edges in the inclusion set (E_1) can be visited more than once by the *NextEdge* procedure (Line 7 in *NextEdge*).

Finally, in Algorithm 5, for simplification, we omit the details on how to handle the two cost functions $g(s, v)$ and $g(v, t)$ associated

Figure 3: **Relative Variance Varying Sample Size**

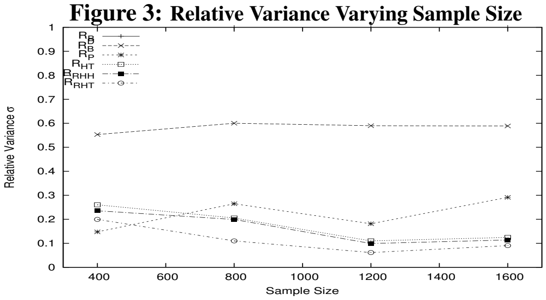
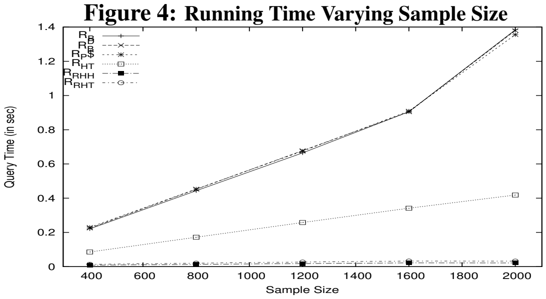


Figure 4: **Running Time Varying Sample Size**



with each vertex v in order to prune the search process. Their updates also need a stack-like mechanism to maintain, which are similar to S_v and S_i. The complete description of $\mathbf{R}^*$ which includes the details of maintaining these two cost functions can be found in Algorithm 6 where the (*) lines maintain $g(s,v)$ and $g(v,t)$.

F. EXPERIMENTAL RESULTS ON SAMPLE SIZE AND SCALABILITY

Varying Sample Size: In this experiment, we study how sample size affect the estimation accuracy and performance. Here, we vary the sample size from 200 to 2000 and run different sampling estimators on the same uncertain graph as in the first experiment. Figure 3 illustrates the relative variance efficiency of different sampling estimator with respect to different sample size. In general, we can see that most of the sampling operators tend to have better variance efficiency as the sample size increases compared with the baseline direct sampling estimator. However, such trend does not hold for the path-based estimator. Based on their variance analysis, we can see both path-based estimator and the direct sampling estimator reduce the variance in the similar rate when the sample size increases. Again, the two recursively sampling estimators are the clear winner as they can reduce the baseline variance by almost 10 times! Figure 4 shows the computational time of different sampling estimators. In general, as the sample size increases, their running time also increases. However, we can see that the increase of the recursive sampling estimator is the smallest.

Table 6: **Scalability: Query Time with Random Graphs (in Seconds)**

	$\widehat{\mathbf{R}}_B$	$\widehat{\mathbf{R}}_B^D$	$\widehat{\mathbf{R}}_P$	$\widehat{\mathbf{R}}_{HT}$	$\widehat{\mathbf{R}}_{RHH}$	$\widehat{\mathbf{R}}_{RHT}$
100k-300k	503	550	677	148	38	71
300k-500k	649	645	1978	174	51	92
500k-700k	691	745	4783	199	59	106
700k-900k	742	756	-	211	64	119
900k-1.1m	809	-	-	215	64	115

Table 7: **Query Time with Power-Law Graphs (in Seconds)**

	$\widehat{\mathbf{R}}_B$	$\widehat{\mathbf{R}}_B^D$	$\widehat{\mathbf{R}}_P$	$\widehat{\mathbf{R}}_{HT}$	$\widehat{\mathbf{R}}_{RHH}$	$\widehat{\mathbf{R}}_{RHT}$
100k-300k	245	287	15312	123	36	59
300k-500k	353	437	-	162	61	92
500k-700k	456	682	-	210	102	151
700k-900k	473	675	-	234	117	159
900k-1.1m	497	780	-	243	122	168

Scalability: To study the scalability of different estimators, we generate queries with very large minimal equivalent DCR subgraphs $\mathcal{G}_s$ with the number of edges ranging from $100,000$ ($100k$) to $1,100,000$ ($1.1m$) on a random graph with $1m$ vertices and $13m$ edges and on a power-law graph with $1m$ vertices and around $2m$ edges. Specifically, we group all the queries into five categories based on the number of edges in $\mathcal{G}_s$: $100k-300k$, $300k-500k$, $500k-700k$, $700k-900k$ and $900k-1.1m$ and each category has 1000 queries. Further, for each estimator, the sample size is 1000. Table 6 reports the query time for large queries on the random graph. In general, we observe that the query time increases with the size of the subgraphs. However, for most of estimators except for $\widehat{\mathbf{R}}_P$, the query time has shown to have sub-linear growth with respect to the subgraph size. The query time of $\widehat{\mathbf{R}}_P$ increases exponentially due to its computational cost to enumerate all the d-paths, which becomes very expensive as the number of edges increases. The estimators $\widehat{\mathbf{R}}_{RHH}$ and $\widehat{\mathbf{R}}_{RHT}$ were 10 times faster than the estimators $\widehat{\mathbf{R}}_B$ and $\widehat{\mathbf{R}}_B^D$. Table 7 reports the query time for large queries on the power-law graph. Similarly, we observe that generally the query time increases as the subgraph size increases. However, the $\widehat{\mathbf{R}}_P$ estimator scales poorly and cannot process the subgraph with number of edges is higher than $300k$ (the cost of computing and storing all d-paths is too large). It is almost 100 times slower than all other methods. The $\widehat{\mathbf{R}}_{RHH}$ and $\widehat{\mathbf{R}}_{RHT}$ estimators are almost 5 times faster than the $\widehat{\mathbf{R}}_B$ and $\widehat{\mathbf{R}}_B^D$ estimators,

G. AN EXTENSION OF DCR QUERY

Here, we consider strengthening the DCR (Distance-Constraint Reachability) query with an additional constraint introduced in [23]. Recall that a s-t path in G with length less than or equal to distance constraint d is referred to as a d-path between s and t. If there is a d-path between s and t, then vertex t is d-reachable from vertex s in graph G, i.e., $dis(s,t|G) \leq d$. If there is no d-path from s to t, then, t is not d-reachable from s ($dis(s,t|G) > d$). However, even when there is a d-path P between s and t, if this path has very small probability $\mathbf{Pr}[P]$, we may not be able to utilize such a path, i.e., we cannot take such a route from s to t because of its low probability. Note that the probability of a path P is simply the product of all its edge existence probabilities. Given this, we introduce the ϵ-d-path from s to t to be a path P with its length no higher than d and its probability no less than ϵ ($\mathbf{Pr}[P] \geq \epsilon$).

DEFINITION 3. (**Distance Constraint ϵ-Path Reachability**)

$$\mathbf{I}_\epsilon^d(G) = \begin{cases} 1, & \textit{if there is a } \epsilon\textit{-}d\textit{-path from } s \textit{ to } t \textit{ in } G \\ 0, & \textit{otherwise} \end{cases}$$

Then, the Distance-constraint ϵ-path reachability (DCPR) in uncertain graph $\mathcal{G}$ with respect to parameter d is defined as

$$\mathbf{R}_\epsilon^d(\mathcal{G}) = \sum_{G \sqsubseteq \mathcal{G}} \mathbf{I}_\epsilon^d(G) \cdot \mathbf{Pr}[G]\,. \tag{5}$$

Basically, if a possible graph $G \sqsubseteq \mathcal{G}$ is considered to be distance-constraint ϵ-path reachable from s to t, it must contain a d-path with probability no less than ϵ (ϵ-d-path). Given this, we note that Algorithm 1 (exact), Algorithm 3 (unequal sampling estimator), and Algorithm 2 (recursive sampling estimator) all can be easily extended to handle the DCPR query. Basically, a straightforward test can determine whether the inclusion edge list E_1 contains a ϵ-d-path or E_2 contains a ϵ-d-cut (defined similarly as d-cut) and these algorithms can easily adopt to the new test condition. More specifically, we can simply extend *FindDPath* to incorporate the additional constraint for the path probability. Thus, we can see that our approaches can be extended to the more constrained DCPR query. Finally, we note that the estimation accuracy results (Theorem 1 and 2) also hold for the new queries.