Cheng YR, Yuan Y, Chen L *et al.* Threshold-based shortest path query over large correlated uncertain graphs. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 30(4): 762–780 July 2015. DOI 10.1007/s11390-015-1559-5

Threshold-Based Shortest Path Query over Large Correlated Uncertain Graphs

Yu-Rong Cheng [1] (成雨蓉), Ye Yuan [1,*] (袁 野), *Member, CCF, ACM, IEEE*
Lei Chen [2] (陈 雷), *Member, ACM, IEEE*, and Guo-Ren Wang [1] (王国仁), *Senior Member, CCF*

[1] *College of Information Science and Engineering, Northeastern University, Shenyang 110819, China*
[2] *Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Hong Kong, China*

E-mail: cyrneu@gmail.com; yuanye@ise.neu.edu.cn; leichen@cse.ust.hk; wanggr@mail.neu.edu.cn

Received January 31, 2015; revised May 28, 2015.

Abstract With the popularity of uncertain data, queries over uncertain graphs have become a hot topic in the database community. As one of the important queries, the shortest path query over an uncertain graph has attracted much attention of researchers due to its wide applications. Although there are some efficient solutions addressing this problem, all existing models ignore an important property existing in uncertain graphs: the correlation among the edges sharing the same vertex. In this paper, we apply Markov network to model the hidden correlation in uncertain graphs and compute the shortest path. Unfortunately, calculating the shortest path and corresponding probability over uncertain graphs modeled by Markov networks is a #P-hard problem. Thus, we propose a filtering-and-verification framework to accelerate the queries. In the filtering phase, we design a probabilistic shortest path index based on vertex cuts and blocks of a graph. We find a series of upper bounds and prune the vertices and edges whose upper bounds of the shortest path probability are lower than the threshold. By carefully picking up the blocks and vertex cuts, the index is optimized to have the maximum pruning capability, so that we can filter a large number of vertices which make no contribution to the final shortest path query results. In the verification phase, we develop an efficient sampling algorithm to determine the final query answers. Finally, we verify the efficiency and effectiveness of our solutions with extensive experiments.

Keywords shortest path, correlated uncertain graph, probabilistic shortest path index

1 Introduction

As one of the most popular graph queries, shortest path search has been widely used in many applications. For example, we often want to find a path in a road network with the least travel time, which is a typical shortest path query. There exist many studies to find efficient solutions for a shortest path query over a graph, but all of them assume the underlying graphs are deterministic, which are not true in real applications. For example, in a road network, the roads or streets sometimes are not available due to the result of road repairing or traffic jams. This causes the uncertainty of the road network data[1]. Another example is Protein-Protein-Interaction (PPI) data, which shows the interactions between each pair of the proteins in biology. When biologists collect these data from experiments, some interactions that should have existed are missed, while some interactions that should not have existed are recorded[2]. Calculating the shortest path over such PPI data infers the similarity between each pair of the proteins. The smaller the length of the shortest path is, the more similar the proteins will be. Other than that, studies have been done to model social networks, ontology networks, XML and RDF data

Regular Paper
Special Section on Data Management and Data Mining
This work is supported in part by the National Natural Science Foundation of China under Grant Nos. 61332006, U1401256, 61328202, 61173029, the Fundamental Research Funds for the Central Universities of China under Grant No. N130504006, the Hong Kong RGC Project under Grant No. N_HKUST637/13, the National Basic Research 973 Program of China under Grant No. 2014CB340300, Microsoft Research Asia Gift Grant and Google Faculty Award 2013.
*Corresponding Author

as uncertain graphs[2-5], all of which have important real applications. Therefore, finding efficient solutions for shortest path query over uncertain graphs is quite important and necessary.

Most previous related studies used an independent uncertain graph model to represent uncertain graphs[1,6]. Specifically, for an uncertain graph $\mathcal{G}(V, E, W, Pr)$, each edge $e \in E$ has a weight $\omega \in W$ and an existence probability $pr \in Pr$. A possible world graph $g_i(V_i, E_i, W_i)$ is an instance of $\mathcal{G}$. In the independent probabilistic graph model, they assumed that the appearance of edges in $\mathcal{G}$ was independent of each other. Thus, the existance probability of a possible world graph, $pr(g_i)$, is

$$pr(g_i) = \prod_{e \in E_i} pr(e) \prod_{e \notin E_i} (1 - pr(e)).$$

In each possible world graph g_i, there exists a path p_i with the least total weight, which is the shortest path. In other words, the probability that p_i is the shortest path of g_i equals the existence probability of g_i. As p_i may be the shortest path of different possible world graphs, the shortest path probability of p_i (denoted as $SPr(p_i)$) equals the sum of probabilities of the possible world graphs where p_i is the shortest path. That is,

$$SPr(p_i) = \sum_i pr(g_i),$$

where g_i is each of the possible world graphs in which p_i is the shortest path between s and t.

The following example illustrates the independent uncertain graph model and the calculation of shortest path probability based on it.

Example 1 (*Independent Uncertain Graph Model*). Fig.1 illustrates an uncertain graph $\mathcal{G}$ and its possible world graphs g_is with their corresponding probabilities $pr(g_i)$s. The existence probability of possible graph g_2 is

$$pr(g_2) = (1 - 0.8) \times 0.5 \times 0.6 = 0.06.$$

The path $p(sAt)$ is the shortest path from s to t in g_1 and g_4, so the shortest path probability of $p(sAt)$ in $\mathcal{G}$ is

$$SPr(p(sAt)) = pr(g_1) + pr(g_4) = 0.22.$$

Unfortunately, as the independent probabilistic graph model considers that the edges are independent of each other, it naturally ignores the internal relationship between the edges. However, in real applications, the edges sharing the same vertex often affect each other. For example, in road networks, for the roads or streets in the daily life, if the traffic is smooth at one intersection, at the next intersection, it is more likely to be smooth than to be stuck. Similar correlations can also be found in PPI data. For the proteins in the same family, it is more likely to have interactions between them rather than to have no interactions. The independent model fails to consider these correlationships, and cannot show these tendencies such as smooth/stuck or having/not having interactions.

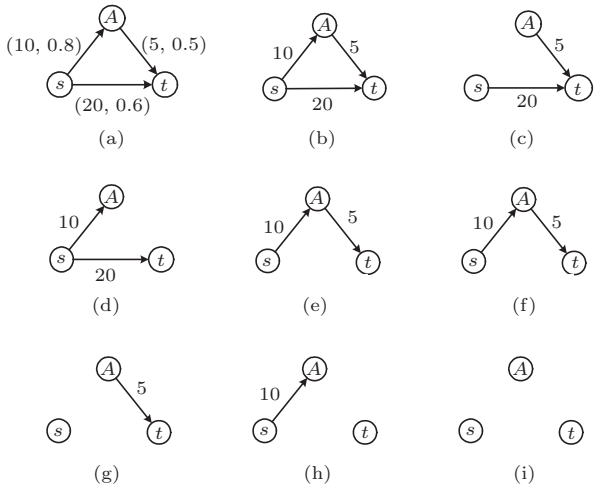


Fig.1. Traditional uncertain graph model. (a) Uncertain graph $\mathcal{G}$. (b) g_1, $pr(g_1) = 0.24$. (c) g_2, $pr(g_2) = 0.06$. (d) g_3, $pr(g_3) = 0.24$. (e) g_4, $pr(g_4) = 0.16$. (f) g_5, $pr(g_5) = 0.06$. (g) g_6, $pr(g_6) = 0.04$. (h) g_7, $pr(g_7) = 0.16$. (i) g_8, $pr(g_8) = 0.04$.

Thus, to address the shortest path query over uncertain graphs, we need to overcome the following changes.

Challenge 1: *How to design a reasonable model?* We propose a novel model based on the Markov network, which captures the correlationship among all the edges using a joint probability table in each clique of a graph (contribution 1). We will illustrate the model in detail in Section 2.

Based on our model, we newly define our threshold-based shortest path problem over correlated uncertain graphs using possible world semantic, which is a widely applied semantic for uncertain graphs. This problem is #P-hard (Theorem 1). A naive solution is to enumerate all the possible graphs, find all the corresponding shortest paths and calculate corresponding shortest path probability in each possible world graph. This naive method is infeasible due to the large graph size and exponential number of possible graphs, which brings the following challenge.

Challenge 2: *As graphs are large, is there an efficient method to get the answers?* We propose a filter-

and-verification framework to reduce the search space and efficiently answer the queries (contribution 2). As we will see, large numbers of vertices and edges make no contribution to the result. In other words, paths through these vertices and edges cannot be the shortest path in any possible graph, or the shortest path probabilities of these paths are too small to exceed the threshold. Thus, it helps to improve the efficiency by finding an algorithm to filter these vertices and edges.

In the filtering phase, we find some tight upper bounds of the shortest path probabilities. To manage these upper bounds, we design a probabilistic shortest path index and propose a pruning algorithm (detailed in Section 4). Furthermore, as building the optimal index is an NP-hard problem, we provide an $O(\log n)$-approximate algorithm (n is the number of vertices in the graph) with polynomial time complexity (detailed in Subsection 4.3).

Although the filtering phase can prune large numbers of "useless" vertices and edges, computing the final shortest path probability is still #P-hard. Thus, in the verification phase, we use a sampling method to estimate the final shortest path probability of each candidate path.

The rest of this paper is organized as follows. In Section 2, we present our new model in detail. In Section 3, we formally propose the problem definition and describe our filtering-and-verification framework. Then in Section 4, we illustrate our pruning algorithm and construction of index. In Section 5, we introduce our sampling algorithm. Finally, we present the experimental study results in Section 6, related work in Section 7, and conclusion in Section 8.

2 Correlated Uncertain Graphs

Regardless of the uncertainty, we use simple graphs just as other studies on graph database[7-11]. Without loss of generality, we mainly process undirected simple graphs.

Definition 1 (Correlated Uncertain Graphs). *A correlated uncertain graph $\mathcal{G}\{g_c(V,E,W),Pr\}$ is composed of a weighted undirected simple graph $g_c(V,E,W)$ (V is the set of vertices, E is the set of edges, and W is the set of weights of the edges) and a probability mass function $Pr : E \to (0,1]$ showing the joint probabilities (denoted as JPr) of the existence of the edges in maximal cliques of $\mathcal{G}$.*

Example 2 (*Correlated Uncertain Graphs*). Fig.2(a) together with Table 1 is a correlated uncertain graph $\mathcal{G}$. The maximal cliques of g_c in Fig.2(a) are $c_1(ABD)$ and $c_2(ACD)$. The joint probability of the edge sets $\{e_{AB}, e_{AD}, e_{BD}\}$ and $\{e_{AC}, e_{AD}, e_{CD}\}$ is shown in Table 1. In this table, edge $e = 1$ means this edge exists, and $e = 0$ means e does not exist. For example, the first line of Table 1 means the probability that none of the edges (e_{BA}, e_{BD} and e_{AD}) exists is 0.05.

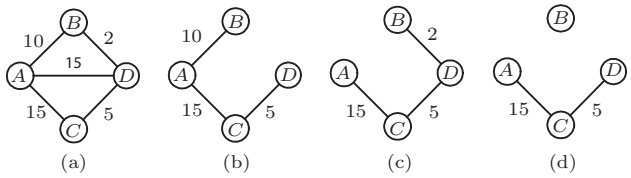


Fig.2. Correlated uncertain graph model. (a) g_c. (b) g_1, 0.005. (c) g_2, 0.010. (d) g_3, 0.015.

Table 1. Joint Probabilities

Clique ABD				Clique ACD			
e_{BA}	e_{BD}	e_{AD}	$JPr(ABD)$	e_{AD}	e_{AC}	e_{DC}	$JPr(ACD)$
0	0	0	0.05	0	0	0	0.10
0	0	1	0.05	0	0	1	0.10
0	1	0	0.10	0	1	0	0.10
0	1	1	0.10	0	1	1	0.10
1	0	0	0.05	1	0	0	0.10
1	0	1	0.50	1	0	1	0.10
1	1	0	0.05	1	1	0	0.20
1	1	1	0.10	1	1	1	0.20

Note that the number of the joint probabilities of the edges in a maximal clique increases exponentially with the number of edges in this clique. Thus when there are too many edges in this clique, the joint probabilities may not be able to be stored in the memory or even disc. There are two situations of the joint probabilities when there are too many edges in one maximal clique. 1) A small part of the joint probabilities are much larger than those of the others. 2) All the joint probabilities are more or less equal to each other. If it is in situation 1, we just store the small part of joint probabilities which are much larger than the others. According to the approximate principles in numeric analysis theory[12], if a joint probability is 1/100 times smaller than the maximum joint probability, it can be treated as 0. If it is in situation 2, it means that all the joint probabilities are nearly equal to 0. For example, if there are 10 edges in the clique, each joint probability is nearly equal to 1/1 028, which is so small that it can be treated as 0. We can just store a 0 for all the joint probabilities.

The reason why we use maximal clique to define the correlationships is that in real applications, whether one edge exists often has more influence on the edges in the same maximal clique than on the other edges. For example, as stated in Section 1, in the road network, whether one road or street is smooth/stuck depends on the roads or streets nearby. If the roads or streets are far from each other, there hardly exists such influence. Another example is in the PPI data. The proteins in the same maximal clique are considered to be in the same family, and such correlationships only exist among the proteins in the same family (i.e., maximal clique).

Given an uncertain graph, a possible world graph is one of its instances. For example, Figs.2(b)∼2(d) are three of all the possible graphs of $\mathcal{G}$. The following definition provides a method to calculate the probability of the possible world graphs.

Definition 2 (Conditional Independent). *In a Markov network, for three vertex sets, X, Y and Z in a graph, if removing the vertices in X, there are no paths between any vertex in Y and any vertex in Z, and then Y and Z are conditionally independent given variables in X*[13].

Thus, according to Definition 2, if $g\{V', E', W'\}$ is a possible world graph of a correlated uncertain graph $\mathcal{G}$, the existence probability of g ($Pr(g)$) equals

$$Pr(g) = \prod_{\substack{i,\{e_j=1|1\leqslant j\leqslant n_e\}\subseteq E',\\ \{e_j=0|1\leqslant j\leqslant n_e\}\subseteq (E-E')}} JPr_i(e_j), \qquad (1)$$

where n_e is the number of edges in the clique.

For example, in Fig.2(a), $\{ABD\}$ is conditionally independent of $\{ACD\}$ on condition $\{AD\}$. From (1), the existence probability of possible graph g_1 is

$$\begin{aligned} Pr(g_1) &= JPr(e_{BA}=1, e_{BD}=0, e_{AD}=0) \times \\ &\quad JPr(e_{AD}=0, e_{AC}=1, e_{DC}=1) \\ &= 0.05 \times 0.10 = 0.005. \end{aligned}$$

Note that the edges in different cliques are conditional independent of each other, not absolute independent of each other. In other words, the edges such as e_{AB} and e_{AC} in Fig.2(a) still have correlations. We can use the methods introduced in [14] to calculate $pr(e_{AB}|e_{AC})$ according to the joint probability tables. Moreover, there is also another correlated uncertain graph model introduced in [15], which is based on Bayesian network. There are three advantages to apply Markov network theory into our correlated uncertain graph model instead of using Bayesian network. 1) Markov network can model any undirected graphs while Bayesian network can only model directed acyclic graphs. 2) The Bayesian network based model can be equivalently changed into our Markov network based model using the methods introduced by [13]. 3) Although both of the two correlated uncertain graph models can show the correlationship among all edges in the graph, the storage size of the correlationships in Markov network is often smaller than that in Bayesian network[14].

Our Correlated Model vs the Independent Model. Our correlated uncertain graph model is more reasonable than the independent model in many real applications. For example, in a social network, a vertex presents a user, and an edge between two users presents that the two users know each other. Specifically, in Fig.2(a), A, B and D are three users, and the edge e_{BA} means that the two users (A and B) know each other. If A knows B and B knows D, under this condition, A and D will have a high probability to know each other. Another example is in a protein-protein interaction (PPI) network. If one protein has a high probability to be similar with two other proteins respectively, then these two proteins would have a high probability to be similar. However, the independent model fails to present these correlationships. Moreover, whether one edge exists or not depends on the other edges in the same clique in our correlated uncertain graph model, and the result of shortest path query calculated by our model should have a high confidence to be a better result. For example, in PPI network, the shortest path between two proteins is often used to evaluate the similarity of the two proteins[2,6]. If there exist paths whose shortest path probability is higher than a given threshold, the two proteins can be treated to be similar. Our experiment in Section 6 proves that the results given by our model are more accurate than those given by the independent model.

3 Problem Statement

Definition 3 (Shortest Path Probability (SPr)). *The shortest path probability of a path p in a correlated uncertain graph $\mathcal{G}$ equals the sum of the probabilities of the possible world graphs in which p is the shortest path.*

For example, in Fig.2, path $p(ACD)$ is the shortest path in g_1, g_2 and g_3 among all possible world graphs of $\mathcal{G}$. Thus,

$$SPr(p(ACD)) = pr(g_1) + pr(g_2) + pr(g_3) = 0.03.$$

The threshold-based shortest path query is a widely used problem for shortest path query over independent uncertain graph model in the previous work[1,17]. Similar to that in the independent model, we propose our problem definition over correlated uncertain graphs.

Definition 4 (Threshold-Based Shortest Path Query). *Given a correlated uncertain graph $\mathcal{G}(g_c(V,E,W),Pr)$, two vertices $s,t \in V$, and a probabilistic threshold τ, a threshold-based shortest path query returns a path set $\{p_1,p_2,\ldots,p_n\}$ in which each path p_i has an $SPr(p_i)$ larger than the threshold.*

Theorem 1. *The calculation of $SPr(p_i)$ under correlated uncertain graph model is #P-hard*[1,15].

Framework of Our Solution. As discussed in Section 1, to efficiently solve our problem, we need to overcome two bottlenecks, the large graph size and the exponential number of possible world graphs. Thus, we design a filtering-and-verification framework, which is outlined in Fig.3.

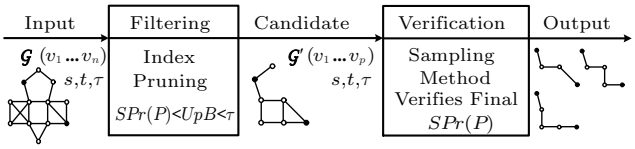


Fig.3. Filtering-and-verification framework.

When query comes, we first conduct the filtering step to prune the vertices and edges, through which SPr can never be larger than τ. We provide a series of upper bounds (UpBs, Theorem 2) for SPr based on path cuts (Definition 7) of g_c. To manage these UpBs, we design a probabilistic shortest path index (Definition 8), which is a tree based on blocks and vertex cuts (Definition 6 and Definition 5). By finding paths from s to t in the index, simultaneously, we can get UpBs from the largest one to the smallest one. Then, we can first test whether the largest upper bound (noted as $UpB_{\max}$) is larger than threshold τ. If $UpB_{\max} < \tau$, we can prune all the vertices and edges in the original graph safely. Otherwise, we test the second largest $UpB\ldots$. Once a $UpB < \tau$, we can prune the remaining vertices and edges. After the filtering step, we get a candidate set of vertices and edges, which is a super set of the shortest path result set.

Then we conduct the verification step based on a sampling algorithm over the candidate set to determine the final result.

4 Filtering Phase

In this section, we first introduce our UpBs and the probabilistic shortest path index in Subsection 4.1. Then we illustrate our pruning algorithm in Subsection 4.2. In Subsection 4.3, we present the construction of the index.

4.1 Upper Bounds and Index

Before giving UpBs and our index, we first illustrate three basic definitions, vertex cut, block, and path cut.

Definition 5 (Vertex Cut). *A vertex cut $\mathcal{VC}$ is a set of vertices whose removal leaves a graph disconnected.*

A vertex cut $\mathcal{VC}(v_1 \ldots v_n)$ can be overall treated as one super node comprised by $v_1,\ldots,v_n$.

Definition 6 (Block). *The connected components separated by $\mathcal{VC}$ are called blocks.*

For example, in Fig.4(a), the removal of $\{CE\}$ makes the graph disconnected, so $\mathcal{VC}(CE)$ is a vertex cut. $\mathcal{B}(BAD)$ and $\mathcal{B}(FGHIJKL)$ are two blocks.

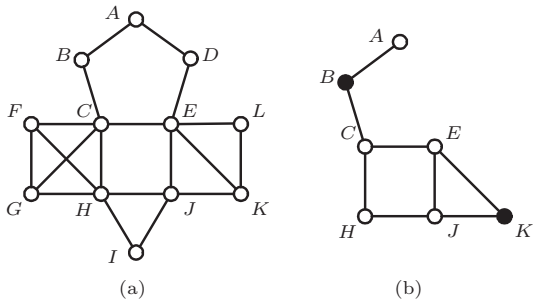


Fig.4. Pruning of a correlated graph. (a) g_c. (b) g_c after pruning.

Definition 7 (Path Cut). *Given a deterministic graph G and a set of paths P, a path cut $\mathcal{PC}$ is a set of vertices that all paths $p \in P$ must pass through.*

Property 1. If two vertices, s and t, in a graph are in the different blocks made by a vertex cut, and we denote the set of all paths from s to t as $P_{s\to t}$, then the vertex cut must be a path cut of $P_{s\to t}$.

This property illustrates the relation between vertex cut and path cut, which stands obviously. For example, in Fig.4(a), as vertices B and K are in different blocks made by $\mathcal{VC}(CE)$, $\mathcal{VC}(CE)$ is a path cut for all paths from B to K. As a path cut is a special vertex cut, it can also be treated as a super node.

If $\mathcal{PC}$ is the path cut in g_c, it must also be the path cut in all possible world graphs of $\mathcal{G}$. Moreover, for the path set $P_{s\to t}$, it can be divided into three parts, 1) the path set from s to $\mathcal{PC}$, 2) the path set from one vertex in $\mathcal{PC}$ to another vertex in $\mathcal{PC}$, and 3) the path set

from $\mathcal{PC}$ to t. We note the three parts as P_1, P_2 and P_3 respectively. Correspondingly, a path $p \in P_{s\to t}$ is divided into three parts, $p_1 \in P_1$, $p_2 \in P_2$ and $p_3 \in P_3$.

Lemma 1. *Given a deterministic graph G, a start point s, and a terminal point t, $\mathcal{PC}$ is a path cut of $P_{s\to t}$. If $p \in P_{s\to t}$ is the shortest path from s to t, then p_1, p_2 and p_3 of p must be the shortest paths of the three parts P_1, P_2 and P_3 respectively.*

Proof. Because p is the shortest path in $P_{s\to t}$, we can see that p_2 must stay in the same vertex of $\mathcal{PC}$. In other words, the length of p_2 must be 0, which is certainly the shortest path in P_2. Then we use the contradiction method to prove p_1 and p_3 are the shortest paths in P_1 and P_3 respectively. The contradiction must be in the following three situations. 1) p_1 is not the shortest but p_3 is; 2) p_1 is the shortest but p_3 is not; 3) neither p_1 nor p_3 is the shortest. For situation 1), if there exists a path p_1' from s to $\mathcal{PC}$ such that p_1' is shorter than p_1, $p_1' + p_3$ must be shorter than $p_1 + p_3$. In other words, $p = (p_1 + p_3)$ is not the shortest path from s to t, which is inconsistent with the proposition. The proof of situations 2 and 3 is similar to that of situation 1. □

Theorem 2. *Given a correlated uncertain graph $\mathcal{G}$, a start point s and a terminal point t, if $\mathcal{PC}$ is a path cut, for any path $p \in P_{s\to t}$ in g_c, the shortest path probability from s to $\mathcal{PC}$ is the upper bound of the shortest path probability from s to t. That is,*

$$SPr(p_{s\to t}) \leqslant SPr(p_1) = UpB.$$

Proof. Lemma 1 means, $SPr(p_{s\to t}) = SPr(p_1 \wedge p_3)$. Because SPr is a kind of marginal probability, $SPr(p_1)$ and $SPr(p_3)$ are independent of each other. Thus,

$$\begin{aligned} SPr(p_1 \wedge p_3) &= SPr(p_1) \times SPr(p_3) \leqslant SPr(p_1) \\ &= UpB. \end{aligned} \tag{2}$$

The equal sign stands when paths reach t. □

If paths from s to t pass through many path cuts, such as $\mathcal{PC}_1$, $\mathcal{PC}_2$, ..., based on Theorem 2, we have

$$\begin{aligned} SPr(p_{s\to t}) &\leqslant UpB_1 = SPr(p_{s\to \mathcal{PC}_1}) \\ &\leqslant UpB_2 = SPr(p_{s\to \mathcal{PC}_1 \to \mathcal{PC}_2}) \leqslant \cdots \\ &\leqslant UpB_x = SPr(p_{s\to \mathcal{PC}_1 \to \mathcal{PC}_2 \to \cdots \to t}). \end{aligned} \tag{3}$$

Once some $UpB < \tau$, we can prune the paths safely. Therefore given any query, we need to find the path cuts and calculate the upper bounds during the pruning process. To manage the upper bounds, we design a probabilistic shortest path index.

Definition 8 (Probabilistic Shortest Path Index). *The probabilistic shortest path index of a graph G is a tree $T = (V(T), E(T))$. $V(T)$ is composed of two kinds of nodes, $\mathcal{VC}$ (the vertex cuts of G), and $\mathcal{B}$ (the blocks made by $\mathcal{VC}$s). $\mathcal{VC}$s and $\mathcal{B}$s are connected alternatively by tree edges $e(\mathcal{B}, \mathcal{VC}) \in E(T)$. The blocks connected with the same vertex cut are called adjacent blocks, and the vertex cuts connected with the same block are called adjacent vertex cuts. Each edge is labelled by a probability which is the maximum SPr from each $v \in \mathcal{B}$ to its adjacent vertex cuts $\mathcal{VC}$. We note these maximum SPr as $SPr(e)_m$.*

Note that to make our statement easier to understand, when we record the blocks, we keep the vertices in the vertex cut that separates this block still in this block. That is, $\mathcal{VC} = \mathcal{B}_i \cap \mathcal{B}_j$ where $\mathcal{B}_i$ and $\mathcal{B}_j$ are two adjacent blocks w.r.t. $\mathcal{VC}$.

Without loss of generality, we consider g_c as a connected graph. Otherwise, we can build an index for each connected component of g_c. Note that the index is a tree that does not have a necessary root. In other words, each node of the tree can be equivalently treated as a root.

Example 3 (Probabilistic Shortest Path Index). The tree in Fig.5 is an index of Fig.4(a). The vertex sets embraced by circles such as $\mathcal{B}(FGCH)$ are blocks. The vertex sets embraced by rectangles such as $\mathcal{VC}(CH)$ are vertex cuts. For simplicity, we use $\mathcal{B}_9$ instead of $\mathcal{B}(FGCH)$. As $\mathcal{B}_7$ and $\mathcal{B}_9$ are separated by $\mathcal{VC}_8$, they are both connected to $\mathcal{VC}_8$. Then, we treat each block as a distinct part and calculate the maximum SPr from vertices in the block to the corresponding vertex cut. For example, Fig.6(a) shows $\mathcal{B}_9$ and $\mathcal{VC}_8$. As $\mathcal{VC}_8$ can be treated as a super node, $\mathcal{B}_9$ is essentially changed into the graph shown in Fig.6(b). Then, we calculate $SPr(P_{F\to \mathcal{VC}_8})$ and $SPr(P_{G\to \mathcal{VC}_8})$ in Fig.6(a) respectively, i.e., $SPr(p(FC))$, $SPr(p(FHC))$, $SPr(p(FGC))$, $SPr(p(FGHC))$, $SPr(p(FHGC))$ ... Among them, we find the maximum SPr and label it on tree edge $e(\mathcal{B}_9, \mathcal{VC}_8)$ of the index, which is assumed as "0.6" in Fig.5.

4.2 Pruning Algorithm

In this subsection, we assume that we have got the index, and discuss the pruning algorithm based on the index.

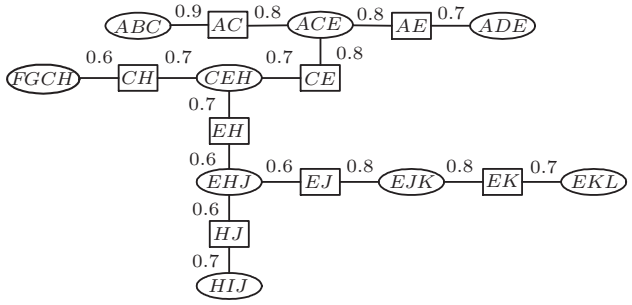
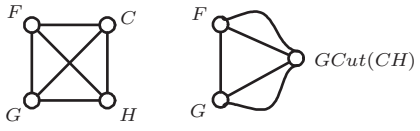


Fig.5. Probabilistic shortest path index.



(a) (b)

Fig.6. Block $\mathcal{B}(FGHC)$ with vertex cut $\mathcal{VC}(CH)$.

4.2.1 Largest and Smallest Upper Bound

We will show that the path from $\mathcal{B}_s$ (the block that s is in) to $\mathcal{B}_t$ (the block that t is in) in the tree provides an upper bound of $SPr(P_{s\to t})$. Some kind of fragment of this tree path provides the largest upper bound. The path that traverses all the vertex cuts in the tree provides the smallest upper bound. To distinguish, we call a path in the index tree a "tree path" (denoted as TP). For example, if $s \in \mathcal{B}_1$, $t \in \mathcal{B}_{13}$, $TP_1(\mathcal{B}_1, \mathcal{VC}_2, \mathcal{B}_3, \mathcal{VC}_6, \mathcal{B}_7, \mathcal{VC}_{10}, \mathcal{B}_{11}, \mathcal{VC}_{12}, \mathcal{B}_{13})$ is a tree path. For simplicity, we use TP_1(1, 2, 3, 6, 7, 10, 11, 12, 13) for short. Moreover, TP_2(1, 2, 3, 4, 5, 4, 3, 6, 7, 10, 11, 12, 13) is also a tree path, and TP_3(1, 2, 7, 10, 13) is a fragment of TP_1. We call the tree path without any repeated nodes, such as TP_1, a main tree path, which is denoted by $mainTP$.

Any tree path in the index can be mapped by a set of paths in the graph, which start from s, pass through all the vertex cuts on the tree path, and terminate at t. For example in Fig.5, if $s = B$ and $t = K$, $mainTP = TP_1(1, 2, 3, 6, 7, 10, 11, 12, 13)$ is mapped by the paths $P_{B\to\mathcal{VC}_2\to\mathcal{VC}_6\to\mathcal{VC}_{10}\to\mathcal{VC}_{12}\to K}$. The paths such as $p(BCHJK)$ in Fig.4(a) satisfy the above path set. Similarly, path $p(BADECHJK)$ is mapped by TP_2. Based on the discussion in Property 1, the vertex cuts on the tree path should be the path cuts of the above paths. According to (3), we can calculate the UpB of these paths by multiplying the $SPr(e)_m$s on the tree edges of the tree path. For example,

$$\begin{aligned} & UpB(TP_1) \\ = \; & SPr(e(\mathcal{B}_1, \mathcal{VC}_2))_m \times \\ & SPr(e(\mathcal{B}_3, \mathcal{VC}_6))_m \times SPr(e(\mathcal{B}_7, \mathcal{VC}_{10}))_m \times \\ & SPr(e(\mathcal{B}_{11}, \mathcal{VC}_{12}))_m \times SPr(e(\mathcal{B}_{13}, \mathcal{VC}_{12}))_m \\ = \; & 0.9 \times 0.8 \times 0.7 \times 0.6 \times 0.8 = 0.241\,92. \end{aligned} \tag{4}$$

Thus, it is obvious that if a path traverses all vertex cuts in the tree, it will provide the smallest upper bound of $SPr(P_{s\to t})$, according to (2) and (3). Now let us discuss what kind of tree path provides the largest upper bound. Intuitively, this kind of tree path should pass though fewer vertex cuts than $mainTP$, such as a fragment of $mainTP$ like $TP_3(1, 2, 7, 10, 13)$, which is mapped by path $p(BCEK)$ in the graph. We define such fragment as follows.

Definition 9 (Shortest Fragment). *A shortest fragment is the fragment of mainTP which contains the smallest number of vertex cuts in the index tree but can still map to at least one path in the original graph. We denote the shortest fragment as TP_{sp}.*

Obviously, such TP_{sp} provides the largest upper bound, because its mapped paths pass through the fewest vertex cuts. The following theorems provide a method to find TP_{sp}.

Theorem 3. *The vertex cuts in TP_{sp} are path cuts of all paths from start vertex s to terminal vertex t.*

Proof. Again, we apply the contradiction method to this proof. Assume, there is a vertex cut $\mathcal{VC}_i$ in TP_{sp} that is not the vertex cut of all paths from s to t. Let TP_{sp} be expressed as $TP_{sp} = (\mathcal{B}_s, \mathcal{VC}_s, \mathcal{B}_1, \mathcal{VC}_1, \ldots, \mathcal{B}_{i-1}, \mathcal{VC}_{i-1}, \mathcal{B}_i, \mathcal{VC}_i, \mathcal{B}_{i+1}, \mathcal{VC}_{i+1}, \ldots, \mathcal{B}_h, \mathcal{VC}_h, \mathcal{B}_t)$. Except $\mathcal{VC}_i$, the other vertex cuts such as $\mathcal{VC}_s$ are path cuts of all paths from s to t. Because of this assumption, there must be a path from s to t that does not pass through $\mathcal{VC}_i$, i.e., there is a shortcut from $\mathcal{VC}_{i-1}$ to $\mathcal{VC}_{i+1}$ directly. Thus, the tree path mapped by this path is $TP' = (\mathcal{B}_s, \mathcal{VC}_s, \mathcal{B}_1, \mathcal{VC}_1, \ldots, \mathcal{B}_{i-1}, \mathcal{VC}_{i-1}, \mathcal{B}_{i+1}, \mathcal{VC}_{i+1}, \ldots, \mathcal{B}_h, \mathcal{VC}_h, \mathcal{B}_t)$, which is shorter than TP_{sp}. Thus, TP' contains fewer $\mathcal{VC}$s than TP_{sp}, which is contradictory to Definition 9. Thus, the vertex cuts in TP_{sp} must be the path cuts of all paths from start vertex s to terminal vertex t. □

According to Theorem 3, finding TP_{sp} equals finding the path cuts of all paths from s to t.

Theorem 4. *The intersection of any two adjacent vertex cuts in TP_{sp} is empty, i.e., $\forall \mathcal{VC}i, \mathcal{VC}_j \in TP_{sp}$, $\mathcal{VC}_i \cap \mathcal{VC}_j = \emptyset$ with adjacent $\mathcal{VC}_i$ and $\mathcal{VC}_j$.*

Proof. Let $TP_{sp} = (\mathcal{B}_s, \mathcal{VC}_s, \ldots, \mathcal{B}_i, \mathcal{VC}_i, \mathcal{B}_j, \mathcal{VC}_j, \mathcal{B}_{j+1}, \mathcal{VC}_{j+1}, \ldots)$. We also apply the contradiction method. Assume $\mathcal{VC}_i \cap \mathcal{VC}_j = Ins \neq \emptyset$, there must

exist a tree path mapped by a path set like $TP = (\mathcal{B}_s, \mathcal{VC}_s, \ldots, \mathcal{B}_i, Ins, \mathcal{B}_j, Ins, \mathcal{B}_{j+1}, \mathcal{VC}_{j+1}, \ldots)$. Thus, there must exist a set of paths that do not pass through $\mathcal{B}_j$ and just stay at Ins. Mapped by these paths, the corresponding tree path can be written by $TP' = (\mathcal{B}_s, \mathcal{VC}_s, \ldots, \mathcal{B}_i, Ins, \mathcal{B}_{j+1}, \mathcal{VC}_{j+1}, \ldots)$. Thus, TP' contains fewer $\mathcal{VC}$s than TP_{sp}, which is contradictory to Definition 9. Therefore the assumption is invalid. □

From Theorem 4, the method to find out the path cuts for all paths from s to t can be achieved by checking the intersection of vertex cuts during the process of traversing the index tree. We conclude this method as $FindTP_{\text{sp}}$, which is shown in Algorithm 1.

Algorithm 1. FindTPsp(T, $\mathcal{B}_s$, $\mathcal{B}_t$, $\mathcal{VC}_{la}$, UpB, τ)

Input: index T, start block $\mathcal{B}_s$, terminal block $\mathcal{B}_t$, the last vertex cut in TP_{sp} $\mathcal{VC}_{la}$, upper bound UpB, threshold τ

Output: TP_{sp}, new upper bound UpB'

```
   // initially VC_la := φ
1  FindTP_sp (T,B_s,B_t,VC_la,UpB,τ) begin
2      while (!visit[B_t]) do
           // Depth-First Traverse the Index Tree
3          if UpB < τ then
4              Return (empty);
5          else if VC_i ∩ VC_la = φ then
6              Insert B_i and VC_i into TP_sp;
7              VC_la := VC_i;
8              UpB* = SPr_m(i);
9              visit[B_{i-1}] := 1;
10     UpB* = SPr_m(e(B_t, VC_last)) ;
11     UpB' := UpB ;
12     Return (TP_sp, UpB');
```

Note the last vertex cut in TP_{sp} is $\mathcal{VC}_{la}$, and initially, $\mathcal{VC}_{la} = \emptyset$. During the process of depth-first traverse on the index tree (lines 2∼9), whenever we meet a vertex cut $\mathcal{VC}_i$, we check whether $\mathcal{VC}_i \cap \mathcal{VC}_{la} = \emptyset$. If so, we insert $\mathcal{B}_i$ and $\mathcal{VC}_i$ into TP_{sp} and multiply current upper bound UpB by $SPr_m(e)$ on tree edge $e(\mathcal{B}_i, \mathcal{VC}_i)$ (lines 5∼9). If the algorithm reaches $\mathcal{B}_t$, we multiply UpB with a $SPr_m(e)$ on tree edge $e(\mathcal{B}_t, \mathcal{B}_{la})$ (line 10, also referenced in (4)). Whenever $UpB < \tau$, we stop the algorithm (line 3).

4.2.2 Algorithm Description

From Algorithm 1, we can find the largest upper bound provided by TP_{sp} during the process of finding the $mainTP$ of the index tree. According to Subsection 4.2.1, we can find the smallest upper bound by traversing the whole index tree. Thus, during the traversing of the index tree, we will first calculate the largest upper bound, then the second largest, ..., the smallest upper bound. Thus, we consider that we first find out the largest upper bound (denoted as $UpB_{\max}$) and test it. If $UpB_{\max} < \tau$, we can prune all blocks and vertex cuts in this tree path safely. If $UpB_{\max} > \tau$, we then test the second largest upper bound. This is the main idea of our pruning algorithm.

Before describing our pruning algorithm, let us first discuss what kind of structures are left in $T - TP_{sp}$.

Property 2. The structures left by $(T - TP_{sp})$ are of two kinds: 1) the fragment of the tree whose blocks and vertex cuts are in $(mainTP - TP_{sp})$, denoted as TF_a, and 2) the fragments of the tree whose blocks and vertex cuts are in $(T - mainTP)$, denoted as TF_b.

For example, TP_3 mentioned above is indeed TP_{sp}. $TP_{a_1} = (3, 6)$ is the tree fragment of 1), and $TP_{b_1} = (5, 4)$ is the tree fragment of 2). TF_as are mapped by the paths passing through not only the vertex cuts in TP_{sp}, but also the vertex cuts in TF_as, such as the path $p(BCHJK)$. Except for going through the vertex cuts in TP_{sp}, TF_bs are mapped by the paths that go through the first vertex cut in TF_b to the last block in TF_b and then go back to the first vertex cut in TF_b, such as $p(BCHIJK)$. Whatever kind of tree fragments is, all these remaining structures can be treated as a new index tree. By recursively looking for TP'_{sp} of each of these structures, we can find the second (third, ...) largest upper bound, which equals $UpB_{\max} \times UpB'_{\max}$ ($UpB_{\max}$ is the largest upper bound provided by TP_{sp}, and $UpB'_{\max}$ is the largest upper bound provided by TP'_{sp}).

Based on the above analysis, the pruning algorithm can be acquired, which is shown in Algorithm 2. Initially, we hash the start vertex s and the terminal vertex t to $\mathcal{B}_s$ and $\mathcal{B}_t$ respectively. If there are more than one block containing s or t, we choose the nearest pair, i.e., there are no other blocks in the $mainTP$ containing s or t except $\mathcal{B}_s$ and $\mathcal{B}_t$. The initial $UpB = 1$. We first perform the function of finding the TP_{sp} of the index tree T, which is denoted as $FindTP_{sp}$ in line 2. The upper bound UpB of TP_{sp} is calculated in function $FindTP_{sp}$. Once $UpB < \tau$, the TP_{sp} returned by $FindTP_{sp}$ will be empty, and the pruning algorithm is interrupted (line 5). Otherwise, we insert TP_{sp} into L, which is a list of blocks and vertex cuts after pruning (line 7) and, check which kind of the remaining structure in $T - TP_{sp}$ is. If it is of kind 1), it must start from a block and terminate at a vertex cut. As $mainTP$ can be written as $mainTP = (\mathcal{B}_s, \mathcal{VC}_s, \ldots, \mathcal{B}_{i-1}, \mathcal{VC}_{i-1}, TF_a, \mathcal{B}_i, \mathcal{VC}_i, \ldots, \mathcal{B}_t)$, to make the TF_a be a sub-tree, we insert $\mathcal{B}_i$ into TF_a. Through this method, the new start block $\mathcal{B}'_s$ is the first block in TF_a, and the new terminal block is $\mathcal{B}_i$ (lines 9∼11). If it is of kind 2), it must start from a vertex cut and terminate at another block, which can be written as $TF_b = (\mathcal{VC}'_s, \ldots, \mathcal{B}'_{t-1},$

$\mathcal{VC}'_{t-1}$, $\mathcal{B}'_t$). Here, $\mathcal{VC}'_s$ must be attached to a block $\mathcal{B}_m$ in $mainTP$. As the paths mapped to TF_b need to go out of $\mathcal{B}_m$ through $\mathcal{VC}'_s$, reach $\mathcal{B}'_t$, and then go back to $\mathcal{B}_m$ through $\mathcal{VC}'_s$, TF_b is essentially $TF_b = (\mathcal{B}_m, \mathcal{VC}'_s, \ldots, \mathcal{B}'_{t-1}, \mathcal{VC}'_{t-1}, \mathcal{B}'_t, \mathcal{VC}'_{t-1}, \mathcal{B}'_{t-1}, \ldots, \mathcal{VC}'_s, \mathcal{B}_m)$. Moreover, the new start block and the terminal block are both $\mathcal{B}_m$ (lines 13∼15). Finally, we recursively call the pruning function (line 16) until TP_{sp} is empty. Following is an example for our pruning algorithm.

Algorithm 2. Pruning(T, s, t, UpB, τ)

Input: index tree T, start vertex s, terminal vertex t, threshold τ

Output: the block and vertex cut list set L after pruning

```
   // Initially, Hash s and t into B_s and B_t, UpB = 1
1  Pruning(T, s, t, τ) begin
2      (TP_sp, UpB'):=FindTP_sp(T, B_s, B_t, UpB, τ);
3      for each TF ⊆ T − TP_sp do
4          if TP_sp is empty then
5              Return (L) ;
6          else
7              Insert TP_sp into L;
8              if TF satisfies TF_a then
9                  B'_s := the first block in TF ;
10                 B'_t = the block in TP_sp which is adjacent to the
                   last vertex cut in TF ;
11                 TF := (B'_s, VC'_s, ..., VC_last, B'_t) ;
12             else if TF satisfies TF_b then
13                 B'_s := the block attached to VC'_s in mainTP ;
14                 B'_t := B'_s ;
15                 TF = (B'_s, VC'_s, ..., B'_{t-1}, VC'_{t-1}, B'_t, VC'_{t-1},
                   B'_{t-1}, ..., VC'_s, B'_s) ;
16             Pruning (TF, B'_s, B'_t, UpB', τ) ;
17     Return (L) ;
```

Example 1 (*Pruning Algorithm*). Assume the start vertex and the terminal vertex of a query are B and K respectively in Fig.4(a), threshold $\tau = 0.4$, and we perform our pruning algorithm on its index shown in Fig.5. TP_{sp} of index tree is shown in Fig.7(a). UpB of TP_{sp} equals $0.9 \times 0.7 \times 0.8 = 0.504 > \tau = 0.4$. Therefore, we put TP_{sp} into candidate set L.

The fragments made by TP_{sp} are shown in Figs.7(b)∼7(f). Specially, the fragments in Figs.7(b) and 7(c) belong to TF_a, and the fragments in Figs.7(d)∼7(f) belong to TF_b. To illustrate clearly, for each tree fragment, we remark its start and terminal blocks or vertex cuts in TP_{sp}, such as $\mathcal{VC}_2$ and $\mathcal{B}_7$ in Fig.7(b). We give each situation an example of searching start vertex and terminal vertex of each fragment. For the fragment shown in Fig.7(b), its start block $\mathcal{B}_3$ is attached to vertex cut $\mathcal{VC}_2$, and its terminal block is $\mathcal{B}_7$. For the fragment shown in Fig.7(e), it essentially equals the tree fragment shown in Fig.7(g).

Recursively executing $TPFind_{sp}$, we can prune each fragment. We still give each situation one example. For TF_{a_1} shown in Fig.7(b), its TP_{sp} is itself, and its UpB equals $0.504 \times 0.8 = 0.403\,2 > \tau = 0.4$. Therefore, this fragment is remained. The TP_{sp} of TF_{b_2} in Fig.7(g) is shown in Fig.7(h). Its UpB equals $0.504 \times 0.8 = 0.403\,2 > \tau = 0.4$. Thus we put this tree fragment into candidate set. Similarly, UpB of the tree fragment shown in Fig.7(i) equals $0.403\,2 \times 0.7 = 0.282\,24 < \tau = 0.4$, and UpB of the tree fragment shown in Fig.7(j) equals $0.504 \times 0.7 = 0.352\,8 < \tau = 0.4$. Thus, we prune these tree fragments. After all, the blocks in candidate set are $\mathcal{B}_1$, $\mathcal{B}_3$, $\mathcal{B}_7$, $\mathcal{B}_{13}$ (omitting vertex cuts). The candidate subgraph after pruning is shown in Fig.4(b).

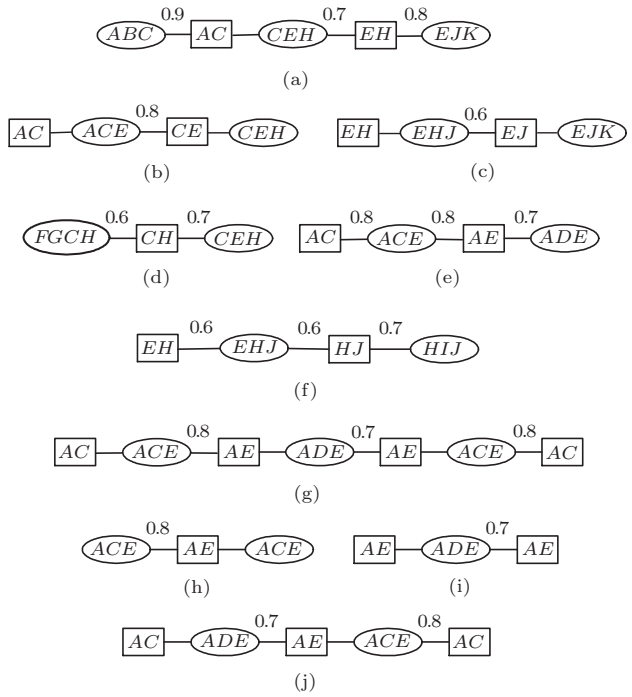


Fig.7. TP_{sp} and fragments from B to K in Fig.4(a). (a) TP_{sp} of index tree T. (b) TF_{a_1} made by TP_{sp}. (c) TF_{a_2} made by TP_{sp}. (d) TF_{b_1} made by TP_{sp}. (e) TF_{b_2} made by TP_{sp}. (f) TF_{b_3} made by TP_{sp}. (g) Equivalent type of (e). (h) TP_{sp} of (g). (i) TF_a of (g). (j) TF_b of (g).

By skillfully using some stacks to store the unvisited blocks and vertex cuts, and the upper bound of each tree fragment, the above pruning algorithm can be completed in $O(b + c)$ time complexity, where b is the number of blocks and c is the number of vertex cuts.

4.3 Optimal Index Construction

Now, we introduce the method to construct our probabilistic shortest path index. A naive construction method is that we can randomly select some vertices in g_c and remove them such that g_c is divided into some

unconnected components. We recursively perform the above step over each component until the graph cannot be further divided, for example, only a vertex or an edge is left. Obviously, taking this method, the constructed index is not unique. In fact, in the worst case, the number of vertex cuts between any two vertices is exponential[18], and the formation of our index can also be exponential. Moreover, different indexes have different pruning capability (the ability to prune vertices and edges in the filtering phase). Thus, we need to consider a method to build an optimal index.

Property 3. An optimal index should have the maximum pruning capability and at least construction time.

To Get the Maximum Pruning Capability. First, we consider the pruning capability. The maximum pruning capability means the number of pruned paths should be as large as possible. According to (2) and (3), given any two vertices s and t, it should have the maximum possibility that s and t are in different blocks or vertex cuts. In other words, s and t should have the least possibility to be mapped into the same block. Thus, the size of blocks and vertex cuts should be as small as possible.

To Get the Least Construction Time. Then, we consider the construction time. In Definition 8, we should calculate $SPr(e)_m$. The number of calculation times of SPr is exponential to the number of vertices in blocks and vertex cuts. From this point, the size of blocks and vertex cuts should also be as small as possible.

Optimal Index. Based on the above analysis, the vertex number in each block and vertex cut should be as small as possible. In addition, the blocks should be cliques, and the vertex cuts should be minimal vertex cuts. Recalling our correlated uncertain graph model introduced in Section 2, constructing the index based on cliques and vertex cuts also can utilize the joint probabilities in a maximal clique well when calculating $SPr(e)_m$ in each clique (blocks). As the smallest cliques are triangles, motivated by the junction tree construction method, if we can divide the original graph into triangles, we can find the smallest blocks. The process to divide the original graph into triangles is called a triangulated graph[19], which is defined as follows.

Definition 10 (Triangulated Graph). *Given a graph g, a triangulated graph is a supergraph of g in which no cycles with four or more vertices exist in which there is no chord. A graph can be triangulated by adding chords. The added edges are called fill-in edges. The clique-width of an undirected graph g is the size of its largest clique or vertex cut.*

For example, the graph in Fig.4(a) is not triangulated, since in the circle $\{CHJE\}$, which has four vertices, neither of its chords $e(E,H)$ or $e(C,J)$ belongs to $E(g_c)$. Its triangulated graph is shown in Fig.8. The edges such as $e(E,H)$ can be called an fill-in edge.

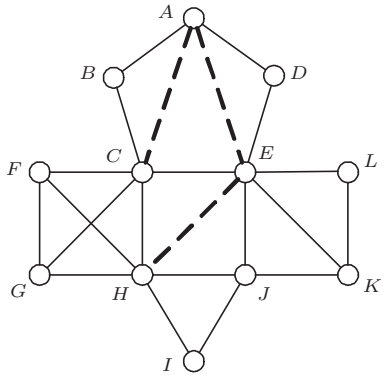


Fig.8. Triangulated graph of Fig.4(a).

Moreover, as each vertex cut can be treated as a super node, it equals adding a chord between each two vertices of the vertex cut. For example, the vertex cut $\mathcal{VC}(AC)$ equals adding a chord $e(A,C)$ into the graph g_c at the above triangulation step, which are the fill-in edges. As the clique-width should be as small as possible, the fill-in edges should be as few as possible. To summarize, our optimal index (also optimal triangulated graph) should satisfy the theorem as follows.

Theorem 5. *In the optimal index of a graph built based on its triangulated graph, both the clique-width and fill-in edges should be minimized. Moreover, finding the optimal triangulated graph is NP-complete. It has been proved that the optimal approximate ratio of an approximate algorithm should be $O(\log n)$, where n is the number of vertices in a graph*[19-20].

For example, the optimal index based on the triangulated graph in Fig.8 is shown in Fig.5. Now, we provide an approximate optimal construction algorithm, which can be concluded in Algorithm 3.

Our index construction algorithm is based on the idea of divide-and-conquer, which is shown in Algorithm 3. Initially, we treat the whole graph g_c as a single block, and put block $\mathcal{B} = V(g_c)$ into $\mathcal{B}(T)$, and edge $(\mathcal{B},\mathcal{B})$ into $E(T)$. In each recursion, we call a 2-way vertex cut algorithm, which is to find out a vertex cut $\mathcal{VC}$ that divides the graph into two connected subgraphs (line 6). There are many algorithms to find 2-way balanced vertex cuts, such as [21]. Then we delete the vertex members in $\mathcal{VC}$ and get two subgraphs sg_1

and sg_2 (lines 7∼8). The vertices in sg_1 and $\mathcal{VC}$ together consist of block $\mathcal{B}_1$, i.e., $\mathcal{B}_1 = (V(sg_1) \cup V(C))$. Similarly, $\mathcal{B}_2 = (V(sg_2) \cup V(C))$. Insert $\mathcal{VC}$ into $\mathcal{VC}(T)$. Now, $\mathcal{B}$ is replaced by $\mathcal{B}_1$ and $\mathcal{B}_2$, and $(\mathcal{B}, \mathcal{B})$ is replace by $(\mathcal{B}_1, \mathcal{VC})$ and $(\mathcal{B}_2, \mathcal{VC})$ (lines 9∼13). We recursively execute the above process on subgraphs sg_1 and sg_2 respectively until the vertex number of each subgraph satisfies $|V(g)| < (2\alpha + 1)k$. Here, α is the ratio between the weight of the 2-way vertex cut found by the algorithm and that of the optimal 2-way vertex cut, and k is the clique-width of g (line 2). Then we will get our index tree T. According to previous work on graph triangulation, we have the following theorem.

Algorithm 3. IndexConstruction(g)

```
Input: graph g_c
Output: index tree T((B(T), VC(T)), E(T))
  // Initially, block B := V(g_c) ∈ B(T), edge (B, B) ∈ E(T)
1 IndexConstruction(g) begin
2   if |V(g)| < (2α + 1)k then
3     Calculate maximum SPr from blocks to vertex
      according to (3) (UpB on each edge);
4     Return (T) ;
5   else
6     Call a 2-way balanced vertex cut function 2-VC(g_c,VC);
7     Delete VC from g_c ;
8     Get two subgraphs sg_1 and sg_2;
9     Block B_1 := (V(sg_1) ∪ V(VC)) ;
10    Block B_2 := (V(sg_2) ∪ V(VC)) ;
11    Insert VC into VC(T) ;
12    Replace edge (B, B) by (B_1, VC) and (B_2, VC) in B(T);
13    Replace block B by B_1 and B_2 in E(T);
14    IndexConstruction(sg_1) ;
15    IndexConstruction(sg_2) ;
```

Theorem 6. *The index built according to Algorithm 3 is $O(\log n)$-approximate optimal*[20].

According to the $(2 - \frac{2}{x})$-approximate algorithm for x-way vertex cut problem provided by [21], we have $\alpha = 1$ in our algorithm, and our approximation algorithm yields a triangulation having a clique-width bounded by $3k$.

The construction time of our index is $O(2^k n \times ploy(n))$, where $ploy(n)$ is the time to run a 2-way balanced vertex cut algorithm (usually $O(n^2)$), k is the maximum clique width (which is very small), n is the vertex number of a graph. The index size is $O(n)$.

5 Verification Phase

After pruning, we get a candidate graph for verification step. As calculating the exact SPr on the candidate graph is still #P-complete, we use a sampling algorithm to estimate the final SPr.

In this phase, we apply the sampling methods introduced in [15] to calculate the final SPr. The main idea is that we sample each edge e in the candidate uncertain subgraph with its probabilistic massive function. After all edges sampled, the candidate subgraph now is changed into a deterministic graph. Then, we run a shortest path algorithm over deterministic graph, such as Dijkstra, and work out a shortest path sp_i. Set a counter y_i for each candidate path, and add "1" to the counter of the sampled sp_i. After n_{samp} times sampling, the SPr for each sp_i equals

$$SPr(sp_i) \approx \widehat{SPr(sp_i)} = \frac{1}{n_{\text{samp}}} \sum_{j=1}^{n_{\text{samp}}} y_i. \tag{5}$$

The detailed algorithms and the evaluation methods are detailed in [15].

6 Experiment

In this section, we will report our experiment results. All the experiments are proceeded on a PC with CPU Inter® Core™ i3-2120, Frequency 3.30 GHz, Memory 4.00 GB, Hard-disk 500 GB. The operation system is Microsoft Windows 7 Ultimate Edition. The development software is Microsoft Visual Studio 2010, using language C++ and its standard template library (STL).

Two kinds of data are used in our experiment, real data and synthetic data. We choose two of the PPI data① (394 and 882) and two of the road network data② (OL and NA), which are listed in Table 2. All the synthetic data are generated by GraphStream③. We use Eclipse Standard 4.3 to develop the graph generator.

Table 2. Parameters of Real Data

Name of Dataset	Vertex Number	Edge Number	Average Degree
394.NGR (394)	788	1 231	3.12
882.DVU (882)	650	2 516	3.94
OLdenburg (OL) Road Network	6 105	7 029	2.25
North America (NA) Road Network	175 813	179 179	2.04

The PPI datasets are real uncertain datasets. Instead of giving the joint probabilities, they provide marginal probabilities. According to [22], the neighbor

①http://string-db.org/, Jan. 2015.

②http://www.cs.utah.edu/∼lifeifei/SpatialDataset.htm/, Jan. 2015.

③http://graphstream-project.org/, Jan. 2015.

edges in PPI data sharing the same vertex are dominated by the strongest interactions among the neighbor edges. We can use the maximum probability among the neighbor edges to calculate the joint probabilities, and then normalize them. For example, given the marginal probabilities $pr(e_1 = 1) = 0.2$, $pr(e_1 = 0) = 0.8$, $pr(e_2 = 1) = 0.7$, and $pr(e_2 = 0) = 0.3$, then the joint probabilities can be calculated as $pr(e_1 = 1, e_2 = 1) = \max(pr(e_1 = 1), pr(e_2 = 1)) = \max(0.2, 0.7) = 0.7$, $pr(e_1 = 1, e_2 = 0) = \max(0.2, 0.3) = 0.3$, $pr(e_1 = 0, e_2 = 0) = \max(0.8, 0.3) = 0.8$, and $pr(e_1 = 0, e_2 = 1) = \max(0.8, 0.7) = 0.8$. Then we need to normalize these joint probabilities. That is, $pr(e_1 = 1, e_2 = 1) = 0.7/(0.7 + 0.3 + 0.8 + 0.8) = 0.7/2.6 \approx 0.26$, $pr(e_1 = 1, e_2 = 0) = 0.3/2.6 \approx 0.12$, $pr(e_1 = 0, e_2 = 0) = 0.8/2.6 \approx 0.31$ and $pr(e_1 = 0, e_2 = 1) = 0.8/2.6 \approx 0.31$. Using this method, we can calculate all the joint probabilities in the same clique.

As the edges in road network datasets are deterministic, we need to do some preprocessing to change the deterministic graphs into uncertain graphs. Our generation method is similar to the one introduced in [23]. We use normal distribution $N(\mu, \sigma)$ to generate the joint probabilities of the edges in the same maximal clique. Here, μ is the expectation of the probabilities, and σ is the variance. After generating all the joint probabilities, we normalize these probabilities to make their sum equal to 1. In default, we make $\mu = 0.5$ and $\sigma = 0.1$.

6.1 Evaluation of Index

In this subsection, we first test the pruning capability and the pruning time of our index in Subsection 6.1.1 and Subsection 6.1.2 respectively. Then we test the index construction time and index space cost in Subsection 6.1.3. All the above experiments are tested on real data. Finally, we test the scalability of all the index in Subsection 6.1.4 using the synthetic data.

6.1.1 Pruning Capability

For different thresholds, we randomly select two vertices in each graph. Firstly, we calculate the shortest path and corresponding SPr using our filtering-and-verification method, and record the vertex number both in candidate set gotten from filtering phase and in final result set (result set 1) gotten from the verification phase. Then, we just use the sampling method in the verification phase and record the vertex number in result set (result set 2). We repeat the above steps 1 000 times and calculate the final query result and average vertex number in candidate set, result set 1, and result set 2. As both the final query result and the average vertex numbers in result set 1 and result set 2 are almost the same, we just use the result set 2 as the comparison to the candidate set. The result of pruning capability vs threshold in each dataset is shown in Fig.9.

From Fig.9, we can see that the vertex number in result set decreases w.r.t. the increase of threshold, and the vertex number in the candidate set also decreases together with the vertex number in the result set. It is reasonable because the higher the threshold is, the fewer paths will pass the threshold. Moreover, the vertex number in the candidate set is always just a little larger than that in final result set, which proves that our upper bounds provided by the index are very tight. For the same threshold, the vertex number in the candidate set (or result set) is the largest in dataset 882, and is the smallest in dataset NA. This is reasonable because the larger a dataset is, the longer path there will possibly be between the two randomly selected vertices when the uncertainty distribution is fixed in different datasets. According to our analysis in Subsection 4.1, a longer path will lead to a smaller SPr. Thus, there will be fewer paths above the threshold in a large graph than those in a smaller one. Thus, the average vertex number in the candidate set (or result set) will be smaller in a large graph than that in a smaller one.

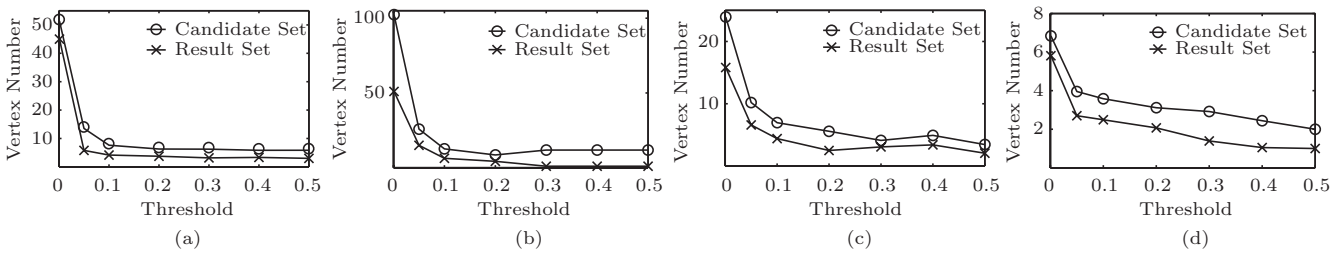


Fig.9. Pruning capability tested on real data. (a) Dataset 394. (b) Dataset 882. (c) Dataset OL. (d) Dataset NA.

6.1.2 Pruning Time

The experiment method of pruning time is also tested on real datasets, and the result is shown in Fig.10. We can see the pruning time decreases with the increase of threshold. The reason is also that fewer paths will pass the threshold when the threshold goes higher. The time cost is the most in dataset NA and the least in dataset 394. The longest pruning time is just less than 40 ms with graph size 175 813 vertices. This indicates that our pruning algorithm is very efficient.

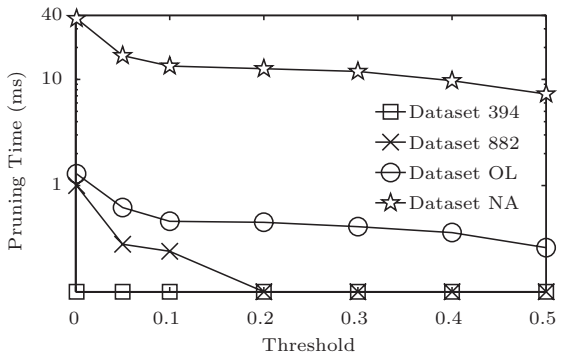


Fig.10. Pruning time.

6.1.3 Cost of Index

In this experiment, we test the index construction time and the index size. We run Algorithm 3 on the real datasets, and test time cost during the process of index construction. The result is shown in Fig.11. After getting the index, we count the node number (including blocks and vertex cuts) of the index tree. The result is shown in Table 3.

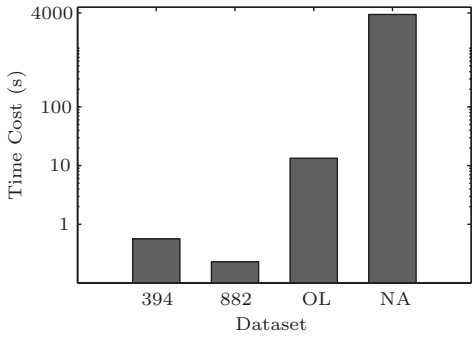


Fig.11. Index building.

From Fig.11, we can see that the index construction time is the least in dataset 882, which is less than 1 s. The time cost of NA is the most, which is about 3 800 s. The index construction time increases with the increase of graph size. From Table 3, we can see that the number of nodes in index is in the same order of magnitude with the vertex number of the original graph. This verifies that the index size is $O(n)$, where n is the number of vertices of the graph.

Table 3. Node Number in Index of Real Datasets

Dataset	Number of Nodes
394	659
882	582
OL	9 455
NA	137 230

6.1.4 Scalability

We test the scalability of pruning capability, pruning time w.r.t. graph size and graph density. When testing the scalability on graph size, we keep the average degree of each edge being 2.5, generate the different graphs with vertex number from 2 thousand to 1 million (noted as "2k" and "1M" for short), and choose threshold $\tau = 0.01$. Similarly, when testing the scalability on graph density, we choose graph size equal to 2k vertices, threshold $\tau = 0.01$, and change the average degree of each vertex from 2 to 20.

In addition, we test the pruning capability and the pruning time w.r.t. uncertainty distribution on each edge. We change the normal distribution of the joint probabilities of the four real datasets, and choose the result of dataset 882 and dataset OL listed here considering the limited space. The conclusion gotten from dataset 394 and dataset NA is the same according to our experiment result. Again, we also choose threshold $\tau = 0.01$. First, we fix $\sigma = 0.1$ and change μ from 0.1 to 0.9. Then, we fix $\mu = 0.5$ and change σ from 0.05 to 0.25.

All the above experiments are repeated 1 000 times and we calculate the average result.

Pruning Capability. The result of the scalability of pruning capability w.r.t. graph size is shown in Fig.12(a). The largest number of vertices left in the candidate set is only 14 while the graph size is 4k. Thus, the pruning capability of our index is very strong when the graph size changes.

Moreover, we can see that the number of vertices in the candidate set and the result set increases when the graph size changes from 2k to 4k, but decreases when the graph size changes from 4k to 1M. This is because the average SPr between two randomly chosen vertices

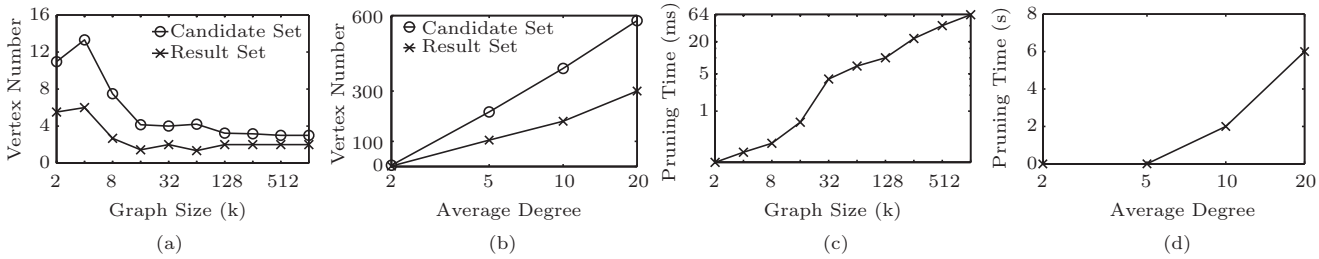


Fig.12. Pruning capability and pruning time vs graph size and graph density. (a) Capability vs graph size. (b) Capability vs graph density. (c) Time vs graph size. (d) Time vs graph density.

in graph with 2k vertices is above the fixed threshold 0.01. When increasing the vertex number from 2k to 4k, the average SPr between two randomly chosen vertices does not decrease so much, and at least it is still above 0.01. Thus, the vertex number in candidate set and result set increases as the graph scale increases. However, when the graph size is increased to 8k or more, the average SPr between any two randomly chosen vertices decreases to less than 0.01. Thus, the vertex number in the candidate set and the result set decreases with the increase of graph size because of the reason mentioned in Subsection 6.1.1.

The result of pruning capability scalability w.r.t. graph density is shown in Fig.12(b). We can see the number of vertices in the candidate set and the result set increases with the increase of graph density. The slope of the candidate set curve is larger than that of result set curve. This indicates that the pruning capability of our index decreases with the increase of graph density. It is reasonable because when the average degree of each vertex in graph increases, the clique width of index also increases, which leads to a weaker pruning capability. It verifies the analysis in Subsection 4.3.

The result of pruning capability scalability w.r.t. μ and σ is shown in Fig.13. The remaining vertices in different datasets are far fewer than those in the original graphs. Thus, the pruning capability is always strong. From Figs.13(a) and 13(b), we can see that the pruning capability increases with the increase of μ. This is because with the increase of μ, when normalizing the joint probabilities in the same maximal clique, each value in it decreases. Thus, different μ essentially changes the average SPr of the two randomly chosen vertices. Thus, the tendency of curves w.r.t. μ is similar to that w.r.t. threshold. From Figs.13(c) and 13(d), we can see curves fluctuate w.r.t. σ, but the number of vertices in the candidate set and the result set are nearly the same. Thus, σ hardly influences pruning capability.

Pruning Time. The result of the scalability of pruning time w.r.t. graph size is shown in Fig.12(c). We can see that the pruning time increases with the increase of graph size. The difference between the longest and the shortest pruning time is less than 64 ms, while the graph size changes from 2k to 1M. Thus, the graph size hardly influences the efficiency of our pruning algorithm.

The result of the scalability of pruning time w.r.t. graph density is shown in Fig.12(d). The pruning time increases with the increase of graph density. When the average degree changes from 2 to 5, the pruning time changes nearly 4 ms. Note that, when the average degree is larger than 5, there is a huge increase in the pruning time. This is because when the graph is too dense, the limited memory cannot store all the joint probability tables of the correlated uncertain graph. We have to leave the joint probability tables in the disk. Thus, when necessary, we have to do some I/Os to load corresponding joint probability tables.

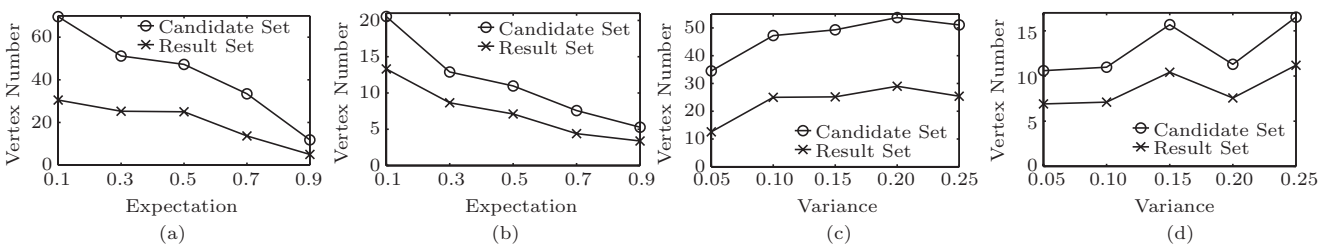


Fig.13. Pruning capability vs μ and σ. (a) μ in 882. (b) μ in OL. (c) σ in 882. (d) σ in OL.

The result of the scalability of pruning time w.r.t. μ and σ is shown in Fig.14. Again, as the influence caused by μ is the same with that caused by threshold, the pruning time just fluctuates slightly because the change of pruning time w.r.t. threshold is also very small. As well, the pruning time just fluctuates slightly when σ changes because σ nearly has no influence on pruning time. Thus, the pruning time is not sensitive to μ and σ.

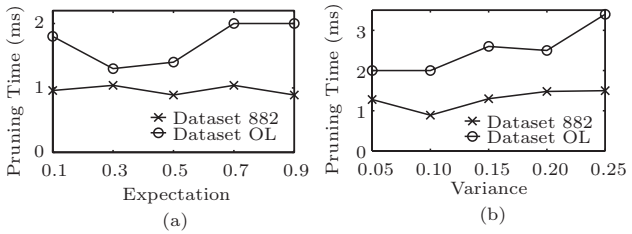


Fig.14. Pruning time vs μ and σ. (a) Pruning time vs μ. (b) Pruning time vs σ.

6.2 Evaluation of Verification Step

The verification experiment includes the verification time and the related error. We choose *H-T* estimator in our verification step, which is the best one in [15]. The calculation method of related error is the same with [15]. Again, we test them on the real datasets, and fix $\mu = 0.5$, $\sigma = 0.1$, threshold $\tau = 0.01$. The results are shown in Fig.15(b).

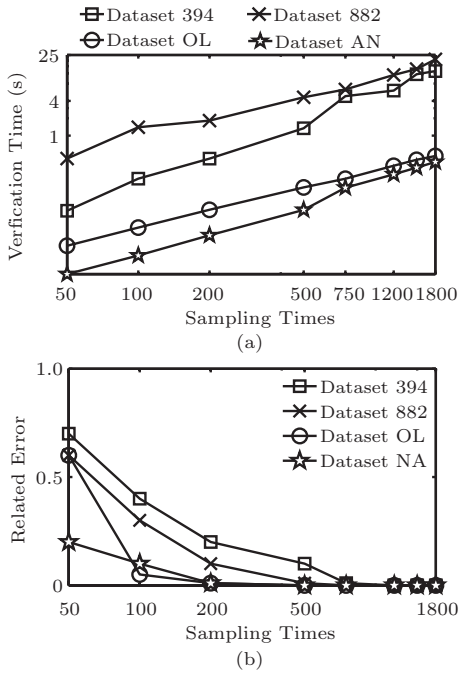


Fig.15. Verification evaluation. (a) Time cost on real datasets. (b) Error of the estimator.

We can see that the verification time increases with the increase of sampling times. The verification time cost is the largest on dataset 882 and the smallest on dataset NA. It is reasonable because the vertex number in candidate set of dataset 882 is the largest while that of dataset NA is the smallest of NA. For the accuracy, the more times of sampling, the smaller related error we will get. It is certainly true because the sampling result will be closer to the real result if we sample more times.

6.3 Total Query Cost

The online query time is the sum of the pruning time in the filtering step and the time cost of the shortest path sampling (SPS) method in the verification step. We still test it on real datasets, and the parameters are fixed as $\mu = 0.5$, $\sigma = 0.1$, $\tau = 0.01$, and sampling times equal 200. We compare the total time cost of our proposed filtering-and-verification method with that of the basic sampling method, which is just using the sampling method in [15] (also using its best estimator H-T) without using index. The result is shown in Fig.16(a). From the result, we can see that the time cost of our proposed filtering-and-verification (denoted as "Index+SPS") method is far less than that of basic

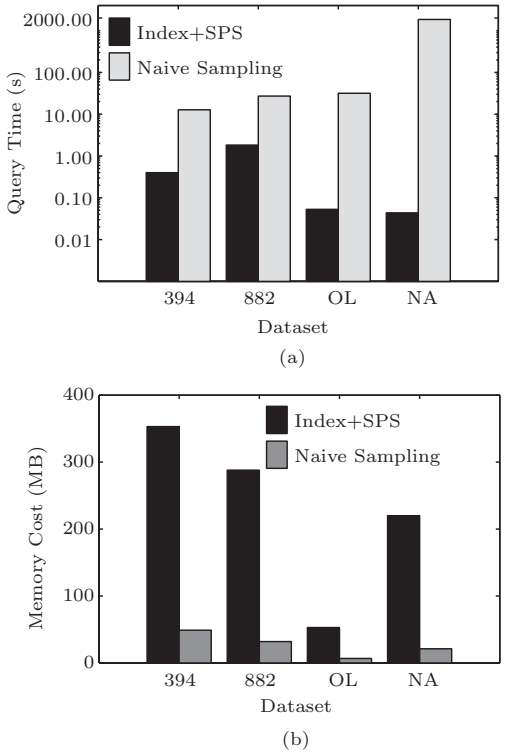


Fig.16. Total online query cost. (a) Query time. (b) Memory cost.

sampling method (denoted as "Naive Sampling"). This proves that our proposed method largely improves the query efficiency.

In this experiment, the total memory cost contains all the memory cost in our filtering-and-verification method, such as graph structure, the joint probabilities, index construction, index, pruning algorithm and the verification algorithm. Again, we compare it with the memory cost of basic sampling method. The result is shown in Fig.16(b). From the result, we can see that our proposed algorithm costs more memory than the basic algorithm. We analyze that the construction step takes large part of memory cost. Though we take more memory cost, the memory cost is still acceptable. However, we improve the query efficiency largely.

6.4 Model Comparison

In this subsection, we verify that the results provided by our correlated model are better than those provided by the independent model, which is mentioned in Section 2. We use the two real PPI data, 394 and 882, to conduct this experiment. As we mentioned in Section 2, the shortest path is used to evaluate the similarity of two proteins. If there exist paths whose SPr is higher than the given threshold between two query proteins (vertices), the two proteins can be treated to be similar. In PPI networks, whether two proteins are similar has been given out by biologists④. We randomly select 100 pairs of proteins given different thresholds on each dataset, and record the number of pairs whose similarity is consistent with that given biologists. These records are the precision of the calculated similarity. We compare the precision of our model and that of the independent model, which is shown in Table 4.

Table 4. Correlated Model vs Independent Model

Threshold	Data 394		Data 882	
	Correlated Model (%)	Independent Model (%)	Correlated Model (%)	Independent Model (%)
0.1	47	35	45	35
0.3	58	50	55	51
0.5	74	62	71	60
0.7	95	88	90	80
0.9	80	77	85	72

From Table 4, we can see that the precision of the results calculated from our correlated model is always higher than that calculated from the independent model. This verifies our statement in Section 2, which means our correlated model is more reasonable than the independent one to model uncertain graphs.

7 Related Work

In this section, we make a summary of previous work, including shortest path query over deterministic graphs (Subsection 7.1) and over uncertain graphs (Subsection 7.2).

7.1 Shortest Path over Deterministic Graphs

In deterministic graph area, the most classic shortest path algorithm is Dijkstra[24]. This algorithm cannot be used in large graph due to large time complexity. Another typical heuristic algorithm is A^*, which uses a heuristic function to accelerate. However, the heuristic function is hard to get and the query precision is unbounded.

Thus, researches consider to build indexes to speed up the query algorithms[25-26]. Cohen *et al.*[27] proposed a 2-hop labeling method. They recorded a table of 2-hop reachable vertices as an index. When a query came, they searched the table and found whether the two query vertices were the shortest path between them. Wei[28] proposed a tree-width decomposition index to accelerate the exact shortest path query. In [29-30], the authors proposed a landmark encoding method, and utilized triangle inequality to get the upper bound and the lower bound of shortest path to estimate the shortest path. This is an approximate method which has a high precision. During the two decennia, shortest path algorithms over deterministic graphs are applied most widely into road network[31-35]. The paper [36] is a good summary about shortest path on road network.

In recent years, some shortest path query studies are done to fit the large graph environment. Jin *et al.*[16] proposed a highway-centric labeling approach to answer the exact shortest path query, which is a combination of improved 2-hop labeling and highway dimension method. Cheng *et al.*[37] used vertex cover method to build the index and efficiently answer the distance query (shortest path query). Gao *et al.*[10] took the shortest path query as an example to introduce their relational FEM framework for the case that graph data cannot fully be loaded into memory.

Essentially, the models and structures of these algorithms above are different from those of our paper; thus it is impossible to extend these algorithms directly

④http://string-db.org and http://thebiogrid.org, Jan. 2015.

into our problem. Moreover, most of the acceleration techniques are aiming at special environment, such as the applications in road networks introduced above. In other application environment, these techniques are not available. The algorithms in our paper have no constraints on application environment.

7.2 Shortest Path over Uncertain Graphs

Many kinds of queries are done over uncertain environment[38-42]. Deshpande *et al.* made great effort on processing the correlation over uncertain data models[43-48]. Especially in [43] and [48], the authors used a junction tree as an index to help the process of join problems over correlated uncertain data. Our work has the following two main differences with [43]. Firstly, even though the optimal index is the same as an optimal junction tree, our index in the filtering phase is not necessarily a junction tree. Our index is based on blocks (not necessarily always be cliques) and vertex cuts, while junction tree is based on cliques. Secondly, the authors use junction tree to solve the join problems, which is a different problem with our shortest path query in graph. The motivation, the method and the purpose of the usage are different.

Among all different queries over uncertain graphs, shortest path query over uncertain graph is important, which was first proposed by Loui[49]. In recent years, many studies such as [1, 6] considered the model in which edges are independent of each other. The shortcoming of the independent model is detailed in the former sections of our paper. Moreover, Hua and Pei[23] built a simple correlated uncertain graph model. They used a joint probability table on each two edges sharing the same vertex to show their correlation. Their model could present simple correlationships between each two edges sharing the same vertex, but when there were more than two edges sharing the same vertices, their model would be confusing.

Until now, there are few studies on approximate shortest path algorithms over uncertain graphs. Jin *et al.*[50] proposed unequal probability sampling method to solve the uncertain reachability problem. They combined unequal probability sampling method with divide-and-concur strategy perfectly, and realized calculating the reachability probability without sampling the whole possible graph. With the help of divide-and-concur strategy, they can know whether two vertices are reachable during the process of sampling. Their solution avoids calling a reachability algorithm on certain graph and highly improves the efficiency. However, the algorithm above cannot be applied into our problem directly, because it is impossible to know which path is the shortest until the last edge on the graph is sampled, even though several reachable paths are sampled out. That is to say, the shortest path query is more complex than the reachability problem. Moreover, [15] is also a work on the shortest path query over correlated uncertain graphs. However, there are two main differences between our work and [15]. Firstly, our correlated uncertain graph model is based on Markov network while the one in [15] is based on Bayesian network. The three advantages of our model compared with the one in [15] have been discussed in Section 2. Secondly, [15] only focused on the sampling method to solve the shortest path query. Although this sampling method can also be applied in the verification phase of our work, only using the sampling method itself is not efficient enough. This is the reason why we design our index and the algorithms in the filtering phase. In Subsection 6.3, we compare our method and the one in [15] in the experiments, and the results show that our method can heavily improve the query efficiency.

8 Conclusions

In this paper, we first proposed a correlated uncertain graph model based on Markov network, which can overcome the shortcomings in the existing models. As calculating the shortest path probability is a #P-hard problem, we proposed a filtering-and-verification method to answer the shortest path query efficiently. In the filtering step, we built a probabilistic shortest path index, utilizing the upper bounds of shortest path probability and filtering large numbers of vertices that make no contribution to the shortest path query. However, as building the optimal index is an NP-hard problem, we provided an $O(\log n)$-approximate algorithm with polynomial time complexity. In the verification phase, we devised a sampling algorithm to verify the final SPr. Our experiments showed that our index has strong pruning capability and our algorithms have high efficiency to get a high precision query answer.

References

[1] Yuan Y, Chen L, Wang G. Efficiently answering probability threshold-based shortest path queries over uncertain graphs. In *Proc. the 15th DASFAA*, Apr. 2010, pp.155-170.

[2] Asthana S, King O D, Gibbons F D, Roth F P. Predicting protein complex membership using probabilistic network reliability. *Genome Research*, 2004, 14(6): 1170-1175.

[3] Nierman A, Jagadish H. ProTDB: Probabilistic data in XML. In *Proc. the 28th VLDB*, Aug. 2002, pp.646-657.

[4] Adar E, Ré C. Managing uncertainty in social networks. *IEEE Data Eng. Bull*, 2007, 30(2): 15-22.

[5] Lian X, Chen L. Efficient query answering in probabilistic RDF graphs. In *Proc. ACM SIGMOD International Conference on Management of Data*, May 2011, pp.157-168.

[6] Zou L, Peng P, Zhao D. Top-K possible shortest path query over a large uncertain graph. In *Proc. the 12th Int. Conf. Web Information System Engineering*, Oct. 2011, pp.72-86.

[7] Tao Y, Sheng C, Pei J. On k-skip shortest paths. In *Proc. ACM SIGMOD International Conference on Management of Data*, Jun. 2011, pp.421-432.

[8] Yao B, Tang M, Li F. Multi-approximate-keyword routing in GIS data. In *Proc. the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, Nov. 2011, pp.201-210.

[9] Fan W, Wang X, Wu Y. Performance guarantees for distributed reachability queries. *Proceedings of the VLDB Endowment*, 2012, 5(11): 1304-1315.

[10] Gao J, Jin R, Zhou J, Yu J X, Jiang X, Wang T. Relational approach for shortest path discovery over large graphs. *Proceedings of the VLDB Endowment*, 2011, 5(4): 358-369.

[11] Yan X, Yu P S, Han J. Substructure similarity search in graph databases. In *Proc. ACM SIGMOD International Conference on Management of Data*, Jun. 2005, pp.766-777.

[12] Atkinson K E. An Introduction to Numerical Analysis. John Wiley & Sons, 2008.

[13] Deshpande A, Getoor L, Sen P. Graphical models for uncertain data. *Managing and Mining Uncertain Data*, Aggarwal C C (ed.), Springer, 1980: 77-112.

[14] Koller D, Friedman N. Probabilistic Graphical Models: Principles and Techniques (1 edition). The MIT Press, 2009.

[15] Cheng Y, Yuan Y, Wang G, Qiao B, Wang Z. Efficient sampling methods for shortest path query over uncertain graphs. In *Proc. the 19th DASFAA*, Apr. 2014, pp.124-140.

[16] Jin R, Ruan N, Xiang Y, Lee V. A highway-centric labeling approach for answering distance queries on large sparse graphs. In *Proc. ACM SIGMOD International Conference on Management of Data*, May 2012, pp.445-456.

[17] Zou Z, Gao H, Li J. Discovering frequent subgraphs over uncertain graph databases under probabilistic semantics. In *Proc. the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Jul. 2010, pp.633-642.

[18] Chandran L S, Ram L S. On the number of minimum cuts in a graph. *SIAM Journal on Discrete Mathematics*, 2004, 18(1): 177-194.

[19] Shoikhet K, Geiger D. A practical algorithm for finding optimal triangulations. In *Proc. the 14th National Conference on Artificial Intelligence and the 9th Innovative Applications of Artificial Intelligence Conference* (*AAAI*), Jul. 1997, pp.185-190.

[20] Becker A, Geiger D. A sufficiently fast algorithm for finding close to optimal junction trees. In *Proc. the 12th International Conference on Uncertainty in Artificial Intelligence*, Aug. 1996, pp.81-89.

[21] Garg N, Vazirani V V, Yannakakis M. Multiway cuts in directed and node weighted graphs. In *Proc. the 21st Int. Colloquium on Automata, Languages and Programming*, Jul. 1994, pp.487-498.

[22] Chua H N, Sung W K, Wong L. Exploiting indirect neighbours and topological weight to predict protein function from protein-protein interactions. *Bioinformatics*, 2006, 22(13): 1623-1630.

[23] Hua M, Pei J. Probabilistic path queries in road networks: Traffic uncertainty aware path selection. In *Proc. the 13th International Conference on Extending Database Technology* (*EDBT*), Mar. 2010, pp.347-358.

[24] Bast H, Funke S, Matijevic D. TRANSIT—Ultrafast shortestpath queries with linear-time preprocessing. In *Proc. the 9th DIMACS Implementation Challenge: Shortest Paths*, Nov. 2006.

[25] Delling D, Sanders P, Schultes D, Wagner D. Engineering route planning algorithms. In *Algorithmics of Large and Complex Networks*, Lerner J, Wagner D, Iweig K A (eds.), Springer, 2009, pp.117-139.

[26] Klein P N, Mozes S, Weimann O. Shortest paths in directed planar graphs with negative lengths: A linear-space $O(n \log^2 n)$-time algorithm. *ACM Transactions on Algorithms* (*TALG*), 2010, 6(2): Article No. 30.

[27] Cohen E, Halperin E, Kaplan H, Zwick U. Reachability and distance queries via 2-hop labels. *SIAM Journal on Comp.*, 2003, 32(5): 1338-1355.

[28] Wei F. TEDI: Efficient shortest path query answering on graphs. In *Proc. ACM SIGMOD International Conference on Management of Data*, Jun. 2010, pp.99-110.

[29] Potamias M, Bonchi F, Castillo C, Gionis A. Fast shortest path distance estimation in large networks. In *Proc. the 18th ACM Conference on Information and Knowledge Management*, Nov. 2009, pp.867-876.

[30] Gubichev A, Bedathur S, Seufert S, Weikum G. Fast and accurate estimation of shortest paths in large graphs. In *Proc. the 19th ACM Conference on Information and Knowledge Management*, Nov. 2010, pp.499-508.

[31] Sankaranarayanan J, Samet H, Alborzi H. Path oracles for spatial networks. *Proceedings of the VLDB Endowment*, 2009, 2(1): 1210-1221.

[32] Samet H, Sankaranarayanan J, Alborzi H. Scalable network distance browsing in spatial databases. In *Proc. ACM SIGMOD International Conference on Management of Data*, Jun. 2008, pp.43-54.

[33] Sankaranarayanan J, Alborzi H, Samet H. Efficient query processing on spatial networks. In *Proc. the 13th International Workshop on Geographic Information Systems* (*GIS*), Nov. 2005, pp.200-209.

[34] Rice M, Tsotras V J. Graph indexing of road networks for shortest path queries with label restrictions. *Proceedings of the VLDB Endowment*, 2010, 4(2): 69-80.

[35] Jing N, Huang Y W, Rundensteiner E A. Hierarchical encoded path views for path query processing: An optimal model and its performance evaluation. *IEEE Transactions on Knowledge and Data Engineering*, 1998, 10(3): 409-432.

[36] Wu L, Xiao X, Deng D, Cong G, Zhu A D, Zhou S. Shortest path and distance queries on road networks: An experimental evaluation. *Proceedings of the VLDB Endowment*, 2012, 5(5): 406-417.

[37] Cheng J, Ke Y, Chu S, Cheng C. Efficient processing of distance queries in large graphs: A vertex cover approach. In *Proc. ACM SIGMOD International Conference on Management of Data*, May 2012, pp.457-468.

[38] Tong Y, Chen L, Cheng Y, Yu P S. Mining frequent itemsets over uncertain databases. *Proceedings of the VLDB Endowment*, 2012, 5(11): 1650-1661.

[39] Tong Y, Chen L, Ding B. Discovering threshold-based frequent closed itemsets over probabilistic data. In *Proc. the 28th International Conference on Data Engineering* (*ICDE*), Apr. 2012, pp.270-281.

[40] Yuan Y, Wang G, Chen L, Wang H. Efficient subgraph similarity search on large probabilistic graph databases. *Proceedings of the VLDB Endowment*, 2012, 5(9): 800-811.

[41] Zou Z, Li J, Gao H, Zhang S. Mining frequent subgraph patterns from uncertain graph data. *IEEE Transactions on Knowledge and Data Engineering*, 2010, 22(9): 1203-1218.

[42] Tong Y, Zhang X, Chen L. Tracking frequent items over distributed probabilistic data. *World Wide Web*, 2015.

[43] Kanagal B, Deshpande A. Indexing correlated probabilistic databases. In *Proc. ACM SIGMOD International Conference on Management of Data*, Jun. 2009, pp.455-468.

[44] Deshpande A, Guestrin C, Hong W H, Madden S. Exploiting correlated attributes in acquisitional query processing. In *Proc. the 21st International Conference on Data Engineering* (*ICDE*), Apr. 2005, pp.143-154.

[45] Deshpande A, Garofalakis M, Rastogi R. Independence is good: Dependency-based histogram synopses for high-dimensional data. *ACM SIGMOD Record*, 2001, 30(2): 199-210.

[46] Sen P, Deshpande A, Getoor L. PrDB: Managing and exploiting rich correlations in probabilistic databases. *The VLDB Journal*, 2009, 18(5): 1065-1090.

[47] Kanagal B, Deshpande A. Lineage processing over correlated probabilistic databases. In *Proc. ACM SIGMOD Int. Conf. Management of Data*, Jun. 2010, pp.675-686.

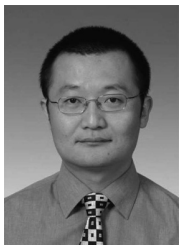
[48] Tzoumas K, Deshpande A, Jensen C S. Lightweight graphical models for selectivity estimation without independence assumptions. *Proceedings of the VLDB Endowment*, 2011, 4(11): 852-863.

[49] Loui R P. Optimal paths in graphs with stochastic or multidimensional weights. *Communications of the ACM*, 1983, 26(9): 670-676.

[50] Jin R, Liu L, Ding B, Wang H. Distance-constraint reachability computation in uncertain graphs. *Proceedings of the VLDB Endowment*, 2011, 4(9): 551-562.

Yu-Rong Cheng received her B.S. degree in computer science from the Department of Computer Science, Northeastern University, Shenyang, in 2012. Currently, she is a Ph.D. candidate of Northeastern University. Her main research interest includes queries over graph data and large graph analysis.

Ye Yuan received his B.S., M.S. and Ph.D. degrees in computer science from Northeastern University, Shenyang, in 2004, 2007 and 2011, respectively. He is now a professor in the Department of Information Science and Engineering in Northeastern University. His research interests include graph databases, probabilistic databases, data privacy-preserving, and cloud computing.



Lei Chen received his B.S. degree in computer science and engineering from Tianjin University, Tianjin, in 1994, M.A. degree from Asian Institute of Technology, Bangkok, Thailand, in 1997, and Ph.D. degree in computer science from the University of Waterloo, Canada, in 2005. He is currently an associate professor in the Department of Computer Science and Engineering, Hong Kong University of Science and Technology. So far, he published over 200 conference and journal papers. He got the Best Paper Awards in DASFAA 2009 and 2010. He is PC Track chairs for SIGMOD 2014, VLDB 2014, ICDE 2012, CIKM 2012, SIGMM 2011. He has served as PC members for SIGMOD, VLDB, ICDE, SIGMM, and WWW. Currently, Prof. Chen is an associate editor-in-chief for IEEE Transactions on Knowledge and Data Engineering and serves on the editorial board of Distributed and Parallel Databases. He is a member of the VLDB endowment committee and the chairman of ACM SIGMOD China Chapter. His research interests include crowdsourcing over social media, social media analysis, probabilistic and uncertain databases, and privacy-preserved data publishing.

Guo-Ren Wang received his B.S., M.S. and Ph.D. degrees in computer science from Northeastern University, Shenyang, in 1988, 1991 and 1996, respectively. Currently, he is a professor in the College of Information Science and Engineering, Northeastern University, Shenyang. His research interests are XML data management, query processing and optimization, bioinformatics, high-dimensional indexing, parallel database systems, and P2P data management.