Accelerating Probabilistic Frequent Itemset Mining: A Model-Based Approach

Liang Wang Reynold Cheng Sau Dan Lee David W. Cheung
Department of Computer Science
The University of Hong Kong
Pokfulam Road, Hong Kong
{lwang, ckcheng, sdlee, dcheung}@cs.hku.hk

ABSTRACT

Data uncertainty is inherent in emerging applications such as location-based services, sensor monitoring systems, and data integration. To handle a large amount of imprecise information, uncertain databases have been recently developed. In this paper, we study how to efficiently discover frequent itemsets from large uncertain databases, interpreted under the *Possible World Semantics*. This is technically challenging, since an uncertain database induces an exponential number of possible worlds. To tackle this problem, we propose a novel method to capture the itemset mining process as a Poisson binomial distribution. This *model-based approach* extracts frequent itemsets with a high degree of accuracy, and supports large databases. We apply our techniques to improve the performance of the algorithms for: (1) finding itemsets whose frequentness probabilities are larger than some threshold; and (2) mining itemsets with the k highest frequentness probabilities. Our approaches support both tuple and attribute uncertainty models, which are commonly used to represent uncertain databases. Extensive evaluation on real and synthetic datasets shows that our methods are highly accurate. Moreover, they are orders of magnitudes faster than previous approaches.

Categories and Subject Descriptors

H.2.8 [**Database management**]: Data applications—*data mining*; G.3 [**Probability and statistics**]: Distribution functions

General Terms

Algorithms

1. INTRODUCTION

In many applications, the underlying databases are uncertain. The locations of users obtained through RFID and GPS systems, for instance, are not precise due to measurement errors [22, 16]. In habitat monitoring systems, data collected from sensors like temperature and humidity are noisy [1]. Customer purchase behaviors, as captured in supermarket basket databases, contain statistical information for predicting what a customer will buy in the future [3, 25]. Integration and record linkage tools associate confidence values to the output tuples according to the quality of matching [11]. In structured information extractors, confidence values are appended to rules for extracting patterns from unstructured data [26]. Recently, *uncertain databases* have been proposed to offer a better support for handling imprecise data in these applications [8, 11, 20, 15, 13].

As an example of uncertain data, consider an online marketplace application (Table 1). Here, the purchase behavior details of customers Jack and Mary are recorded. The value associated with each item represents the chance that a customer may buy that item in the near future. These probability values may be obtained by analyzing the users' browsing histories. For instance, if Jack visited the marketplace ten times in the previous week, out of which *video* products were clicked five times, the marketplace may conclude that Jack has a 50% chance of buying *videos*. This *attribute-uncertainty* model, which is well-studied in the literature [22, 8, 15, 25], associates confidence values with data attributes. It is also used to model location and sensor uncertainty in GPS and RFID systems.

Customer	Purchase Items
Jack	(*video*:1/2), (*food*:1)
Mary	(*clothing*:1), (*video*:1/3); (*book*:2/3)

Table 1: An Uncertain Database

To interpret uncertain databases, the *Possible World Semantics* (or PWS in short) is often used [11]. Conceptually, a database is viewed as a set of deterministic instances (called *possible worlds*), each of which contains a set of tuples. A possible world w for Table 1 consists of two tuples, $\{food\}$ and $\{clothing\}$, for Jack and Mary respectively. Since $\{food\}$ occurs with a probability of $(1-\frac{1}{2})\times 1 = \frac{1}{2}$, and $\{clothing\}$ has a probability of $1\times(1-\frac{1}{3})\times(1-\frac{2}{3}) = \frac{2}{9}$, the probability that w exists is $\frac{1}{2}\times\frac{2}{9}$, or $\frac{1}{9}$. Any query evaluation algorithm for an uncertain database has to be correct under PWS. That is, the results produced by the algorithm should be the same as if the query is evaluated on every possible world [11].

Although PWS is intuitive and useful, querying or mining under this notion is costly. This is because an uncertain database has an exponential number of possible worlds. For example, the database in Table 1 has $2^3 = 8$ possible worlds. Performing data mining under PWS can thus be technically

CIKM'10, October 25–29, 2010, Toronto, Ontario, Canada.

challenging. In fact, the mining of uncertain data has recently attracted research attention [3]. For example, in [17], efficient clustering algorithms were developed for uncertain objects; in [14] and [27], naïve Bayes and decision tree classifiers designed for uncertain data were studied. Here, we develop scalable algorithms for finding frequent itemsets (i.e., sets of attribute values that appear together frequently in tuples) for uncertain databases. Our algorithms can be applied to two important uncertainty models: *attribute uncertainty* (e.g., Table 1); and *tuple uncertainty*, where every tuple is associated with a probability to indicate whether it exists [11, 20, 13, 10, 21].

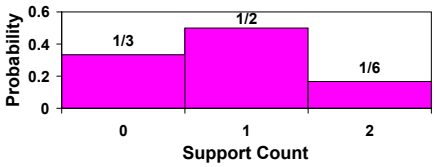


Figure 1: s-pmf of PFI {*video*} from Table 1.

The frequent itemsets discovered from uncertain data are naturally probabilistic, in order to reflect the confidence placed on the mining results. Figure 1 shows a *Probabilistic Frequent Itemset* (or *PFI*) extracted from the table in Table 1. A *PFI* is a set of attribute values that occur frequently with sufficiently-high probabilities. In Figure 1, the *support probability mass function* (or *s-pmf* in short) for the *PFI* {*video*} is shown. This is the pmf for the number of tuples (or *support count*) that contain an itemset. Under PWS, a database induces a set of possible worlds, each giving a (different) support count for a given itemset. Hence, the support of a frequent itemset is described by a pmf. In Figure 1, if we consider all possible worlds where itemset {*video*} occurs twice, the corresponding probability is $\frac{1}{6}$.

A simple way of finding PFIs is to mine frequent patterns from every possible world, and then record the probabilities of the occurrences of these patterns. This is impractical, due to the exponential number of possible worlds. To remedy this, some algorithms have been recently developed to successfully retrieve PFIs without instantiating all possible worlds [28, 25]. These algorithms are able to find a PFI in $O(n^2)$ time (where n is the number of tuples contained in the database). However, our experimental results reveal that they require a long time to complete (e.g., with a 300k real dataset, the dynamic programming algorithm in [25] needs 30.1 hours to complete). We observe that the s-pmf of a PFI can be modeled by a Poisson binomial distribution, for both attribute- and tuple-uncertain data. We make use of this intuition to propose a method for approximating a PFI's pmf with a Poisson distribution, which can be efficiently and accurately estimated. This *model-based algorithm* runs in $O(n)$ time, and is thus more scalable to large datasets. In fact, our algorithm only needs 9.2 seconds to find all PFIs, which is four orders of magnitudes faster.

We demonstrate how the model-based algorithm can work under two semantics of PFI, proposed in [25]: (1) *threshold-based*, where PFIs with probabilities larger than some user-defined threshold are returned; and (2) *rank-based*, where PFIs with the k highest probabilities are returned. As we will show, these algorithms can be adapted to the attribute and tuple uncertainty models. For mining threshold-based PFIs, we demonstrate how to reduce the amount of effort of scanning the database. For mining rank-based PFIs, we optimize the previous algorithm to improve the performance. We derive the time and space complexities of our approaches. As our experiments show, model-based algorithms can significantly improve the performance of PFI discovery, with a high degree of accuracy. To summarize, our contributions are:

- Derive the probability model for the s-pmf of a PFI;
- Study an approximate representation of s-pmf, by using a common probability model;
- Develop a more efficient method to verify a threshold-based PFI;
- Use our techniques to enhance the performance of threshold and rank-based PFI discovery algorithms, for both attribute and tuple uncertainty models; and
- Perform extensive experiments on real and synthetic datasets.

This paper is organized as follows. In Section 2, we review the related works. Section 3 discusses the problem definition. Section 4 describes efficient and accurate methods for computing s-pmf. Then, in Sections 5 and 6, we present our algorithms for discovering threshold- and rank-based PFIs respectively. Section 7 presents our experimental results. We conclude in Section 8.

2. RELATED WORK

Mining frequent itemsets is often regarded as an important first step of deriving association rules [4]. Many efficient algorithms have been proposed to retrieve frequent itemsets, such as Apriori [4] and FP-growth [12]. While these algorithms work well for databases with precise and exact values, it is interesting to extend them to support uncertain data. Our algorithms are based on the Apriori. We believe that they can be used by other algorithms (e.g., FP-growth) to support uncertain data.

For uncertain databases, [9, 2] developed efficient frequent pattern mining algorithms based on the expected support counts of the patterns. However, [28, 25] found that the use of expected support may render important patterns missing. Hence, they proposed to compute the probability that a pattern is frequent, and introduced the notion of PFI. In [25], dynamic-programming-based solutions were developed to retrieve PFIs from attribute-uncertain databases, for both threshold- and rank-based PFIs. However, their algorithms have to compute exact probabilities, and find a PFI in $O(n^2)$ time. By using probability models, our algorithms avoid the use of dynamic programming, and can find a PFI much faster (in $O(n)$ time). In [28], approximate algorithms for deriving threshold-based PFIs from tuple-uncertain data streams were developed. While [28] only considered the extraction of singletons (i.e., sets of single items), our solution discovers patterns with more than one item. More recently, [24] developed an exact threshold-based PFI mining algorithm. However, it does not support rank-based PFI discovery. Here we also study the retrieval of rank-based PFIs from tuple uncertain data. To our best knowledge, this has not been examined before. Table 2 summarizes the major work done in PFI mining.

Uncertainty	Threshold-PFI	Rank-PFI
Attribute	Exact [25] Approx. [√]	Exact [25] Approx. [√]
Tuple	Exact [24] Approx. (singleton) [28] Approx. (multiple items) [√]	Approx. [√]

Table 2: Our Contributions (marked [√]).

Other works on the retrieval of frequent patterns from imprecise data include: [7], which studied approximate frequent patterns on noisy data; [18], which examined association rules on fuzzy sets; and [19], which proposed the notion of a "vague association rule". However, none of these solutions are developed on the uncertainty models studied here.

3. PROBLEM DEFINITION

We now discuss the uncertainty models used in this paper, in Section 3.1. Then, we describe the notions of threshold- and rank-based PFIs in Section 3.2.

3.1 Attribute and Tuple Uncertainty

Let V be a set of items. In the **attribute uncertainty model** [22, 8, 15, 25], each attribute value carries some uncertain information. Here we adopt the following variant [25]: a database D contains n tuples, or transactions. Each transaction, t_j is associated with a set of items taken from V. Each item $v \in V$ exists in t_j with an existential probability $Pr(v \in t_j) \in (0, 1]$, which denotes the chance that v belongs to t_j. In Table 1, for instance, the existential probability of *video* in t_{Jack} is $Pr(video_{Jack}) = 1/2$. This model can also be used to describe uncertainty in binary attributes. For instance, the item *video* can be considered as an attribute, whose value is one, for Jack's tuple, with probability $\frac{1}{2}$, in tuple t_{Jack}.

Under the Possible World Semantics (PWS), D generates a set of possible worlds $\mathcal{W}$. Table 3 lists all possible worlds for Table 1. Each world $w_i \in \mathcal{W}$, which consists of a subset of attributes from each transaction, occurs with probability $Pr(w_i)$. For example, $Pr(w_2)$ is the product of: (1) the probability that Jack purchases *food* but not *video* (equal to $\frac{1}{2}$); and (2) the probability that Mary buys *clothing* and *video* only (equal to $\frac{1}{9}$). As shown in Table 3, the sum of possible world probabilities is one, and the number of possible worlds is exponentially large. Our goal is to discover frequent patterns without expanding D into possible worlds.

$\mathcal{W}$	Tuples in $\mathcal{W}$	Prob.
w_1	{*food*}; {*clothing*}	1/9
w_2	{*food*}; {*clothing, video*}	1/18
w_3	{*food*}; {*clothing, book*}	2/9
w_4	{*food*}; {*clothing, book, video*}	1/9
w_5	{*food, video*}; {*clothing*}	1/9
w_6	{*food, video*}; {*clothing, video*}	1/18
w_7	{*food, video*}; {*clothing, book*}	2/9
w_8	{*food, video*}; {*clothing, book, video*}	1/9

Table 3: Possible Worlds of Table 1.

In the **tuple uncertainty model**, each tuple or transaction is associated with a probability value. We assume the following variant [10, 21]: each transaction $t_j \in D$ is associated with a set of items and an existential probability $Pr(t_j) \in (0, 1]$, which indicates that t_j exists in D with probability $Pr(t_j)$. Again, the number of possible worlds for this model is exponentially large. Table 4 summarizes the symbols used in this paper.

Notation	Description
D	An uncertain database of n tuples
n	The number of tuples contained in D
V	The set of items that appear in D
v	An item, where $v \in V$
t_j	The j-th tuple with a set of items, where $j = 1, \ldots, n$
$\mathcal{W}$	The set of all possible worlds.
w_j	A possible world $w_j \in W$
I	An itemset, where $I \subseteq V$
minsup	An integer between $(0, n]$
minprob	A real value between $(0, 1]$, used in threshold-based PFI
k	An integer greater than zero, used in ranked-based PFI
$Pr^I(i)$	Support probability (i.e. probability that I has a support count of i)
$Pr_{freq}(I)$	Frequentness probability of I
p_j^I	$Pr(I \subseteq t_j)$
μ^I	Expected value of X^I
μ_l^I	Expected X^I, up to l tuples

Table 4: Summary of Notations

3.2 Probabilistic Frequent Itemsets (PFI)

Let $I \subseteq V$ be a set of items, or an *itemset*. The *support* of I, denoted by $s(I)$, is the number of transactions in which I appears in a transaction database [4]. In precise databases, $s(I)$ is a single value. This is no longer true in uncertain databases, because in different possible worlds, $s(I)$ can have different values. Let $S(w_j, I)$ be the support count of I in possible world w_j. Then, the probability that $s(I)$ has a value of i, denoted by $\Pr^I(i)$, is:

$$Pr^I(i) = \sum_{w_j \in \mathcal{W}, S(w_j, I) = i} Pr(w_j) \quad (1)$$

Hence, $Pr^I(i)(i = 1, \ldots, n)$ form a *probability mass function* (or *pmf*) of $s(I)$. We call Pr^I the *support pmf* (or *s-pmf*) of I. In Table 3, for instance, $Pr^{video}(2) = Pr(w_6) + Pr(w_8) = \frac{1}{6}$, since $s(I) = 2$ in possible worlds w_6 and w_8. Figure 1 shows the s-pmf of {*video*}.

Now, let $minsup \in (0, n]$ be an integer. An itemset I is said to be *frequent* if $s(I) \geq minsup$ [4]. For uncertain databases, the *frequentness probability* of I, denoted by $Pr_{freq}(I)$, is the probability that an itemset is frequent [25]. Notice that $Pr_{freq}(I)$ can be expressed as:

$$Pr_{freq}(I) = \sum_{i \geq minsup} Pr^I(i) \quad (2)$$

In Figure 1, if $minsup = 1$, then $Pr_{freq}(\{video\}) = Pr^{\{video\}}(1) + Pr^{\{video\}}(2) = \frac{2}{3}$.

Using frequentness probabilities, we can determine whether an itemset is frequent. We identify two classes of *Probabilistic Frequent Itemsets* (or *PFI*) below:

- I is a **Threshold-based PFI** if its frequentness probability is larger than some threshold [25]. Formally, given a real value $minprob \in (0, 1]$, I is a threshold-based PFI, if $Pr_{freq}(I) \geq minprob$. We call *minprob* the *frequentness probability threshold*.
- I is a **Rank-based PFI** if its frequentness probability satisfies some ranking criteria. The *top-k* PFI, pro-

posed in [25], belongs to this class. Given an integer $k > 0$, I is a top-k PFI if $Pr_{freq}(I)$ is at least the k-th highest, among all itemsets. We focus on top-k PFI in this paper.

Before we move on, we would like to mention the following theorem, which was discussed in [25]:

THEOREM 1. **Anti-Monotonicity:** Let S and I be two itemsets. If $S \subseteq I$, then $Pr_{freq}(S) \geq Pr_{freq}(I)$.

This theorem will be used in our discussions.

We next derive efficient s-pmf computation methods in Section 4. Based on these methods, we present algorithms for retrieving threshold-based and rank-based PFIs, in Sections 5 and 6 respectively.

4. EVALUATING S-PMF

From the last section, we can see that the s-pmf $s(I)$ of itemset I plays an important role in determining whether I is a PFI. However, directly computing $s(I)$ (e.g., using the dynamic programming approaches of [25, 28]) can be expensive. We now investigate an alternative way of computing $s(I)$. In Section 4.1 we study some statistical properties of $s(I)$. Section 4.2 exploits these results by approximating $s(I)$ in a computationally efficient way.

4.1 Statistical Properties of s-pmf

An interesting observation about $s(I)$ is that it is essentially the number of successful *Poisson trials* [23]. To explain, we let X_j^I be a random variable, which is equal to one if I is a subset of the items associated with transaction t_j (i.e., $I \subseteq t_j$), or zero otherwise. Notice that $Pr(I \subseteq t_j)$ can be easily calculated in our uncertainty models:

- For *attribute-uncertainty*,

$$Pr(I \subseteq t_j) = \prod_{v \in I} Pr(v \in t_j) \tag{3}$$

- For *tuple-uncertainty*,

$$Pr(I \subseteq t_j) = \begin{cases} Pr(t_j) & \text{if } I \subseteq t_j \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

Given a database of size n, each I is associated with random variables $X_1^I, X_2^I, \ldots, X_n^I$. In both uncertainty models considered in this paper, all tuples are independent. Therefore, these n variables are independent, and they represent n Poisson trials. Moreover, $X^I = \sum_{j=1}^{n} X_j^I$ follows a Poisson binomial distribution.

Next, we observe an important relationship between X^I and $Pr^I(i)$ (i.e., the probability that the support of I is i):

$$Pr^I(i) = Pr(X^I = i) \tag{5}$$

This is simply because X^I is the number of times that I exists in the database. Hence, the s-pmf of I, i.e., $Pr^I(i)$ is the pmf of X^I, a Poisson binomial distribution.

Using Equation 5, we can rewrite Equation 2, which computes the frequentness probability of I, as:

$$Pr_{freq}(I) = \sum_{i \geq minsup} Pr(X^I = i) \tag{6}$$

$$= Pr(X^I \geq minsup) \tag{7}$$

Therefore, if the cumulative distribution function (cdf) of X^I is known, $Pr_{freq}(I)$ can also be evaluated. Next, we discuss an approach to approximate this cdf, in order to compute $Pr_{freq}(I)$ efficiently.

4.2 Approximating s-pmf

From Equation 7, we can express $Pr_{freq}(I)$ as:

$$Pr_{freq}(I) = 1 - Pr(X^I \leq minsup - 1) \tag{8}$$

For notational convenience, let p_j^I be $Pr(I \subseteq t_j)$. Then, the expected value of X^I, denoted by μ^I, can be computed by:

$$\mu^I = \sum_{j=1}^{n} p_j^I \tag{9}$$

Since a Poisson binomial distribution can be well-approximated by a Poisson distribution [6], Equation 8 can be written as:

$$Pr_{freq}(I) \approx 1 - F(minsup - 1, \mu^I) \tag{10}$$

where F is the cdf of the Poisson distribution with mean μ^I, i.e., $F(minsup-1, \mu^I) = 1 - \frac{\Gamma(minsup, \mu^I)}{(minsup-1)!}$, where $\Gamma(minsup, \mu^I) = \int_{\mu^I}^{\infty} t^{minsup-1} e^{-t} dt$. A theoretical analysis of the approximation quality is shown in Appendix A. The upper bound of the error is small. According to our experimental results, the approximation is quite accurate.

To estimate $Pr_{freq}(I)$, we can first compute μ^I by scanning D once and summing up p_j^I's for all tuples t_j in D. Then, $F(minsup - 1, \mu^I)$ is evaluated, and Equation 10 is used to approximate $Pr_{freq}(I)$.

We have also observed an important property of the frequentness probability:

THEOREM 2. $Pr_{freq}(I)$, if approximated by Equation 10, increases monotonically with μ^I.

PROOF. The cdf of a Poisson distribution, $F(i, \mu)$, can be written as: $F(i, \mu) = \frac{\Gamma(i+1, \mu)}{i!} = \frac{\int_{\mu}^{\infty} t^{(i+1)-1} e^{-t} dt}{i!}$

Since *minsup* is fixed and independent of μ, let us examine the partial derivative w.r.t. μ.

$$\begin{aligned} \frac{\partial F(i,\mu)}{\partial \mu} &= \frac{\partial}{\partial \mu} \left(\frac{\int_{\mu}^{\infty} t^{(i+1)-1} e^{-t} dt}{i!} \right) \\ &= \frac{1}{i!} \frac{\partial}{\partial \mu} \left(\int_{\mu}^{\infty} t^i e^{-t} dt \right) \\ &= \frac{1}{i!} (-\mu^i e^{-\mu}) \\ &= -f(i, \mu) \leq 0 \end{aligned}$$

Thus, the cdf of the Poisson distribution $F(i, \mu)$ is monotonically decreasing w.r.t. μ, when i is fixed. Consequently, $1 - F(i-1, \mu)$ increases monotonically with μ. Theorem 2 follows immediately by substituting $i = minsup$. □

Intuitively, Theorem 2 states that the higher value of μ^I, the higher is the chance that I is a PFI. Next, we will illustrate how this theorem avoids the costly computations of F, and improves the efficiency of finding both threshold- and rank-based PFIs.

5. THRESHOLD-BASED PFI MINING

Can we quickly determine whether an itemset I is a threshold-based PFI? Answering this question is crucial, since in typical PFI mining algorithms (e.g.,[25]), candidate itemsets are first generated, before they are tested on whether they are PFI's. In Section 5.1, we develop a simple method of testing whether I is a threshold-based PFI, without computing its frequentness probability. We then enhance this method in Section 5.2. We demonstrate an adaptation of these techniques in an existing PFI-mining algorithm, in Section 5.3.

5.1 PFI Testing

Given the values of *minsup* and *minprob*, we can test whether I is a threshold-based PFI, in three steps:

Step 1. Find a real number μ_m satisfying the equation:

$$minprob = 1 - F(minsup - 1, \mu_m) \tag{11}$$

The above Equation can be solved efficiently by employing numerical methods, thanks to Theorem 2.

Step 2. Use Equation 9 to compute μ^I. Notice that the database D has to be scanned once.

Step 3. If $\mu^I \geq \mu_m$, we conclude that I is a PFI. Otherwise, I must not be a PFI.

To understand why this works, first notice that the right side of Equation 11 is the same as that of Equation 10, an expression of frequentness probability. Essentially, Step 1 finds out the value of μ_m that corresponds to the frequentness probability threshold (i.e., *minprob*). In Steps 2 and 3, if $\mu^I \geq \mu_m$, Theorem 2 allows us to deduce that $Pr_{freq}(I) > minprob$. Hence, these steps together can test whether an itemset is a PFI.

In order to verify whether I is a PFI, once μ_m is found, we do not have to evaluate $Pr_{freq}(I)$. Instead, we compute μ^I in Step 2, which can be done in $O(n)$ time. This is a more scalable method compared with solutions in [28, 25], which evaluate $Pr_{freq}(I)$ in $O(n^2)$ time. Next, we study how this method can be further improved.

5.2 Improving the PFI Testing Process

In Step 2 of the last section, D has to be scanned once to obtain μ^I, for every itemset I. This can be costly if D is large, and if many itemsets need to be tested. For example, in the Apriori algorithm [25], many candidate itemsets are generated first before testing whether they are PFIs. We now explain how the PFI testing can still be carried out without scanning the whole database.

Let $\mu_l^I = \sum_{j=1}^{l} p_j$, where $l \in (0, n]$. Essentially, μ_l^I is the "partial value" of μ^I, which is obtained after scanning l tuples. Notice that $\mu_n^I = \mu^I$. Suppose that μ_m has been obtained from Equation 11, we first claim the following:

LEMMA 1. Let $i \in (0, n]$. If $\mu_i^I \geq \mu_m$, then I is a threshold-based PFI.

PROOF. Notice that μ_i^I monotonically increases with i. If there exists a value of i such that $\mu_i^I \geq \mu_m$, we must have $\mu^I = \mu_n^I \geq \mu_i^I \geq \mu_m$, implying that I is a PFI. □

Using Lemma 1, a PFI can be verified by scanning only a part of the database. We next show the following.

LEMMA 2. If I is a threshold-based PFI, then:

$$\mu_{n-i}^I \geq \mu_m - i \ \forall i \in (0, \lfloor \mu_m \rfloor] \tag{12}$$

PROOF. Let D_l be a set of tuples $\{t_1, \ldots, t_l\}$. Then,

$$\mu^I = \sum_{j=1}^{n} Pr(I \subseteq t_j); \quad \mu_l^I = \sum_{j=1}^{l} Pr(I \subseteq t_j)$$

Since $Pr(I \subseteq t_j) \in [0, 1]$, based on the above equations, we have:

$$i \geq \mu^I - \mu_{n-i}^I \tag{13}$$

If itemset I is a PFI, then $\mu^I \geq \mu_m$. In addition, $\mu_{n-i}^I \geq 0$. Therefore,

$$i \geq \mu^I - \mu_{n-i}^I \geq \mu_m - \mu_{n-i}^I \ for\ 0 < i \leq \lfloor \mu_m \rfloor$$
$$\therefore \mu_{n-i}^I \geq \mu_m - i \text{ for } 0 < i \leq \lfloor \mu_m \rfloor$$

□

This lemma leads to the following corollary.

COROLLARY 1. An itemset I cannot be a PFI if there exists $i \in (0, \lfloor \mu_m \rfloor]$ such that:

$$\mu_{n-i}^I < \mu_m - i \tag{14}$$

We use an example to illustrate Corollary 1. Suppose that $\mu_m = 1.1$ for the database in Table 1. Also, let $I = \{clothing,\ video\}$. Using Corollary 1, we do not have to scan the whole database. Instead, only the tuple t_{Jack} needs to be read. This is because:

$$\mu_1^I = 0 < 1.1 - 1 = 0.1 \tag{15}$$

Since Equation 14 is satisfied, we confirm that I is not a PFI without scanning the whole database.

We use the above results to improve the speed of the PFI testing process. Specifically, after a tuple has been scanned, we check whether Lemma 1 is satisfied; if so, we immediately conclude that I is a PFI. After scanning $n - \lfloor \mu_m \rfloor$ or more tuples, we examine whether I is not a PFI, by using Corollary 1. These testing procedures continue until the whole database is scanned, yielding μ^I. Then, we execute Step 3 (Section 5.1) to test whether I is a PFI.

5.3 Case Study: The Apriori Algorithm

The testing techniques just mentioned are not associated with any specific threshold-based PFI mining algorithms. Moreover, these methods support both attribute- and tuple-uncertainty models. Hence, they can be easily adopted by existing algorithms. We now explain how to incorporate our techniques to enhance the Apriori [25] algorithm, an important PFI mining algorithms.

The resulting procedure (Algorithm 1) uses the "bottom-up" framework of the Apriori: starting from $k = 1$, size-k PFIs (called k-PFIs) are first generated. Then, using Theorem 1, size-$(k+1)$ candidate itemsets are derived from the k-PFIs, based on which the k-PFIs are found. The process goes on with larger k, until no larger candidate itemsets can be discovered.

The main difference of Algorithm 1 compared with that of Apriori [25] is that all steps that require frequentness probability computation are replaced by our PFI testing methods. In particular, Algorithm 1 first computes μ_m (Line 2). Then, for each candidate itemset I generated on Line 3 and Line 17, we scan D and compute its μ_i^I (Line 10). If Lemma 1 is satisfied, then I is put to the result (Lines 11-13). However,

Algorithm 1: **Apriori-based PFI Mining**

Input: Uncertain database D, *minsup*, *minprob*
Output: All PFI: $F = \{F_1, F_2, \ldots, F_m\}$ // F_k is set of k-PFIs

```
1  begin
2    μ_m = MinExpSup(minsup, minprob);
3    C_1.GenerateSingleItemCandidates(D);
4    k = 1; j = 0;
5    while |C_k| != 0 do
6      for each I ∈ C_k do
7        I.μ = 0;
8      while (++j) ≤ n and |C_k| != 0 do
9        for each I ∈ C_k do
10         I.μ += Pr(I ⊆ t_j);
11         if I.μ ≥ μ_m then
12           F_k.push(I);
13           C_k.remove(I);
14         else if j ≥ n − ⌊μ_m⌋ then
15           if Pruning(I, μ_m, j, n) == true then
16             C_k.remove(I);
17     C_{k+1}.GenerateCandidates(F_k);
18     k += 1; j = 0
19   return F;
20 end
```

if Corollary 1 is satisfied, I is pruned from the candidate itemsets (Lines 14-16). This process goes on until no more candidates itemsets are found.

Complexity. In Algorithm 1, each candidate item needs $O(n)$ time to test whether it is a PFI. This is much faster than the Apriori [25], which verifies a PFI in $O(n^2)$ time. Moreover, since D is scanned once for all k-PFI candidates C_k, at most a total of n tuples is retrieved for each C_k (instead of $|C_k| \cdot n$). The space complexity is $O(|C_k|)$ for each candidate set C_k, in order to maintain μ^I for each candidate.

6. RANK-BASED PFI MINING

Besides mining threshold-based PFIs, the s-pmf approximation methods presented in Section 4 can also facilitate the discovery of rank-based PFIs (i.e., PFIs returned based on their relative rankings). In this section we investigate an adaptation of our methods for finding top-k PFIs, a kind of rank-based PFIs which orders PFIs according to their frequentness probabilities.

Our solution (Algorithm 2) follows the framework in [25]: A bounded priority queue, Q, is maintained to store candidate itemsets that can be top-k PFIs, in descending order of their frequentness probabilities. Initially, Q has a capacity of k itemsets, and single items with the k highest probabilities are deposited into it. Then, the algorithm iterates itself k times. During each iteration, the first itemset I is popped from Q, which has the highest frequentness probability among those in Q. Based on Theorem 1, I must also rank higher than itemsets that are supersets of I. Hence, I is one of the top-k PFIs. A *generation step* is then performed, which produces candidate itemsets based on I. Next, a *testing step* is carried out, to determine which of these itemsets should be put to Q, with Q's capacity decremented. The top-k PFIs are produced after k iterations.

We now explain how the generation and testing steps in [25] can be redesigned, in respectively Sections 6.1 and 6.2.

6.1 Candidate Itemset Generation

Given an itemset I retrieved from Q, in [25] a candidate itemset is generated from I by unioning I with every single item in V. Hence, the maximum number of candidate itemsets generated is $|V|$, which is the number of single items.

Algorithm 2: **top-k PFI Mining**

Input: Uncertain database D, *minsup*, k
Output: Top-k PFI Set: T

```
1  begin
2    C.GenerateSingleItemCandidates(D);
3    C.TestCandidates(D);
4    Q.size = k;
5    Q.UpdateQueue(C);
6    while T.size < k do
7      I = Q.pop();
8      T.push(I);
9      C.GenerateCandidates(I, Q);
10     C.TestCandidates(D);
11     Q.size = Q.size − 1;
12     Q.UpdateQueue(C);
13   return T
14 end
```

We argue that the number of candidate itemsets can actually be fewer, if the contents of Q are also considered. To illustrate this, suppose $Q = \{\{abc\}, \{bcd\}, \{efg\}\}$, and I=$\{abc\}$ has the highest frequentness probability. If the generation step of [25] is used, then four candidates are generated for I: $\{\{abcd\}, \{abce\}, \{abcf\}, \{abcg\}\}$). However, $Pr_{freq}(\{abce\}) \leq Pr_{freq}(\{bce\})$, according to Theorem 1. Since $\{bce\}$ is not in Q, $\{bce\}$ must also be not a top-k PFI. Using Theorem 1, we can immediately conclude that $\{abce\}$, a superset of $\{bce\}$, cannot be a top-k PFI. Hence, $\{abce\}$ cannot be a top-k PFI candidate. Using similar arguments, $\{abcf\}$ and $\{abcg\}$ do not need to be generated, either. In this example, only $\{abcd\}$ should be a top-k PFI candidate.

Algorithm. Based on this observation, we redesign the generation step (Line 9 of Algorithm 2) as follows: for any itemsets $I' \in Q$, if I and I' contain the same number of items, and there is only one item that differentiates I from I', we generate a candidate itemset $I \cup I'$. This guarantees that $I \cup I'$ contains items from both I and I'.

Since Q has at most k itemsets, the new algorithm generates at most k candidates. In practice, the number of single items ($|V|$) is large compared with k. For example, in the *accidents* Dataset used in our experiments, $|V| = 572$. Hence, our algorithm generates fewer candidates, and significantly improves the performance of the solution in [25], as shown in our experimental results.

6.2 Candidate Itemset Testing

After the set of candidate itemsets, C, is generated, [25] performs testing on them by first computing their exact frequentness probabilities, then comparing with the minimal frequentness probability, Pr_{min}, for all itemsets in Q. Only those in C with probabilities larger than Pr_{min} are added to Q. As explained before, computing frequentness probabilities can be costly. Here we propose a faster solution based on the results in Section 4. The main idea is that instead of ranking the itemsets in Q based on their frequentness probabilities, we order them according to their μ^I values. Also, for each $I' \in C$, instead of computing $Pr_{freq}(I')$, we evaluate $\mu^{I'}$. These values are then compared with μ_{min}, the minimum value of μ^I among the itemsets stored in Q. Only candidates whose $\mu^{I'}$s are not smaller than μ_{min} are

placed into Q. An itemset I' selected in this way has a good chance to satisfy $Pr_{freq}(I') \geq Pr_{min}$, according to Theorem 2, as well as being a top-k PFI. Hence, by comparing the μ^I values, we avoid computing any frequentness probability values.

Complexity. The details of the above discussions are shown in Algorithm 2. As we can see, $\mu^{I'}$, computed for every itemset $I' \in C$, is used to determine whether I' should be put into Q. During each iteration, the time complexity is $O(n+k)$. The overall complexity of algorithm is $O(kn + k^2)$. This is generally faster than the solution in [25], whose complexity is $O(kn^2 + k|V|)$.

We conclude this section with an example. Suppose we wish to find top-2 PFIs from the dataset in Table 1. There are four single items, and the initial capacity of Q is $k = 2$. Suppose that after computing the μ^Is of single items, $\{food\}$ and $\{clothing\}$ have the highest values. Hence, they are put to Q (Table 5). In the first iteration, $\{food\}$, the PFI with the highest μ^I is returned as the top-1 PFI. Based on our generation step, $\{food,\ clothing\}$ is the only candidate. However, $\mu^{\{food,\ clothing\}}$ is smaller than μ_{min}, which corresponds to the minimum μ^I among itemsets in Q. Hence, $\{food,\ clothing\}$ is not put to Q. Moreover, the capacity of Q is reduced to one. Finally, $\{clothing\}$ is popped from Q and returned.

Iteration	Answer	Priority Queue (Q)
0	$\emptyset$	$\{\{food\},\{clothing\}\}$
1	$\{food\}$	$\{\{clothing\}\}$
2	$\{clothing\}$	$\emptyset$

Table 5: Mining Top-2 PFIs

7. RESULTS

We now present the experimental results on two datasets. The first one, called *accidents*, comes from the Frequent Itemset Mining (FIMI) Dataset Repository[1]. This dataset is obtained from the National Institute of Statistics (NIS) for the region of Flanders (Belgium), for the period of 1991–2000. The data are obtained from the 'Belgian Analysis Form for Traffic Accidents', which are filled out by a police officer for each traffic accident occurring on a public road in Belgium. The dataset contains 340,184 accident records, with a total of 572 attribute values. On average, each record has 45 attributes.

We use the first 10k tuples as our default dataset. The default value of *minsup* is 20% of the database size n.

The second dataset, called *T10I4D100k*, is produced by the IBM data generator[2]. The dataset has a size n of 100k transactions. On average, each transaction has 10 items, and a frequent itemset has four items. Since this dataset is relatively sparse, we set *minsup* to 1% of n.

For both datasets, we consider both attribute and tuple uncertainty models. For attribute uncertainty, the existential probability of each attribute is drawn from a Gaussian distribution with mean 0.5 and standard deviation 0.125. This same distribution is also used to characterize the existential probability of each tuple, for the tuple uncertainty model. The default values of *minprob* and k are 0.4 and 10 respectively. In the results presented, *minsup* is shown as a percentage of the dataset size n. Notice that when the values of *minsup* or *minprob* are large, no PFIs can be returned; we do not show the results for these values. Our experiments were carried out on the Windows XP operating system, on a machine with a 2.66 GHz Intel Core 2 Duo processor and 2GB memory. The programs were written in C and compiled with Microsoft Visual Studio 2008.

We first present the results on the real dataset. Sections 7.1 and 7.2 describe respectively the results for mining threshold- and rank-based PFIs, on attribute-uncertain data. We summarize the results for tuple-uncertain data and synthetic data, in Section 7.3.

7.1 Results on Threshold-based PFI Mining

We now compare the performance of three threshold-based PFI mining algorithms mentioned in this paper: (1) `DP`, the Apriori algorithm used in [25]; (2) `MB`, the modified Apriori algorithm that employs the PFI testing method (Section 5.1); and (3) `MBP`, the algorithm that uses the improved version of the PFI testing method (Section 5.2).

(i) Accuracy. Since `MB` approximates s-pmf by a Poisson distribution, we first examine its accuracy with respect to `DP`, which yields PFIs based on exact frequentness probabilities. Here, we use the standard *recall* and *precision* measures [5], which quantify the number of negatives and false positives. Specifically, let F_{DP} be the set of PFIs generated by `DP`, and F_{MB} be the set of PFIs produced by `MB`. Then, the recall and the precision of `MB`, relative to `DP`, can be defined as follows:

$$recall = \frac{|F_{DP} \cap F_{MB}|}{|F_{DP}|} \tag{16}$$

$$precision = \frac{|F_{DP} \cap F_{MB}|}{|F_{MB}|} \tag{17}$$

In these formulas, both recall and precision have values between 0 and 1. Also, a higher value reflects a better accuracy.

Table 6 shows the recall and the precision of `MB`, for a wide range of *minsup*, n and *minprob* values. As we can see, the precision and recall values are always higher than 98%. Hence, the PFIs returned by `MB` are highly similar to those returned by `DP`. Since `MBP` returns the same PFIs as `MB`, it is also highly accurate.

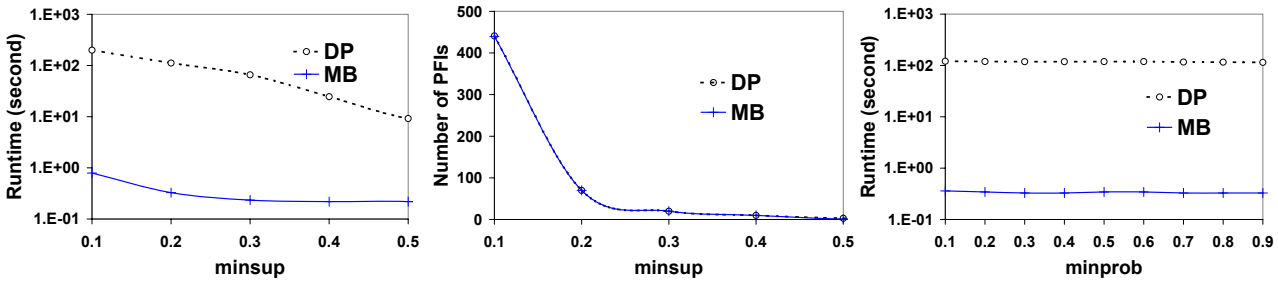
minsup/n	0.1	0.2	0.3	0.4	0.5
Recall	1	1	1	1	1
Precision	0.997	1	1	1	1
(a)*Recall* & *Precision* vs. *minsup*					
minprob	0.1	0.3	0.5	0.7	0.9
Recall	1	1	1	1	1
Precision	0.986	1	0.985	1	1
(b)*Recall* & *Precision* vs. *minprob*					
n	1k	4k	10k	50k	100k
Recall	1	1	1	1	1
Precision	0.987	0.988	1	1	1
(c)*Recall* & *Precision* vs. n					

Table 6: Recall and Precision of `MB`

(ii) `MB` vs. `DP`. Next, we compare the performance (in log scale) of `MB` and `DP`, in Figure 2(a). Observe that `MB` is about two orders of magnitude faster than `DP`, over a wide range of *minsup*. This is because `MB` does not compute exact frequentness probabilities as `DP` does; instead, `MB` only computes the μ^I values, which can be obtained faster. We also notice that the running times of both algorithms decrease with a higher

[1] `http://fimi.cs.helsinki.fi/`

[2] `http://www.almaden.ibm.com/cs/disciplines/iis/`



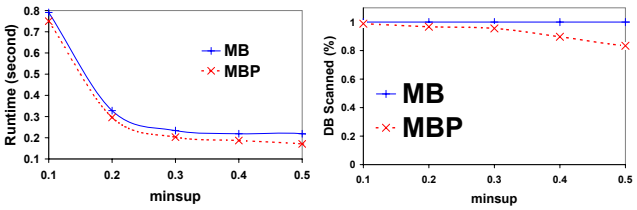
(a) Runtime vs. *minsup* (b) $|PFI|$ vs. *minsup* (c) Runtime vs. *minprob*

Figure 2: Threshold-based PFI Mining: Efficiency of model-based algorithm MB vs. dynamic programming DP

minsup. This is explained by Figure 2(b), which shows that the number of PFIs generated by the two algorithms, $|PFI|$, decreases as *minsup* increases. Thus, the time required to compute the frequentness probabilities of these itemsets decreases. We can also see that $|PFI|$ is almost the same for the two algorithms, reflecting that the results returned by MB closely resemble those of DP.

Figure 2(c) examines the performance of MB and DP (in log scale) over different *minprob* values. Their execution times drop by about 6% when *minprob* changes from 0.1 to 0.9. We see that MB is faster than DP. For instance, at *minprob* = 0.5, MB needs 0.3 seconds, while DP requires 118 seconds, delivering an almost 400-fold performance improvement.

(iii) MB vs. MBP. We then examine the benefit of using the improved PFI testing method (MBP) over the basic one (MB). Figure 3(a) shows that MBP runs faster than MB over different *minsup* values. For instance, when *minsup* = 0.5, MBP has about 25% of performance improvement. Moreover, as *minsup* increases, the performance gap increases. This can be explained by Figure 3(b), which presents the fraction of the database scanned by the two algorithms. When *minsup* increases, MBP examines a smaller fraction of the database. For instance, at *minsup* = 0.5, MBP scans about 80% of the database. This reduces the I/O cost and the effort for interpreting the retrieved tuples. Thus, MBP performs better than MB.

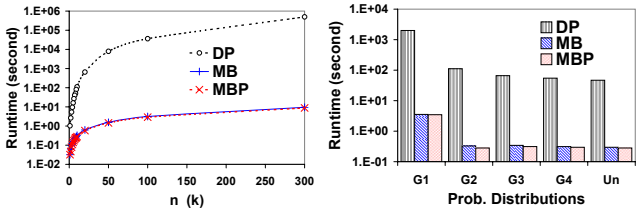


(a) Runtime vs. *minsup* (b) DB Scanned(%) vs. *minsup*

Figure 3: Efficiency of MBP vs. MB

(iv) Scalability. Figure 4(a) examines the scalability of the three algorithms. Both MB and MBP scale well with n. The performance gap between MB/MBP and DP also increases with n. At $n = 20k$, MB and DP need 0.62 seconds and 657.7 seconds respectively; at $n = 100k$, MB finishes in 3.1 seconds while DP spends 10 hours. Hence, the scalability of our approaches is better than that of DP.

(v) Existential probability. We also examine the effect of using different distributions to characterize an attribute's probability, in Figure 4(b). We use *Un* to denote a uniform distribution, and G_i $(i = 1, \ldots, 4)$ to represent a Gaussian



(a) Scalability (b) Prob. Distributions

Figure 4: Other Results for Threshold-Based PFIs

distribution. The details of these distributions are shown in Table 7. We observe that MB and MBP perform better than DP over different distributions. All algorithms run comparatively slower on G_1. This is because G_1 has high mean (0.8) and low standard deviation (0.125), which generates high existential probability values. As a result, many candidates and PFIs are generated. Also note that G_3 and Un, which have the same mean and standard deviation, yield similar performance. Lastly, we found that the precision and the recall of MB and MBP over these distributions are the same, and are close to 1. Hence, the PFIs retrieved by our methods closely resemble those returned by DP.

	Mean	Standard Deviation
G_1	0.8	0.125
G_2(default)	0.5	0.125
G_3	0.5	$\sqrt{1/12} \approx 0.289$
G_4	0.5	0.5
Un	0.5	$\sqrt{1/12} \approx 0.289$

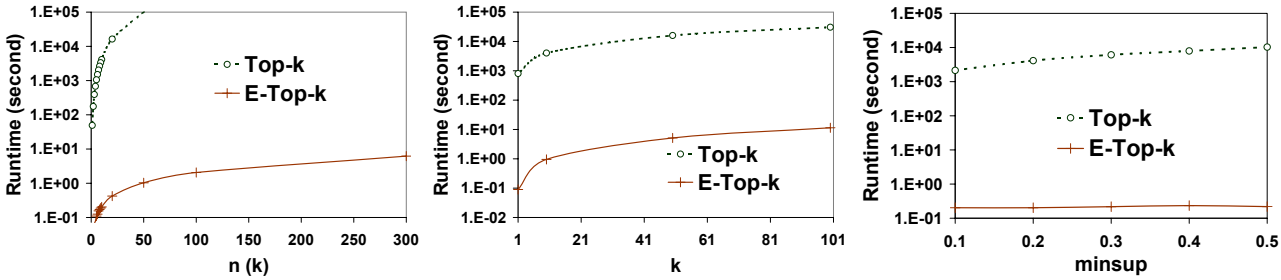
Table 7: Existential Probability (Experiment (v))

7.2 Results on Rank-based PFI Mining

We now compare the first top-k solution proposed in [25] (called top-k) and our enhanced algorithm (called E-top-k).

(vi) Accuracy. We measure the recall and precision of E-top-k relative to that of top-k, by using formulas similar to Equation 16 and 17. Table 8 shows the recall and the precision of E-top-k for a wide range of n, k and *minsup* values. We observe that the recall values are equal to the precision values. This is because number of PFIs returned by top-k and E-top-k are the same. As We can also see the recall and precision values are always higher than 98%. Hence, E-top-k can accurately return top-k PFIs.

(vii) Performance. Figures 5(a), (b), and (c) compare the two algorithms under different values of n, k, and *minsup*. Similar to the case of threshold-based PFIs, our solution runs much faster than top-k. When $n = 10k$, for

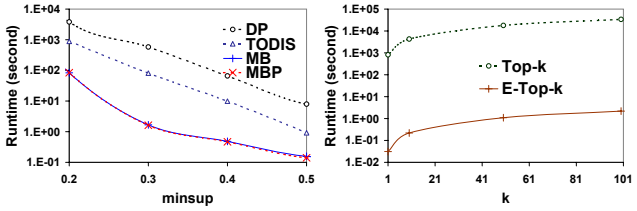


(a) Runtime vs. n (b) Runtime vs. k (c) Runtime vs. *minsup*

Figure 5: Performance of Rank-Based PFI Mining Algorithms

n	1k	5k	10k	50k
Recall & Precision	1	1	1	1
(a)*Recall & Precision* vs. n				
k	1	10	50	100
Recall & Precision	1	1	0.98	0.99
(b)*Recall & Precision* vs. k				
minsup/n	0.1	0.2	0.3	0.4
Recall & Precision	1	1	1	1
(c)*Recall & Precision* vs. *minsup*				

Table 8: Recall and Precision of `E-top-k`



(a) Runtime vs. *minsup* (b) Runtime vs. k

Figure 6: Results for Tuple Uncertainty

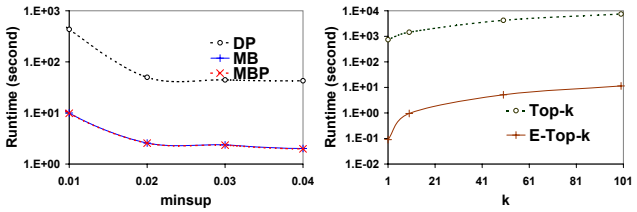
example, `E-top-k` finishes in only 0.2 seconds, giving an almost 20000-fold improvement over that of `top-k`, which completes in 1.1 hours. In Figure 5(a), we see that the runtime of `top-k` increases faster than that of `E-top-k` with a bigger database. In Figure 5(b), observe that the `E-top-k` is about four orders of magnitude faster than `top-k`.In Figure 5(c), with the increase of *minsup*, `top-k` needs more time, but the runtime of E-top-k only slightly increases. This is because (1) fewer candidates are produced by our generation step; and (2) the testing step is significantly improved by using our model-based methods.

7.3 Other Experiments

We have also performed experiments on the tuple uncertainty model and the synthetic dataset. Since they are similar to the results presented above, we only describe the most representative ones. For the accuracy aspect, the recall and precision values of approximate results on these datasets are still higher than 98%. Thus, our approaches can return accurate results.

Tuple uncertainty. We compare the performance of `DP`, `TODIS`, `MB` and `MBP` in Figure 6(a). `TODIS` [24] is proposed to retrieve threshold-based PFIs from tuple-uncertain databases. It shows that both `MB` and `MBP` perform much better than `DP` and `TODIS`, under different *minsup* values. For example, when $minsup = 0.3$, `MB` needs 1.6 seconds, but `DP` and `TODIS` complete in 581 and 81 seconds respectively. In Figure 6(b), we also see that `E-top-k` consistently outperforms `top-k` by about two orders of magnitude. This means that our approach, which avoids computing frequentness probabilities, also works well for the tuple uncertainty model.

Synthetic Dataset. Finally, we run the experiments on the synthetic dataset. Figure 7(a) compares the performance of `MB`, `MBP` and `DP`, for attribute uncertainty model. We found that `MB` and `MBP` still outperform `DP`. Figure 7(b) tests the performance of `top-k` and `E-top-k`, for tuple uncertainty model. Again, `E-top-k` runs faster than `top-k`, by around two orders of magnitude. Thus, our approach also works well for this dataset.



(a) Runtime vs. *minsup* (b) Runtime vs. k

Figure 7: Results for Synthetic Dataset

8. CONCLUSIONS

We propose a model-based approach to discover PFIs from uncertain databases. Our method represents the s-pmf of a PFI as some probability models, in order to find PFIs quickly. This method can efficiently extract threshold- and rank-based PFIs. It also supports attribute and tuple uncertainty models, which are two common data models. We develop new approximation methods to evaluate frequentness probabilities efficiently. As shown theoretically and experimentally, our algorithms are more efficient and scalable than existing ones. They are also highly accurate. In the future, we will examine how the model-based approach can be used to handle other mining problems (e.g., clustering) in uncertain databases. We will also study how other approximation techniques, such as sampling, can be used to mine PFIs.

9. ACKNOWLEDGMENT

This work was supported by the Research Grants Council of Hong Kong (GRF Projects 513508 and 711309E). We would like to thank the anonymous reviewers for their insightful comments.

10. REFERENCES

[1] A. Deshpande et al. Model-driven data acquisition in sensor networks. In *VLDB*, 2004.

[2] C. Aggarwal, Y. Li, J. Wang, and J. Wang. Frequent pattern mining with uncertain data. In *KDD*, 2009.

[3] C. Aggarwal and P. Yu. A survey of uncertain data algorithms and applications. *TKDE*, 21(5), 2009.

[4] R. Agrawal, T. Imieliński, and A. Swami. Mining association rules between sets of items in large databases. In *SIGMOD*, 1993.

[5] C. J. van Rijsbergen. *Information Retrieval*. Butterworth, 1979.

[6] L. L. Cam. An approximation theorem for the Poisson binomial distribution. In *Pacific Journal of Mathematics*, volume 10, 1960.

[7] H. Cheng, P. Yu, and J. Han. Approximate frequent itemset mining in the presence of random noise. *SCKDDM*, 2008.

[8] R. Cheng, D. Kalashnikov, and S. Prabhakar. Evaluating probabilistic queries over imprecise data. In *SIGMOD*, 2003.

[9] C. K. Chui, B. Kao, and E. Hung. Mining frequent itemsets from uncertain data. In *PAKDD*, 2007.

[10] G. Cormode and M. Garofalakis. Sketching probabilistic data streams. In *SIGMOD*, 2007.

[11] N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. In *VLDB*, 2004.

[12] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *SIGMOD*, 2000.

[13] J. Huang et al. MayBMS: A Probabilistic Database Management System. In *SIGMOD*, 2009.

[14] J. Ren and S. Lee and X. Chen and B. Kao and R. Cheng and D. Cheung. Naive Bayes Classification of Uncertain Data. In *ICDM*, 2009.

[15] R. Jampani, L. Perez, M. Wu, F. Xu, C. Jermaine, and P. Haas. MCDB: A Monte Carlo Approach to Managing Uncertain Data. In *SIGMOD*, 2008.

[16] N. Khoussainova, M. Balazinska, and D. Suciu. Towards correcting input data errors probabilistically using integrity constraints. In *MobiDE*, 2006.

[17] H. Kriegel and M. Pfeifle. Density-based clustering of uncertain data. In *KDD*, 2005.

[18] C. Kuok, A. Fu, and M. Wong. Mining fuzzy association rules in databases. *SIGMOD Record*, 1998.

[19] A. Lu, Y. Ke, J. Cheng, and W. Ng. Mining vague association rules. In *DASFAA*, 2007.

[20] M. Mutsuzaki et al. Trio-one: Layering uncertainty and lineage on a conventional dbms. In *CIDR*, 2007.

[21] M. Yiu et al. Efficient evaluation of probabilistic advanced spatial queries on existentially uncertain data. *TKDE*, 21(9), 2009.

[22] P. Sistla et al. Querying the uncertain position of moving objects. In *Temporal Databases: Research and Practice*. Springer Verlag, 1998.

[23] C. Stein. Approximate Computation of Expectations. *Institute of Mathematical Statistics Lecture Notes - Monograph Series*, 7, 1986.

[24] L. Sun, R. Cheng, D. W. Cheung, and J. Cheng. Mining Uncertain Data with Probabilistic Guarantees. In *SIGKDD*, 2010.

[25] T. Bernecker et al. Probabilistic frequent itemset mining in uncertain databases. In *KDD*, 2009.

[26] T. Jayram et al. Avatar information extraction system. *IEEE Data Eng. Bulletin*, 29(1), 2006.

[27] S. Tsang, B. Kao, K. Y. Yip, W. Ho, and S. Lee. Decision Trees for Uncertain Data. In *ICDE*, 2009.

[28] Q. Zhang, F. Li, and K. Yi. Finding frequent items in probabilistic data. In *SIGMOD*, 2008.

APPENDIX

A. QUALITY OF APPROXIMATION (SECTION 4.2)

In Section 4.2, we use Poisson distribution to approximate Poisson binomial distribution. Here, we summarize the results of this approximation quality, discussed in [23].

Let $X_1, X_2, ..., X_n$ be a set of Poisson trials such that $Pr(X_j = 1) = p_j$ and $X = \sum_{j=1}^{n} X_j$. Then X follows a Poisson binomial distribution. Suppose $\mu = E[X] = \sum_{j=1}^{n} p_j$. The probability of $X = i$ and $X \leq i$ can be approximated by probability distribution function (*pdf*) and cumulative distribution function (*cdf*) of Poisson distribution,

$$Pr(X = i) \approx f(i, \mu) = \frac{\mu^i}{i!} e^{-\mu}$$

$$Pr(X \leq i) \approx F(i, \mu) = \frac{\Gamma(i+1, \mu)}{i!}$$

[23] gives an upper bound of the error of the approximation:

$$|Pr(X \leq i) - F(i, \mu)| \leq (\mu^{-1} \wedge 1) \sum_{j=1}^{n} p_j^2 \qquad (18)$$

for $i = 0, 1, 2, ..., n$ where $\mu^{-1} \wedge 1 = min(\mu^{-1}, 1)$.

Now, we want to compute a bound on expression on the right hand side. Since $\mu = \sum_{j=1}^{n} p_j$,

$$(\mu^{-1} \wedge 1) \sum_{j=1}^{n} p_j^2 = min(\frac{1}{\sum_{j=1}^{n} p_j}, 1) \sum_{j=1}^{n} p_j^2$$

Obviously the above expression is greater than or equal to 0.

When $0 \leq \sum_{j=1}^{n} p_j \leq 1$,

$$(\mu^{-1} \wedge 1) \sum_{j=1}^{n} p_j^2 = \sum_{j=1}^{n} p_j^2 \leq \sum_{j=1}^{n} p_j \leq 1$$

When $\sum_{j=1}^{n} p_j > 1$,

$$(\mu^{-1} \wedge 1) \sum_{j=1}^{n} p_j^2 = \frac{\sum_{j=1}^{n} p_j^2}{\sum_{j=1}^{n} p_j} \leq \frac{\sum_{j=1}^{n} p_j}{\sum_{j=1}^{n} p_j} = 1$$

So, in either case:

$$0 < (\mu^{-1} \wedge 1) \sum_{i=1}^{n} p_i^2 \leq 1 \qquad (19)$$

The upper bound of the error is very small. As also verified by our experiments, the Poisson binomial distribution can be approximated well.