Mining High Utility Itemsets in Uncertain Databases

Yuqing Lan, Yang Wang, Shengwei Yi, Dan Yu, and Simin Yu

School of Computer Science and Engineering, Beihang University, China
{lanyuqing, yangwang, yishengwei, yudan, siminyu}@buaa.edu.cn

Abstract. Recently, with the growing popularity of Internet of Things (IoT) and pervasive computing, a large amount of uncertain data, i.e. RFID data, sensor data, real-time monitoring data, etc., has been collected. As one of the most fundamental issues of uncertain data mining, the problem of mining uncertain frequent itemsets has attracted much attention in the database and data mining communities. Although some efficient approaches of mining uncertain frequent itemsets have been proposed, most of them only consider each item in one transaction as a random variable following the binary distribution and ignore the unit values of items in the real scenarios. In this paper, we focus on the problem of mining probabilistic high utility itemsets in uncertain databases (MPHU), in which each item has a unit value. In order to solve the MPHU problem, we propose a novel mining framework, called UUIM, which not only includes an efficient mining algorithm but also contains an effective pruning technique. Extensive experiments on both real and synthetic datasets verify the effectiveness and efficiency of proposed solutions.

1 Introduction

Recently, with the growing popularity of Internet of Things (IoT) and pervasive computing, a large amount of uncertain data, i.e. RFID data, sensor data, real-time monitoring data, etc., has been collected. As one of the most fundamental issues of uncertain data mining, the problem of mining uncertain frequent itemsets has attracted much attention in the database and data mining communities. Although some efficient approaches of mining uncertain frequent itemsets have been proposed, most of them only consider each item in one transaction as a random variable following the binary distribution and ignore the unit values of items in the real scenarios. In recommender systems (e.g. music, video) analysis, mining frequent itemsets from an uncertain database refers to discovery of the itemsets which may frequently appear together in the records (transactions). However, the unit values (profits) of items are not considered in the framework of uncertain frequent itemsets mining. Hence, it cannot satisfy the requirement of the user who is interested in discovering the itemsets with high enjoyment values. For example, consider the table of song data shown in Table 1. A and B in the table have different scores which depend on the evaluation of most users. The table of record is shown in the Table 2. Jack had heard A (Big Big World). The probable value that Jack enjoys Big Big World is 0.8. When songs are recommended, not only the frequency of songs in different users record should be considered, but the popularity of songs also ought to be taken into account. In view of this, utility mining will emerges as an important topic in data mining for discovering the itemsets with high utility like values (profits) in uncertain databases.

Table 1. A Table of Song

ID	Song Name	Score
A	Big Big World	3
B	Someone Like You	4
C	Dont You Remember	1
D	My Love	1
E	Time to say goodbye	8
F	Right Here Waiting	2
G	Can You	1

Table 2. A Table of Record

User	Record and matching analysis
Jack	(A, 0.8) (B, 0.8) (C, 0.7) (D, 0.7)
Rose	(A, 0.9) (C, 0.8) (G, 0.8)
Tom	(A, 0.6) (B, 0.7) (C, 0.6) (D, 0.5) (E, 0.8) (F, 0.5)
Peter	(B, 0.7) (C, 0.8) (D, 0.5)
Linda	(B, 0.8) (C, 0.7) (F, 0.4)

Mining high utility itemsets from the databases refers to finding the itemsets with high utilities. The basic meaning of utility is the interestedness / importance / profitability of items to the users. Before finding high utility itemsets over uncertain databases, the definition of the high utility itemset is the most essential issue. In deterministic data, the utility of items in a transaction database consists of two aspects: (1) the importance of distinct items, which is called external utility, and (2) the importance of the items in the transaction, which is called internal utility. The utility of an itemset is defined as the external utility multiplied by the internal utility. An itemset is called a high utility itemset if its utility is no less than a user-specified threshold. However, different from the deterministic case, the utility of items in an uncertain transaction database only contain the importance of distinct items, which is called external utility in deterministic database. The definition of a high utility itemset over uncertain data has two different semantic explanations: expected support-based high utility itemset and probabilistic high utility itemset. However, the two definitions are different on using the random variable to define high utility itemsets. In the definition of the expected support-based high utility itemset, the expectation of the utility of an itemset is defined as the measurement, called as the expected utility of this itemset. In this definition, an itemset is high utility if and only if the expected utility of such itemset is no less than a specified minimum expected utility threshold, min eutil. In the definition of probabilistic utility itemset, the probability that the utility of an itemset is no less than the threshold is defined as the measurement, called as the high utility probability of an itemset, and an itemset is high utility if and only if the high utility probability of such itemset is larger than a given probabilistic threshold.

Mining high utility itemsets from uncertain databases is an important task which is essential to a wide range of applications such as most of recommender systems analysis, Internet of Things (IoT) and even biomedical applications. The definition of probabilistic high utility itemset includes the complete probability distribution of the utility of an itemset. Although the expectation is known as an important statistic, it cannot show the complete probability distribution. Hence, we mainly discuss mining probabilistic high utility itemsets in this paper.

To sum up, we make the following contributions:

- To the best of our knowledge, this is the first work to formulate the problem of mining probabilistic high utility itemsets in uncertain databases (MPHU).
- Due to the challenges from utility constraints, we propose a novel mining framework, called UUH-mine, which not only includes an efficient mining algorithm but also contains an effective pruning technique.
- We verify the effectiveness and efficiency of the proposed methods through extensive experiments on real and synthetic datasets.

The rest of the paper is organized as follows. Preliminaries and our problem formulation are introduced in Section **??**. In Section 3, we present a novel mining framework, called UUH-mine. Based on this framework, an efficient mining algorithm and an effective pruning technique is devised in Section 4. In Section 5, experimental studies on both real and synthetic datasets are reported. In Section 6, we review the existing works. Finally, we conclude this paper in Section 7.

2 Preliminaries and Problem Definitons

In this section, we use directed graphs due to the wide applications in real life. For example, in a road network, it is common to see vehicles stuck in one direction, while the opposite direction has a very light traffic. What's more, in other domains, such as semantic Web (RDF), social networks, and bioinformations, graphs are often directed. Specifically, we take *DAG(Directed Acyclic Graph)* typically in our model, since the cyclic graphs can equivalently be changed into *DAG*s by merging the strong connected components into a single vertices [29]. We give the definition of *DAG* as follows.

Definition 1 (Directed Acyclic Graph (DAG)). *A Directed Acyclic Graph is denoted as $DAG(V, E, W)$, where V is the vertex set, E is the edge set, and each edge is given a weight $w \in W$. For any path P from v_1 to v_2 in a DAG, there should be no such paths P's from v_2 to v_1. If there is an edge $e(v_1, v_2) \in E$, it is called **outgoing** v_1 and **injecting** v_2, and v_1 is called a **parent vertex** of v_2. If there are two edges $e_1(v_1, v_2)$ and $e_2(v_2, v_3)$, e_1 is called a **parent edge** of e_2. The total number of edges outgoing a vertex is called the **out-degree** of this vertex, and the total number of edges injecting a vertex is called its **in-degree**. The vertices with zero in-degree are called **root vertices**.*

In this section, we first describe our new uncertain graph model in Section 2.1. Then, in Section 2.2, we formally define our problem.

2.1 Uncertain Graph Model

Definition 2 (Uncertain Graph Model). *An Uncertain Graph is denoted as $\mathcal{G}(g_c, Pr)$, where $g_c(V, E, W)$ is a* DAG *and Pr is a* probability mass function (pmf) *showing the correlation among edges and their parent edges. Similar to* Bayesian Network*, the* pmf *is represented by a conditional probability CPr_e that edge e exists (or not) on the condition that its parent edges exist (or not). That is,*

$$CPr_{state(e)} = Pr(state(e)|state(e_1), \dots, state(e_i), \dots, state(e_{in})) \quad (1)$$

where $e_1 \dots e_{in}$ are the parent edges of e, and $state(e)$ is the function presenting whether e exists. If e exists, $state(e) = 1$. Otherwise, $state(e) = 0$.

A conditional probability table CPT *is used on e to list all its $CPrs$.*

Example 1 (Uncertain Graph Model). Fig. 2 shows an uncertain graph, and Gi. 2(a) is its g_c. As there are 9 edges in g_c, there should be 9 *CPT*s in total, and Fig. 2(b)-(f) show 5 of them. The edge e_{sA} and e_{sB} are outgoing a root vertex s, and their conditional probabilities are shown in Fig. 2(b) and (c). The state of edge e_{BE} depends on the state of the edge e_{sB}. Similarly, the state of edge e_{Ct} depends on the states of both e_{AC} and e_{BC}. For instance in Fig 2(e), the value 0.2 means the probability that e_{BE} exists on the condition that e_{sS} exists.

Fig. 1. Uncertain Graph Model

This correlated uncertain graph model can be used in road network to show the fluency correlationship among neighbor roads. For example, in Fig. 2(a), road e_{Ct} is more possible to be fluent (i.e., e_{Ct} exists) if both the roads e_{AC} and e_{BC} are fluent (i.e., e_{AC} and e_{BC} both exist). Besides, if this model is used in website society, for example, in Fig. 2(a), if a user visit a website A from its similar website s, he is more probable to continue visit its another similar website C or D than not to visit them. These correlationships can only be presented by our correlated uncertain graph model other than the existing independent one.

Given the above uncertain graph model, how to calculate the probability of a *possible world graph*? This would apply the following definition of *Conditional Independent*.

Definition 3 (Conditional Independent). *X, Y, Z are three sets of random variables. Y is conditionally independent of Z given X (denoted as $Y \perp Z|X$) in distribution Pr if $\forall\ x \in X$, $y \in Y$ and $z \in Z$, that is,*

$$Pr(Y = y, Z = z|X = x) = Pr(Y = y|X = x) \times Pr(Z = z|X = x)$$

From Definition 3, the probability of a *possible world graph g* equals to

$$Pr(g) = \prod_{1 \leq i \leq m} CPr(state(e_i)) \tag{2}$$

where m is the number of edges in a graph.

For example, the probability of g_c shown in Fig. 2(a) equals to

$$\begin{aligned} Pr(g_1) &= CPr(state(e_{sB}) = 1) \times CPr(state(e_{BE}) = 1) \times \cdots \\ &= Pr(state(e_{sB}) = 1) \times Pr(state(e_{BE}) = 1)|state(e_{sB}) = 1) \times \cdots \\ &= 0.7 \times 0.2 \times \cdots \end{aligned} \tag{3}$$

2.2 Problem Statement

Definition 4 (Shortest Path Probability (SPr)). *For any path P of g_c, the probability of the event that P is the shortest path in the uncertain graph $\mathcal{G}(g_c(V, E, W), Pr)$ is called its Shortest Path Probability, which is calculated as*

$$SPr(P) = \sum_{P \text{ is the shortest path in } g_i \ (1 \leq i \leq n)} Pr(g_i) \tag{4}$$

Definition 5 (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, the threshold-based shortest path query returns a path set $SP = \{P_1, P_2, \ldots, P_n\}$ in which each path P_i has a $SPr(P_i)$ larger than the threshold τ.*

Apparently, if we want to quickly answer the *threshold-based shortest path query*, we have to efficiently calculate the SPr. However, the calculation of SPr under the correlated uncertain graph model is #P-hard. This is because calculating the reachability probability under the independent uncertain graph model is #P-complete [11][23]. When considering the correlation among edges, the calculation of the reachability probability under our model will be harder, whose complexity is at least #P-complete. Apparently, the calculation of the shortest path probability under our model is even harder than calculating the reachability probability. So our problem is at least #P-complete, which is #P-hard.

3 Baseline Sampling Algorithm

As discussed in the last section, calculating SPr is a #P-hard problem, so there is no exact solution in polynomial time unless P=NP. Thus, in the rest of this paper, we use sampling methods to get an approximate result. In this section, we give a baseline sampling method based on *Monte Carlo* theory.

During the sampling process, we sample N *possible world graphs*, G_1, G_2, $\ldots$, G_N, according to the *pmf* of each edge. Then, on each sampled possible world graph, we run a shortest path algorithm over deterministic graphs to find out its shortest path. Finally, we set a flag y_{P_i} for each path, so that

$$y_{P_i} = \begin{cases} 1 \text{ if } P_i \text{ is the shortest path in the sampled } \textit{possible world graph} \\ 0 \text{ otherwise} \end{cases}$$

Thus, the estimator $\widehat{SPr_B}$ equals to,

$$SPr_B \approx \widehat{SPr_B} = \frac{\sum_{i=1}^{N} y_{P_i}}{N} \tag{5}$$

For any sampling method, The *Mean Square Error (MSE)* incorporates both the bias and the precision of an estimator into a measure of overall accuracy. It is calculated as,

$$MSE(\widehat{\theta}) = E[(\widehat{\theta} - \theta)^2] = Var(\widehat{\theta}) + Bias(\widehat{\theta}, \theta)$$

The bias of an estimator is given by,

$$Bias(\widehat{\theta}) = E(\widehat{\theta}) - \theta$$

An estimator of θ is unbiased if its bias is 0 for all values of θ, that is, $E(\widehat{\theta}) = \theta$ As the estimator of *Monte Carlo* method is unbiased [6], thus,

$$MSE(\widehat{SPr_B}) = Var(\widehat{SPr_B}) = \frac{1}{N} SPr_B(1 - SPr_B) \approx \frac{1}{N}\widehat{SPr_B}(1 - \widehat{SPr_B}) \tag{6}$$

The pseudo code is shown in Algorithm 1.

If the sampling number N is far smaller than the number of *possible world graphs*, the *Baseline Sampling Algorithm* is more efficient than the naive exact method discussed in Section 1. But as a consequence, the calculation result is less accurate. According to our experiment result in Section 5, if we want to get a more accurate result, we need to spend many hours to calculate the result using the *Baseline Sampling* algorithm. Whereas, if we want to quickly access the result in seconds, the error may be as large as more than 20%.

The total time cost t_{total} of the sampling algorithm equals to $t_{total} = t_{once} \times N$ (t_{once} is the time cost in one sampling). **So, there are two bottlenecks we need to break. The first one is how to get a more accurate result using a smaller sampling size (reduce N). The other one is whether we can reduce the time cost in one sampling (reduce t_{once}).** Taking the above consideration, we propose an improved sampling method based on *Unequal Probability Sampling*.

Algorithm 1: Baseline Algorithm

```
   Input: Start Point $s$, Termination Point $t$
   Output: $\widehat{SPr_B}$, $Var(\widehat{SPr_B})$
1  for $i$ from 1 to $N$ do
2      Sample a possible graph;
3      $Dijkstra$(PossibleGraph PG);
4      $C_{P_i}$++;
       // to count the number of 1 in $y_{P_i}$
5  return $(\widehat{SPr_B}, Var(\widehat{SPr_B}))$ according to formula (5) and (6);
```

4 Improved Sampling Algorithm

In this section, we first introduce some knowledge on *Unequal Probability Sampling* in Section 4.1. Then, in Section 4.2, we illustrate our improved sampling algorithm, which is called *DijSampling* algorithm.

4.1 Unequal Probability Sampling

Unequal probability sampling is when some units in the population have probabilities of being selected from others. Suppose a sample of size N is selected randomly from a population S but that on each draw, unit i is sampled according to any probability q_i, where $\sum_{i=1}^{S} q_i = 1$ [14]. Let y_i be the response variable measured on each unit selection, i.e.,

$$y_i = \begin{cases} 1 \text{ if } i \text{ is selected in one sampling} \\ 0 \text{ otherwise} \end{cases}$$

Note that if a unit is selected more than once, it is used as many times as it is selected.

There are two classical unbiased estimators, *Hansen-Hurwitz* (*H-H*) and *Horvitz-Thompson* (*H-T*) in unequal probability sampling method. The *H-H* estimator is for random sampling with replacement, while the *H-T* estimator is a more general estimator, which can be used for any probability sampling plan, including both sampling with and without replacement [14]. For *H-H* estimator, an unbiased estimator of the population total $\xi = \sum_{i=1}^{S} y_i$ is given by

$$\hat{\xi_q} = \frac{1}{N}\sum_{i=1}^{N}\frac{y_i}{q_i}$$

The variance of *H-H* estimator is

$$\widehat{Var}(\hat{\xi_q}) = \frac{1}{N(N-1)}\sum_{i=1}^{N} q_i(\frac{y_i}{q_i} - \hat{\xi_q})^2$$

For *H-T* estimator, the population total $\xi = \sum_{i=1}^{S} y_i$ is estimated by

$$\hat{\xi_\pi} = \frac{1}{d}\sum_{i=1}^{N}\frac{y_i}{\pi_i}$$

where π_i is the probability that the i^{th} unit of the population is included in the sample (*inclusion probability*), and d is the distinct unit number in the sample, which is called *effective sample size*. In addition, the variance of *H-T* estimator is,

$$\widehat{Var}(\hat{\xi_\pi}) = \sum_{i=1}^{d}(\frac{1-\pi_i}{\pi_i^2})y_i^2 + \sum_{i=1}^{d}\sum_{j=1,j\neq i}^{d}(\frac{\pi_{ij}-\pi_i\pi_j}{\pi_i\pi_j})\frac{y_i y_j}{\pi_{ij}}$$

where π_{ij} is the *joint inclusion probability* of units i and j (the probability that i and j are sampled at the same time).

As both of the above estimators are unbiased, i.e., $Bias(\widehat{\theta}, \theta) = 0$, the variance can be used to evaluate the accuracy of the estimators.

4.2 *DijSampling Algorithm*

Reviewing the two bottlenecks in Section 3, the underlying cause of the first one is that the *Baseline Sampling* algorithm only samples one instance of the sample space in one sampling. **We consider a method to sample a lot of *possible world graphs* together in one sampling in order to reduce N.** Then, for the second bottleneck, when sampling once, the *Baseline Sampling* algorithm samples the whole *possible world graph*, and the followed *Dijkstra* algorithm is run on the whole *possible world graph*. Thus, **we should think of a method that only samples a part of a *possible world graph* each time to reduce t_{once}.** These are the main purposes of our *DijSampling* algorithm. We use an example to explain these advantages in detail.

Example 2 (Advantages of Improved Sampling Algorithm). Fig. 3 shows 4 *possible world graphs*. These *possible world graphs* have the same shortest path, P_{sBEt}. (Actually, there are 16 such *possible world graphs*, and we only show 4 of them.) We apply shortest path algorithm to sample edges e_{sB}, e_{BE}, and e_{Et}, and the 16 *possible world graphs* are totally the *possible world graphs* that contain P_{SBEt} as the shortest path. Thus, our *DijSampling* algorithm only needs one sampling and can sample all these 16 *possible world graphs* together. But in *Baseline Sampling* algorithm, it takes 16 times to sample them. In other words, sampling once in *DijSampling* algorithm has the same effect with sampling 16 times in *Baseline Sampling*. If the *Baseline Sampling* needs 1600 times of sampling to get an accurate result, the *DijSampling* algorithm only needs 100 times of sampling. Thus, by together sampling the *possible world graphs* containing the same shortest path, the *DijSampling* algorithm can effectively reduce the sample size N. Moreover, since the 4 edges, e_{AC}, e_{AD}, e_{Ct} and e_{Dt}, are not sampled in one sampling of *DijSampling* algorithm, the *DijSampling* algorithm has a smaller t_{each} than the *Baseline Sampling* algorithm.

Fig. 2. Possible World Graphs in which P_{sBEt} is the shortest path

As indicated in the above example, the main idea of our *DijSampling* algorithm is to sample the edges at the same time with the process of *Dijkstra* algorithm. The pseudocode is shown in Algorithm 2. The *DijSampling* is proceeded during the process of *Dijkstra* algorithm. When the *Dijkstra* algorithm needs the messages of an edge such as its existence or weight, we sample it according to its *pmf* (Line 3). Note that if this edge is outgoing the source query point s, we need to determine whether its in-degree is zero. If it is so ($s = s_z$), we just sample it according to its *pmf* (Line 5). Otherwise (Line 7), we sample this edge with the probability estimated by the following formula

$$Pr(state(e_s) = 1) = \frac{\sum_{1 \leq i \ldots \leq m_{in}; k=0,1} CPr(state(e_s) = 1 | state(e_i) = k, \cdots)}{\sum_{1 \leq i \ldots \leq m_{in}; k=0,1} CPr(state(e_s) = k | state(e_i) = k, \cdots)} \quad (7)$$

This formula means we sample the edge e_s using the average probability when $state(s) = 1$ in its *CPT* as its existence probability, while using the average probability when $state(s) = 0$ as its nonexistence probability. The reason why we treat the e_s specially will be explained latter in this subsection.

The following is an example illustrating the process of *DijSampling* algorithm.

Algorithm 2: DijSampling Algorithm

```
   Input: Start Point $s$, Termination Point $t$, sampling times $N$
   Output: $\widehat{SPr_{H-H}}, Var(\widehat{SPr_{H-H}}), \widehat{SPr_{H-T}}, Var(\widehat{SPr_{H-T}})$
1  for $i$ from $1$ to $N$ do
2      running Dijkstra begin
3          if Dijkstra needs the messages of an edge then
4              if $s$==$s_z$ then
5                  Sample the edges outgoing $s$ according to their CPTs ;
6              else
7                  Sample the edges outgoing $s$ according to formula (7) ;
8              Sample the other edges according to their CPTs ;
9          else
10             continue the Dijkstra algorithm ;
11         return (result shortest path $P_i$) ;
12     $y_{P_i}$++ ;
13 calculate $\widehat{SPr_{H-H}}, Var(\widehat{SPr_{H-H}}), \widehat{SPr_{H-T}}, Var(\widehat{SPr_{H-T}})$;
```

Example 3. Fig. 4 shows the process of *DijSampling* algorithm. In this figure, we use dashed lines to denote the unsampled edges, solid lines to denote the existing edges after sampling, and no lines (empty) to denote the nonexisting edges after sampling. Specifically, Fig. 4(a) is the g_c of the uncertain graph in Fig. 2 before sampling, and Fig. 4(b)-(d) are the graphs after sampling the designated edges.

Assume the start query point is s and terminal query point is t. We first sample e_{sA} and e_{sB} according to Fig. 2(b) and Fig. 2(c) respectively. For example, we sample the e_{sB} with probability 0.7 existing and 0.3 nonexisting. After that, we assume that e_{sA} does not exists and e_{sB} exists (Fig. 4(b)). Then, we sample edge e_{BE} and e_{BC}. For example, for edge e_{BE}, as we have sampled edge e_{sB} exists, then $state(e_{sB}) = 1$. So we sample e_{BE} with probability 0.2 existing, and probability 0.8 nonexisting (Fig. 4(c)). Repeat the above process, after we sample to Fig. 4(d), we can know P_{sBEt} is the shortest path. Thus, there is no need to sample the other edges (dashed edges in Fig. 4(d)).

Fig. 3. *DijSampling* Process

In addition, if the query is B to t instead, the first edges should be sampled are e_{BE} and BC. The sampling existence probability of e_{BE} is $(0.2 + 0.857)/(0.2 + 0.857 + 0.8 + 0.143) = 0.5285$, and similarly the nonexistence probability is 0.4715. Then, the remained sampling steps are the same with those mentioned above (from s to t).

After describing the algorithm, let us see how to estimate the SPrs. According to Section 4.1, the *H-H* estimator of SPr can be calculated as

$$SPr_{H-H} \approx \widehat{SPr_{H-H}} = \frac{1}{N}\sum_{i=1}^{N}\frac{Pr_{P_i}y_{P_i}}{q_i} \tag{8}$$

where N is the sampling times, and Pr_{P_i} is the probability of the sampled *possible world graph* that P_i is the shortest path in it, which can be calculated as

$$Pr_{P_i} = \prod_{1\leq i\leq m_{samp}} CPr(state(e_i)) \tag{9}$$

where m_{samp} is the number of sampled edges in one sampling process.

The variance of this estimator equals to

$$Var(\widehat{SPr_{H-H}}) = \frac{1}{N(N-1)}\sum_{i=1}^{N} q_i(\frac{Pr_{P_i}}{q_i} - \widehat{SPr_{H-H}})^2 \tag{10}$$

When $q_i = Pr_{P_i}$, i.e., we sample each edge according to its *pmf* [11], it has been proved that, the variance in formula (10) is minimum. At this moment, the $\widehat{SPr_{H-H}}$ in formula (8) equals to

$$min(\widehat{SPr_{H-H}}) = \frac{\sum_{i=1}^{N} y_{P_i}}{N} \tag{11}$$

and its corresponding variance is

$$min(Var(\widehat{SPr_{H-H}})) = \frac{1}{N}\widehat{SPr_{H-H}}(1 - \widehat{SPr_{H-H}}) \tag{12}$$

The ***H-H*** estimator and its variance equals to those of the baseline algorithm.

Similarly, we give the formula to compute ***H-T*** estimator. Suppose, in the N times sampling, there are v different sampling results. Thus the probability that each one of the v results may be sampled is $\pi_i = 1 - (1 - q_i)^N$. So the estimator $\widehat{SPr_{H-T}}$ is,

$$SPr_{H-T} \approx \widehat{SPr_{H-T}} = \frac{1}{N}\sum_{i=1}^{N}\frac{Pr_{P_r}y_{P_i}}{\pi_i} \tag{13}$$

This estimator is unbiased as well [11][14]. $\pi_{ij} = 1 - (1 - q_i)^n - (1 - q_j)^N - (1 - q_i - q_j)^N$ is the probability that π_i and π_j are in the result set at the same time, and the variance of this estimator is,

$$Var(\widehat{SPr_{H-T}}) = \sum_{i\in v}(\frac{1-\pi_i}{\pi_i})(Pr_{P_i})^2 + \sum_{i,j\in v, i\neq j}(\frac{\pi_{ij} - \pi_i\pi_j}{\pi_i\pi_j})Pr_{P_j}Pr_{P_j} \tag{14}$$

Again, when $q_i = Pr_{P_i}$, $Var(\widehat{SPr_{H-T}})$ gets its minimum value. However, at this time, $\widehat{SPr_{H-T}} \neq \widehat{SPr_B}$. Moreover, $Var(\widehat{SPr_{H-T}}) \leq Var(\widehat{SPr_B})$ [11].

The Special e_s in Line 7 of Algorithm 2 The reason why we treat the e_s specially is that the edges outgoing a vertex always depend its parent edges under the uncertain graph model introduced in Section 2.1. Just like Example 4, we cannot know the sampling probability of e_{BE} if the state of e_{sB} is unknown. Actually, we can just randomly choose a root vertex and sample from it. After it reach the source query point, we continue our algorithm. However, this method is apparently uneconomical since the long journey before the source query point has no contribution to the result set except for a start probability. Thus, we use formula (7) to quickly estimate this probability instead.

Our algorithm can achieve $q_i = Pr_{P_i}$ except for this step in Line 7 of Algorithm 2. In this situation, if we still use the estimators with the minimum variance, they are no longer unbiased. But fortunately, we can prove that this bias is close to zero when the number of sampled edges are large, which is Theorem 1 in follows.

Theorem 1. *The bias of estimators equals nearly to zero when the number of sampled edges are large.*

Fig. 4. *Dependency Schematic Diagram for Derivation of Bias*

Proof. We take the situation in which s has only one injection edge as an example to derivate the bias. The derivation when s has more than one injection edges are similar. A schematic diagram is shown in Fig. 5. e_1 is the first edge we want to sample. Let $Pr(e_i|e_j) = Pr(state(e_i) = 1|state_{e_j} = 1)$, $Pr(e_i|\overline{e_j}) = Pr(state(e_i) = 1|state_{e_j} = 0)$, $Pr(\overline{e_i}|e_j) = Pr(state(e_i) = 0|state_{e_j} = 1)$, and $Pr(\overline{e_i}|\overline{e_j}) = Pr(state(e_i) = 0|state_{e_j} = 0)$, then the real sampling probability of e_1 equals to

$$\begin{aligned} Pr(e_1) &= Pr(e_1|e_0) \times Pr(e_0) + Pr(e_1|\overline{e_0}) \times Pr(\overline{e_0}) \\ &= Pr(e_1|e_0) \times Pr(e_0) + Pr(e_1|\overline{e_0}) \times (1 - Pr(\overline{e_0})) \end{aligned} \tag{15}$$

But our estimated probability of e_1 equals to,

$$\widehat{Pr(e_1)} = Pr(e_1|e_0) \times \widehat{Pr(e_0)} + Pr(e_1|\overline{e_0}) \times (1 - \widehat{Pr(\overline{e_0})})$$

Thus, the bias after sampling the first edge $Bias^{(1)}$ equals to,

$$Bias^{(1)} = \widehat{Pr(e_1)} - Pr(e_1) = (Pr(e_1|e_0) - Pr(e_1|\overline{e_0})) \times (Pr(e_0) - \widehat{Pr(e_0)})$$

Similarly, we can get the bias after sampling the second edge,

$$\begin{aligned} Bias^{(2)} &= \widehat{Pr(e_2)} - Pr(e_2) \\ &= (Pr(e_2|e_1) - Pr(e_2|\overline{e_1})) \times (Pr(e_1|e_0) - Pr(e_1|\overline{e_0})) \times (Pr(e_0) - \widehat{Pr(e_0)}) \end{aligned}$$

Recursively, we can get the bias after sampling m_{sample} edges,

$$\begin{aligned} Bias^{(m_{sample})} &= Pr(\widehat{e_{m_sample}}) - Pr(e_{m_sample}) \\ &= \prod_{i=1}^{m_{sample}} (Pr(e_i|e_j)^{(i)} - Pr(e_i|\overline{e_j})^{(i)}) \times (Pr(e_0) - \widehat{Pr(e_0)}) \end{aligned}$$

where $Pr(e_i|e_j)^{(i)}$ and $Pr(e_i|\overline{e_j})^{(i)}$ are conditional probabilities of i^{th} sampled edge.

Apparently, $Bias^{(m_{sample})} \to 0$ when m_{sample} is large enough. □

5 Performance Evaluations

In this section, we will report and analyze our experiment results. All the experiments are proceeded on a PC with CPU Inter(R) Core(TM)i7-2600, frequency 3.40GHz, memory 8.00GB, hard disk 500GB. The Operation System is Microsoft Windows 7 Enterprise Edition. The development software is Microsoft Visual Studio 2010, using language C++ and its standard template library (STL).

The Dataset The real data used in this paper can be classified into two categories. One is sparse dataset such as Road Network data[1], and the other one is Social Network data[2]. The numbers of vertices and edges are listed in Table 4.

In real datasets, the edges are certain, so we need to change the certain graphs into uncertain graphs. In Road Network Datasets, we use the method which is introduced in [10]. Use Normal Distribution $N(\mu, \sigma)$ to generate the weights on each edge. Here, μ is the edge weight in original datasets, and σ is the variance of the generated weights. σ is different according to different edges, which is normally distributed as $N(\mu_\sigma, \sigma_\sigma)$. Here, $\mu_\sigma = xR$, and the value range of x is $[1\%, 5\%]$. In default condition, $\mu_\sigma = 1\%R$, and R is the value range of all weights in original datasets. In the same way, we generate the weights and probabilities on edges in Social Network datasets.

Table 3. Real Dataset Parameters

Name of Dataset	Vertex Number	Edge Number
OLdenburg (OL) Road Network	6,105	7,035
San Francisco Road Network (SF)	174,956	223,001
wiki-Vote	7,115	103,689

For all the datasets, we choose 100 pairs of vertices as starting points and termination points. After the 100 tests, we calculate the average time cost, memory cost, *Mean Square Error (MSE)* and relative error as the experiment results. We set the threshold defaulted to be 0.5 and compare the proformance of baseline sampling algorithm (denoted as *BS*) and our improved *DijSampling* algorithm (denoted as *DS*).

Running Time Fig. 6 shows the running time vs sample size for the two sampling algorithms on different real datasets. From the results, we can observe that with the increase of sample size, the time cost for the two algorithms all increases. The time cost of *BS* is always largest than that of *DS*. In addition, we observe that the larger the graph is, the more time will be cost.

(a)(b)(c)
OLSwiki-
FVote

Fig. 5. Running time vs Sample Size

The results above are reasonable. The running time depends on the number of sampled edges. The more edges sampled, the more time an algorithm will cost. As the *BS* Algorithm samples more edges, the whole possible graph, its running time is longer.

(a)(b)(c)
OLSwiki-
FVote

Fig. 6. Memory Cost vs Sample Size

[1] http://www.cs.utah.edu/ lifeifei/SpatialDataset.htm

[2] http://snap.stanford.edu/data

Memory Cost Fig. 7 shows the memory cost vs sample size on different datasets. It can be seen that with the increase of sample times, the memory cost of two algorithms increases. The memory cost of the two algorithms is almost the same.

The phenomena above is reasonable that as sample size increases, the program needs to explore more space to find the shortest path and calculate estimator with corresponding variance. Thus, the memory cost increases with the increase of sample times. Moreover, both the algorithms need to save the structures of graph data and some queues for *Dijkstra* algorithm, which are the same. The only difference between the two methods are the flags showing whether each edge is sampled, which takes little memory cost. Thus, the memory cost of the two algorithms is nearly the same.

Accuracy We test *Mean Square Error (MSE)* and relative error of the estimators to show the accuracy of different algorithms. As the bias caused by formula (7) is always 0 in the experiment result, we do not show it in our result figures. Since the variance of *BS* estimator is the same as the *H-H* estimator of *DS*, we show them using the same line in result figure. The method of calculating *relative error* is the same as that in [11].

(a)(b)(c)
OS wiki-
F Vote

Fig. 7. Mean Square Error vs Sample Size

From Fig. 8, it can be seen that no matter how the size of datasets change, the variance of estimators decreases as sample size increases. Moreover, the variance of $\widehat{SPr}_{H-T}$ always keeps smaller than that of $\widehat{SPr}_{H-H}$. This result means the $\widehat{SPr}_{H-T}$ estimator is better than $\widehat{SPr}_{H-T}$, which verifies the discussion in Section 4.1.

(a)(b)(c)
OS wiki-
F Vote

Fig. 8. Relative Error vs Sample Size

From Fig. 9, the relative error of *BS* is always the largest, and *H-T* of *DS* is always the smallest. This means the stability and accuracy of $\widehat{SPr}_{H-T}$ is strongest. This result verifies our analysis in Section 4.2. That is, we need fewer sampling times by applying *DijSampling* Algorithm to get the same accuracy as baseline algorithm.

As we cannot get the distribution of unequal probability sampling, the error cannot be bounded. However, from Figure 9, in the 4 datasets used in our experiments, the error of both $\widehat{SPr}_{H-T}$ and $\widehat{SPr}_{H-H}$ is very low. Moreover, with the increase of sample times, the error fluctuates very gently. Thus, the estimators can approximate the exact answer well.

6 Related Works

The most popular shortest path algorithms over deterministic graphs are *Dijkstra* [3] and A^* [7]. These algorithms has a large time complexity. Thus, there are large number of works focusing on building indexes to speed up the algorithms. To accelerate the exact shortest path query, Cohen et al. [5] proposed a 2-hop labeling method, and F.Wei [24] proposed a tree-width decomposition index. Moreover, authors in [9] proposed a landmark encoding method to provide an approximate answer for shortest path query. In addition, there are also some shortest path algorithms over deterministic graphs designed typically for road networks such as [20][19][18][13], and [25] is a good summary. In recent years, some shortest path algorithms are also designed to for large graph environment, such as [12][4][8].

Essentially, the models of these algorithms above are all different from ours, so it is impossible to extend these algorithms directly.

Different queries are addressed in uncertain environment for a long time such as [21] [22][28][27]. Among them, shortest path query over uncertain graph is important. This query is first proposed by Loui [16]. Many works such as [26][30] considered an independent model, which is argued in detail in Section 1. Moreover, Ming Hua et.al [10] built a simple correlated model. It modeled simple correlation in uncertain graphs between each two edges sharing the same vertex. But when there are more than two edges sharing the same vertex, it would be confusing.

In addition, ruoming Jin et.al [11] applied *Unequal Probability Sampling* method, which solved the uncertain reachability problem efficiently and effectively. However, their algorithms cannot be applied into our problem directly for three reasons. First, their model was independent uncertain graph model, and their algorithms cannot handle the correlated uncertain graph model. Secondly, their query was reachability, which is different from our shortest path query. As discussed in Section 2.2, our problem is harder than theirs. Thirdly, they applied their sampling algorithm in a *divided and conquer* framework. If this framework is extended to our problem directly, we cannot find out the shortest path unless the last edge is sampled. Then, their algorithms would lose the advantages.

7 Conclusion

In this paper, we first propose a new uncertain graph model, which considers the hidden correlation among edges sharing the same vertex. As calculating the *shortest path probability* is a #P-hard problem, we use sampling methods to approximately compute it. We propose a baseline algorithm and an improved algorithm. Our improved algorithm is more efficient than the baseline algorithm with more accurate answers when sampling the same times. Moreover, we preform comprehensive experiments to verify the efficiency and accuracy of our algorithms.

Acknowledgments. Yurong Cheng, Ye Yuan and Guoren Wang were supported by the NSFC (Grant No.61025007, 61328202, 61202087, 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). Baiyou qiao was supported by the NSFC (Grant No.61073063).

References

1. Adar, E., Ré, C.: Managing uncertainty in social networks. IEEE Data Eng. Bull 30(2), 15–22 (2007)

2. Asthana, S., King, O.D., Gibbons, F.D., Roth, F.P.: Predicting protein complex membership using probabilistic network reliability. Genome Research 14(6), 1170–1175 (2004)
3. Bast, H., Funke, S., Matijevic, D.: Transitłultrafast shortest-path queries with linear-time preprocessing. 9th DIMACS Implementation Challenge [1] (2006)
4. Cheng, J., Ke, Y., Chu, S., Cheng, C.: Efficient processing of distance queries in large graphs: a vertex cover approach. In: SIGMOD. pp. 457–468. ACM (2012)
5. Cohen, E., Halperin, E., Kaplan, H., Zwick, U.: Reachability and distance queries via 2-hop labels. SIAM Journal on Comp. 32(5), 1338–1355 (2003)
6. Fishman, G.S.: A monte carlo sampling plan based on product form estimation. In: Proceedings of the 23rd conference on Winter simulation. pp. 1012–1017. IEEE Computer Society (1991)
7. Fu, L., Sun, D., Rilett, L.R.: Heuristic shortest path algorithms for transportation applications: state of the art. Computers & Operations Research 33(11), 3324–3343 (2006)
8. Gao, J., Jin, R., Zhou, J., Yu, J.X., Jiang, X., Wang, T.: Relational approach for shortest path discovery over large graphs. PVLDB 5(4), 358–369 (2011)
9. Gubichev, A., Bedathur, S., Seufert, S., Weikum, G.: Fast and accurate estimation of shortest paths in large graphs. In: CIKM. pp. 499–508. ACM (2010)
10. Hua, M., Pei, J.: Probabilistic path queries in road networks: traffic uncertainty aware path selection. In: EDBT. pp. 347–358. ACM (2010)
11. Jin, R., Liu, L., Ding, B., Wang, H.: Distance-constraint reachability computation in uncertain graphs. PVLDB 4(9), 551–562 (2011)
12. Jin, R., Ruan, N., Xiang, Y., Lee, V.: A highway-centric labeling approach for answering distance queries on large sparse graphs. In: SIGMOD. pp. 445–456. ACM (2012)
13. Jing, N., Huang, Y.W., Rundensteiner, E.A.: Hierarchical encoded path views for path query processing: An optimal model and its performance evaluation. TKDE 10(3), 409–432 (1998)
14. K.Thompson, S.: Sampling the Third Edition. WILEY SERIES IN PROBABILITY AND STATISTICS, WILEY (2012)
15. Lian, X., Chen, L.: Efficient query answering in probabilistic rdf graphs. In: SIGMOD. pp. 157–168. ACM (2011)
16. Loui, R.P.: Optimal paths in graphs with stochastic or multidimensional weights. CACM 26(9), 670–676 (1983)
17. Nierman, A., Jagadish, H.: Protdb: Probabilistic data in xml. In: Proceedings of the 28th international conference on Very Large Data Bases. pp. 646–657. VLDB Endowment (2002)
18. Rice, M., Tsotras, V.J.: Graph indexing of road networks for shortest path queries with label restrictions. PVLDB 4(2), 69–80 (2010)
19. Samet, H., Sankaranarayanan, J., Alborzi, H.: Scalable network distance browsing in spatial databases. In: SIGMOD. pp. 43–54. ACM (2008)
20. Sankaranarayanan, J., Samet, H., Alborzi, H.: Path oracles for spatial networks. PVLDB 2(1), 1210–1221 (2009)
21. Tong, Y., Chen, L., Cheng, Y., Yu, P.S.: Mining frequent itemsets over uncertain databases. PVLDB 5(11), 1650–1661 (2012)
22. Tong, Y., Chen, L., Ding, B.: Discovering threshold-based frequent closed itemsets over probabilistic data. In: ICDE. pp. 270–281. IEEE (2012)
23. Valiant, L.G.: The complexity of enumeration and reliability problems. SIAM Journal on Comp. 8(3), 410–421 (1979)
24. Wei, F.: Tedi: efficient shortest path query answering on graphs. In: Proceedings of SIGMOD. pp. 99–110. ACM (2010)
25. Wu, L., Xiao, X., Deng, D., Cong, G., Zhu, A.D., Zhou, S.: Shortest path and distance queries on road networks: an experimental evaluation. PVLDB 5(5), 406–417 (2012)
26. Yuan, Y., Chen, L., Wang, G.: Efficiently answering probability threshold-based shortest path queries over uncertain graphs. In: DASFAA. pp. 155–170. Springer (2010)
27. Yuan, Y., Wang, G., Chen, L., Wang, H.: Efficient keyword search on uncertain graph data. Knowledge and Data Engineering, IEEE Transactions on 25(12), 2767–2779 (2013)
28. Yuan, Y., Wang, G., Wang, H., Chen, L.: Efficient subgraph search over large uncertain graphs. In: International Conference on Very Large Data Bases (2011)
29. Zhang, Z., Yu, J.X., Qin, L., Chang, L., Lin, X.: I/o efficient: computing sccs in massive graphs. In: Proceedings of the 2013 international conference on Management of data. pp. 181–192. ACM (2013)
30. Zou, L., Peng, P., Zhao, D.: Top-k possible shortest path query over a large uncertain graph. In: WISE, pp. 72–86. Springer (2011)