Efficient Keyword Search on Uncertain Graph Data

Ye Yuan, Guoren Wang, Lei Chen, and Haixun Wang

Abstract—As a popular search mechanism, keyword search has been applied to retrieve useful data in documents, texts, graphs, and even relational databases. However, so far, there is no work on keyword search over uncertain graph data even though the uncertain graphs have been widely used in many real applications, such as modeling road networks, influential detection in social networks, and data analysis on PPI networks. Therefore, in this paper, we study the problem of top-k keyword search over uncertain graph data. Following the similar answer definition for keyword search over deterministic graphs, we consider a subtree in the uncertain graph as an answer to a keyword query if 1) it contains all the keywords; 2) it has a high score (defined by users or applications) based on keyword matching; and 3) it has low uncertainty. Keyword search over deterministic graphs is already a hard problem as stated in [1], [2], [3]. Due to the existence of uncertainty, keyword search over uncertain graphs is much harder. Therefore, to improve the search efficiency, we employ a *filtering-and-verification* strategy based on a probabilistic keyword index, PKIndex. For each keyword, we offline compute path-based top-k probabilities, and attach these values to PKIndex in an optimal, compressed way. In the *filtering* phase, we perform existence, path-based and tree-based probabilistic pruning phases, which filter out most false subtrees. In the *verification*, we propose a sampling algorithm to verify the candidates. Extensive experimental results demonstrate the effectiveness of the proposed algorithms.

Index Terms—Database, algorithm, uncertain data, graph data

1 Introduction

Due to its flexibility and no requirement on background knowledge of searched data, keyword search has been applied to find not only useful web documents, but also relational and graph data. Especially recently, quite a lot efforts have been put for keyword search over graphs [1], [4], [5], [2], [6], [3], [7]. However, in these works, all graphs in the database are assumed to be certain or accurate, and in real-life applications, this assumption is often invalid. For example, Resource Description Framework (RDF) is a standard language for describing web resources, and RDF data are modeled as graphs. In practice, RDF data can be highly unreliably [8] due to errors in the web data or data expiration. In the application of the data integration, we need to incorporate such RDF data from various data sources into an integrated database. In this case, uncertainties/inconsistencies often exist. In social networks, each link between any two persons is often associated with a probability that represents the uncertainty of the link [9] or the strength of influence a person has over another person in viral marketing [10]. In XML data (a tree or graph structure), uncertainties are incorporated in XML documents known as probabilistic XML document (p-document) [11], [12]. There are quite some works having been proposed to manage p-documents [11], [12], [13]. Keyword search over RDF data, social networks and XML data have many important applications [14], [15], [7]. Therefore, in this work, we want to relax the strict assumption of deterministic graphs and study keyword search over uncertain graphs.

1.1 Probabilistic Keyword Search

In this paper, we focus on *threshold-based probabilistic keyword search* (T-KS) over large uncertain graph data. Fig. 1A shows an uncertain graph g with each node attached some text, i.e., node 8 containing two keywords $\{b, c\}$. A real number associated with each edge represents the existence probability of the edge, i.e., 0.5 denoting the existence probability of edge $(1, 3)$. Fig. 1B shows a keyword query, $q = (c, d)$.

The first question we must answer is: What is the semantics of T-KS query over an uncertain graph? To answer this question, we employ the *possible world semantics* [16], [17], which has been widely used for modeling semantics of probabilistic databases. A possible world graph (PWG) of an uncertain graph is a possible *instance* of the uncertain graph. It contains all nodes and a subset of edges of the uncertain graph, and its existence probability is the product of the probabilities of all the edges it has. Fig. 2 lists partial PWGs (the total number is $2^{12} = 4,096$) and corresponding probabilities of uncertain graph in Fig. 1A.

Based on the possible world semantics, given a query $q = (w_1, \ldots, w_m)$, we first perform q on each PWG, i.e., performing traditional keyword search over deterministic

- *Y. Yuan and G. Wang are with the College of Information Science and Engineering, Northeastern University, Wenhua Road 11-3, Heping District, Shenyang, Liaoning 110819, China. E-mail: {yuanye, wanggr}@ise.neu.edu.cn.*
- *L. Chen is with the Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong, China. E-mail: leichen@cse.ust.hk.*
- *H. Wang is with Microsoft Resarch Asia, Microsoft Research Asia Building 2, No. 5 Dan Ling Street, Haidian District, Beijing 100080, P.R. China. E-mail: haixunw@microsoft.com.*

Manuscript received 26 May 2012; revised 26 Aug. 2012; accepted 24 Oct. 2012; published online 7 Nov. 2012.
Recommended for acceptance by B. Cui.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDE-2012-05-0370.
Digital Object Identifier no. 10.1109/TKDE.2012.222.

1041-4347/13/$31.00 © 2013 IEEE Published by the IEEE Computer Society

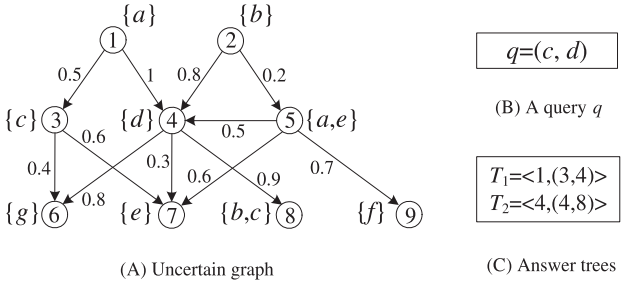


Fig. 1. Example of query and answers.

graphs (PWGs) [1], [4], [5]. The query answers in each PWG are k subtrees whose scores are largest. The score of each subtree is based on the sum of edge weights and node weights, which is formally defined in Definition 4. Second, for each query answer, T, we sum up the probabilities of PWGs that T appears as one of the top-k answer. The summarized probability is called *top-k score probability* of T, denoted by $Pr^k(T)$. If $Pr^k(T)$ is not smaller than a probability threshold ϵ, T is an answer of T-KS query. In other words, a T-KS query returns all subtrees whose top-k score probabilities are at least ϵ. For example, Fig. 1C shows two answer subtrees T_1 and T_2. To calculate $Pr^3(T_1)$, we first find all of g's PWGs whose top-3 query answers contains T_1. In Fig. 2, the results are PWGs (1), (2), ... Next, we add up the probabilities of these PWGs: $0.0011 + 0.0008 + \cdots = 0.382$. If, for example, the query specifies a threshold of 0.6, then T_1 is not an answer, since $Pr^3(T_1) = 0.382 < 0.6$.

The above example also illustrates a naive solution to T-KS query processing, which enumerates all the PWGs and checks top-k lists in each PWG. Clearly, the solution is very inefficient due to the exponential number of PWGs. To address this issue, in this paper, we propose a *filtering-and-verification* method to avoid searching all the PWGs.

1.2 Overview of Our Approach and Contributions

Fig. 3 illustrates a general framework for a keyword search $(w_1, \ldots, w_m)$ over an uncertain graph g, which consists of four phases, *Existence probability pruning*, *Path-based probability pruning*, *Tree-based probability pruning*, and *Verification*, where the first three phases belong to filtering step and the last one is the verification step. We briefly present each step in the following.

1.2.1 Existence Probabilistic Pruning

In this paper, we use g^c to denote the corresponding deterministic graph after we remove all the uncertain information from g and use $subT$, $Ext(T)$, and $Ext(subT)$ to denote a subtree of T, the existence probability of T, and the existence probability of $subT$, respectively. The existence probabilistic pruning can be written as,

Pruning 1. $Pr^k(T) \leq Ext(T) \leq Ext(subT)$.

Pruning 1 means if the existence probability of T or $subT$ is less than threshold ϵ, then $Pr^k(T) < \epsilon$, i.e., T being not an answer.

Suppose D is the set of all subtrees, in g^c, matching query keywords. Based on pruning 1, we can prune some subtrees in D that do not satisfy the probability threshold.[1] Assume the results after pruning 1 is C_e, then C_e is the input for path-based probabilistic pruning in the next step.

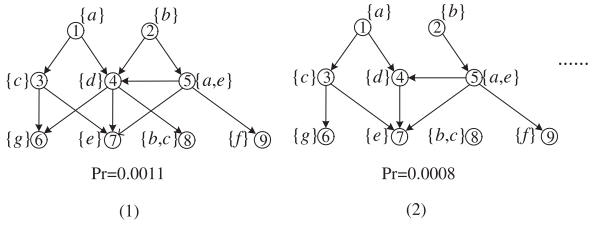


Fig. 2. Partial possible worlds of uncertain graph in Fig. 1A.

```
Procedure keywordsearch_Framework {
  Input: an uncertain graph g, query keywords {w1, ..., wm}, a threshold ε
  Output: subtrees whose top-k score probabilities are at least ε
  (1) perform existence probability pruning
  (2) perform path-based probability pruning
  (3) perform tree-based probability pruning
  (4) perform verification
}
```

Fig. 3. Framework for keyword search on an uncertain graph.

1.2.2 Path-Based Probabilistic Pruning

To further prune the results, we propose a *probabilistic keyword index* (PKIndex) for path-based probabilistic pruning. Specifically, for each distinct keyword, w, we store all shortest paths, in g, reaching w. Each path P has a score which is computed using the same ranking function for returned subtrees. Thus, for each keyword w, PKIndex stores the top-k score probabilities of P that can reach w. Based on precomputed top-k score probabilities, we can conduct path pruning as follows:

Pruning 2. For a subtree $T \in C_e$ and a path P from the root of T to its leaf node, we have $Pr^k(T) \leq Pr^k(P)$. Thus, if $Pr^k(P) < \epsilon$, then $Pr^k(T) < \epsilon$. We can remove T from C_e.

To save space cost of PKIndex, we partition the domain of top-k probabilities (0, 1] into several intervals, so that a very small number of values of k can support the path-based probabilistic pruning. Otherwise, we should store a very large k, i.e., n to support the path-based pruning. We also optimize the partition to acquire the greatest pruning power based on a cost model. In addition, the paths may have structural overlapping, and thus, the probability distributions of the paths are correlated. Obviously, the state-of-the-art approaches cannot be applied to compute $Pr^k(P)$. In our approach, we first build a graph model that captures the correlated probability distribution of the paths. Then, we construct a random repair process, that goes over the graph model, to compute $Pr^k(P)$.

After applying pruning 2 to $T \in C_e$, we get a result set C_p which is the input for tree-based probabilistic pruning.

1.2.3 Tree-Based Probabilistic Pruning

Let $D = \{T_1, \ldots, T_n\}$, and subtrees in D are sorted in a non-increasing score order, i.e., $socre(T_1) > score(T_n)$. For a subtree $T \in C_p$, let its score order in D be i, i.e., $T = T_i$. We select a subset from $\{T_1, \ldots, T_{i-1}\}$, say $IND(T_i) = \{T_{i_1}, \ldots, T_{i_2}\}$ in which any two subtrees are structurally disjoint. Then, the tree-based probabilistic pruning rule can be written as,

1. We compute D while we apply pruning 1.

Pruning 3. For a subtree $T \in C_p$, we have $Pr^k(T) \leq UpperB[T_{i_1}, \ldots, T_{i_2}]$.

$UpperB[T_{i_1}, \ldots, T_{i_2}]$ is a constant which is derived from the top-k probability distribution of subtrees in $IND(T_i)$, and will be introduced in Section 5. The subtrees in C_p are ranked in a nonincreasing score order. Let $\mu = \sum_{T \in IND(T_i)} Pr(T)$. Then, we derive a halting condition as follows:

Halting condition. Given a threshold ϵ and a query k, if $2 \ln \frac{1}{\epsilon} \leq \mu(1 - \frac{k}{\mu})^2 holds$, all subtrees, of C_p, ranked after T can be pruned.

The key effectiveness issue of tree-based probabilistic pruning is to derive a tight $UpperB[T_{i_1}, \ldots, T_{i_2}]$. In our approach, we use *Poisson binomial distribution* to approximate $UpperB[T_{i_1}, \ldots, T_{i_2}]$.

Suppose the result is C_t after we apply the tree-based probabilistic pruning.

1.2.4 Verification

In this step, we calculate $Pr^k(T)$ for $T \in C_t$ to make sure T is really an answer, i.e., $Pr^k(T) \geq \epsilon$. Since the exact calculation of $Pr^k(T)$ needs exponential time, thus, in the verification step, we propose an efficient sampling algorithm, based on the Monte Carlo theory, to estimate $Pr^k(T)$ with a high quality.

1.3 Paper Organization

The rest of this paper is organized as follows: We formally define T-KS queries over uncertain graphs in Section 2. We present existence, path-based, tree-based probabilistic pruning in Sections 3, 4, and 5, respectively. We propose the sampling algorithm for calculating $Pr^k(T)$ in Section 6. We discuss the results of performance tests on real data sets in Section 7 and the related works in Section 8. We conclude our work in Section 9.

2 Background Information

2.1 Problem Definition

Definition 1 (Uncertain Graph). *Let $g^c = (V, E)$ be a directed deterministic graph.[2] Each node $v \in V$ is associated with a set of keywords and a weight. Each edge $e \in E$ is directed and weighted. An uncertain graph is defined as $g = (g^c, Pr)$, where $Pr : E \rightarrow (0, 1]$ is a function that determines the probability of existence of edge e in the graph.*

Definition 2 (Possible World Graph). *A possible world graph $g' = (V', E')$ is a deterministic graph which is a possible instance of the random variables representing the edges of the uncertain graph $g = (V, E, Pr)$, where $V' \subseteq V$, $E' \subseteq E$. We denote the relationship between g' and g as $g \Rightarrow g'$.*

Both g' and g^c are deterministic graphs. An uncertain graph g corresponds to one g^c and multiple possible world graphs. We use $PWG(g)$ to denote the set of all possible world graphs derived from g.

Following the same assumption of the existing uncertain graph models [18], [19], [20], [21], [22], we assume that uncertain variables of different edges are independent of each another. Then, the probability of g' is

$$Pr(g \Rightarrow g') = \prod_{e \in E'} Pr(e) \prod_{e \in E \setminus E'} (1 - Pr(e)). \quad (1)$$

Clearly, for any possible world graph g', we have $Pr(g \Rightarrow g') > 0$ and $\sum_{g' \in PWG(g)} Pr(g \Rightarrow g') = 1$, that is, each possible world graph has an existence probability, and the sum of these probabilities is 1.

Definition 3 (Answer Tree). *Given a query $q = (w_1, \ldots, w_m)$ and an uncertain graph g, an answer to q is a pair $<r, (n_1, \ldots, n_m)>$, where r and n_is are nodes of g^c satisfying: 1) For every i, node n_i contains keyword w_i; 2) For every i, there exists a directed path of g^c from r to n_i.*

Definition 4 (Score of Answer Tree [5]). *The overall answer score is a function of the node score and the edge score of the answer tree. The node score is determined by the sum of the leaf/root weights. The edge score of an answer is the sum of the path lengths (total weight of edges in the path from r to n_i). Without loss of the generality, we ignore the node score and focus on edge score.*

Note that in this paper, we inherit the mainstream definition of answer tree score from traditional works on keyword search over deterministic graph [5], [2], [6]. This definition has been proven to be very effective for real applications [1], [4], [3], [7]. The score of an answer is a function of edge score, and the function is often a decreasing function, i.e., an answer tree with smaller path lengths having a higher score.

Then, given a subtree T, we define its top-k probability, based on possible world semantics, as follows:

Definition 5 . (Top-k score probability of T)

$$Pr^k(T) = \sum_{g' \in Top-k(g)} Pr(g \Rightarrow g'), \quad (2)$$

where Top-$k(g)$ = {g' | T is one of the k distinct root subtrees in g' with highest scores}.

In other words, Top-$k(g)$ is a set of possible world graphs g' s.t. the score of T is among the largest k answer scores in g'.

Note that $Pr^k(T)$ follows the sematic of widely accepted definitions of top-k query on uncertain relational data [23], [24], [25], [26].

Now, we can define probabilistic keyword search:

Definition 6 (Probabilistic Keyword Search). *Given an uncertain graph g, a query q, and a probability threshold ϵ, probabilistic keyword search returns answer trees $T =< r, (n_1, \ldots, n_m)>$ such that the top-k score probability of T is at least ϵ, i.e., $Pr^k(T) >= \epsilon$.*

2.2 Preliminaries

The proposed algorithms and index are based on *BLINKS* [5] that is used to support efficient keyword search over deterministic graphs. The detail introduction to BLINKS can be found in the Appendix, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TKDE.2012.222.

2. This is consistent with works on deterministic graph keyword search [1], [4], [5].

3 Existence Probabilistic Pruning

Following the general framework of keyword search on an uncertain graph (Fig. 3), the first step is existence probability pruning.

Pruning 1. $Pr^k(T) \leq Ext(T) \leq Ext(subT)$.

Proof. From the possible world semantics, we have

$$Ext(T) = \sum_{g' \in \Omega_{(T)}} Pr(g \Rightarrow g'),$$
$$Ext(subT) = \sum_{g' \in \Omega_{(subT)}} Pr(g \Rightarrow g'),$$

where $\Omega_{(T)}$ and $\Omega_{(subT)}$ are the possible world graphs that contain T and $subT$, respectively.

In addition, since some edges of T are missing in $subT$, thus, $\Omega(T) \subseteq \Omega(subT)$:

$$Pr^k(T) = \sum_{g' \in Top-k(g)} Pr(g \Rightarrow g')$$
$$\leq Ext(T) = \sum_{g' \in \Omega_{(T)}} Pr(g \Rightarrow g')$$
$$\leq Ext(subT) = \sum_{g' \in \Omega_{(subT)}} Pr(g \Rightarrow g').$$

□

To apply pruning 1 online, we need to compute $Ext(T)$ and $Ext(subT)$ efficiently. The following theorem states how to compute $Ext(T)$ and $Ext(subT)$.

Theorem 1.

$$Ext(T) = \prod_{e \in T} Pr(e),$$
$$Ext(subT) = \prod_{e \in subT} Pr(e). \quad (3)$$

Proof. Based on possible world semantics, we can obtain

$$Ext(T) = \sum_{g' \in \Omega_{(T)}} Pr(g \Rightarrow g'). \quad (4)$$

By substituting (1) into (4), (4) is written as

$$Ext(T) = \sum_{g' \in \Omega_{(T)}} \prod_{e \in g'} Pr(e) \prod_{e \in g \setminus g'} (1 - Pr(e)).$$

Since g' contains T, we obtain

$$Ext(T) = \prod_{e \in T} Pr(e) \sum_{g' \in \Omega_{(T)}} \prod_{e \in g' \setminus T} Pr(e) \prod_{e \in g \setminus g'} (1 - Pr(e)). \quad (5)$$

Observe that the item $\sum_{g' \in \Omega_{(T)}}(\cdot)$ in the RHS of (5) equals 1. Thus, (3) holds. The same conclusion can also be derived for $Ext(subT)$. □

Now, we can compute $Ext(T)$ and $Ext(subT)$ by using BLINKS. As introduced in Section 2, both backward search and forward search scan one edge e in each step. In each step, we use Theorem 1 to accumulate probabilities of $Ext(T)$ and $Ext(subT)$ by multiplying $Pr(e)$.

During the search, if $Ext(T) \leq Ext(subT) < \epsilon$, then T is pruned. Thus, we apply pruning 1 to T while we are computing T. BLINKS can find the set of all distinct root subtrees with best scores in g^c [5], say D. All subtrees in D are ranked in a nonincreasing score order. Therefore, after we perform BLINKS and pruning 1, we get a set C_e which equals the result after we use pruning 1 to each subtree in D. Similar to D, subtrees in C_e are also ranked in a nonincreasing score order.

The set C_e is the input for path-based probabilistic pruning.

Example 1. Consider the subtree $T_1 = <1, (3, 4)>$ in Fig. 1. The existence probability of T_1 is $0.5 \times 1 = 0.5$, and its top-2 probability is 0.42. Then, $Pr^2(T_1) < Ext(T_1)$.

4 Path-Based Probabilistic Pruning

4.1 Pruning Condition

This section introduces the path-based probabilistic pruning applied to each subtree in C_e. For query keywords $q = (w_1, \ldots, w_m)$, a subtree $T \in C_e$ has m leaf nodes with each containing a keyword w. For a keyword w, there is a path P from the root of T to its leaf node containing w. For path P, we can also define top-k probability of P, $Pr^k(P)$, as Definition 5. $Pr^k(P)$ is interpreted as the top-k probability of all distinct root paths containing keyword w. Assume, we have calculated $Pr^k(P)$, and stored it to probabilistic keyword index, PKI. The details about calculating $Pr^k(P)$ and PKI will be introduced the next two sections.

Let the score order of T in the distinct root subtree set, $D = \{T_1, \ldots, T_i, \ldots, T_n\}$, be i, i.e., $T = T_i$, and the score order of P in the distinct root path set, $\{P'_1, \ldots, P'_{i'}, \ldots, P'_n\}$, be i', i.e., $P = P'_{i'}$, then we have path-based probabilistic pruning as follows:

Pruning 2. For a subtree $T \in C_e$ and a path P from the root of T to its leaf node, if $i \geq i'$, we have $Pr^k(T) \leq Pr^k(P)$. If $Pr^k(P) < \epsilon$, T can be pruned from C_e.

Proof. Recall that subtrees in $D = \{T_1, \ldots, T_n\}$ are sorted in a nonincreasing score order. Set a Boolean variable bT_l for each $T_l \in D$, and $Pr(bT_l = true) = Ext(T_l)$. It also means $bT_l = 1$ iff T exists (all uncertain edges of T exist), otherwise $bT_l = 0$.

Assume the score order of subtree T_i in D is i. Based on possible world semantics in Definition 5, $Pr^k(T_i)$ is the probability that T_i exists (i.e., $bT_i = 1$) and fewer than k subtrees with scores higher than T_i are present (i.e.,$\sum_{l=1}^{i-1} bT_l < k$). In other words,

$$Pr^k(T_i) = Pr(bT_1 + \cdots + bT_{i-1} < k \wedge bT_i = 1). \quad (6)$$

For the subtree T_i, there are m paths from the root of T_i to its leaf nodes. Set a Boolean variable bP_l for each path P_l, and $Pr(bP_l = 1) = Ext(P_l)$. Then, bT_i can be written as

$$bT_i = \begin{cases} 1 & \text{if } bP_1 = 1 \wedge \cdots \wedge bP_m = 1, \\ 0 & \text{if } bP_1 = 0 \vee \cdots \vee bP_m = 0. \end{cases} \quad (7)$$

Equation (7) can also be written as

$$bT_i = \begin{cases} 1 \\ 0 \end{cases} \quad \text{if } bP_1 + \cdots + bP_m \leq m. \quad (8)$$

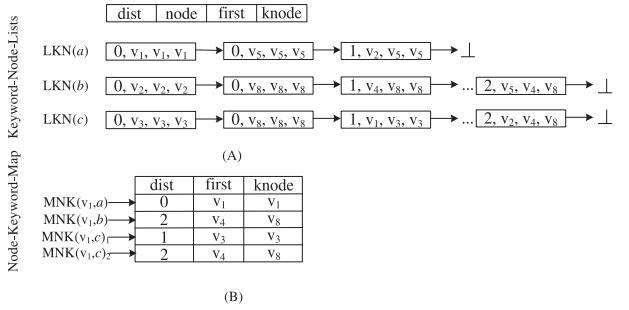


Fig. 4. Keyword-node lists and node-keyword map.

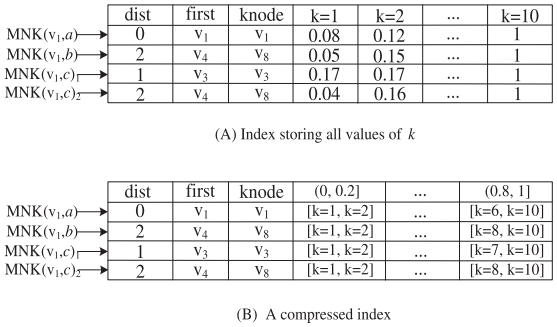


Fig. 5. Probabilistic keyword index.

Similarly, for $bT_1, \ldots, bT_{i-1}$, we have

$$bT_1 = \begin{cases} 1 & \text{if } bP_{11} + \cdots + bP_{1m} \leq m \\ 0 & \end{cases}$$

$$\vdots$$

$$bT_{i-1} = \begin{cases} 1 & \text{if } bP_{(i-1)1} + \cdots + bP_{(i-1)m} \leq m. \\ 0 & \end{cases}$$

Based on these results, event $bT_1 + \cdots + bT_{i-1} < k$ becomes

$$(bP_{11} + \cdots + bP_{(i-1)1}) + \cdots + (bP_{1m} + \cdots + bP_{(i-1)m}) < km. \tag{9}$$

Thus, there is at least one $(\cdot) < k$, say

$$bP_{1j} + \cdots + bP_{(i-1)j} < k. \tag{10}$$

The jth item $(\cdot) < k$ is corresponding to the jth keyword k_j. Then, the distinct root paths reaching k_j are $RP = \{P'_{1j}, \ldots, P'_{(i'-1)j}, P'_{i'j}, \ldots\}$, and $P'_{i'j} = P_{ij}$. If $i \geq i'$, $\{P_{1j}, \ldots, P_{(i-1)j}\} \supseteq \{P'_{1j}, \ldots, P'_{(i'-1)j}\}$. Thus, we have

$$bP_{1j} + \cdots + bP_{(i-1)j} < k \subseteq bP'_{1j} + \cdots + bP'_{(i'-1)j} < k. \tag{11}$$

Also $bT_{i-1} = 1 \subseteq bP'_{(i'-1)j} = 1$, since $P'_{(i'-1)j}$ is a path of T_{i-1}. Then, we obtain

$$\begin{aligned} Pr^k(T_i) &= Pr[bT_1 + \cdots + bT_{i-1} < k \wedge bT_i = 1] \\ &= Pr[(bP_{11} + \cdots + bP_{(i-1)1}) + \cdots \\ &\quad + (bP_{1m} + \cdots + bP_{(i-1)m}) < km \wedge bT_i = 1] \\ &\leq Pr[bP_{1j} + \cdots + bP_{(i-1)j} < k \wedge bT_i = 1] \\ &\leq Pr[bP'_{1j} + \cdots + bP'_{(i'-1)j} < k \wedge bP'_{i'j} = 1] \\ &= Pr^k(P_{ij}). \end{aligned} \tag{12}$$

Since j can be any value in $\{1, \ldots, m\}$, $Pr^k(T_i) \leq Pr^k(P_i)$. □

Example 2. Consider the subtree T_1 in Fig. 1 and compute $Pr^k(T_1)$. Suppose T_1 has a smaller score than T_2, then i for T_1 is 2. The path $P_1 = (1, 4)$ is from root 1 to leaf node 4 that contains keyword d. There is another path $P_2 = (2, 4)$ reaching keyword d. Suppose the score of P_1 is higher than P_2, then i' for P_1 is 1. Based on pruning 2, we have $Pr^k(T_1) < Pr^k(P_1)$.

For a subtree T and k, it takes $O(1)$ time to choose a path, and then another $O(1)$ time to locate the value of $Pr^k(T)$ in the index. If there are C_1 subtrees that will be pruned in this step, the total time complexity of path-based pruning is $O(C_1)$.

Note that there must exist a path P in T such that $i \geq i'$. To apply this pruning rule, we choose $Pr^k(P)$ whose value is largest among m paths. Usually, m is very small (e.g., 2 or 3) since m is the number of query keywords, thus, the largest $Pr^k(P)$ leads to a very tight upper bound for $Pr^k(T)$. Moreover, the value of $Pr^k(P)$ can be computed offline for any keyword in g, which does not depend on query keywords. Thus, $Pr^k(P)$ can be easily integrated into index.

4.2 Probabilistic Keyword Index

In this section, we introduce the probabilistic keyword index. Recall that *node-keyword map* (M_{NK}) in BLINKS stores, for each node u, the shortest path distance from u to every keyword. PKI extends M_{NK} by attaching the values of $Pr^k(P)$ to each shortest path P. For the path P, a naive approach is to store all possible values of k in PKI.

For example, Fig. 5A shows the structure of PKI for the uncertain graph in Fig. 1. Fig. 4B is the node-keyword map for the deterministic case of this uncertain graph. Fig. 5A adds $Pr^k(P)$, for all possible values of k, to the node-keyword map.

We now use an example to show the path-based probabilistic pruning. Consider subtree $T = <v_1, (a, b, c)>$ and path $P = <v_1, a>$ in Fig. 5A, if threshold is 0.3 and $k = 2$, we can prune the subtree because $Pr^2(T) < Pr^2(P) = 0.12 < 0.3$.

To answer any top-k query, the naive approach should store n (number of nodes in g) values, in the worst case, of k for each path. Obliviously, the naive approach is quite space consuming for a large uncertain graph. Therefore, in the following, we introduce a compression method for PKI.

For path P, we divide the domain of $Pr^k(P)$ (0, 1] into h prob-intervals, and each has size $1/h$. Formally, given a granularity parameter $h > 0$, PKI for P contains a set of prob-intervals $\{b_1, \ldots, b_h\}$, where $b_i = (\frac{i-1}{h}, \frac{i}{h}](1 \leq i \leq h)$. We observe that the value of $Pr^k(P)$ increases with the increasing of k. Thus, b_i contains a list of increasing integers, say $KL = \{k_{i1}, \ldots, k_{im}\}$, such that $\frac{i-1}{h} < Pr^{k_{i1}}(P) \leq \cdots \leq Pr^{k_{im}}(P) \leq \frac{i}{h}$.

For example, Fig. 5B shows the compression result of the index in Fig. 5A. In the example, $b_1 = (0, 0.2]$ and $b_h = (0.8, 1]$. b_1 covers values of $Pr^1(P)$ and $Pr^2(P)$, and thus stores $[1, 2]$.

Note that PKI inherits the structure, from BLINKS, that consists of small size blocks. Thus, after the compression, PKI is small enough to be loaded into memory.

To support path-based probabilistic pruning, each b_i only needs to contain the largest value of k, i.e., k_{im}. Based on this setting, we revise the pruning 2 as follows:

Pruning 2'. For k within b_i, i.e., $k_{(i-1)m} < k \leq k_{im}$, then $Pr^k(T) \leq Pr^k(P) \leq \frac{i}{h}$.

$k_{(i-1)m}$ is the largest value contained in b_{i-1}, and hence $k_{(i-1)m} < k \leq k_{im}$ guarantees $k \in b_i$. In the partition, PKI only indices h values to support pruning. The value of h is much smaller than n used in the naive approach. In practice, we can set h according to the space capacity of the system.

Following the previous example for path-based pruning, since $k = 2$, we locate interval $b_1 = (0, 0.2]$ in compressed PKI. Based on b_1, we can prune the subtree since $Pr^2(T) < Pr^2(P) < \frac{i}{h} = 0.2 < 0.3$.

Clearly, given h, different domain partition strategies will result in different pruning power. In the following, we introduce an optimal partition strategy for obtaining the maximum pruning power.

Optimal partition. We first devise a cost model to estimate the path-based probabilistic pruning power. Suppose we divide the domain $(0, 1]$ into h prob-intervals such that $\{(0, U_1], \ldots, (U_{i-1}, U_i], \ldots, (U_{h-1}, U_h = 1]\}$. Note that $U_i \neq \frac{i}{h}$. Also suppose a prob-interval $b_i = (U_{i-1}, U_i]$ indices values of k, $\{k_{i1}, \ldots, k_{im_i}\}$. Before partition, $Pr^{k_i}(P)$ is an upper bound for a subtree and is used in the pruning. After partition, U_i is the upper bound for the subtree. Since $Pr^{k_i}(P) < U_i$, the pruning power decreases after partition. Therefore, we define a cost model to measure the decrease of the pruning power after partition. We use the difference between probability upper bounds of $Pr^k(T)$, (U_i), and $Pr^{k_i}(P)$ to estimate the decrease degree of the pruning power after partition. Specifically, for each interval b_i, the decreased pruning power is

$$\begin{aligned} C_i &= \left(U_i - Pr^{k_{i1}}(P)\right) + \cdots + \left(U_i - Pr^{k_{im_i}}(P)\right) \\ &= m_i U_i - \sum_{j=1}^{m_i} Pr^{k_{ij}}(P). \end{aligned} \tag{13}$$

Equation (21) gives a measurement on the decrease of pruning power for prob-interval b_i after partition. Similarly, we can get the cost models $\{C_1, \ldots, C_h\}$ for prob-intervals $\{b_1, \ldots, b_h\}$. Then, the sum of these cost models measures the decrease of pruning power for path P after partition. The summarized cost is written as

$$C(P) = \sum_{i=1}^{h} m_i U_i - \sum_{i=1}^{h} \sum_{j=1}^{m_i} Pr^{k_{ij}}(P).$$

Given P, the second item is a constant, which can be ignored; Both m_i and U_i have constraints, thus, for path P, we have

$$\begin{aligned} &C(P) = \sum_{i=1}^{h} m_i U_i \\ &\text{s.t.} \sum_{i=1}^{h} m_i = n, 0 < U_1 \leq \cdots \leq U_h = 1. \end{aligned} \tag{14}$$

Summarize (14) for total n paths, we get the total decrease of pruning power for PKI after partition. We want to minimize the total decrease of pruning power. Then, for PKI, we want to achieve

$$\begin{aligned} &minimize \quad \sum_{l=1}^{n} \sum_{i=1}^{h} m_{li} U_i \\ &\text{s.t.} \quad \sum_{li=1}^{h} m_{li} = n, \quad \text{for } l = \{1, \ldots, n\}, \\ &0 < U_1 \leq \cdots \leq U_h = 1. \end{aligned} \tag{15}$$

Equation (15) can be solved by *quadratic programming*. In this paper, we apply the solution in [27] to obtain $\{U_1, \ldots, U_h\}$ that is used to partition domain $(0, 1]$.

4.3 Calculation of Top-k Probabilities of Path

Now, we present how to compute top-k probabilities of a path, i.e., $Pr^k(P)$, offline. For a keyword indexed in g, we use BLINKS to determine distinct root paths $E = \{P_1, \ldots, P_n\}$[3] that reach the keyword, in a nonincreasing score order. Each path $P_i \in E$ has an existence probability, and thus, we use the traditional top-k algorithms for uncertain relational databases [23], [24], [28], [25], [26], to compute $Pr^k(P_i)$. These methods all assume tuples or attributes probabilistic independence. However, paths in E overlap and, thus, their existence probabilities are correlated. We should develop a new approach to calculate $Pr^k(P)$. In this section, we propose an efficient sampling algorithm to calculate $Pr^k(P)$.

Consider an uncertain graph g with edge set E and vertex set V. Associated each edge $e \in E$ with a Boolean variable X_e. The probability, $Pr(X_e = 1)$, of X_e being assigned true is equal to its existence probability, i.e., $Pr(X_e = 1) = Pr(e)$. Similarly, the probability, $Pr(X_e = 0)$, of X_e being assigned false is equal to $Pr(X_e = 0) = 1 - Pr(e)$. We label the edges from 1 to a, and call the vector $\mathbf{X} = (X_1, \ldots, X_a)$ a *state* of edge set E. Let S be a set of all possible states of E. For the path P reaching a keyword w, we define

$$\Phi(\mathbf{X}) = \begin{cases} 1 & \text{If path } P \text{ is in top-}k \text{ list of } \mathbf{X}, \\ 0 & \text{otherwise,} \end{cases} \tag{16}$$

where the top-k list of $\mathbf{X}$ contains k paths, with largest scores, reaching w.

Now, we can obtain

$$\begin{aligned} Pr^k(P) &= E[\Phi(\mathbf{X})] \\ &= \sum_{\mathbf{x} \in S} \Phi(\mathbf{x}) Pr(\mathbf{X} = \mathbf{x}). \end{aligned} \tag{17}$$

Now, let us consider a random repair process for E in which each edge e has an exponential repair time with repair rate $\lambda(e) = -\log(Pr(e))$. This process is similar as random walk. At time $t = 0$ all edges do not exist, i.e., $X_i = 0$ for $1 \leq i \leq a$. Since different X_i are independent, we assume that all repair times are independent of each other. The state of e at time t is denoted by $X_e(t)$. $X_e(t) = 1$ denotes that e has been repaired (become existent) at time t, and $X_e(t) = 0$ denotes that e has not been repaired.

3. We consider the worst case in which all nodes (i.e., n nodes) reach the keyword. In practice, the number will be much smaller.

Similarly, the vector $\mathbf{X}(t) = (X_1(t), \ldots, X_a(t))$ denotes the state of the edge set E at time t. Therefore, $\mathbf{X}(t)$ is a *Markov process* with the state space $\{0,1\}^a$.

Let Π denote the order in which the edges are repaired (become existent), and let C_0, $C_0 + C_1, \ldots, C_0 + \cdots C_{a-1}$ be the times at which those edges have been repaired. Π is a random variable which takes values in the space of permutations of E.

For any permutation $\pi = (e_1, e_2, \ldots, e_a)$ define

$$\begin{aligned} E_0 &= E, \\ E_i &= E_{i-1} \backslash e_i, 1 \le i \le a-1, \\ \lambda(E_i) &= \sum_{e \in E_i} \lambda(e), \end{aligned}$$

and let

$$c(\pi) = \min_i \{\Phi(E_i) = 1\}.$$

From the general theory of Markov processes [29], it is not difficult to obtain

$$Pr(\Pi = \pi) = \prod_{j=1}^{a} \frac{\lambda(e_j)}{\lambda(E_{j-1})}. \tag{18}$$

Moreover, conditioned on $\{\Pi = \pi\}$, the times $C_0, \ldots, C_{a-1}$ are independent and each C_i is exponentially distributed with parameter $\lambda(E_i)$ for $i = 0, \ldots, a-1$.

Note that the probability of each edge e being existent at time $t = 1$ is $Pr(e)$. It follows that the value of $Pr^k(P)$ at time $t = 1$ is the same as in (17). Hence, based on conditioning on π, we have

$$\begin{aligned} Pr^k(P) &= E[Pr(\Phi(\mathbf{X}(1))] \\ &= \sum_{\pi} Pr[\Pi = \pi] Pr[\Phi(\mathbf{X}(1)) = 1 | \Pi = \pi]. \end{aligned} \tag{19}$$

Using the definitions of C_i and $c(\pi)$, the last probability of (19) can be written in terms of convolutions of exponential distributions. Namely, for any $t \ge 0$, we have

$$\begin{aligned} Pr[\Phi(\mathbf{X}(1)) = 1 | \Pi = \pi] &= Pr[C_0 + \cdots + C_{c(\pi)-1} < t | \Pi = \pi] \\ &= \underset{0 \le i \le c(\pi)}{\mathbf{Conv}} \{1 - exp[-\lambda(E_i)t]\}. \end{aligned} \tag{20}$$

Let

$$\begin{aligned} H(\pi) &= Pr[\Phi(\mathbf{X}(1)) = 1 | \Pi = \pi] \\ &= \underset{0 \le i \le c(\pi)}{\mathbf{Conv}} \{1 - exp[-\lambda(E_i)t]\}. \end{aligned} \tag{21}$$

Equation (19) can then be rewritten to be

$$Pr^k(P) = E[H(\pi)]. \tag{22}$$

A closer look at the repair process for the uncertain graph reveals that many of the above results remain valid when we combine various states into super states at various stages of the process. This is known as the merge process in stochastic process [29]. We apply this principle to obtain more accurate estimates.

For a subset of edges $F \subseteq E$, the proper partition of the subgraph $g(V, F)$ is denoted by $\sigma = \{V_1, V_2, \ldots, V_B\}$. Let I_i denote the edge-set of the induced subgraph $g(V_i)$. The set $I_\sigma = \{I_1 \cup \cdots \cup I_a\}$ of the edges is the closure of F. Denote its complement by $E_\sigma = E \setminus I_\sigma$.

Let $L(g)$ be the collection of all proper partitions of $g(V, E)$. It is not difficult to see that $L(g)$ is a lattice, ordered by the relation $I_\eta \subset I_\sigma$ for $\eta \prec \sigma$.

Consider the repair process $(\mathbf{X}(t))$ of the uncertain graph. By restricting the process $(\mathbf{X}(t))$ to $L(g)$, we obtain another Markov process $(\mathbf{X}'(t))$. This process starts from the initial state σ_0 of isolated nodes and ends at the terminal state σ_α corresponding to $g(V, E)$.

For each σ, the repairing time in σ has an exponential distribution with parameter $\lambda(\sigma) = \sum_{e \in E_\sigma} \lambda(e)$, independent of anything else. Moreover, the transition probability from η to σ (where σ is a direct successor of η) is given by

$$\lambda(\sigma) - \lambda(\eta)/\lambda(\sigma).$$

Next, compare to the results for the repair process, we define a trajectory of $(\mathbf{X}'(t))$ as a sequence $\theta = (\sigma_0, \sigma_1, \ldots, \sigma_c)$, where $c = c(\theta)$ is the first index i such that σ_i is up, that is, $\Phi(\mathbf{X}) = 1$. By defining $\Phi(\mathbf{X}'(t)) = \Phi(\mathbf{X}(t))$, we thus have

$$Pr[\Phi(\mathbf{X}(1)) = 1] = E[G(\Theta)], \tag{23}$$

where Θ is a random trajectory of $(\mathbf{X}'(t))$. For each output $\theta = (\sigma_0, \sigma_1, \ldots, \sigma_c)$ of Θ, $G(\theta)$ is given by

$$\begin{aligned} G(\theta) &= Pr[\Phi(\mathbf{X}(1)) = 1 | \Theta = \theta] \\ &= Pr[C_0 + \cdots + C_{c(\theta)-1} < 1 | \Theta = \theta] \\ &= \underset{0 \le i \le c(\theta)}{\mathbf{Conv}} \{1 - exp[-\lambda(\sigma_i)t]\}. \end{aligned} \tag{24}$$

Finally, we can obtain the approximate value of $Pr^k(P)$ as

$$Pr^k(P) = E[G(\theta)]. \tag{25}$$

This result shows how the random repair simulation works. Namely, let $G(\theta_1), \ldots, G(\theta_N)$ be independent-identically distributed random permutations, according to distribution given in (24). Then, the approximated value of $Pr^k(P)$ is given by

$$\widehat{Pr^k(P)} = \frac{1}{N} \sum_{i=1}^{N} G(\theta_i). \tag{26}$$

When the value of N is large enough, the approximation quality given by (26) is guaranteed from the well-known Chernoff-Hoeffding bound [30].

Lemma 1. *For any $\xi (0 < \xi < 1)$ and $\tau (\tau > 0)$, if $N \ge (4 \ln \frac{2}{\xi})/\tau^2$, then*

$$Pr[|CNT/N - Pr^k(T)| < \varepsilon Pr^k(T)] \ge 1 - \delta.$$

5 Tree-Based Probabilistic Pruning

5.1 Pruning Condition

Assume the result set is C_p after we apply path-based probabilistic pruning. C_p is the input for tree-based probabilistic pruning. This section introduces the pruning condition for tree-based probabilistic pruning.

For $T_i \in D = \{T_1, \ldots, T_i, \ldots, T_n\}$, recall that $Pr^k(T_i)$ is calculated as follows:

$$Pr^k(T_i) = Pr(bT_1 + \cdots + bT_{i-1} < k \wedge bT_i = 1).$$

We select a subset, $\{bT_{i_1}, \ldots, bT_{i_2}\}$, of $\{bT_1, \ldots, bT_{i-1}\}$. Then, $Pr^k(T_i)$ is upper bounded by

$$\begin{aligned} Pr^k(T_i) &= Pr(bT_1 + \cdots + bT_{i-1} < k \wedge bT_i = 1) \\ &\leq Pr(bT_1 + \cdots + bT_{i-1} < k) \\ &\leq Pr(bT_{i_1} + \cdots + bT_{i_2} < k). \end{aligned} \quad (27)$$

If subtrees $\{T_{i_1}, \ldots, T_{i_2}\}$ do not overlap each other, $\{bT_{i_1}, \ldots, bT_{i_2}\}$ are independent of each other. In this case, $\{bT_{i_1}, \ldots, bT_{i_2}\}$ are called *Poisson trials*, and $sum_i = bT_{i_1} + \cdots + bT_{i_2}$ follows a *Poisson binomial distribution*, s.t. $E(sum_i) = Pr(T_{i_1}) + \cdots + Pr(T_{i_2})$.

In the Poisson binomial distribution $sum_i = k$, its probability can be approximated by

$$Pr(sum_i = k) \approx f(k, \mu) = \frac{\mu^k}{k!} e^{-\mu}, \quad (28)$$

where $f(k, \mu)$ is the Poisson probability mass function, and $\mu = E(sum_i) = Pr(T_{i_1}) + \cdots + Pr(T_{i_2})$.

Thus, the probability of $sum_i < k$ can be approximated by [29]

$$Pr(sum_i < k) \approx F(k, \mu) = \frac{\Gamma(\lfloor k+1 \rfloor, \mu)}{\lfloor k \rfloor !}, \quad (29)$$

where $F(k, \mu)$ is the cumulative distribution function corresponding to $f(k, \mu)$, and $\Gamma(x, y) = \int_y^\infty t^{x-1} e^{-t} dt$ is the upper incomplete gamma function.

Based on (27) and (29), we obtain

$$\begin{aligned} Pr^k(T_i) &\leq Pr(sum_i = bT_{i_1} + \cdots + bT_{i_2} < k) \\ &= \frac{\Gamma(\lfloor k+1 \rfloor, \mu)}{\lfloor k \rfloor !}. \end{aligned} \quad (30)$$

Then, we have tree-based probabilistic pruning as follows:

Pruning 3. For a subtree $T \in C_p$, we have $Pr^k(T) \leq \frac{\Gamma(\lfloor k+1 \rfloor, \mu)}{\lfloor k \rfloor !}$. If $\frac{\Gamma(\lfloor k+1 \rfloor, \mu)}{\lfloor k \rfloor !} < \epsilon$, T can be pruned from C_p.

Example 3. Consider subtrees T_1 and T_2 in Fig. 1 and calculate $Pr^2(T_1)$. Assume T_2 has a higher score than T_1, then the rank is $\{T_2, T_1\}$. Based on pruning 3, we have $Pr^2(T_1) < f(2, \mu)$, where $\mu = Pr(T_1) + Pr(T_2) = 0.5 + 0.9 = 1.4$.

The upper bound $F(k, \mu)$ of $Pr^k(T)$ depends on subtrees that do not overlap. However, among $\{T_1, \ldots, T_{i-1}\}$, there are many groups of subtrees that are disjoint, which leads to different upper bounds. We want to get a tight upper bound to increase the pruning power. Next, we introduce how to obtain tightest upper bound.

Obtain Tightest Upper Bound. We construct an undirected graph, G_T, with each vertex representing a subtree, $T \in \{T_1, \ldots, T_{i-1}\}$, and an edge connecting two *disjoint* subtrees (vertices). We assign each vertex a weight, $Pr(T)$. In G_T, a *clique* is such a set of vertices that any two vertices of the set are adjacent. We define the weight of a clique as the sum of vertex weights in the clique. Given a clique cl with weight $\mu = \sum_{T \in cl} Pr(T)$, the upper bound is $F(k, \mu)$. $F(k, \mu)$ corresponds to disjoint subtrees represented by cl. For given k, $F(k, \mu)$ is a decreasing function of μ [29]. Thus, the larger μ is, the tighter (smaller) the upper bound is. To obtain a tight upper bound, we should find a clique whose weight is largest, which is exactly the *maximum weight clique* problem. Here, we use the efficient solution in [31] to solve the maximum clique problem, and the algorithm returns the largest weight z. Therefore, we use $F(k, z) = \frac{\Gamma(\lfloor k+1 \rfloor, z)}{\lfloor k \rfloor !}$ as the tightest upper bound.

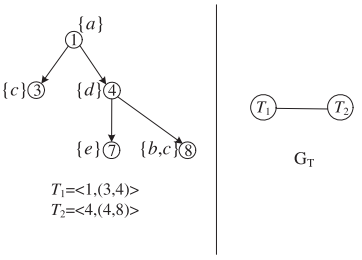


Fig. 6. Tightest upper bound for tree-based pruning.

Setting up an example in Fig. 6, this figure shows two subtrees, in Fig. 1, which are $T_1 = <1, (3, 4)>$ and $T_2 = <4, (4, 8)>$. Though they share a common vertex 4, their edges are disjoint. This implies T_1 and T_2 are probability independent. Thus, G_T derived from T_1 and T_2 are a line connecting these two nodes and each node has a weight $Pr(T)$. Then, we can calculate the tightest upper bound.

For a subtree T_i with order i, we spend $O(i^2)$ time to compare subtrees whose orders are smaller than T_i and choose the maximum number of disjoint subtrees. Then, we spend $O(i)$ time to obtain the upper bound and apply the tree-based pruning rule. Thus, the total time taken for pruning or filtering T_i is $O(i^2)$. If C_2 subtrees are pruned in this step, the total time complexity of tree-based pruning is $O(1^2 + \cdots + C_2^2) = O(C_2^3)$.

5.2 Halting Condition

To apply pruning 3, we progressively read subtrees in C_p in the descending ranking order. In this section, we will derive a halting condition for the reading. That is, once the halting condition holds, all unread subtrees in C_p cannot satisfy the T-KS query and can be pruned.

We use $sum_i < k$ in Inequality 30 to upper bound $Pr^k(T_i)$. To obtain the halting condition, we apply an inequality for top-k queries on relational uncertain data [32]. This inequality combines the Chernoff bound [29]:

$$Pr[sum_i < (1 - \omega)\mu] < e^{-\frac{\mu\omega^2}{2}}, \quad (31)$$

where $\mu = E(sum_i)$ and $0 < \omega \leq 1$, to upper bound $Pr(sum_i < k)$ as $Pr[sum_i < k] \leq Pr[sum_i < (1 - \omega)\mu]$. Thus, the inequality is [32]

$$2 \ln \frac{1}{\epsilon} \leq \mu \left(1 - \frac{k}{\mu}\right)^2. \quad (32)$$

This inequality is correct for uncertain data with independent probability distribution. One question is that how to apply the inequality to subtrees whose probability distributions are correlated? We can solve the problem as follows:

Consider a subtree $T \in C_p$ whose score order, in C_p, is larger than that of T_i in C_p. Note that subtrees in C_p are ranked in a nonincreasing score order. Let the score order of

T in D be j. Then, the score order of T_j in D is also larger than that of T_i in D, i.e., $j > i$. Thus, the set $\{T_1, \ldots, T_{j-1}\}$ must contain a set of disjoint subtrees, $\{T_{j_1}, \ldots, T_{j_2}\}$, such that $\{T_{j_1}, \ldots, T_{j_2}\} \supseteq \{T_{i_1}, \ldots, T_{i_2}\}$. Let $sum_j = bT_{j_1} + \cdots + bT_{j_2}$. Then, we obtain

$$Pr^k(T_j) \leq Pr(sum_j < k) \leq Pr(sum_i < k) < e^{-\frac{\mu\omega^2}{2}} < \epsilon, \quad (33)$$

when Inequality 32 holds.

Inequality 33 shows once T_i is pruned on the condition given in Inequality 32, T_j can also be pruned.

Finally, we have the halting condition as follows:

Halting Condition. Given a threshold ϵ, query k and subtree $T \in C_p$, if $2\ln\frac{1}{\epsilon} \leq \mu(1-\frac{k}{\mu})^2$, we can stop and avoid retrieving further subtrees, of C_p, ranked after T.

In pruning 3, we have calculated the value of μ for the read T. Then, we input μ into Inequality 32 to test if the halting condition is satisfied.

Assume the result is C_t, after we use the tree-based probabilistic pruning.

6 Verification

In this section, we present the algorithms to compute $Pr^k(T)$ for $T \in C_t$. Then, we can obtain the query answers, i.e., $A_q = \{T | Pr^k(T) \geq \epsilon, T \in C_t\}$.

Equation (6) gives a method to calculate $Pr^k(T)$. However, subtrees are structurally overlapping, and probability distributions of the subtrees are correlated. It is quite expensive to calculate correlated probability distributions in (6) (unfolding it needs exponential steps.), which may lead to bottleneck in the query processing. Thus, we introduce a sampling approach to estimate $Pr^k(T)$ in the following.

We sample an instance g' of uncertain graph g with edge probability distributions. An indicator function is defined as follows:

$$\Phi(g') = \begin{cases} 1 & \text{if } T \text{ is in the top}-k \text{ score list of } g', \\ 0 & \text{otherwise.} \end{cases}$$

For the query keywords, we use the backward and forward search in BLINKS [5] to quickly determine if $\Phi(g') = 1$. Then, the sampling approach is given in Algorithm 1.

Algorithm 1. Calculate $Pr^k(T)$.
1: $CNT = 0$;
2: $N = (4ln2/\xi)/\tau^2$;
3: **for** 1 to N **do**
4: sample $X_e = 1$ with probability $Pr(e)$, and sample $X_e = 0$ with probability $1 - Pr(e)$ for each edge $e \in g$. Then we obtain an instance g';
5: **if** $\Phi(g') = 1$ **then**
6: $CNT = CNT + 1$;
7: **end if**
8: **end for**
9: **return** CNT/N;

The main idea of this algorithm is to sample possible worlds, g', of uncertain graph g and to test query condition (Definition 5) in each g'. Then, we set the number of generated g' to guarantee the quality of estimated value. In the algorithm, we first set a counter and the number of generated g' (Lines 1-2). Second, we sample g' from g based on distributions of g (Line 4), where X_e is a Boolean variable for edge e. In each generated g', we test query condition, and accumulate the counter if the condition is satisfied (Lines 5-6). Finally, we return the estimated value (Line 9).

Based on Monte Carlo theory, the adopted value of sampling number ($N = (4ln2/\xi)/\tau^2$) guarantees the estimated quality [29]:

Lemma 2. *For any $\xi(0 < \xi < 1)$ and $\tau(\tau > 0)$, if $N \geq (4\ln\frac{2}{\xi})/\tau^2$, then*

$$Pr[|CNT/N - Pr^k(T)| < \varepsilon Pr^k(T)] \geq 1 - \delta.$$

In Monte Carlo theory, the values of ξ and τ are usually both set to 0.1 [29], [22].

7 Performance Evaluation

In this section, we report the effectiveness and efficiency test results of our new proposed techniques. Our methods are implemented on a Windows XP machine with a Core 2 Duo CPU (2.8 and 2.8 GHz) and 4-GB main memory. Programs are compiled by Microsoft Visual C++ 6.0.

Real data sets. We use two different data sets, the DBLP data and the Internet Movie database (IMDB), that have been widely used in examination of deterministic keyword graph search [1], [4], [5], [2]. We generate a node-labeled directed graph from the DBLP data (http://dblp.uni-trier.de/xml/) with 985M of the raw XML data. The original XML data are a tree in which each paper is a small subtree. To make it a graph, we add two types of nontree edges. First, we connect papers through citations. Second, we make the same author under different papers share a common node. Finally, we get a graph containing 152K papers, 1,356K nodes, 3,158K edges, and 178K distinct keywords. For the IMDB data set (http://www.imdb.com), we generate a graph from the movie-link table using movie titles as nodes and links between movies as edges. The graph contains 72K nodes, 250K edges, and 48K distinct keywords. To simulate uncertain graphs, we generate existence probabilities for edges of the DBLP and IMDB graphs following Gaussian distribution $(mean, var)$ [19], [20], [22]. In the Gaussian distribution, the mean value (*mean*) is 0.3-0.7, and the default value is 0.5; the variance (var) is 0.1.

We list typical queries for the two data sets in Table 1, and the default queries are DQ3 and IQ3. The querying probability threshold is 0.3-0.7, and the default value is 0.5. The value of top-k is 5-25, and the default value is 15.

The existence, path-based and tree-based probabilistic pruning techniques are, respectively, called *Ext*, *Path* and *Tree* in experiments. *Tree* consists of pruning 3 and halting condition. The method of using tightest upper bound in the tree-based probabilistic pruning is called *OPT-Tree*. As introduced in Section 4, we divide the domain of $Pr^k(P)$ (0, 1] into h prob-intervals. The value of h is set to {150, 200, 250}, and corresponding methods are called *Path-150*, *Path-200*, and *Path-250*. The default method in experiments is *Path-200*, i.e., $Path = Path - 200$. The method of fully materializing values of k is called *All*. In verification, the

TABLE 1
List of Queries

Dataset	Query	Keywords	# Keyword nodes
DBLP	DQ1	image segmentation	(38426, 9752)
DBLP	DQ2	algorithm graph	(25572, 12495)
DBLP	DQ3	query XML database	(8153, 5856, 2047)
DBLP	DQ4	gray data ming	(1865, 77619, 12498)
DBLP	DQ5	westbrook software cloud performance	(72, 37943, 3284, 27966)
IMDB	IQ1	spielberg violence	(9, 175)
IMDB	IQ2	Keanu matrix Nicole	(2, 215, 658)
IMDB	IQ3	world university join	(456, 64, 3752)
IMDB	IQ4	street party heaven	(1227, 861, 548)
IMDB	IQ5	Hepburn summer dream love	(12, 225, 536, 1096)

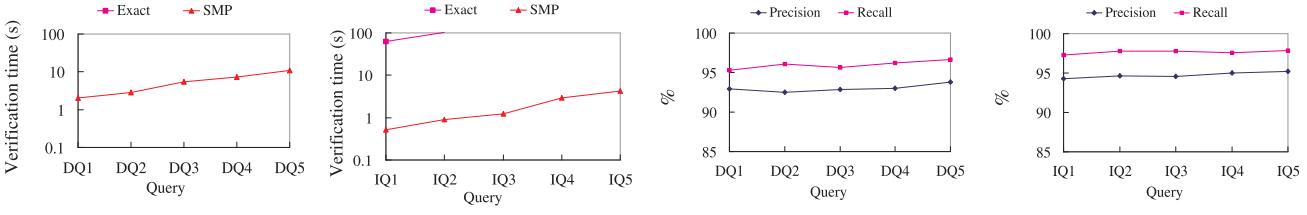
sampling algorithm is called *SMP*, and the method given by (6) is called *Exact*. Since there are no pervious works on the topic studied in this paper, we also compare the proposed algorithms with *Exact* that scans subtrees in D one by one. The complete algorithm proposed in this paper is called *PKI*, i.e., the query procedure given in Fig. 3.

In the first experiment, we demonstrate the efficiency of *SMP* against *Exact* in verification step. We first run three probabilistic pruning rules against the DBLP data set to create candidate sets. The candidate sets are then verified for calculating $Pr^k(T)$ using proposed algorithms. Fig. 7 reports the result, from which we know *SMP* is efficient at both data sets, while the curve of *Exact* grows exponentially. The approximation quality of *SMP* is measured by the *precision* and *recall* metrics with respect to query size shown in Figs. 7c and 7d. Precision is the percentage of true subtrees in the result subtrees. Recall is the percentage of returned subtrees in the true subtrees. The experimental results verify that *SMP* has a very high approximation quality with very high precision and recall (both larger than 90 percent) at the DBLP or IMDB data set. In the rest experiments, we use *SMP* for verification.

Fig. 8 reports candidate sizes and pruning time of *Ext*, *Path*, *Tree*, and *OPT-Tree* with respect to probability thresholds. From the results, we know that candidate sizes of four pruning methods all decrease with the increase of probability threshold, since larger thresholds can remove more false subtrees with low confidences. As shown in Figs. 8a and 8b, the candidate sizes gradually decrease after each filtering step, and the candidate size reaches 15 at the IMDB data set (we search top-10 answers) after tree-based probabilistic pruning at the threshold of 0.7. *Path*, *Tree* and *OPT-Tree* have great pruning powers and filter out a great number of subtrees. *OPT-Tree* has a smaller candidate size than *Tree* due to the tighter upper bound used in *OPT-Tree*. But *OPT-Tree* takes more pruning time than *Tree*, as shown in Figs. 8c and 8d, this is because additional time is required for computing the tightest bound in *OPT-Tree*. Figs. 8c and 8d also show that, *Path* takes least time, since *Path* only needs to look up $Pr^k(P)$ in the index while others have to perform online computation.

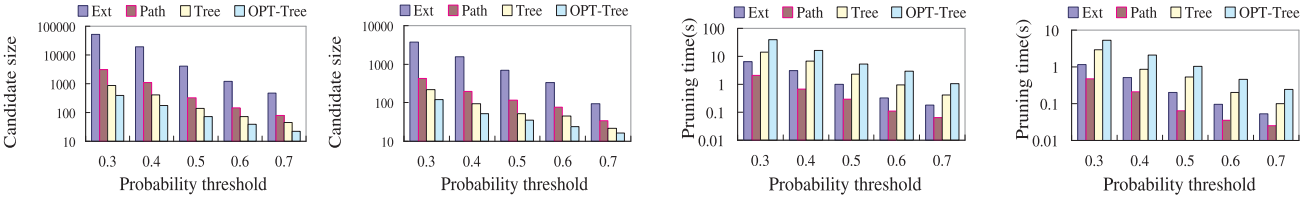
Fig. 9 gives candidate sizes and pruning time of *Ext*, *Path*, *Tree*, and *OPT-Tree* with respect to different top-k. From the results, we know that bars in four figures grow with the increase of k due to that more querying answers need more computing time. We also know that the increasing tendencies of bars in these figures are slower, which shows algorithms are not very sensitive to values of k. The pruning time and candidate size of the DBLP data set are larger than those of the IMDB data set due to the larger graph generated from the DBLP data set.

Fig. 12 shows the results of examination on probabilistic keyword index with respect to different queries. Fig. 12a shows the building time of PKI. The building time consists of time of building index in BLINKS, calculating $Pr^k(P)$, and compressing PKI. From the figure, we observe that different numbers of divided prob-intervals (Path-h) do not have a large impact on building time and *All* has almost the same building time as others. However, as given in Fig. 12b,



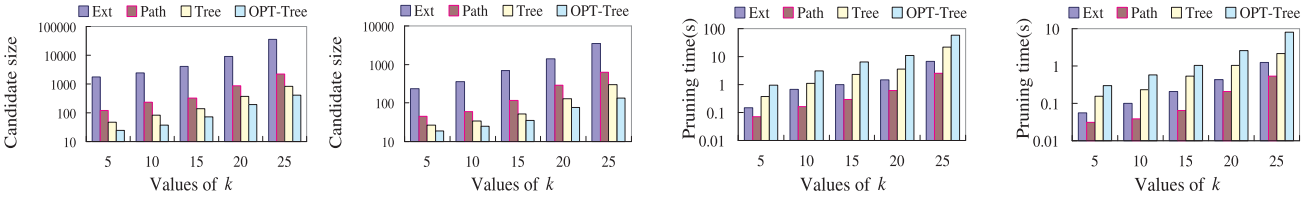
(a) Runtime on DBLP (b) Runtime on IMDB (c) Query quality on DBLP (d) Query quality on IMDB

Fig. 7. Performance on the verification.



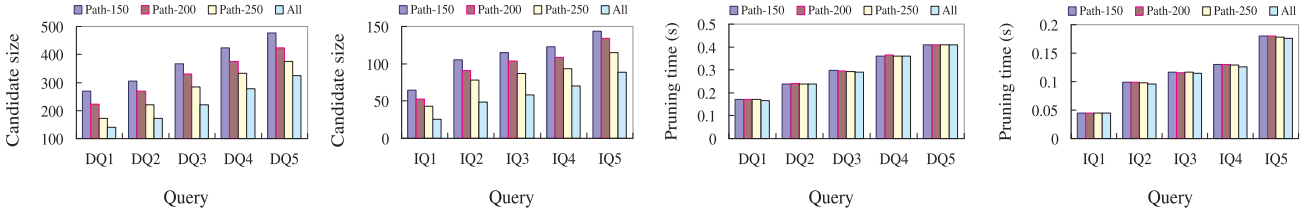
(a) Candidate size on DBLP (b) Candidate size on IMDB (c) Runtime on DBLP (d) Runtime on IMDB

Fig. 8. Scalability to probability threshold.



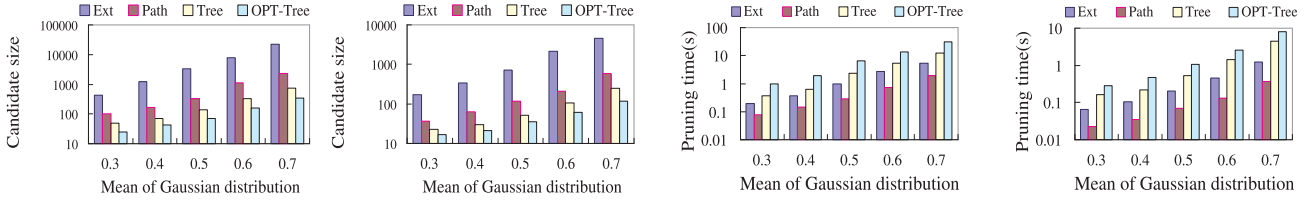
(a) Candidate size on DBLP (b) Candidate size on IMDB (c) Runtime on DBLP (d) Runtime on IMDB

Fig. 9. Scalability to values of top-k.



(a) Candidate size on DBLP (b) Candidate size on IMDB (c) Runtime on DBLP (d) Runtime on IMDB

Fig. 10. Scalability to query.



(a) Candidate size on DBLP (b) Candidate size on IMDB (c) Runtime on DBLP (d) Runtime on IMDB

Fig. 11. Scalability to uncertainty.

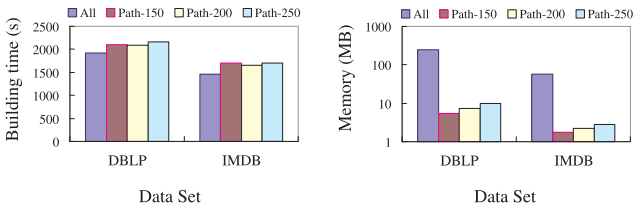
All consumes as much as 50 times memories than the compressed index, which indicates our proposed compressing approach is effective.

Fig. 10 examines the pruning power of probabilistic keyword index with respect to different queries. As expected, *All* has the smallest candidate size shown in Figs. 10a and 10b, since it materializes all values of k. Different h holds a similar expectation that the larger h, the larger pruning power. But PKIs of different h have almost same pruning time as shown in Figs. 10c and 10d.

Fig. 11 gives the impact of uncertainties on the efficiency with respect to varying $mean$ from 0.3 to 0.7. The changes of $mean$ result in a change of the existence probability of an edge. From the figure, we observe that all bars increase with the increase of $mean$, because high existence probabilities increase (weaken) upper bounds used in each probabilistic step, which results in more candidates.

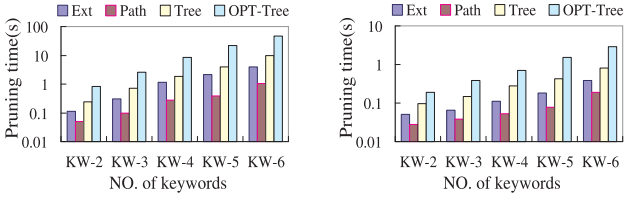
Fig. 13 shows the performance of queries with different number of keywords. To achieve this, we generate five groups of queries randomly at the graph and each group has 10 queries. Number of keywords used in each group of queries is 2, 3, 4, 5, and 6, respectively. We report average runtime in each group of queries in Fig. 13. From the result, we know that the processing time of each pruning step grows, because the increase of the number of keywords leads to more candidate subtrees. The result also shows that all runtime are scalable and efficient at both data sets.

Finally, Fig. 14 reports total query processing time with respect to different probability thresholds. *PKI* denotes the complete algorithm, that is, a combination of *Ext*, *Path*, *OPT-Tree*, and *SMP*. From the result, we know all curves decrease with the increase of probability threshold, because a larger threshold can remove more failed answers with



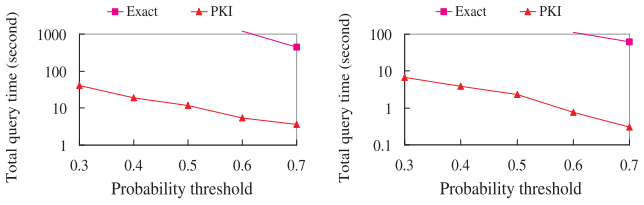
(a) Building time (b) Memory

Fig. 12. Efficiency of probabilistic keyword index.



(a) DBLP (b) IMDB

Fig. 13. Efficiency of queries with different number of keywords.



(a) DBLP (b) IMDB

Fig. 14. Total query processing time.

pruning condition $UpperBound < \epsilon$. In this evaluation, *PKI* has quite efficient runtime and avoids the huge cost of computing the exact $Pr^k(T)$ at both data sets. *PKI* can process queries within 10 seconds on average even at the DBLP data set with more than 1 million nodes. But the runtime of *Exact* increases exponentially, and has gone beyond 1,000 seconds at the threshold of 0.6 at the DBLP data set. The result of this experiment validates the design of our solution.

8 Related Work

The most related work to ours is keyword search over deterministic graphs, that is, given a set of keywords, we are required to return top-k subtress that contain these keywords [1], [4], [5], [2], [6], [3], [7]. Formally, it is NP-hard to compute the exact top-k subtrees because it is an instance of the group Steiner tree problem [33]. To avoid the hard problem, greedy algorithms are used to find the approximate top-k results. The typical approaches are backward search in BANKS-I [1], and bidirectional search, in BANKS-II [2], which is used to avoid the hub nodes found in the backward search. Ding et al. [4] introduced a dynamic programming approach to find the minimum subtree and approximate top-k subtrees. Kimelfeld and Sagiv [3] and Golenberg and Sagiv [7] proposed a framework to find an approximate result in polynomial time under the data and query complexity. To avoid the NP-completeness of the group Steiner tree problem, producing subtrees with distinct roots are introduced in recent years [2], [6]. BLINKS improves the work of [2] by using an efficient indexing structure [5]. Our work is to find distinct root subtrees, and extends the index in BLINKS with probabilistic information.

Another most related topic is managing and mining uncertain graphs. Papapetrou et al. [19] considered the problem of finding frequent subgraphs of an input uncertain graph under the probabilistic semantics. The authors in [19] extend the mining definition to probabilistic expectation semantics and give an efficient randomized algorithm to guarantee its quality [20]. Zou et al. [21] and Jin et al. [22] studied finding top k-cliques and distance-constraint reachability query in a single uncertain graph. Hua and Pei [34] and Kollios et al. [35] study shortest path queries and clustering in uncertain graphs, respectively. Potamias et al. [18] proposed new distance functions between nodes in uncertain graphs that extend shortest path distances from deterministic graphs and studied k-nearest neighbor queries (k-NN) over uncertain graphs. Yuan et al. [36], [37] studied subgraph search on uncertain graphs. Different from this paper, [36] and [37] developed pruning techniques derived from the Inclusion-Exclusion principle and constructed index based on frequent subgraph features, because query types in this paper and [36] are completely different. Thus, techniques in [36] and [37] cannot be used to process probabilistic keyword search. Deshpande et al. [38] used graphical models, i.e., Bayesian network, to manage uncertain data. However, techniques in [38] focus on process SQL queries on probabilistic data and cannot be used to manage uncertain graph data.

9 Conclusion

Uncertain graphs pervasively exist in real applications such as RDF, where data often exhibit uncertainties. In this paper, we study the problem of keyword search over uncertain graphs. To efficiently tackle this problem, we employ the filtering-and-verification framework to answer the query. During the filtering phase, existence, path-based, and tree-based probabilistic rules are proposed to filter out false candidates as many as possible. To support the effective probabilistic pruning rules, we propose a probabilistic keyword index that integrates discriminative structural information with probabilistic features of uncertain graphs. We also develop an optimal partition strategy for PKI, which enables PKI take very small spacial costs and have the maximum pruning power. Moreover, we use the solution to the maximum clique problem to obtain tightest probabilistic bounds for pruning. During the verification phase, a sampling algorithm is designed to compute the final answers. Finally, we confirm our designs through an extensive experimental study.

Acknowledgments

Ye Yuan and Guoren Wang were supported by the NSFC (Grant No. 61025007, 60933001, 61332006, and 61100024), National Basic Research Program of China (973, Grant No. 2011CB302200-G), National High Technology Research and Development 863 Program of China (Grant No. 2012AA011004), and the Fundamental Research Funds for the Central Universities (Grant No. N110404011). Lei Chen was supported by the NSFC (Grant No. 61328202) and the Microsoft Research Asia Theme-based Grant under project MRA11EG05 and Hong Kong RGC GRF 611411.

References

[1] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan, "Keyword Searching and Browsing in Databases Using Banks," *Proc. 18th Int'l Conf. Data Eng. (ICDE)*, pp. 431-440, 2002.

[2] V. Kacholia, S. Pandit, S. Chakrabarti, R. Desai, and H. Karambelkar, "Bidirectional Expansion for Keyword Search on Graph Databases," *Proc. 31st Int'l Conf. Very Large Data Bases (VLDB)*, pp. 505-516, 2005.

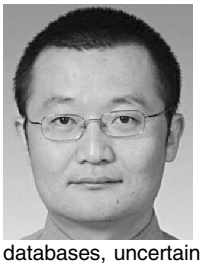
[3] B. Kimelfeld and Y. Sagiv, "Finding and Approximating Top-K Answers in Keyword Proximity Search," *Proc. 25th ACM SIGMOD-SIGACT-SIGART Symp. Principles Database Systems (PODS)*, 2006.

[4] B. Ding, J.X. Yu, S. Wang, L. Qin, X. Zhang, and X. Lin, "Finding Top-k Min-Cost Connected Trees in Databases," *Proc. IEEE 23rd Int'l Conf. Data Eng. (ICDE)*, pp. 836-845, 2007.

[5] H. He, H. Wang, J. Yang, and P.S. Yu, "Blinks: Ranked Keyword Searches on Graphs," *Proc. ACM SIGMOD Int'l Conf. Management of Data,* pp. 305-316, 2007.
[6] B.B. Dalvi, M. Kshirsagar, and S. Sudarshan, "Keyword Search on External Memory Data Graphs," *Proc. VLDB Endowment,* vol. 1, pp. 1189-1204, 2008.
[7] B.K.K. Golenberg and Y. Sagiv, "Keyword Proximity Search in Complex Data Graphs," *Proc. ACM SIGMOD Int'l Conf. Management of Data,* 2008.
[8] H. Huang and C. Liu, "Query Evaluation on Probabilistic Rdf Databases," *Proc. 10th Int'l Conf. Web Information Systems Eng. (WISE),* pp. 307-320, 2009.
[9] D. Liben-Nowell and J. Kleinberg, "The Link Prediction Problem for Social Networks," *Proc. 12th Int'l Conf. Information Knowledge Management (CIKM),* pp. 556-569, 2003.
[10] E. Adar and C. Re, "Managing Uncertainty in Social Networks," *IEEE Data Eng. Bull.,* vol. 30, no. 2, pp. 15-22, June 2007.
[11] A. Nierman and H.V. Jagadish, "ProTDB: Probabilistic Data in Xml," *Proc. Int'l Conf. Very Large Data Bases (VLDB),* 2002.
[12] P. Senellart and S. Abiteboul, "On the Complexity of Managing Probabilistic Xml Data," *Proc. 26th ACM SIGMOD-SIGACT-SIGART Symp. Principles Database Systems (PODS),* 2007.
[13] B. Kimelfeld, Y. Kosharovsky, and Y. Sagiv, "Query Efficiency in Probabilistic Xml Models," *Proc. ACM SIGMOD Int'l Conf. Management of Data,* 2008.
[14] P. Domingos and M. Richardson, "Mining the Network Value of Customers," *Proc. Seventh ACM SIGKDD Int'l Conf. Knowledge Discovery Data Mining (KDD),* 2001.
[15] D.J. Abadi, A. Marcus, S.R. Madden, and K. Hollenbach, "Scalable Semantic Web Data Management Using Vertical Partitioning," *Proc. 33rd Int'l Conf. Very Large Data Bases (VLDB),* pp. 411-422, 2007.
[16] D. Suciu and N.N. Dalvi, "Foundations of Probabilistic Answers to Queries," *Proc. ACM SIGMOD Int'l Conf. Management of Data,* p. 963, 2005.
[17] N.N. Dalvi and D. Suciu, "Management of Probabilistic Data: Foundations and Challenges," *Proc. ACM SIGMOD-SIGACT-SIGART Symp. Principles Database Systems (PODS),* pp. 1-12, 2007.
[18] M. Potamias, F. Bonchi, A. Gionis, and G. Kollios, "K-Nearest Neighbors in Uncertain Graphs," *Proc. VLDB Endowment,* vol. 3, pp. 997-1008, 2010.
[19] O. Papapetrou, E. Ioannou, and D. Skoutas, "Efficient Discovery of Frequent Subgraph Patterns in Uncertain Graph Databases," *Proc. 14th Int'l Conf. Extending Database Technology (EDBT),* 2011.
[20] Z. Zou, H. Gao, J. Li, and S. Zhang, "Mining Frequent Subgraph Patterns from Uncertain Graph Data," *IEEE Trans. Knowledge and Data Eng.,* vol. 22, no. 9, pp. 1203-1218, Sept. 2010.
[21] Z. Zou, J. Li, H. Gao, and S. Zhang, "Finding Top-K Maximal Cliques in an Uncertain Graph," *Proc. 26th Int'l Conf. Data Eng. (ICDE),* 2010.
[22] R. Jin, L. Liu, B. Ding, and H. Wang, "Distance-Constraint Reachability Computation in Uncertain Graphs," *Proc. VLDB Endowment,* vol. 4, pp. 551-562, 2011.
[23] G. Cormode, F. Li, and K. Yi, "Semantics of Ranking Queries for Probabilistic Data and Expected Ranks," *Proc. IEEE 25th Int'l Conf. Data Eng. (ICDE),* 2009.
[24] M. Soliman, I. Ilyas, and K. Chang, "Top-K Query Processing in Uncertain Databases," *Proc. IEEE 23rd Int'l Conf. Data Eng. (ICDE),* 2007.
[25] C. Re, N. Dalvi, and D. Suciu, "Efficient Top-K Query Evaluation on Probabilistic Data," *Proc. IEEE 23rd Int'l Conf. Data Eng. (ICDE),* 2007.
[26] K. Yi, F. Li, D. Srivastava, and G. Kollios, "Efficient Processing of Top-K Queries in Uncertain Databases," *Proc. IEEE 24th Int'l Conf. Data Eng. (ICDE),* 2008.
[27] M. Kozlov, S. Tarasov, and L. Hacijan, "Polynomial Solvability of Convex Quadratic Programming," *Math. Doklady,* vol. 20, pp. 1108-1111, 1979.
[28] J. Li, B. Saha, and A. Deshpande, "A Unified Approach to Ranking in Probabilistic Databases," *Proc. VLDB Endowment,* vol. 2, pp. 502-513, 2009.
[29] M. Mitzenmacher and E. Upfal, *Probability and Computing: Randomized Algorithms and Probabilistic Analysis.* Cambridge Univ. Press, 2005.
[30] D. Angluin and L.G. Valiant, "Fast Probabilistic Algorithms for Hamiltonian Circuits and Matchings," *Proc. Ninth Ann. ACM Symp. Theory Computing (STOC),* 1977.
[31] E. Balas and J. Xue, "Weighted and Unweighted Maximum Clique Algorithms with Upper Bounds from Fractional Coloring," *Algorithmica,* vol. 15, pp. 397-412, 1996.
[32] M. Hua, J. Pei, W. Zhang, and X. Lin, "Ranking Queries on Uncertain Data: A Probabilistic Threshold Approach," *Proc. ACM SIGMOD Int'l Conf. Management of Data,* 2008.
[33] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman, 1979.
[34] M. Hua and J. Pei, "Probabilistic Path Queries in Road Networks: Traffic Uncertainty Aware Path Selection," *Proc. 13th Int'l Conf. Extending Database Technology (EDBT),* pp. 347-358, 2010.
[35] G. Kollios, M. Potamias, and E. Terzi, "Clustering Large Probabilistic Graphs," *IEEE Trans. Knowledge and Data Eng.,* vol. 25, no. 2, pp. 325-336, Feb. 2013.
[36] Y. Yuan, G. Wang, L. Chen, and H. Wang, "Efficient Subgraph Similarity Search on Large Probabilistic Graph Databases," *Proc. VLDB Endowment,* vol. 5, pp. 800-811, 2012.
[37] Y. Yuan, G. Wang, H. Wang, and L. Chen, "Efficient Subgraph Search over Large Uncertain Graphs," *Proc. VLDB Endowment,* vol. 4, pp. 876-886, 2011.
[38] A. Deshpande, L. Getoor, and P. Sen, *Graphical Models for Uncertain Data.* Springer, 2009.

Ye Yuan received the BS, MS, and PhD degrees in computer science from Northeastern University in 2004, 2007, and 2011, respectively. He is currently an associate professor in the Department of Computer Science, Northeastern University, China. His research interests include probabilistic database and graph database.

Guoren Wang received the BSc, MSc, and PhD degrees from the Department of Computer Science, Northeastern University, China, in 1988, 1991, and 1996, respectively. Currently, he is a professor in the Department of Computer Science, Northeastern University, China. His research interests include XML data management, query processing and optimization, bioinformatics, high-dimensional indexing, parallel database systems, and P2P data management. He has published more than 100 research papers.



Lei Chen received the bachelor's degree in computer science from Tianjin University, China, in 1994, the master's degree in computer science from the Asian Institute of Technology in 1997, and the PhD degree in computer science from the University of Waterloo, Canada. He is currently an associate professor of computing science at Hong Kong University of Science and Technology, China. His research interests include multimedia databases, graph databases, uncertain and probabilistic databases.

Haixun Wang received the BS and MS degrees in computer science from Shanghai Jiao Tong University in 1994 and 1996, respectively, and the PhD degree in computer science from the University of California, Los Angeles, in 2000. He is currently a senior researcher in Microsoft Research Asia. His interests include graph database and graph mining.

▷ **For more information on this or any other computing topic, please visit our Digital Library at** www.computer.org/publications/dlib.