Scalable Evaluation of k-NN Queries on Large Uncertain Graphs

Xiaodong Li[1], Reynold Cheng[1], Yixiang Fang[1], Jiafeng Hu[1], Silviu Maniu[2]
[1]The University of Hong Kong, China, [2]Université Paris-Sud, France
{xdli,ckcheng,yxfang,jhu}@cs.hku.hk,silviu.maniu@lri.fr

ABSTRACT

Large graphs are prevalent in social networks, traffic networks, and biology. These graphs are often inexact. For example, in a friendship network, an edge between two nodes u and v indicates that users u and v have a close relationship. This edge may only exist with a probability. To model such information, the *uncertain graph model* has been proposed, in which each edge e is augmented with a probability that indicates the chance e exists. Given a node q in an uncertain graph $\mathcal{G}$, we study the k-NN query of q, which looks for k nodes in $\mathcal{G}$ whose distances from q are the shortest. The k-NN query can be used in friend-search, data mining, and pattern-recognition. Despite the importance of this query, it has not been well studied. In this paper, we develop a tree-based structure called the *U-tree*. Given a k-NN query, the *U-tree* produces a compact representation of $\mathcal{G}$, based on which the query can be executed efficiently. Our results on real and synthetic datasets show that our algorithm can scale to large graphs, and is 75% faster than state-of-the-art solutions.

1 INTRODUCTION

Graphs are prevalent in social networks [10, 12], traffic networks [11], biological networks [32], and mobile ad-hoc networks [13]. Due to noisy measurements [1], hardware limitation [2], inference models [9], and privacy-preserving perturbation [4, 23], these graphs are inherently uncertain. To model this error, *uncertain graphs* have been studied [1, 6, 15, 19]. In these graphs, each edge is associated with a probability distribution. Figure 1(a) shows the protein-protein interaction (PPI) network as an uncertain graph, where each node denotes a protein, and each edge is an interaction between a pair of proteins. The value on each edge denotes the probability that the interaction exists (called *existential probability*). For instance, the existential probability between nodes A and B is 0.7.

k-NN query. In this paper, we study the efficient evaluation of *k-nearest neighbor* (k-NN) queries on uncertain graphs [1]. Given a node q, the k-NN query returns k nodes with the shortest "*distances*" from q, based on a distance function (Table 1). The k-NN query can help biologists to perform tasks such as protein complex detection [2, 21], link discovery [17, 32], and protein function prediction [24]. Security experts can also use k-NN queries to design privacy-preserving algorithms to protect nodes in a graph from being identified [4].

Although k-NN queries are useful, the issues of evaluating them efficiently have only been briefly touched, e.g., [19]. Our experiments found that they are also not very efficient on large uncertain graphs. The major reason is that for correct query execution, queries running on the uncertain graph should follow the *Possible World Semantics* (*PWS* in short). Figure 1(b) shows a *possible world* instance sampled from $\mathcal{G}$ in Figure 1(a). A simple way to evaluate the query is to get the answer from each possible world and then collect all these answers to form the final answer. For example, given a k-NN query of computing the k nearest neighbors from node A in Figure 1(a), we obtain 2^4 possible worlds, then for each of them, we compute the k nearest neighbors, and finally obtain the k nodes which are the closest to A considering all the possible worlds (according to a function in Table 1). Because an uncertain graph has an exponential number of *possible worlds*, a naïve solution can be extremely inefficient. Although existing solutions try to avoid enumerating all the possible worlds, they are still expensive.

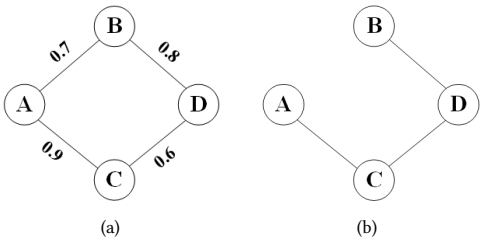


Figure 1: (a) an uncertain graph $\mathcal{G}$ and (b) a possible world from $\mathcal{G}$ with existence probability $0.3\times0.9\times0.6\times0.8 = 0.1296$.

Table 1: Distance functions for uncertain graphs.

Function	Formula
Most-probable [32]	$d_{mp}(s,t) = \arg\max \prod_{e\in PATH(s,t)} p(e)$
Reliability [35]	$d_{re}(s,t) = \sum_{d\|d<\infty} P_{s,t}(d)$
Median [30]	$d_{me}(s,t) = \arg\max\{\sum_{d=0}^{D} P_{s,t}(d) \le \frac{1}{2}\}$
Majority [30]	$d_{ma}(s,t) = \arg\max_d\{P_{s,t}(d)\}$
Expected [32]	$d_{ex}(s,t) = \sum_{d=0}^{\infty} P_{s,t}(d)d$
Expected-reliable [35]	$d_{er}(s,t) = \sum_{d\|d<\infty} d\frac{P_{s,t}(d)}{1-P_{s,t}(\infty)}$

The U-tree framework. In this paper, we develop a new indexing framework for k-NN requests, which (i) allows efficient and scalable k-NN query evaluation under the PWS; and (ii) can be easily adapted to different distance functions (e.g., Table 1). As shown in Figure 2, our framework consists of two stages:

- *offline index construction (Figure 2(a)).* Given an uncertain graph $\mathcal{G}$ and a distance function d, we first employ *decomposition techniques* (Step A), which converts $\mathcal{G}$ into a succinct index structure, whose edges are encoded with the probability information of $\mathcal{G}$ [5, 14, 16, 22, 31, 33]. These techniques were not designed for k-NN queries. Hence, we perform *customization* of the index (Step B), which adjusts the information and structure of the index, in order to enhance the performance of k-NN queries.
- *online query evaluation (Figure 2(b)).* This stage is used to evaluate a k-NN request for a node q online. Particularly, we design an efficient query algorithm (Step C) that uses the U-tree developed

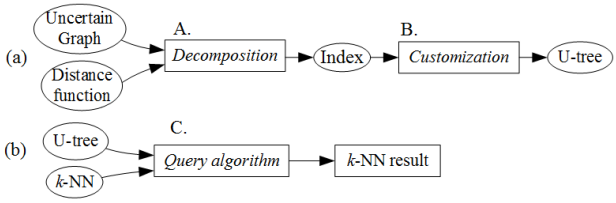


Figure 2: Answering k-NN query with the U-tree.

in the offline stage, which then yields the k nodes that are the closest to q according to distance function d.

Cost model. Upon receiving a k-NN request, the U-tree will generate another uncertain graph g (Step C above), which is a proper subgraph of $\mathcal{G}$, but contains the probability information essential to answering the k-NN query. Because g is often much smaller than $\mathcal{G}$, the query execution time can be significantly reduced. To achieve this goal, more information needs to be incorporated to the index during customization, which also renders a larger U-tree (Step B). As we will point out later, there is a trade-off between the U-tree's size and the query cost. We will discuss a cost model that allows us to balance between these two factors, in order to improve query efficiency without significantly increasing the U-tree size.

Experiments. To evaluate the performance of the proposed methods, we conduct experiments on both real and synthetic datasets. We have also tested different decomposition methods and distance functions in the U-tree framework. The results show that U-tree is superior to the state-of-the-art algorithms, and the query time is significantly reduced by 75%; the overhead of U-tree is only 23% more than the index generated in Step A. Our solution is also scalable to large uncertain graphs with over one million nodes.

Organization. The rest of paper is organized as follows. We review the related works in Section 2. In Section 3, we present the formal definition of the problems, and discuss some basic techniques. We present the U-tree framework in Section 4. In Section 5, we present our experiment results. Section 6 concludes.

2 RELATED WORK

We review the existing queries for uncertain graphs, and then discuss the probabilistic distance functions.

2.1 Queries for Uncertain Graphs

In recent years, the k-NN query for uncertain graphs has received plenty of attention [19, 29, 30]. As mentioned before, to answer queries on uncertain graphs, the Possible World Semantics (PWS) are often used, which assumes that an uncertain graph can be expressed as a number of graph instances. Because there is an exponential no. of possible world instances, three methods are proposed in the literature: (1) The first one finds some representative deterministic graphs from the uncertain graph and answer the queries based on them [28]. (2) The second one is to reduce the size of the uncertain graph by removing some weak nodes and edges, or only sample part of the uncertain graph [30]. This, however, will lead to information loss. (3) The last method is to build some elegant index structures, which can significantly speed up the query process without information loss, and thus has received plenty of attention recently [19, 22]. Thus, in this paper we adopt the third method for the k-NN query.

In [30], Potamias et al. studied the k-NN query on uncertain graphs by using incremental Dijkstra [29] with *Monte Carlo* (MC) sampling. In [19], Khan et al. proposed a novel index for the probabilistic reliability search problem (reliable set query), which aims to find all the nodes reachable from a given source node with probability over a user-defined threshold. Moreover, it can be adapted for answering the top-k reliability query. However, it only focuses on reliability search, and it is not clear how to support other distance functions. Recently, Maniu et al. [22] proposed a novel tree index, called ProbTree, and studied how to perform source-to-target query (STQ) using ProbTree. However, as it is mainly designed for STQ, it works poorly for k-NN queries.

In summary, none of these works can answer the k-NN queries with arbitrary probabilistic distance functions, and thus they are not general enough. Therefore, it is desirable to develop a generic k-NN query framework for any probabilistic distance functions and graphs with different probability distributions.

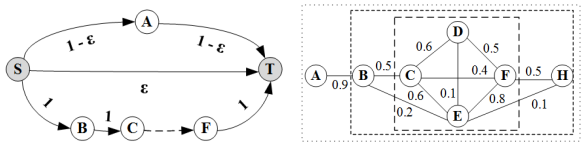
2.2 Probabilistic Distance Functions

All the distance functions that can be used for k-NN queries are summarized in Table 1, where $P_{s,t}(a)=\sum_{G|d_G(s,t)=a} Pr(G)$ is the probability that the shortest path distance (SPD) between two nodes s and t equals to a [30]. Given a query node s, a k-NN query can return the top-k nodes with the smallest values of d_{me}, d_{ma}, d_{ex}, and d_{er} from s, or the top-k nodes with the biggest values of d_{mp} or d_{re} from s.

The function d_{mp} measures the length of the most-probable-path (i.e., a path with the highest probability) between nodes s and t. The function d_{re} measures the probability that there exists a path between nodes s and t, which is meaningful in situations such as delivering packages in a sensor network, but it cannot deal with the cases that prefer distance rather than reliability. The function d_{me} measures the median SPD among all the possible worlds, and it can be used when the user wants to get any k-th order statistic, but its value may be infinite. The function d_{ma} computes the *SPD* that is the most likely to be observed when sampling a graph from $\mathcal{G}$. It is useful in uncertain graphs with irregular distance distributions on edges, where the value has a limited discrete domain, but it cannot deal with infinity, e.g., most values of d_{ma} will become infinite when searching in the uncertain graphs with small probability on each edge. The functions d_{ex} and d_{er} are often used to compute the expected distances, and d_{er} is better when there are infinite distances in uncertain graphs. It is worth mentioning that, there is no consensus on which distance function is the best, because different functions can be used in different applications.

Consider the example graph in Figure 3(a), where the number on an edge represents the probability that it exits and ϵ is an infinitely small number. Suppose that our goal is to compute the SPD from S to T. Then, d_{me} will return the length of the path on the top, while d_{mp} will return the length of the path on the bottom as the SPD although it may contain infinite edges in the path.

Because of space limitations, we mainly focus on *reliability* and *expected-reliable* distances in this paper, as they are the most well studied ones in recent years [19, 22, 30], but other distance functions can also be easily incorporated into our indexing framework.



(a) (b)

Figure 3: Examples for (a) distance functions and (b)PCD results.

3 PRELIMINARIES

In this section, we first introduce the problem we studied in Section 3.1, then discuss the MC sampling in Section 3.2, and finally present the uncertain graph decomposition (UGD) and UGD-based index in Sections 3.3 and 3.4.

3.1 Problem Definition

In line with previous studies [19, 30], we consider an uncertain graph as follows. Note that all the notations frequently used in the paper are summarized in Table 2.

Definition 1 (Uncertain Graph). *An* uncertain graph $\mathcal{G}$ *is a triple* (V, E, p)*, where* V *is the set of nodes,* E *is the set of edges, and* p *is the function of assigning probabilities, i.e., for any edge* $e \in E$*, the probability that it has a value* w *is* $p(w|e)$*, and* $\sum_{w \in W} p(w|e)=1$*, where* W *contains all the possible values on the edges.*

Let us take the graph in Figure 1(a) as an example, where each edge follows a Bernoulli distribution (e.g., for e=(A, C) we have $p(0|e)$=0.1 and $p(1|e)$=0.9). For each edge, if its value w=0, it means that the edge does not exist; otherwise, it exists. Notice that the function p allows any kind of probability distributions (e.g., multi-valued or normal distributions) to be assigned for an edge.

According to the possible world semantics [19, 22, 30], an uncertain graph $\mathcal{G}$ can be thought of as a probabilistic distribution, containing an exponential number of possible worlds, each having different probabilities. The probability of a possible world is defined as follows:

Definition 2. *Given an uncertain graph* $\mathcal{G}$=(V, E, p)*, the probability of observing a possible world* G=(V, E_G) *is*

$$Pr(G) = \prod_{e \in E_G \& i \in W \setminus \{0\}} p(i|e) \prod_{e \in E \setminus E_G} p(0|e). \quad (1)$$

For simplicity, if the edges follow Bernoulli distributions (see Figure 1(a)), the probability of observing a possible world is:

$$Pr(G) = \prod_{e \in E_G} p(e) \prod_{e \in E \setminus E_G} (1 - p(e)). \quad (2)$$

Next, we formally define STQ and k-NN query as follows[1]. We illustrate them via Example 1.

Definition 3. *Given an uncertain graph* $\mathcal{G}(V, E, p)$*, a distance function* d*, and two nodes* s *and* t *(*$s, t \in V$*), the source-to-target query (STQ)* $q_d(s, t)$ *aims to compute the distance between* s *and* t *based on the distance function* d*.*

Definition 4. *Given an uncertain graph* $\mathcal{G}(V, E, p)$*, a function* d*, a node* s *(*$s \in V$*), and an integer* $k > 0$*, the k-nearest neighbors query (k-NN)* $q_k(s)$ *aims to find a set of nodes* C *such that for any* $r \in C$*, the value of* $d(s, r)$ *is in the top-k list w.r.t. the function* d*.*

[1]For convenience, the subscript d can be omitted when the type of distance function is not required.

Table 2: Notations used in this paper.

Notation	Meaning
$\mathcal{G}(V, E, p)$	An uncertain graph
n, m	The sizes of V and E respectively
$G(V, E_\mathcal{G})$	A possible world from $\mathcal{G}$
STQ	The source-to-target query
SPD	The shortest path distance
$q(s, t)$	An STQ between two nodes s and t
$q_k(s)$	An k-NN query for a node s
$\phi(G)$	The diameter of the uncertain graph $\mathcal{G}$
$(\mathcal{B}, \mathcal{T})$	The tree index of an uncertain graph with the bag set $\mathcal{B}$ and tree structure $\mathcal{T}$
$g(\mathcal{G}, \epsilon, \delta)$	The sampling times for $\mathcal{G}$ with error rate ϵ and failure rate δ
$f(\mathcal{G})$	The cost function of the uncertain graph $\mathcal{G}$

In this paper, we will focus on the k-NN query. One naïve way to answer k-NN query is to run $(n-1)$ times of STQs like the way to answer $q_1(A)$ in Example 1. These STQs are called sub-queries of $q_k(s)$.

Example 1. *Consider the uncertain graph in Figure 1(a). (1) Let s=A, t=B and* $d = d_{er}$*. There are 3 kinds of possible SPD values between A and B on* $\mathcal{G}$*, that is,* $Pr(SPD = 1) = 0.7$*,* $Pr(SPD = 3) = 0.3 \times 0.9 \times 0.6 \times 0.8 = 0.1296$ *(see Figure 1(b)) and* $Pr(SPD = 0) = 0.3 \times (1 - 0.9 \times 0.6 \times 0.8) = 0.1704$*. Therefore,* $q(A, B) = (0.7 + 0.1296 \times 3)/(1 - 0.1704) = 1.31$*. (2) Let s=A, k=1 and* $d = d_{er}$*. We first calculate the three SPD values* $d_{er}(A, B) = 1.31$*,* $d_{er}(A, C) = 1.07$ *and* $d_{er}(A, D) = 2$*. Therefore,* $q_1(A) = \{C\}$*.*

Since the number of possible worlds is exponentially large, it is impractical to enumerate all the possible worlds. To alleviate this issue, people often sample a small number of possible worlds, then perform queries on these sampled graphs, and obtain the final answer by accumulating the results in a particular manner. Next, we will introduce Monte Carlo Sampling, which is the popular way for approximate query processing in an efficient way.

3.2 Monte Carlo Sampling

To address the curse of exponentiation, the *Monte Carlo (MC) sampling* is used to sample a small number of graph instances [22, 30], and estimate the query results by performing queries on these sampled graph instances. To sample an instance graph G, we can sequentially consider each edge of $\mathcal{G}$ and sample it as a deterministic edge in G following the probability distribution on the edge. Intuitively, if a possible world is repeatedly sampled multiple times, it should have a high probability to exist.

To achieve a theoretical estimation accuracy, the *Chernoff bound* [30] can be applied to determine the number of possible worlds needed for a k-NN query. Given an uncertain graph $\mathcal{G}$, a distance function d, and a pair of nodes s and t, the accuracy of estimating the value of $d(s, t)$ by MC sampling can be well guaranteed by Lemma 1:

Lemma 1. *[30] To achieve an error rate of* $\epsilon > 0$ *with a failure probability of* $\delta > 0$*, i.e.,* $\Pr(|d(s, t) - d'(s, t)| \geq \epsilon d(s, t)) \leq \delta$*, the number of samples needed is*

$$g(\mathcal{G}, s, t, \epsilon, \delta) = \max\left\{\frac{3}{\epsilon^2 d(s, t)}, \frac{\phi(\mathcal{G})^2}{2\epsilon^2}\right\} \cdot ln\left(\frac{2}{\delta}\right), \quad (3)$$

Table 3: UGD methods.

Method	Lossless	Time	Pros&cons
JSD [33]	Yes	Cubic	Smaller decomposition result, accurate but slow.
SPQR [14]	No	Linear	Smaller decomposition result but not lossless.
FWD [31]	Yes	Linear	Information lossless when $\mathcal{W} = 2$ with more redundancy.
LIN [22]	Yes	Linear	Compromise between SPQR and FWD.
PCD [5]	Yes	#*P*-hard	Layered decomposition without information loss.
PTD [16]	Yes	#*P*-hard	Layered decomposition without information loss.

where $d'(s,t)$ is the estimated value of $d(s,t)$, and the function $\phi(\mathcal{G}) = \max_{(s,t)\in V\times V} d(s,t)$ is the diameter of $\mathcal{G}$.

In practice, one usually focuses on finding the pairs with a given threshold ρ. Note that in general ρ is not too small [30], and thus we have $\frac{\phi(\mathcal{G})^2}{2\epsilon^2} \geq \frac{3}{\epsilon^2\rho}$. Therefore, the number of needed samples is:

$$g(\mathcal{G},\epsilon,\delta) = \frac{\phi(\mathcal{G})^2}{2\epsilon^2} ln\left(\frac{2}{\delta}\right). \quad (4)$$

3.3 Uncertain Graph Decomposition

To enable efficient k-NN queries, offline index are usually used [34, 36–38], often based on graph decomposition methods. In Table 2, we list all the uncertain graph decomposition (UGD) methods, including *junction scan* decomposition (JSD), SPQR decomposition [14], *lineage tree* decomposition (LIN) [22], *probabilistic core* decomposition (PCD) [5], and *probabilistic truss* decomposition (PTD) [16].

In [33], Na et al. proposed JSD[2] for dividing a deterministic graph into several partitions by finding the junction nodes. This method can also be adapted for UGD by simply regarding the probabilities of edges as their weights.SPQR [14] is named from the optimal tree obtained by decomposing the graph into a tree of tri-connected components. FWD [31] also decomposes the tree, but limits itself at bags which have at most a limited number of nodes, which is called their treewidth $\mathcal{W}$. LIN [22] behaves the same as FWD, but keeps more information in the bags for better query processing. SPQR can decompose the uncertain graph optimally, but during the process it will lose some information. FWD computes probabilities exactly in the bags, but can lead to decompositions that do not reduce much the graphs. LIN can both reduce the graph drastically and allow exact computation of probabilities, but comes at a high cost in space.

PCD [5] and PTD [16] are recently proposed layered uncertain graph decomposition methods. They can extract a dense part of the uncertain graph called (k,η)-core and (k,η)-truss respectively, which has higher probability to exist. This extraction can be executed iteratively. Note that even though the exact algorithm for finding (k,η)-core or (k,η)-truss is #P-hard, approximation algorithms are provided so that they can be completed in linear time [5, 16]. For example, there are three (k,η)-cores in Figure 3(b), and the smaller ones are more dense and reliable in the uncertain graph.

[2]Also called *partition-based road network index* in [33]. We call it JSD because it scans the junction nodes.

3.4 UGD-based Tree Index

A naïve index is to pre-compute all the pairwise distances and store them in a matrix, and then answer a query by simply looking up the matrix. This index, however, has a space complexity of $O(n^2)$, and it is not affordable if $\mathcal{G}$ is large. Thus, it is desirable to develop more effective indexes. Recently, people often build an index based on UGD. The rationality behind is that, by decomposing $\mathcal{G}$ into several sub-graphs, the k-NN query can be performed on a few sub-graphs, rather than the entire graph, resulting in high query efficiency.

To build an index for $\mathcal{G}$, a specific UGD method from Table 3 is used to decompose it into several bags, each of which contains a set of nodes of $\mathcal{G}$. The bags are then organized into a tree structure. Below, we present a formal definition of the tree index.

Definition 5. *Given an uncertain graph $\mathcal{G}(V,E,p)$, the index of $\mathcal{G}$ is a tuple $(\mathcal{B},\mathcal{T})$, where $\mathcal{B}=\{B_i|i=1,2,\cdots,l\}$ is a set of l bags from the decomposition of $\mathcal{G}$ and $\mathcal{T}$ is a tree, such that:*

(1) $\cup_{B_i\in\mathcal{B}} B_i = V$;
(2) *For each $(u,v)\in E$, there is $B_i\in\mathcal{B}, s.t.\ u,v\in B_i$;*
(3) *There is a link between B_i and B_j in $\mathcal{T}$ if $B_i\cap B_j\neq\emptyset$.*

We illustrate the index by taking FWD decomposition as an example. We first adopt FWD to compute all the bags limited by the tree width $\mathcal{W}=2$. After that, for each pair of bags, if they share some nodes, we link them with an edge. Since only one bag serves as the root, this index is a tree structure. For example, the tree index in Figure 5 is an example from the FWD decomposition of Figure 4(a). Especially, for PCD and PTD, we only link the two bags when a bag is directly a subset of another bag, to keep the tree structure of the index. For example, if (k_1,η_1)-core$\subset$ (k_2,η_2)-core and (k_2,η_2)-core $\subset$ (k_3,η_3)-core, we only link (k_1,η_1)-core with (k_2,η_2)-core and (k_2,η_2)-core with (k_3,η_3)-core.

Recall that we have discussed six UGD methods in Section 3.3. Generally, all these methods can be adopted in the index. Due to the space limitation, in this paper we mainly focus on FWD and PCD, since most of existing k-NN queries on uncertain graphs are based on expected distance search [30] or density search [35]. For the expected distance based k-NN queries (e.g., the k-NN queries based on Expected-reliable distance [30]), FWD can be used; for the density based k-NN queries (e.g., top-k reliability query [35]), PCD can be used, since it is useful for extracting the dense components (e.g., (k,η)-core in Figure 3(b)) with high existence probabilities.

4 THE U-TREE INDEXING FRAMEWORK

Even with the sampling methods or indexes proposed in Section 3.3, it is challenging to answer k-NN queries in a large uncertain graphs. However, with an advanced index, we can speed up the searching process by generating an uncertain subgraph in much smaller size, with enough information to answer the k-NN query. Since the uncertain subgraph is smaller, query answering will be more efficient.

For example, the subgraph with essential information (Figure 4(b)) to answer the query $d_{er}(B,E)$ is much smaller compared to the whole graph in Figure 4(a). Since the reduction on nodes will lead to an exponential decrease in the sampling times based on Definition 3.2, the tree index will speed up the probabilistic searching in a dramatic way. In another aspect, a tree index has a reasonable cost compared to other indexes. It can decrease the number of distances to be computed compared to the matrix index. For example, we only need to compute 3 d_{er} values in

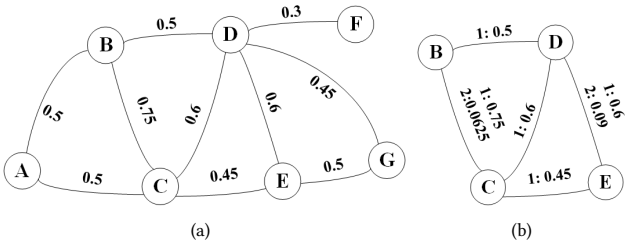


Figure 4: Tree index demonstration (a) An uncertain graph $\mathcal{G}$ (b) $\mathcal{G}(q)$ for $q = d_{er}(B, E)$.

Figure 4(a) rather than the 21 d_{er} values for each pair of nodes in $\mathcal{G}$.

We will answer two questions in this section: how to build an index to efficiently answer the k-NN queries on uncertain graphs (Step A, B in Figure 2), and how to query the index when faced with a k-NN query (Step C in Figure 2). To answer the second question, we propose a novel structure and a cost evaluation model.

4.1 U-tree Index

In this section, we propose *U-tree* for k-NN search on an uncertain graph. After decomposing the uncertain graph into several bags by Definition 5, we want to dig more on the index structure.

There are two steps to build the U-tree: the basic index construction step, directly after the decomposition (Step A), and the index customization step (Step B), which will refine the index for efficient k-NN search.

Algorithm 1: Basic index construction

Input : Uncertain graph $\mathcal{G}$
Output: $(\mathcal{B}, \mathcal{T})$

1 $\mathcal{B} \leftarrow \emptyset, \mathcal{T} \leftarrow \emptyset$;
2 $G \leftarrow$ undirected, unweighted graph of $\mathcal{G}$;
3 **for** $d \leftarrow 1$ *to* 2 **do**
4 **while** $degree(v) = d$ & $v \in G$ **do**
5 create bag B;
6 $V(B) \leftarrow v$ and all its neighbors;
7 **for** *all unmarked* $e \in V(B) \times V(B) \cap E(G)$ **do**
8 $E(B) \leftarrow E(B) \cup \{e\}$;
9 mark e;
10 **end**
11 Delete v and link v's neighbors in G;
12 $\mathcal{B} \leftarrow \mathcal{B} \cup \{B\}$;
13 **end**
14 **end**
15 Create the root bag R with all unmarked edges;
16 $\mathcal{B} \leftarrow \mathcal{B} \cup \{R\}$;
17 Organize the bags in $\mathcal{B}$ into $\mathcal{T}$ by their generating orders;
18 Add an edge between two bags in $\mathcal{T}$ if they share nodes;
19 Calculate distance distributions between nodes $v \in G$;
20 return $(\mathcal{B}, \mathcal{T})$;

Step A. First we adapt the STQ index from [22] into a k-NN index, representing the basic index construction method (see Algorithm 1). After the initialization (line 1 to 2), we iteratively add the node in degree 0 or 1 into a bag with its neighbors[3], as well as the edges among them to form a bag (line 3 to 14). Every time we add the node into one bag, it will be deleted from the original graph G. Then we create the root bag with the nodes left (line 15 to 16). Next, we link the bags and form a tree (line 17 to 18). This basic index is evaluated in the indexing framework named PTI in Section 5.

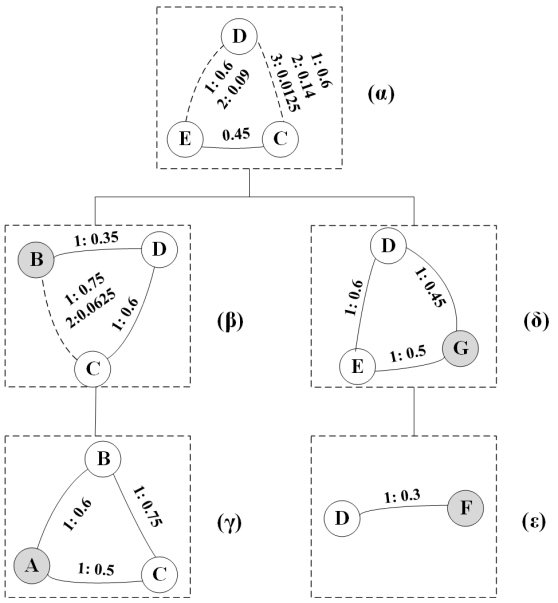


Figure 5: The basic tree index for the uncertain graph in Figure 4(a).

The index is a bottom-up structure, the root of the tree is where the query will be answered, so computations will be launched in a bottom-up way up to the root. Normally, we need several bags to answer a single sub-query; in the following, we study whether it is possible to use fewer bags to answer it after adding extra information into the index.

Note that a bag in higher level of U-tree will contain more information, because we aggregate all the information into a root bag in a bottom-up way in Algorithm 1. Consequently, only the root bag contains the complete distance distribution from the uncertain graph $\mathcal{G}$ with respect to the nodes and edges in this bag. Then we can use the root bag itself to reduce an uncertain graph; this is not true for any other bag $\mathcal{B}$ in the graph.

For example, node B in bag β of Figure 5 can only see the information from its child bag γ. Assume now we need to query $q_{er}(B, C)$. Even though we have known the distance distribution on the edge (B, C) in bag β, before we search the root bag α, we do not know if there exists another path in another part of the uncertain graph $\mathcal{G}$, that is, the subgraph of nodes $\{C, D, E, F, G\}$ in $\mathcal{G}$. So if the starting bag is far from the root, then we must search several bags before reaching the root bag, thus making the reduced sub-graph relatively large in size.

Step B. To reduce the bags to be scanned, we propose a novel method to customize the basic index from Step A. After generating the root bag, we make each edge contain distance information for both sides. Here both-side information means the bag can see the information of its child bags as well as the parent bags.

[3]We use here FWD ($\mathcal{W} = 2$) to decompose the uncertain graph into several bags. Other decomposition methods can be used with minor changes.

Then each bag will obtain a global view on $\mathcal{G}$. Therefore, if the sub-query $q(s,t)$ only concerns the nodes in one bag, we can use this single bag to answer $q(s,t)$, and thus the size of the uncertain subgraph will be significantly reduced.

We summarize U-tree in Algorithm 2. First we obtain the set of bags $\mathcal{B}$ and the tree structure $\mathcal{T}$ from the basic index (line 1). Then we initialize a queue Q and add the root bag R of $\mathcal{T}$ into Q (line 2 to 3). Next, we iteratively pop out the head P of Q until it is not a leaf node of $\mathcal{T}$ (line 4 to 7). For every edge that is shared by P and its child bags, we calculate the distance distribution in the corresponding child bags, with the help of the information provided by P (line 8 to 14).

Different from the basic index $(\mathcal{B},\mathcal{T})$ from Algorithm 1, two points are developed in the index from Algorithm 2. First, each bag is embedded with more information. Second, we actually change the structure of $\mathcal{T}$. Since every bag now can see the whole uncertain graph and thus can serve as the root, we can terminate the index searching earlier. Because fewer bags are searched and a smaller sub-graph is generated, query time is saved when running the query on this smaller sub-graph.

Algorithm 2: U-tree construction

```
   Input  : Uncertain graph G
   Output : (B, T)
1  (B, T) = Basic index construction(G);
2  Q ← ∅, P ← null;
3  Q.add(R);
4  while Q is not empty do
5      while P is null or P is the leafnode of T do
6          P ← Q.pop();
7      end
8      for each child S of P do
9          if edge e is shared by P and S then
10             Compute the distance distribution of e in S;
11             Q.add(S);
12         end
13     end
14 end
15 return (B, T);
```

For example, when answering $q_3(S)$, we need to answer the sub-query $q_{er}(A,D)$. So we start searching the U-tree in Figure 6 from bag γ and stop searching when reaching bag β. However, if we answer it by the index in Figure 5, we can only terminate the searching when reaching the root bag α. Thus, instead of generating the uncertain subgraph from $\{\alpha,\beta,\gamma\}$, a smaller uncertain subgraph is generated from $\{\beta,\gamma\}$.

Compared to the basic index, U-tree may help us utilize *graph locality* [27]. This is because the nodes, which may be queried by the users, are usually localized in some area of $\mathcal{G}$ rather than uniformly distributed. For example, if s and t is within the same bag, a single bag is enough to answer the sub-query $q(s,t)$. From graph locality, only few bags are searched when answering each sub-query and thus the query time is significantly reduced.

The resulting extra cost is reasonable: the space complexity is still linear, and the index updating cost is only increased linearly. Using this efficient index, we can design the query algorithm for k-NN queries now.

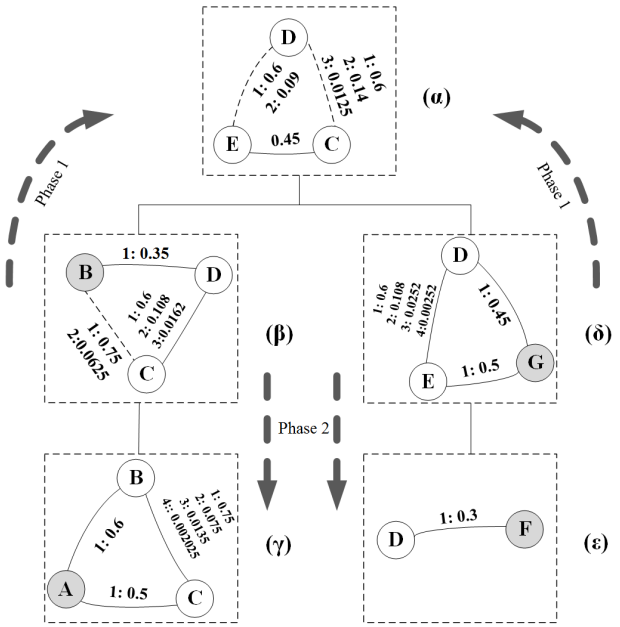


Figure 6: U-tree for the uncertain graph in Figure 4(a).

Algorithm 3: U-tree query algorithm

```
   Input  : U-tree (B, T), source node s ∈ V, integer k
   Output : the k-NN set C
1  C ← ∅;
2  From s to get r on G;
3  Generate E-table and update the bags information;
4  Re-order E-table;
5  Initiate a queue Q and load the tuples into Q;
6  Perform r-pruning;
7  if Q = ∅ then
8      return NULL
9  end
10 p ← Q.pop();
11 while Q ≠ ∅ do
12     q ← Q.pop();
13     if Cost(p ∪ q) > Cost(p) + Cost(q) then
14         p ← merge(p, q);
15     else
16         Generate G_p and answer tuple p by sampling G_p;
17         p ← q;
18     end
19     Perform r-pruning in Q;
20 end
21 Add the nodes with top-k smallest d_er values into C;
22 return C;
```

4.2 Query processing by E-table

To analyze the search process on U-tree, and then develop an efficient query algorithm, we propose a novel data structure called *E-table* (from *execution table*) to keep track of the indexing querying process.

Definition 6. *Given a k-NN query $q_k(s)$, an Execution-table (E-table) is a collection of tuples $u = \{id, t, r, B\}$ that:*
1. $u.id$ is the identifier for u.

2. $u.t$ is the target node in the sub-query $q(s,t)$ mapped to u.
3. $u.r$ is the lower bound for $d(s,t)$.
4. $u.B$ is the set of minimum bags to answer $q(s,t)$.

Note that each tuple u in the E-table corresponds to a sub-query $q(s,t)$ for the given k-NN query $q_k(s)$. A k-NN query $q_k(s)$ can be divided into $(n-1)$ sub-queries $q(s,t_i)$ where $t_i \in V$ and $t_i \neq s$, which are actually STQs. For example, the d_{er} based NN query[4] can be divided into $(n-1)$ STQs and then return the node with the minimum d_{er} value. We demonstrate the use of E-table in Example 2.

Example 2. *To find the nearest neighbors for node B in Figure 4(a), we can build a E-table like Table 4, in which each row is a sub-query and each column is an attribute of the sub-query, including the id, the target node, the lower bound for d_{er}, and the bags needed to answer this sub-query. Note that for $d_{er}(s,t)$, r can be calculated from $SPD(s,t)$ since $SPD(s,t)$ can serve as the lower bound of $d_{er}(s,t)$.*

Table 4: E-table for NN(B).

id	t	r	B	Sub-query
1	A	1	$\{\alpha,\beta,\gamma\}$	$d(B,A)$
2	C	1	$\{\alpha,\beta\}$	$d(B,C)$
3	D	1	$\{\alpha,\beta\}$	$d(B,D)$
4	E	2	$\{\alpha,\beta\}$	$d(B,E)$
5	F	2	$\{\alpha,\beta,\delta,\epsilon\}$	$d(B,F)$
6	G	2	$\{\alpha,\beta,\delta\}$	$d(B,G)$

Obviously, it will be very time-consuming if we run every STQ in the E-table. Alternatively, we find that a tuple in the E-table can actually answer several sub-queries. For example, in Table 4, we can find that bag α and β contain enough information to answer three sub-queries, that is, the $d_{er}(B,C)$, $d_{er}(B,D)$ and $d_{er}(B,E)$.

For a k-NN query to find the nodes t with top-k smallest $d(s,t)$, we can filter the tuples u_i whose r_i is bigger than the current maximum $d(s,t_k)$ named as d_{max}. Here t_k is the node that has been searched and added in the k-NN set to be returned. We name the k-NN set to be returned as C. It is actually the candidate set before it is returned. Note that if C is not full, the current maximum $d(s,t_k)$ is set to be $+\infty$. The filtering theory can be formalized by Lemma 2.

Lemma 2. *All the tuples t_i whose $r_i > d(s,C)$ can be safely pruned if $|C| > k$ is satisfied. Here $d(s,C) = \max_{i \in C} d(s,i)$.*

The lemma can be easily proved by the fact that the k-NN solution can only be found in the subset whose r value is not bigger than the current $d_{max} = d_{er}(s,C)$. We call this pruning technique r-pruning.

Thus, our goal is to find the proper C and use it to prune the tuples in the E-table as more as possible, especially the tuple with huge amount of bags. For example, it will be nice if tuple 5 and 6 in Table 4 are all pruned at the very first stage because these two tuples will generate uncertain graphs of bigger size.

We follow two steps to generate the candidate set C. First, we rank the E-table according to r. Then, we rank the tuples according to the number of bags, that is, the tuple with fewer bags will be searched first even they have the same r value. It is from the instinct that the tuple u_i with fewer bags will generate a smaller uncertain sub-graph and thus may terminate in an earlier stage. We demonstrate the process in Example 3.

[4]NN query is the k-NN query when $k = 1$.

Table 5: Executed tuples in Table 4.

id	t	r	B	sub-query	$d_{er}(s,t)$
1	A	1	$\{\alpha,\beta,\gamma\}$	$d(B,A)$	1.3 340
2	C	1	$\{\alpha,\beta\}$	$d(B,C)$	1.1631
3	D	1	$\{\alpha,\beta\}$	$d(B,D)$	1.4708

Example 3. *For a NN query q_1s, we load the first tuple u_1 and generate the corresponding reduced uncertain graph $\mathcal{G}_1$ from B_1. Then, we sample the reduced uncertain graph $\mathcal{G}_1$ and get the result of $d_{er}(s,t_1)$. We add t_1 into C and update $d_{max} = d_{er}(s,t_1)$. Next, we prune the tuples whose r value is even bigger than $d_{er}(s,t_1)$. So on so forth, we load the tuple u_i which is not filtered, and generate $\mathcal{G}_i$ from B_i to answer $d_{er}(s,t_i)$ and then launch tuple filtering and if a smaller $d_{er}(s,t_i)$ is found, update $C = t_i$ and $d_{max} = d_{er}(s,t_i)$. After searching all the tuples that are not filtered, we can return C.*

4.3 Query scheduling by cost evaluation

In the steps above, we notice that the set of bags of one sub-query maybe the upper set of another sub-query, that is $B_i \subset B_j$ happens time to time. At this time, we can safely use the upper set B_j to answer u_i and u_j. For example, the uncertain graph generated from u_1 contains all the information needed to answer u_2, u_3 and u_4. However, some sub-queries' bag sets have overlap but not belong to each other, e.g., B_1 and B_6. Faced with this condition, we have two choices. The first choice is to combine these two tuples and generate an uncertain graph of big size to answer these two sub-queries. The second choice is to generate two small uncertain graphs and answer the two sub-queries respectively. Thus, we can choose the one with smaller cost. The cost evaluation is summarized by the function below.

Definition 7. *Cost Evaluation is a function f to compare the cost of combing two tuples with the cost of answering them respectively. Here each uncertain graph $\mathcal{G}_i$ is generated by the corresponding tuple t_i, and $\mathcal{G}_{i\cup j}$ is generated by $B_i \cup B_j$. If $f(B_i \cup B_j) < f(B_i) + f(B_j)$ then we will use B_i and B_j to generate a single uncertain subgraph. Otherwise, we will generate two uncertain subgraphs respectively.*

$$f(B_i \cup B_j) = g(\mathcal{G}_{i\cup j}, \epsilon, \delta) \cdot |E(\mathcal{G}_{i\cup j})| \tag{5}$$

$$f(B_i) + f(B_j) = \sum_{p=i,j} g(\mathcal{G}_p, \epsilon, \delta) \cdot |E(\mathcal{G}_p)| \tag{6}$$

If we combine all the possible tuples to generate a single uncertain graph which contains all the information to answer the k-NN query, then this reduced uncertain graph is called *Equivalent Uncertain Graph* (EUG) of $\mathcal{G}$ for the given query. Figure 7 is the EUG for $q_1(B) = NN(B)$ on the uncertain graph in Figure 4(a). When the candidate set is big in size or the k-NN query has a large k value, the EUG cannot be reduced much. This shows another reason why we need a cost evaluation.

According to section 3.2, $\phi(\mathcal{G})$ can be roughly approximated by $(n-1)$, then the cost function can be written as $f(\mathcal{G}) = (n-1)^2 \cdot m$.

Example 4. *If there comes three tuples in the E-table, and they need the bags $\{\alpha,\beta,\gamma\}$, $\{\alpha,\delta,\epsilon\}$ and $\{\alpha,\beta,\delta\}$ respectively (see Figure 8). Thus we get three reduced uncertain graph $\mathcal{G}_1$, $\mathcal{G}_2$ and $\mathcal{G}_3$.*

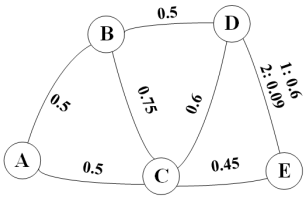
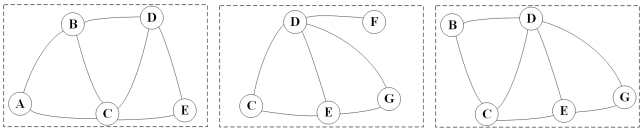


Figure 7: The EUG for Figure 4(a) considering NN(B).



(a) (b) (c)

Figure 8: The uncertain graph generated from bags (a) $\{\alpha, \beta, \gamma\}$, (b) $\{\alpha, \delta, \epsilon\}$ and (c) $\{\alpha, \beta, \delta\}$.

We can merge these reduced graphs to answer multiple tuples or answer them one by one. There are 5 ways to answer these tuples and the costs are listed below.

$f(\mathcal{G}_1, \mathcal{G}_2, \mathcal{G}_3) = f(\mathcal{G}_1) + f(\mathcal{G}_2) + f(\mathcal{G}_3) = 4^2 \times (7 + 6 + 7) = 320$
$f(\mathcal{G}_1 \cup \mathcal{G}_2, \mathcal{G}_3) = f(\mathcal{G}_1 \cup \mathcal{G}_2) + f(\mathcal{G}_3) = 6^2 \times 10 + 4^2 \times 7 = 472$
$f(\mathcal{G}_1, \mathcal{G}_2 \cup \mathcal{G}_3) = f(\mathcal{G}_1) + f(\mathcal{G}_2 \cup \mathcal{G}_3) = 4^2 \times 7 + 5^2 \times 8 = 312$
$f(\mathcal{G}_2, \mathcal{G}_1 \cup \mathcal{G}_3) = f(\mathcal{G}_2) + f(\mathcal{G}_1 \cup \mathcal{G}_3) = 4^2 \times 6 + 5^2 \times 9 = 321$
$f(\mathcal{G}_1 \cup \mathcal{G}_2 \cup \mathcal{G}_3) = f(\mathcal{G}_1 \cup \mathcal{G}_2 \cup \mathcal{G}_3) = 6^2 \times 10 = 360$

Here we see the optimal solution for this E-table is not to answer each tuple respectively or to generate an EUG, but only to merge $\mathcal{G}_2$ and $\mathcal{G}_3$. Therefore, we form Algorithm 3 to find the local optimal solution of the minimum cost. We find that the local optimal solution works well; on the other hand, finding the global optimal solution is relatively cumbersome.

Given the initial uncertain graph $\mathcal{G}_0 = (V_0, E_0, p)$ and two reduced uncertain graph $\mathcal{G}_1 = (V_1, E_1, p)$, $\mathcal{G}_2 = (V_2, E_2, p)$, the function $merge(\mathcal{G}_1, \mathcal{G}_2)$ generates an uncertain graph $\mathcal{G}_{1,2} = (V1 \cup V_2, E, p)$ where $E = (V1 \cup V_2)^2 \cap E_0$. The function *pop* will pop up the queue's head and delete it from the queue.

Discussion. We notice that the basic index has a fixed root, that is, every time when the index is queried, we need to reach the root. But every bag in the U-tree can serve as the root, meaning the index retrieval can be done in any part of the tree. So when the query node is far away from the root, the query time from the basic index will become too high, because many bags are required and the reduced subgraph will not decrease much in size. U-tree overcomes this drawback but the index construction time is increased. Therefore, we study whether there is any trade-off and balance between the two indexes, and then we can enjoy both the quick indexing and efficient querying. We answer this question by graph locality. With the help of the history queries, we can obtain the popular nodes which may be frequently queried [18]. And with the help of graph locality, most k-NN queries will be around these nodes. Then we can build multiple basic indexes rooted on each popular node. Every time a query comes, it can be assigned to an index with the smallest cost. In this way, the cost of index updating is reduced and the query is accelerated.

5 EXPERIMENTAL EVALUATION

We now present the experimental results. We first describe the datasets in Section 5.1, then introduce the competitors in Section 5.2, and finally report the experimental results on indexing cost, query time, time proportion, accuracy and generality in Sections 5.3, 5.4, 5.5, 5.6, and 5.7 respectively.

5.1 Datasets

5.1.1 Real-world Graphs. We use six real-world datasets. Table 6 reports their statistics. All datasets are transformed into undirected uncertain graphs with meaningful probability distribution on each edge. Note that PPI, DBLP and FBN have Bernoulli distributions on each edge, whereas BJT has no limitation on the type of the distribution on each edge, and they do not follow any standard distributions [8].

For each uncertain graph, we calculate the diameter to better show the size and density of the dataset. To calculate the diameter, we ignore the probabilities and treat each uncertain graph as an unweighted graph, and then find the largest SPD. For the dataset with multiple connected components, we report the one with the biggest diameter. We evaluate scalability of the indexing framework by testing its performance when faced with datasets with different size (see Table 6) and different density (See Table 7).

PPI[5] We have two protein-protein interaction (PPI) networks (PPIC, PPIK) [25]. We use these networks in which a node denotes a protein and an edge denotes a possible interaction encoded with a specific probability calculated from biology experiments, meaning the existence confidence of this edge. *Collins*'s network PPIC is the core dataset which is believed to have strong interactions among the nodes [7]. *Krogan*'s network PPIK, however, contains less reliable interactions and thus is bigger in size [20].

DBLP[6] It is a subset of the co-authorship network. The nodes denote the authors and if two authors have coauthored a paper, there will be an edge between them. We follow the popular way of generating co-authorship probability between two authors [19]. If two authors coauthored c times, the probability is $1-e^{-c/\mu}$ on the corresponding edge where smaller μ means smaller probabilities in general. Following the trend [19], we adopt $\mu = 5$ which makes the most edges with an existing probability around 0.05.

FBN[7] The dataset called Facebook-like Social Network is from an online community for students at University of California, Irvine [26]. It includes the users who sent or received at least one message. Each node denotes a user and if there are messages between two users, there will be an edge. The probability on an edge means the probability of the corresponding users having an interaction. The method to generate the probability is same as the method in DBLP but we set $\mu = 2$ to make the most edges with a probability around 0.4.

BJT[8] The dataset is generated from mapping 15 million Beijing Taxi trajectories [39, 40] on a Beijing map with about 3 million road segments (BJT) and 1 million road segments (BJTA, only the arterial roads) respectively [33]. The probability distribution on an edge means the speed distribution of the vehicles on the corresponding road segment. It can be generated by analyzing the trajectories [8]. Note that the diameters of BJT and BJTA is obviously bigger than the other datasets, since long roads (tens

[5] http://www.nature.com.eproxy1.lib.hku.hk/nmeth/journal/v9/n5/full/nmeth.1938.html
[6] http://www.informatik.uni-trier.de/~ley/db/
[7] https://toreopsahl.com/datasets/
[8] https://www.microsoft.com/en-us/research/publication/t-drive-trajectory-data-sample/

Table 6: Statistics of Real-world Datasets.

Graph	*#Nodes*	*#Edges*	*Diameter*	*Degree*
PPIC	1,622	9,074	15	11.2
PPIK	3,672	14,317	10	7.8
FBN	22,016	58,595	10	5.3
DBLP	317,080	1,033,668	23	6.6
BJTA	426,196	946,434	3,851	4.4
BJT	1,285,215	2,690,296	10,529	4.2

Table 7: Statistics of Synthetic Datasets.

Graph	*#Nodes*	*#Edges*	*Diameter*
S_1	1,000,000	1,115,373	15
S_2	1,000,000	1,672,561	15
S_3	1,000,000	2,433,185	15
S_4	1,000,000	3,128,529	15
S_5	1,000,000	3,728,130	15
S_6	1,000,000	4,213,833	15

of kilometers) will be separated into many road segments (ten meters).

5.1.2 Synthetic Graphs. Given an unweighted graph, algorithms from [4] can generate a graph with disturbances on the edges and calculate a probability of existence for each edge. The algorithm from [23] can encrypt a given weighted graph into an uncertain graph encoded with a probability on each edge.

Here we encrypt a synthetic graph generated from the *Barabási-Albert model* [3], which is a widely used model to simulate real graphs. To vary the density of the graph, we set the diameter as 15 but vary the number of the edges for each synthetic graph. Then the unweighted graphs with different size are encrypted by $(20, 10^{-3})$-*obfuscation* [4]. We use these graphs to test the scalability of the indexes.

5.2 Competitors

In our experiments, we try to find the k-nearest neighbors for 100 randomly generated query nodes. We use three competitors: Incremental k-NN (IKNN) from [30], *RQ*-tree reliability search (RQRS) from [19], and the basic index (see Algorithm 1) adapted from *ProbTree* indexing framework (PTI) [22].

We adapt PTI into a single-source version to deal with k-NN queries. They have been discussed in the related works. IKNN is popular in finding k-NN in the uncertain graphs, and RQRS is the state-of-the-art probabilistic k-NN index based on reliability search. PTI is the current indexing framework which can adopt different decomposition methods and distance functions.

We implement the whole U-tree indexing framework in Java, and run experiments on a machine having a 4-core Intel i5-3570 3.40GHz processor and 16GB of memory, with Ubuntu installed.

5.3 Evaluation on Indexing Cost

Here we evaluate the space cost and the index construction time of U-tree, PTI and RQRS.

The space cost is composed by the tree structure and the distance distribution stored in the bags. From Figure 9, we see that the U-tree index needs more memory since it stores more distance distributions into each bag, and RQRS needs more memory because it stores the whole node set at each level of the tree index. Compared with the query efficiency improvement in Figure 13,

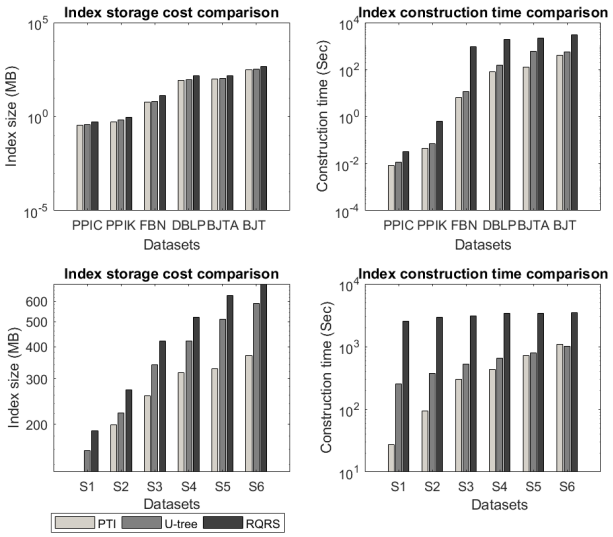


Figure 9: Scalability Evaluation on index size and index construction Time on real-world datasets (top) and synthetic datasets (bottom).

we find that the extra memory cost in U-tree index is reasonable, because the space cost rises 23% but the query time is 75% faster on average.

We compare the time cost to construct each index in Figure 9. To avoid affecting the comparison, here we only evaluate the time to construct the tree index and the uncertain graphs are assumed to be decomposed beforehand. Note that RQRS needs to recursively perform balanced bi-partition clustering, which can also be seen as a decomposition method.

To test the scalability of U-tree, we use the synthetic graph with the graph anonymization technique. We vary the size of the synthetic graph (see Table 7) and compare the index construction time and index storage in Figure 9. *Semilog* is used on the *y-axis* because of the big difference of the index size and construction time when the number of nodes sharply changes. From the figure, we find the index construction cost of U-tree is linear to the number of nodes in the uncertain graph. This shows good scalability over uncertain graphs of different size.

5.4 Evaluation on Query Time

Here we evaluate the query time on six real-world networks and six synthetic datasets. We use the popular probabilistic k-NN algorithm IKNN [30] to show the efficiency of the query time without index acceleration, and the ability of U-tree, PTI and RQRS to speed up the searching. This comparison makes sense because they all need to find the probability distribution among nodes, which is the most expensive part in the algorithms.

In Figure 13, we vary the k value to see the changes of each curve. For each k value, we randomly pick 100 nodes as the source nodes to initiate 100 probabilistic k-NN queries, and calculate the average query time as the value on *y-axis*. We use *semilog* on the query time because of the wide range between different methods.

From Figure 13, U-tree is superior to the other methods, and the index based methods have a much smaller query time than IKNN (about two magnitudes). However, the curve of RQRS and that of PTI index twist together. This might be because both RQRS

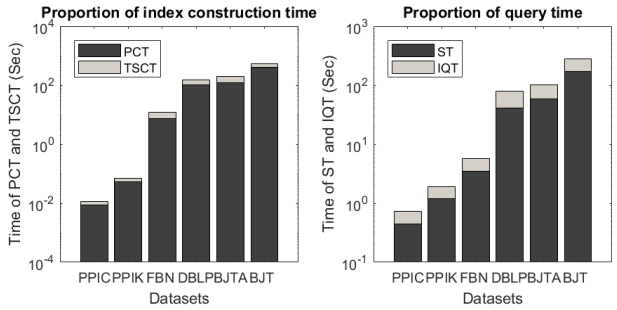


Figure 10: Proportion of (a) probability computation time out of index construction time (b) index retrieval time out of query time.

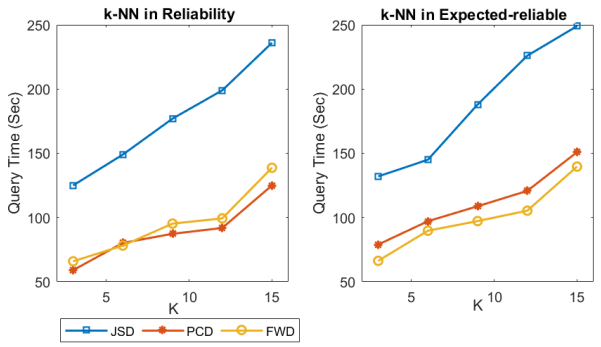


Figure 11: Comparison of (a) k-NN query in d_{re} and (b) k-NN query in d_{er} on the U-tree from JSD, PCD and FWD.

and PTI construct a tree whose height $h \in [10, 20]$ and they all have to search on the tree from the leaves until reaching the big root. These similarities make their retrieval time in the same level. Also, the figures show good scalability of U-tree despite of the dataset size and density.

5.5 Evaluation on Time Proportion

From Figure 10 we see that the Probability Computation Time (PCT) takes the most part of the index construction time and the Sampling Time (ST) takes the most part of the query time. Note that the query time minus ST is the Index Query Time (IQT) and the index construction time minus PCT is the Tree Structure Construction Time (TSCT).

It makes sense because in the index construction period, to calculate the distance distributions is required in both phase one and phase two which asks for sampling the corresponding subgraphs. Also, when we run the queries on the index, the bags containing essential information are collected and then sample the reduced uncertain subgraph.

Since bigger proportion of TSCT means more complex index structure, e.g., a bigger tree height, and thus requires more time to answer a k-NN query. Also, IQT is the time to be spent on searching the index. The bigger proportion of an IQT, the bigger probability that more bags are required to generate the uncertain subgraph, thus making the query time unbearable. From Figure 10, U-tree performs good when searching the index and answering the k-NN queries.

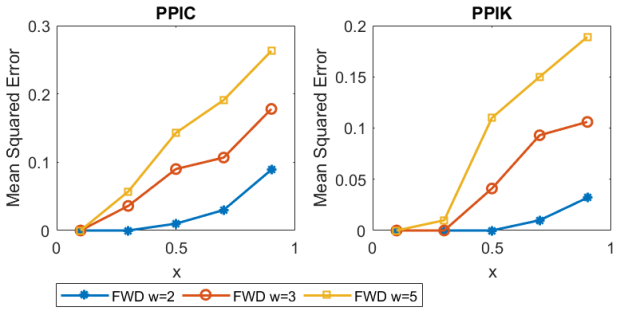


Figure 12: Indexing framework accuracy evaluation.

5.6 Evaluation on Accuracy

In U-tree indexing framework, there are two parts that may cause precision loss. The first part is the choice of decomposition methods. If a decomposition method which may lose information is used, e.g., FWD with $\mathcal{W} > 2$, the uncertain subgraph to be generated from searching the index will become not accurate, thus making error in k-NN results. The second part is the sampling method. Since sampling is a technique for approximate query processing, it will lose information while accelerating the querying process. In general, less sampling times will lead to bigger error in the k-NN results.

Here we vary the FWD tree width $\mathcal{W}$ and construct the corresponding U-tree respectively. Then, we run k-NN query on these U-trees with different pair of parameters (ϵ, δ) in the sampling. We set $(\epsilon, \delta) = (x, x)$ and vary x from 0.1 to 0.9.

The k-NN results are collected and compared with the k-NN result from a baseline which simply runs big number of samplings times to find the k-nearest neighbors on uncertain graphs. We run the experiment on PPIC and PPIK and d_{er} is used. Here k is set to 100 and the Mean Squared Error is reported.

From Figure 12, we see that the Mean Squared Error is generally small even when the sampling parameters are setting in a bad way, e.g., ϵ and δ are very near to 1. Especially, with parameters below 0.25, we can achieve high accurate k-NN results even with decomposition methods that may lose information.

5.7 Evaluation on Framework Generality

To show the generality of the indexing framework, we will use another distance function and another decomposition method to construct U-tree. We run two different k-NN queries on each U-tree, i.e., the k-NN query which asks for the set of nodes with top-k biggest reliabilities d_{re} from the query node, and the k-NN query which asks for the k-nearest neighbors in Expected-reliable distance d_{er}.

We use three different decomposition methods to construct U-tree, i.e., JSD which can find the nodes and edges to divide the graph into several partitions [33], PCD (see Figure 3(b)), and FWD where we set the tree width $\mathcal{W} = 2$ [22].

We run this evaluation on DBLP. The results are collected in Figure 11. We can see that the U-tree with PCD and the U-tree with FWD are similar in performance,and PCD has a little advantage over FWD for k-NN in d_{re} while FWD is better for k-NN in d_{er}. The above two U-trees have significant advantage over the U-tree with JSD, since JSD performs poor in capturing hierarchical structure of an uncertain graph.

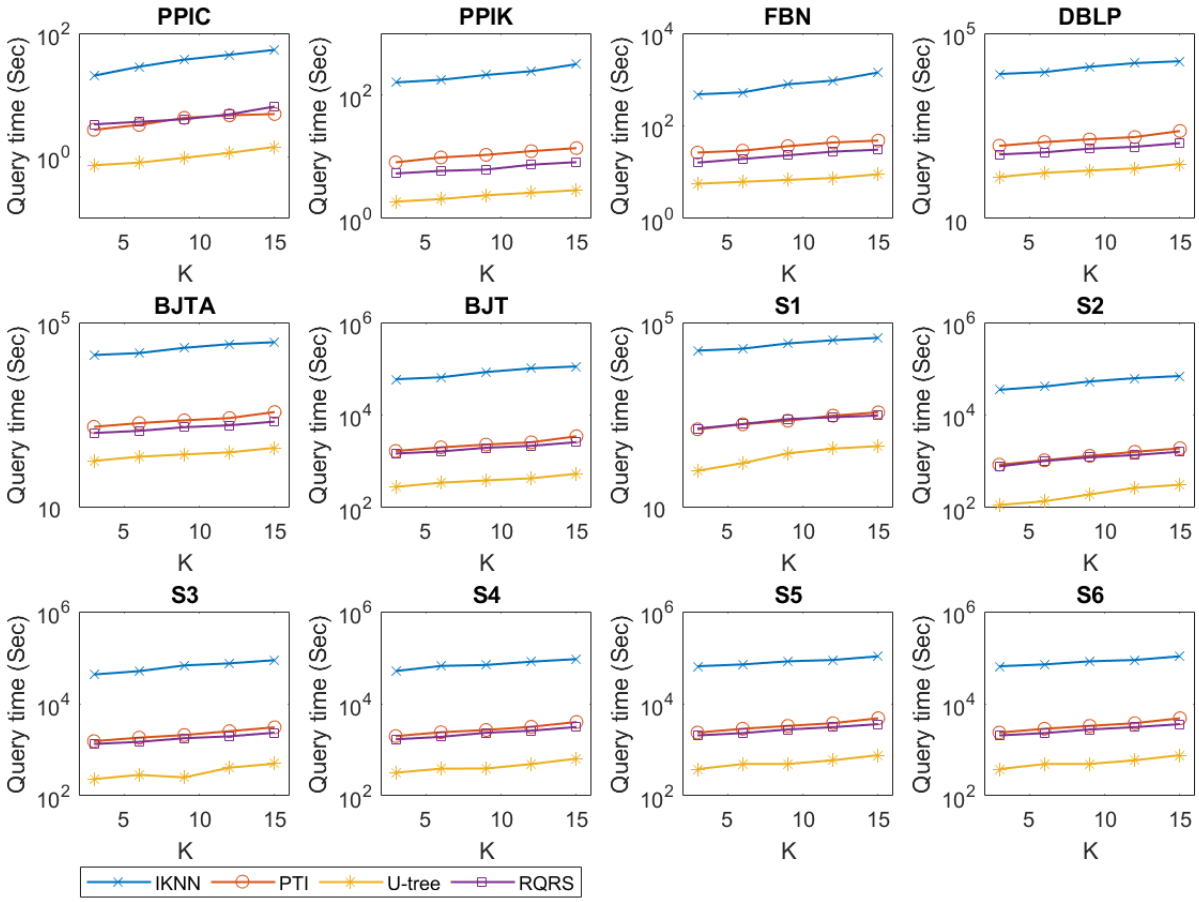


Figure 13: Query efficiency comparison.

6 CONCLUSION

In this paper, we examine the k-NN query on uncertain graphs. We first propose a generic index which can be built with several uncertain graph decomposition methods. Then, based on this index, we develop an efficient k-NN query algorithm, which supports various probabilistic distance functions. Finally, we evaluate the proposed indexing frameworks on both real and synthetic large uncertain graphs. The experimental results show that our index is effective and scalable to large graphs, and our query algorithm is very efficient.

In the future, we will study how to efficiently maintain the index for dynamic graphs, in which the nodes and edges are inserted and deleted frequently. Another interesting direction is to study how to extend the indexing framework for answering k-NN queries on a distributed platform. The research on the insight of the uncertain graph decomposition will be valuable, since a better decomposition method will largely benefit the index performance. We will also perform more experimental evaluations on other real uncertain graphs.

A APPENDIX

In this appendix, we present the algorithm listing of the competitors from the research literature.

Algorithm 4: Incremental k-NN (IKNN)

Input : uncertain graph $\mathcal{G} = (V, E, P, W)$, query node $s \in V$, sampling times r, distance increment γ, k
Output : k-NN result T_k

$T_k \leftarrow \emptyset, D \leftarrow 0$;
Initiate r executions of *Dijkstra* from s;
while $|T_k| < k$ **do**
  $D \leftarrow D + \gamma$;
  for $i \leftarrow 1$ *to* r **do**
    Continue visiting nodes until reaching D by *Dijkstra*;
    for *each node* $t \in V$ *visited* **do**
      Update the distance distribution and get d_{er};
    end
  end
  for *node* $t \notin T_k$ **do**
    if $d_{er}(s, t) < D$ **then**
      $T_k \leftarrow T_k \cup \{t\}$
    end
  end
end
return T_k;

Algorithm 5: *RQ*-tree reliability search (RQRS)

Input : uncertain graph $\mathcal{G} = (V, E, P)$, query node $s \in V$, sampling times r, k

Output : k-NN result T_k

$T_k \leftarrow \emptyset$;

Initiate *RQ-tree* based on binary clustering;

Compute the upper bound U_{out} for each cluster C;

Generate the candidate set C^*;

Travel the clusters in *RQ-tree* in bottom-up until $C^*(\{s\}, \eta) = \arg \max_{\{S\} \subseteq C, U_{out}(\{s\}, C) < \eta} U_{out}(\{s\}, C)$;

Get the reduced graph G' by C^*;

Initiate r executions of *DFS* on G' from s;

Calculate the reliability and select the biggest *top-k* as T_k;

return T_k;

Algorithm 6: *ProbTree* indexing (PTI) ($\mathcal{G}$)

Input : uncertain graph $\mathcal{G} = (V, E, P)$, query node $s \in V$, sampling times r, k

Output : k-NN result T_k

 # Index construction:

Run Algorithm 1 to build the *FWD* tree with $\mathcal{W} = 2$;

 # Information Retrieval:

$T_k \leftarrow \emptyset$;

for $v \in V$ **do**

  Calculate $d_{er}(s, v)$ with the help of the *FWD* tree;

end

Put the k nodes with the *top-k* smallest $d_{er}(s, v)$ in T_k;

return T_k;

REFERENCES

[1] Eytan Adar and Christopher Re. 2007. Managing uncertainty in social networks. *IEEE Data Eng. Bull.* 30, 2 (2007), 15–22.

[2] Saurabh Asthana, Oliver D King, Francis D Gibbons, and Frederick P Roth. 2004. Predicting protein complex membership using probabilistic network reliability. *Genome research* 14, 6 (2004), 1170–1175.

[3] Albert-László Barabási and Réka Albert. 1999. Emergence of scaling in random networks. *science* 286, 5439 (1999), 509–512.

[4] Paolo Boldi, Francesco Bonchi, Aristides Gionis, and Tamir Tassa. 2012. Injecting uncertainty in graphs for identity obfuscation. *Proceedings of the VLDB Endowment* 5, 11 (2012), 1376–1387.

[5] Francesco Bonchi, Francesco Gullo, Andreas Kaltenbrunner, and Yana Volkovich. 2014. Core decomposition of uncertain graphs. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 1316–1325.

[6] Reynold Cheng, Yixiang Fang, and Matthias Renz. 2014. *Data Classification: Algorithms and Applications.* Chapman & Hall CRC Data Mining and Knowledge Discovery Series, New York, USA, Chapter Uncertain Data Classification.

[7] Sean R Collins, Patrick Kemmeren, Xue-Chu Zhao, Jack F Greenblatt, Forrest Spencer, Frank CP Holstege, Jonathan S Weissman, and Nevan J Krogan. 2007. Toward a comprehensive atlas of the physical interactome of Saccharomyces cerevisiae. *Molecular & Cellular Proteomics* 6, 3 (2007), 439–450.

[8] Jian Dai, Bin Yang, Chenjuan Guo, Christian S Jensen, and Jilin Hu. 2016. Path cost distribution estimation using trajectory data. *Proceedings of the VLDB Endowment* 10, 3 (2016), 85–96.

[9] Yixiang Fang, Reynold Cheng, Xiaodong Li, Siqiang Luo, and Jiafeng Hu. 2017. Effective community search over large spatial graphs. *Proceedings of the VLDB Endowment* 10, 6 (2017), 709–720.

[10] Yixiang Fang, Reynold Cheng, Siqiang Luo, and Jiafeng Hu. 2016. Effective community search for large attributed graphs. *Proceedings of the VLDB Endowment* 9, 12 (2016), 1233–1244.

[11] Yixiang Fang, Reynold Cheng, Wenbin Tang, Silviu Maniu, and Xuan Yang. 2016. Scalable algorithms for nearest-neighbor joins on big trajectory data. *IEEE Transactions on Knowledge and Data Engineering* 28, 3 (2016), 785–800.

[12] Yixiang Fang, Haijun Zhang, Yunming Ye, and Xutao Li. 2014. Detecting hot topics from Twitter: A multiview approach. *Journal of Information Science* 40, 5 (2014), 578–593.

[13] Joy Ghosh, Hung Q Ngo, Seokhoon Yoon, and Chunming Qiao. 2007. On a routing problem within probabilistic graphs and its application to intermittently connected networks. In *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*. IEEE, 1721–1729.

[14] Carsten Gutwenger and Petra Mutzel. 2001. A linear time implementation of SPQR-trees. In *Graph Drawing*. Springer, 77–90.

[15] Jiafeng Hu, Reynold Cheng, Zhipeng Huang, Yixiang Fang, and Siqiang Luo. 2017. On Embedding Uncertain Graphs. In *26th ACM Conf. on Information and Knowledge Management (ACM CIKM 2017)*.

[16] Xin Huang, Wei Lu, and Laks VS Lakshmanan. 2016. Truss decomposition of probabilistic graphs: Semantics and algorithms. In *Proceedings of the 2016 International Conference on Management of Data*. ACM, 77–90.

[17] Zhipeng Huang, Yudian Zheng, Reynold Cheng, Yizhou Sun, Nikos Mamoulis, and Xiang Li. 2016. Meta structure: Computing relevance in large heterogeneous information networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 1595–1604.

[18] Theodore Johnson and Dennis Shasha. 1994. X3: A low overhead high performance buffer management replacement algorithm. In *Proceedings of VLDB*.

[19] Arijit Khan, Francesco Bonchi, Aristides Gionis, and Francesco Gullo. 2014. Fast Reliability Search in Uncertain Graphs.. In *EDBT*. 535–546.

[20] Nevan J Krogan, Gerard Cagney, Haiyuan Yu, Gouqing Zhong, Xinghua Guo, Alexandr Ignatchenko, Joyce Li, Shuye Pu, Nira Datta, Aaron P Tikuisis, et al. 2006. Global landscape of protein complexes in the yeast Saccharomyces cerevisiae. *Nature* 440, 7084 (2006), 637–643.

[21] Xiang Li, Yao Wu, Martin Ester, Ben Kao, Xin Wang, and Yudian Zheng. 2017. Semi-supervised clustering in attributed heterogeneous information networks. In *Proceedings of the 26th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee.

[22] Silviu Maniu, Reynold Cheng, and Pierre Senellart. 2017. An Indexing Framework for Queries on Probabilistic Graphs. *ACM Transactions on Database Systems (TODS)* 42, 2 (2017), 13.

[23] Xianrui Meng, Seny Kamara, Kobbi Nissim, and George Kollios. 2015. GRECS: graph encryption for approximate shortest distance queries. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*.

[24] Naoki Nariai, Eric Kolaczyk, and Simon Kasif. 2007. Probabilistic protein function prediction from heterogeneous genome-wide data. *PLoS One* (2007).

[25] Tamás Nepusz, Haiyuan Yu, and Alberto Paccanaro. 2012. Detecting overlapping protein complexes in protein-protein interaction networks. *Nature methods* 9, 5 (2012), 471–472.

[26] Tore Opsahl and Pietro Panzarasa. 2009. Clustering in weighted networks. *Social networks* 31, 2 (2009), 155–163.

[27] Dominic Pacher, Robert Binna, and Günther Specht. 2011. Data Locality in Graph Databases through N-Body Simulation.. In *Grundlagen von Datenbanken*. Citeseer, 85–90.

[28] Panos Parchas, Francesco Gullo, Dimitris Papadias, and Franceseco Bonchi. 2014. The pursuit of a good possible world: extracting representative instances of uncertain graphs. In *Proceedings of the 2014 ACM SIGMOD international conference on management of data*. ACM, 967–978.

[29] Sriram Pemmaraju and Steven S Skiena. 2003. *Computational Discrete Mathematics: Combinatorics and Graph Theory with Mathematica®*. Cambridge university press.

[30] Michalis Potamias, Francesco Bonchi, Aristides Gionis, and George Kollios. 2010. K-nearest neighbors in uncertain graphs. *Proceedings of the VLDB Endowment* 3, 1-2 (2010), 997–1008.

[31] Neil Robertson and Paul D Seymour. 1984. Graph minors. III. Planar tree-width. *Journal of Combinatorial Theory, Series B* 36, 1 (1984), 49–64.

[32] Petteri Sevon, Lauri Eronen, Petteri Hintsanen, Kimmo Kulovesi, and Hannu Toivonen. 2006. Link discovery in graphs derived from biological databases. In *International Workshop on Data Integration in the Life Sciences*. Springer.

[33] Na Ta, Guoliang Li, Yongqing Xie, Changqi Li, Shuang Hao, and Jianhua Feng. 2017. Signature-based trajectory similarity join. *IEEE Transactions on Knowledge and Data Engineering* 29, 4 (2017), 870–883.

[34] Silke Trißl and Ulf Leser. 2007. Fast and practical indexing and querying of very large graphs. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*. ACM, 845–856.

[35] Leslie G Valiant. 1979. The complexity of enumeration and reliability problems. *SIAM J. Comput.* 8, 3 (1979), 410–421.

[36] Haixun Wang, Hao He, Jun Yang, Philip S Yu, and Jeffrey Xu Yu. 2006. Dual labeling: Answering graph reachability queries in constant time. In *Proceedings of the 22nd International Conference on Data Engineering*. IEEE, 75–75.

[37] Fang Wei. 2010. TEDI: efficient shortest path query answering on graphs. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*. ACM, 99–110.

[38] Yanghua Xiao, Wentao Wu, Jian Pei, Wei Wang, and Zhenying He. 2009. Efficiently indexing shortest paths by exploiting symmetry in graphs. In *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology*. ACM, 493–504.

[39] Jing Yuan, Yu Zheng, Xing Xie, and Guangzhong Sun. 2011. Driving with knowledge from the physical world. In *Proceedings of the 17th SIGKDD international conference on Knowledge discovery and data mining*. ACM, 316–324.

[40] Jing Yuan, Yu Zheng, Chengyang Zhang, Wenlei Xie, Xing Xie, Guangzhong Sun, and Yan Huang. 2010. T-drive: driving directions based on taxi trajectories. In *Proceedings of the 18th SIGSPATIAL International conference on advances in geographic information systems*. ACM, 99–108.