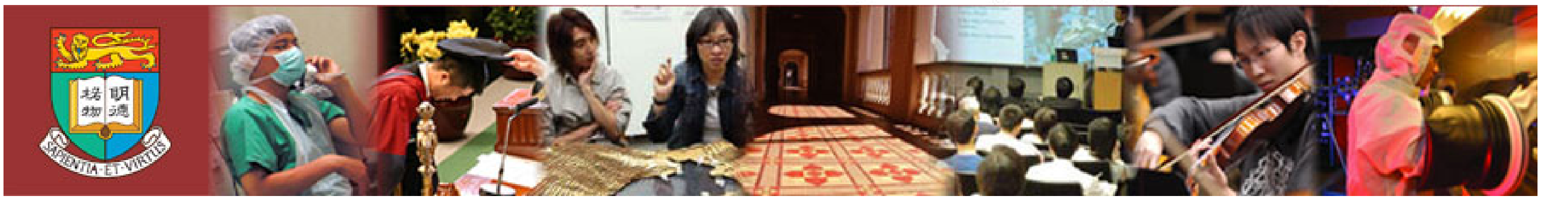


Efficient Mining of Frequent Item Sets on Large Uncertain Databases

Liang Wang, David Wai-Lok Cheung, Reynold Cheng, *Member*, *IEEE*, Sau Dan Lee, and Xuan S. Yang

Abstract—The data handled in emerging applications like location-based services, sensor monitoring systems, and data integration, are often inexact in nature. In this paper, we study the important problem of extracting frequent item sets from a large uncertain database, interpreted under the *Possible World Semantics (PWS)*. This issue is technically challenging, since an uncertain database contains an exponential number of possible worlds. By observing that the mining process can be modeled as a Poisson binomial distribution, we develop an approximate algorithm, which can efficiently and accurately discover frequent item sets in a large uncertain database. We also study the important issue of maintaining the mining result for a database that is evolving (e.g., by inserting a tuple). Specifically, we propose *incremental mining algorithms*, which enable Probabilistic Frequent Item set (PFI) results to be refreshed. This reduces the need of re-executing the whole mining algorithm on the new database, which is often more expensive and unnecessary. We examine how an existing algorithm that extracts exact item sets, as well as our approximate algorithm, can support incremental mining. All our approaches support both tuple and attribute uncertainty, which are two common uncertain database models. We also perform extensive evaluation on real and synthetic data sets to validate our approaches.

Index Terms—Frequent item sets, uncertain data set, approximate algorithm, incremental mining

1 Introduction

The databases used in many important and novel applications are often uncertain. For example, the locations of users obtained through RFID and GPS systems are not precise due to measurement errors [22], [28]. As another example, data collected from sensors in habitat monitoring systems (e.g., temperature and humidity) are noisy [17]. Customer purchase behaviors, as captured in supermarket basket databases, contain statistical information for predicting what a customer will buy in the future [3], [6]. Integration and record linkage tools also associate confidence values to the output tuples according to the quality of matching [16]. In structured information extractors, confidence values are appended to rules for extracting patterns from unstructured data [31]. To meet the increasing application needs of handling a large amount of uncertain data, *uncertain databases* have been recently developed [10], [16], [19], [20], [27].

Fig. 1 shows an online marketplace application, which carries probabilistic information. Particularly, 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 10 times in the previous week, out of which *video* products were clicked five times, the marketplace may conclude that Jack has a 50 percent chance of buying *videos*.

• *The authors are with the Department of Computer Science, The University of Hong Kong, Room 301, Chow Yei Ching Building, Pokfulam Road, Hong Kong. E-mail: {lwang, dcheung, ckcheng, sdlee, xyang2}@cs.hku.hk.*

Manuscript received 27 Nov. 2010; revised 20 Apr. 2011; accepted 25 May 2011; published online 19 July 2011.
Recommended for acceptance by H. Wang.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDE-2010-11-0635.
Digital Object Identifier no. 10.1109/TKDE.2011.165.

This *attribute-uncertainty* model, which is well studied in the literature [6], [10], [20], [28], associates confidence values with data attributes. It is also used to model location and sensor uncertainty in GPS and RFID systems.

To interpret uncertain databases, the *Possible World Semantics* (PWS) is often used [16]. 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 Fig. 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 [16].

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 Fig. 1 has $2^3 = 8$ possible worlds. Performing data mining under PWS can, thus, be technically challenging. In fact, the mining of uncertain data has recently attracted research attention [3]. For example, in [23], efficient clustering algorithms were developed for uncertain objects; in [21] and [32], naïve Bayes and decision tree classifiers designed for uncertain data were studied. Here, we develop scalable algorithms for finding frequent item sets (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., Fig. 1); and *tuple uncertainty*, where every tuple is associated with a probability to indicate whether it exists [15], [16], [19], [27], [34].

The frequent item sets discovered from uncertain data are naturally probabilistic, in order to reflect the confidence placed on the mining results. Fig. 2 shows a *Probabilistic*

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

Fig. 1. Illustrating an uncertain database.

Frequent Item set (*PFI*) extracted from Fig. 1. A *PFI* is a set of attribute values that occurs frequently with a sufficiently high probability. In Fig. 2, the *support probability mass function* (*s-pmf*) for the *PFI* {*video*} is shown. This is the pmf for the number of tuples (or *support count*) that contain an item set. Under PWS, a database induces a set of possible worlds, each giving a (different) support count for a given item set. Hence, the support of a frequent item set is described by a pmf. In Fig. 2, if we consider all possible worlds where item set {*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 [6], [30], [35]. These algorithms can verify whether an item set is 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 can require a long time to complete (e.g., with a 300k real data set, the dynamic programming algorithm in [6] needs 30.1 hours to find all PFIs). We observe that the s-pmf of a PFI can be captured 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* can verify a PFI in $O(n)$ time, and is thus more suitable for large databases. We demonstrate how our algorithm can be used to mine *threshold-based* PFIs, whose probabilities of being true frequent item sets are larger than some user-defined threshold [6]. Our algorithm only needs 9.2 seconds to find all PFIs [33], which is four orders of magnitudes faster than the method in [6].

Mining evolving databases. We also study the important problem of *maintaining* mining results for changing, or *evolving*, databases. The type of evolving data that we address here is about the appending, or insertion of a batch of tuples to the database. Tuple insertion is common in the applications that we consider. For example, a GPS system may have to handle location values due to the registration of a new user; in an online marketplace application, information about new purchase transactions may be appended to the database for further analysis. Fig. 3 shows

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

Fig. 3. The new database after inserting new customer information.

a new database, which is the result of appending the purchase information of Tony, a new customer, to the database in Fig. 1. Notice that these new tuples may induce changes to the mining result. For example, if the new database (Fig. 3) is considered, the s-pmf of the PFI {*video*} (Fig. 2) becomes the one shown in Fig. 4. Hence, we need to derive the PFIs for the new database. A straightforward way of refreshing the mining results is to re-evaluate the whole mining algorithm on the new database. This can be costly, however, when new tuples are appended to the database at different time instants. In fact, if the new database D^+ is similar to its older version, D, it is likely that most of the PFIs extracted from D remain valid for D^+. Based on this intuition, we develop *incremental mining algorithms*, which use the PFIs of D to derive the PFIs of D^+, instead of finding them from scratch. In this paper, we propose an incremental mining algorithm for the method studied in [6], which discovers exact PFIs. We also examine how our model-based algorithm, which discovers approximate PFIs, can be extended to handle evolving data. As our experiments show, when the change of the database is small, running our incremental mining algorithms on D^+ is much faster than finding PFIs on D^+ from scratch. In an experiment on a real data set, our model-based, incremental mining algorithm addresses a fivefold performance improvement over its nonincremental counterpart.

To summarize, we develop a model-based algorithm, which can reduce the amount of effort of scanning the database for mining threshold-based PFIs. We also develop two incremental mining algorithms, for extracting exact and approximate PFIs. All our algorithms can support both attribute and tuple uncertainty models. We study the time complexity of our approaches. Experiments on both real and synthetic data sets reveal that our methods significantly improve the performance of PFI discovery, with a high degree of accuracy.

The rest of the paper is organized as follows: in Section 2, we review the related works. Section 3 defines the problems to be studied. Section 4 describes efficient and accurate methods for computing s-pmf. In Section 5, we present our algorithm for discovering threshold-based PFIs. The exact and approximate algorithms for maintaining PFIs on evolving databases are, respectively, presented in Sections 6 and 7. Section 8 reports our experimental results. We conclude in Section 9.

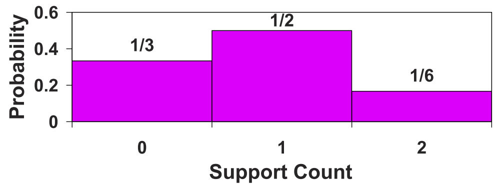


Fig. 2. s-pmf of PFI {*video*} from Fig. 1.

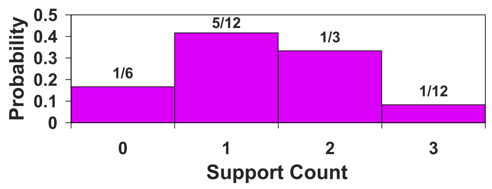


Fig. 4. s-pmf of PFI {*video*} from Fig. 3.

TABLE 1
Our Contributions (Marked [$\surd$])

Uncertainty Model	Static Algorithms	Incremental Algorithms
Attribute	Exact [6] Approx. [$\surd$]	Exact [$\surd$] Approx. [$\surd$]
Tuple	Exact [30] Approx. (singleton) [35] Approx. (multiple items) [$\surd$]	Exact [$\surd$] Approx. [$\surd$]

TABLE 2
Possible Worlds of Fig. 1

$\mathcal{W}$	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

2 Related Work

Mining frequent item sets is an important problem in data mining, and is also the first step of deriving association rules [4]. Hence, many efficient item set mining algorithms (e.g., Apriori [4] and FP-growth [18]) have been proposed. While these algorithms work well for databases with precise values, it is not clear how they can be used to mine probabilistic data. Here we develop algorithms for extracting frequent item sets from uncertain databases. Although our algorithms are developed based on the Apriori framework, they can be considered for supporting other algorithms (e.g., FP-growth) for handling uncertain data.

For uncertain databases, Aggarwal et al. [2] and Chui et al. [14] developed efficient frequent pattern mining algorithms based on the expected support counts of the patterns. However, Bernecker et al. [6], Sun et al. [30], and Yiu et al. [35] 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 [6], dynamic-programming-based solutions were developed to retrieve PFIs from attribute-uncertain databases. However, their algorithms compute exact probabilities, and verify that an item set is a PFI in $O(n^2)$ time. Our model-based algorithms avoid the use of dynamic programming, and are able to verify a PFI much faster (in $O(n)$ time). In [35], approximate algorithms for deriving threshold-based PFIs from tuple-uncertain data streams were developed. While Zhang et al. [35] only considered the extraction of singletons (i.e., sets of single items), our solution discovers patterns with more than one item. Recently, Sun et al. [30] developed an exact threshold-based PFI mining algorithm. However, it does not support attribute-uncertain data considered in this paper. In a preliminary version of this paper [33], we examined a model-based approach for mining PFIs. Here, we study how this algorithm can be extended to support the mining of evolving data.

Other works on the retrieval of frequent patterns from imprecise data include: [9], which studied approximate frequent patterns on noisy data; [24], which examined association rules on fuzzy sets; and [26], which proposed the notion of a "vague association rule." However, none of these solutions are developed on the uncertainty models studied here.

For evolving databases. A few incremental mining algorithms that work for exact data have been developed. For example, in [11], the Fast Update algorithm (FUP) was proposed to efficiently maintain frequent item sets, for a database to which new tuples are inserted. Our incremental mining framework is inspired by FUP. In [12], the FUP$_2$ algorithm was developed to handle both addition and deletion of tuples. ZIGZAG [1] also examines the efficient maintenance of maximal frequent item sets for databases that are constantly changing. In [13], a data structure, called CATS Tree, was introduced to maintain frequent item sets in evolving databases. Another structure, called CanTree [25], arranges tree nodes in an order that is not affected by changes in item frequency. The data structure is used to support mining on a changing database. To our best knowledge, maintaining frequent item sets in evolving uncertain databases has not been examined before. We propose novel incremental mining algorithms for both exact and approximate PFI discovery. Our algorithms can also support attribute and tuple uncertainty models.

Table 1 summarizes the major work done in PFI mining. Here, "Static Algorithms" refer to algorithms that do not handle database changes. Hence, any change in the database necessitates a complete execution of these algorithms.

3 Problem Definition

We now discuss the uncertainty models used in this paper, in Section 3.1. The problem of mining threshold-based PFIs is then described in Section 3.2.

3.1 Attribute and Tuple Uncertainty

Let V be a set of items. In the **attribute uncertainty model** [6], [10], [20], [28], each attribute value carries some uncertain information. Here, we adopt the following variant [6]: 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 Fig. 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, D generates a set of possible worlds $\mathcal{W}$. Table 2 lists all possible worlds for Fig. 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 2, the sum of possible world

TABLE 3
Summary of Notations

Notation	Description
D	An uncertain database of n tuples
V	The set of items that appear in D
v	An item, where $v \in V$
t_j	The j-th tuple in D
$\mathcal{W}$	The set of all possible worlds.
w_j	A possible world $w_j \in W$
I	An itemset, where $I \subseteq V$
$minsup$	A real value between $(0, 1]$
$msc(D)$	The minimal support count in D
$s(I)$	The support count of I in D
$minprob$	A real value between $(0, 1]$
$Pr^I(i)$	Support prob. (prob. 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 in D
μ_l^I	Expected value of X^I, for the first l tuples in D
Notations used in Sections 6 and 7	
d	Delta database with n' tuples
D^+	New database with n^+ tuples; $D^+ = D \cup d$
F^D	Set of all PFIs in D
F_k^D	Set of k-PFIs in D
C_k^+	Set of size-k candidates for D^+
F^+	Set of all PFIs in D^+
F_k^+	Set of k-PFIs for D^+
DB	A database, can be D, d, or D^+
$s^{DB}(I)$	The support count of I in DB
$Pr_{freq}^{DB}(I)$	The frequentness probability of I in DB
$\mu^I(DB)$	The expected value of X^I in DB

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.

In the **tuple uncertainty model**, each tuple or transaction is associated with a probability value. We assume the following variant [15], [34]: 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 3 summarizes the symbols used in this paper.

3.2 Probabilistic Frequent Item Sets

Let $I \subseteq V$ be a set of items, or an *item set*. 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

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

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

Now, let $minsup \in (0, 1]$ be a percentage value, which is generally used to define minimal support in a deterministic database. An item set I is said to be *frequent* in a database D if $s(I) \geq msc(D)$, where $msc(D) = minsup \times n$ is called the *minimal support count* of D [4]. For uncertain databases, the *frequentness probability* of I, denoted by $Pr_{freq}(I)$, is the probability that an item set is frequent [6]. Notice that $Pr_{freq}(I)$ can be expressed as

$$Pr_{freq}(I) = \sum_{i \geq msc(D)} Pr^I(i). \tag{2}$$

In Fig. 2, if $minsup = 1$, then $msc(D) = 2$. Thus, $Pr_{freq}(\{\text{video}\}) = Pr^{\{\text{video}\}}(1) + Pr^{\{\text{video}\}}(2) = \frac{2}{3}$.

Using frequentness probabilities, we can determine whether an item set I is frequent. In this paper, we adopt the definition in [6]: I is a **Threshold-based PFI** if its frequentness probability is larger than some user-defined threshold [6]. 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*.

Here, we would like to mention the following theorem, which was discussed in [6].

Theorem 1 (Antimonotonicity). *Let S and I be two item sets. If* S $\subseteq$ I, *then* Pr_{freq}(S) $\geq$ Pr_{freq}(I).

This theorem will be used in our discussions.

We derive efficient s-pmf computation methods in Section 4. Then, Section 5 examines how these methods facilitate efficiency discovery of approximate threshold-based PFIs. We examine the maintenance of exact and approximate PFIs on evolving data, in Sections 6 and 7.

4 Evaluating s-pmf

From the last section, we can see that the s-pmf $s(I)$ of item set 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 [6] and [35]) 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 manner.

4.1 Statistical Properties of s-pmf

An interesting observation about $s(I)$ is that it is essentially the number of successful *Poisson trials* [29]. 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,

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

- For tuple-uncertainty,

$$Pr(I \subseteq t_j) = \begin{cases} Pr(t_j), & \text{if } I \subseteq t_j, \qquad (4a) \\ 0, & \text{otherwise.} \qquad (4b) \end{cases}$$

Given a database of size n, each I is associated with random variables $X_1^I, X_2^I, \dots, X_n^I$. 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 (5), we can rewrite (2), which computes the frequentness probability of I, as

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

$$= Pr(X^I \geq msc(D)). \tag{7}$$

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

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

For notational convenience, let p_j^I be $Pr(I \subseteq t_j)$. Then, the expected value of X^I in D, 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 [8], (8) can be written as

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

where F is the cdf of the Poisson distribution with mean μ^I, i.e., $F(msc(D) - 1, \mu^I) = 1 - \frac{\Gamma(msc(D), \mu^I)}{(msc(D)-1)!}$, expressed using the incomplete gamma function $\Gamma(s, x) = \int_x^{\infty} t^{s-1} e^{-t} dt$. Empirical results (see Appendix A which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TKDE.2011.165) show that the errors introduced by this approximation is small in practice.

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, $F(msc(D) - 1, \mu^I)$ is evaluated, and (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 (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 $i = msc(D)$. □

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 threshold-based PFIs.

5 Mining Threshold-Based PFIs

Can we quickly determine whether an item set I is a threshold-based PFI? Answering this question is crucial, since in typical PFI mining algorithms (e.g., [6]), candidate item sets 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:

$$minprob = 1 - F(msc(D) - 1, \mu_m). \tag{11}$$

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

Step 2. Use (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 (11) is the same as that of (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 $Pr_{freq}(I) \geq minprob$. Hence, these steps together can test whether an item set 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 [6] and [35], 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 item set I. This can be costly if D is large, and if many item sets need to be tested. For example, in the Apriori algorithm [6], many candidate item sets 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 (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,

$$\mu^I = \sum_{j=1}^{n} Pr(I \subseteq t_j),$$

$$\mu_l^I = \sum_{j=1}^{l} Pr(I \subseteq t_j).$$

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

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

$$\therefore \quad \mu_{n-i}^I \geq \mu_m - i \; for \; 0 < i \leq \lfloor \mu_m \rfloor.$$

□

This lemma leads to the following corollary.

Corollary 1. *An item set 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 Fig. 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 (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 [6] 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 item sets are derived from the k-PFIs, based on which the $(k+1)$-PFIs are found. The process goes on with larger k, until no larger candidate item sets can be discovered.

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      $\mu_m$ = MinExpSup($minsup$, $minprob$, $D$);
3      $C_1$.GenerateSingleItemCandidates($D$);
4      $k$ = 1; $j$ = 0;
5      while $|C_k| \neq 0$ do
6          foreach $I \in C_k$ do
7              $I.\mu$ = 0;
8          while (++$j$) $\leq n$ and $|C_k| \neq 0$ do
9              foreach $I \in C_k$ do
10                 $I.\mu = I.\mu + Pr(I \subseteq t_j)$;
11                 if $I.\mu \geq \mu_m$ then
12                     $F_k$.push($I$);
13                     $C_k$.remove($I$);
14                 else if $j \geq n - \lfloor \mu_m \rfloor$ then
15                     if Pruning($I$, $\mu_m$, $j$, $n$) == true then
16                         $C_k$.remove($I$);
17         $C_{k+1}$.GenerateCandidate($F_k$);
18         $k = k + 1$; $j$ = 0
19     return $F$;
20 end
```

The main difference of Algorithm 1 compared with that of Apriori [6] 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 item set I generated on Lines 3 and 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, if Corollary 1 is satisfied, I is pruned from the candidate item sets (Lines 14-16). This process goes on until no more candidates item sets 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 [6], 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

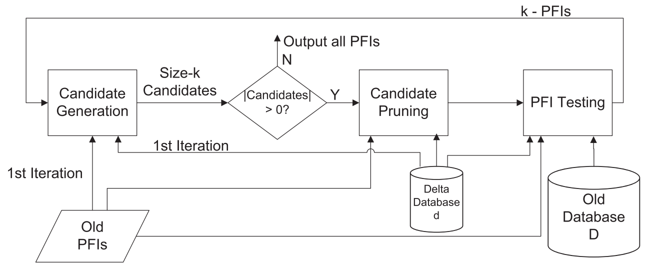


Fig. 5. Solution framework for uFUP and $\mathtt{uFUP}_{\mathrm{app}}$.

(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.

Next, we examine how to maintain PFIs in a database that is constantly evolving.

6 Exact Incremental Mining

We now examine how to efficiently maintain a set of PFIs in an *evolving database*, where new tuples, or transactions, are constantly appended to it. We assume that every tuple has a timestamp attribute, which indicates the time that it is created. This timestamp is not used for mining; it is only used to differentiate new tuples from existing ones. Let D be the "old" database that contains n tuples, and d be a *delta database* of n' tuples, whose timestamps are larger than those of tuples in D. Let D^+ be a "new" database, which is a concatenation of the tuples in D and d, and has a size of $n^+ = n + n'$. Given the set of PFIs and their s-pmfs in D, our goal is to discover PFIs on D^+, under the same $minsup$ and $minprob$ values used to mine the PFIs of D. We use $s^{DB}(I)$ and $Pr^{DB}_{freq}(I)$ to respectively denote the support count and the frequentness probability of item set I in some database DB, where DB is any of $\{D, d, D^+\}$.

Before we go on, we would like to remark that the incremental mining problem described above can be treated as a special case of *stream mining*, which refers to the maintenance of mining results for stream data. Particularly, we can view the database d as the arrival of $|d|$ data units from a stream source. Moreover, we assume that the sliding window initially contains D, which then expands to incorporate new stream units. Mining D^+ is then equivalent to updating the mining results for the arrival of $|d|$ stream units. In Section 8.3, we study an adaptation of a stream algorithm in [35] for use in incremental mining.

A simple way of obtaining PFIs from D^+ is to simply rerun a PFI-mining algorithm on it. However, this approach is not very economical, since 1) running a PFI algorithm on a large database is not trivial; and 2) the same algorithm has to be frequently executed if a lot of update activities occur. In fact, if only a few tuples in d are appended to D, it may not be necessary to compute all PFIs on D^+ from scratch. This is because the PFIs found in D^+ should not be very different from those discovered in D. Based on this intuition, we design an *incremental mining algorithm* that finds PFIs in D^+, without rerunning a complete PFI algorithm. This algorithm works the best when the size of d is very small compared with that of D; nevertheless, it works with any size of d. We next discuss the framework of our solution, which discovers exact PFIs in D^+, based on the PFIs found in D. We extend this solution to discover approximate PFIs in Section 7. Table 3 summarizes the symbols used in these sections.

6.1 Algorithm uFUP

The design of our **u**ncertain **F**ast **UP**date algorithm (or uFUP), is inspired by FUP [11]. That algorithm maintains frequent item set results in an evolving database, whose attribute values are exact. The uFUP algorithm extracts frequent item sets in an "Apriori" fashion: it utilizes a bottom-up approach, where $(k+1)$-PFIs are generated from k-PFIs. Moreover, it supports both attribute and tuple uncertainty models. As shown in Fig. 5, uFUP undergoes three phases in the kth iteration, starting from $k = 1$.

1. **Candidate generation.** In the first iteration, size-1 item sets that can be 1-PFIs are obtained, using the PFIs discovered from D, as well as the delta database d. In subsequent iterations, this phase produces size-$(k+1)$ candidate item sets, based on the k-PFIs found in the previous iteration. If no candidates are found, uFUP halts.
2. **Candidate pruning.** With the aid of d and the PFIs found from D, this phase filters the candidate item sets that must not be a PFI.
3. **PFI testing.** For item sets that cannot be pruned, they are *tested* to see whether they are the true PFIs. This involves the use of database D^+, as well as the s-pmfs of PFIs on D.

Notice that in Phases 1 and 2, only d and the PFIs of D are needed. Since these pieces of information are relatively small in size (compared with D or D^+), they are usually not very expensive to evaluate. Phase 3 involves deriving the s-pmfs of item sets, with the use of D^+, and is thus more expensive than other phases. If Phase 2 successfully removes a lot of candidates from consideration, the cost of executing Phase 3 can be reduced. This solution framework can also be used to extract approximate PFIs, which will be revisited in Section 7.

The above discussion is formalized in Algorithm 2, which uses the databases D and d, as well as the set of exact PFIs F^D collected from D (e.g., using the method of [6]). The output of uFUP is a set F^+ of PFIs for D^+, where $F^+ = \{F^+_1, F^+_2, \ldots, F^+_m\}$, and F^+_k is the set of k-PFIs for D^+. Let C^+_k be a set of size-k candidates found from D^+. Initially, $k = 1$. Line 3 generates C^+_1 (Phase 1). In the kth iteration (Lines 5-11), we first remove candidate item sets that cannot be k-PFIs, from C^+_k (Line 5; Phase 2). If C^+_k is not empty, we perform testing on these candidates, in order to find out the true k-PFIs (i.e., F^+_k), in Line 7 (Phase 3). Line 10 then generates size (k+1)-candidate item sets by using the k-PFIs. The whole process is repeated until no more candidates are found. Line 12 returns the set of PFIs of different sizes.

Algorithm 2. uFUP

```
Input: $D$, $d$, $F^D$, $minsup$, $minprob$
Output: Exact PFIs of $D^+$: $F^+ = \{F^+_1, F^+_2, \ldots, F^+_m\}$ // $F^+_k$ is
        set of $k$-PFIs
1  begin
2      $F^+ = \emptyset$;
3      $C^+_1$.GenerateSingleton($d, F^D_1$);
4      $k$ = 1; while $|C^+_k| \neq 0$ do
5          $C^+_k$.Prune($d, F^D_k, minsup$);
6          if $|C^+_k| \neq 0$ then
7              $F^+_k \leftarrow C^+_k$.Test($D, d, F^D_k, minsup, minprob$);
8          else
9              break;
10         $C^+_{k+1}$.GenerateCandidate($F^+_k$);
11         $k$ = $k$ + 1;
12     return $F^+ = \{F^+_1, F^+_2, ..., F^+_{k-1}\}$;
13 end
```

We next discuss the details of Phase 1, in Section 6.2. Then, Sections 6.3 and 6.4 present Phases 2 and 3, respectively. We discuss other issues of `uFUP` in Section 6.5.

6.2 Phase 1: Candidate Generation

We consider two cases of generating size-k candidate item sets in this phase: 1) $k = 1$ and 2) $k > 1$.

Case 1: $k = 1$. We invoke `GenerateSingleton`, in Line 3 of Algorithm 2. This subroutine simply returns the union of all single items in d and the 1-PFIs of D (i.e., F_1^D), as the set of size-1 candidate item sets (C_1^+).

To understand why `GenerateSingleton` covers all possible size-1 candidates, first notice that if an item set is a 1-PFI in D, it should naturally be considered as a candidate item set in D^+. We then claim that it suffices to include all single items of d to C_1^+, using the following lemma.

Lemma 3. *Suppose item set I is not a PFI of D. If I does not exist in any tuple of d, I is not a PFI of D^+.*

Proof. Since I is not a PFI in D, we have

$$Pr_{freq}^D(I) = Pr\big[s^D(I) \geq msc(D)\big] < minprob. \tag{16}$$

If I does not exist in d, its s-pmf will not be changed in D^+. Thus,

$$Pr\big[s^{D^+}(I) \geq msc(D)\big] = Pr\big[s^D(I) \geq msc(D)\big]. \tag{17}$$

Moreover, since $msc(D^+) \geq msc(D)$, we obtain

$$Pr\big[s^{D^+}(I) \geq msc(D^+)\big] \leq Pr\big[s^{D^+}(I) \geq msc(D)\big]. \tag{18}$$

Using Inequalities 16, 18 and (17), we can deduce that $Pr_{freq}^{D+}(I) < minprob$. Thus, I cannot be a PFI in D^+. □

Using Lemma 3, if a singleton I is not a 1-PFI in D, and does not appear in d, then I must not be a 1-PFI in D^+. Thus, by including F_1^D and all singletons in d as members of C_1^+, we will not miss any true size-1 candidate for D^+.

Case 2: $k > 1$. We use the typical `Apriori-gen` method [4] to generate size-k candidates from $(k-1)$-PFIs. Particularly, subroutine `GenerateCandidate` (Line 10 in Algorithm 2) performs the following: for any two $(k-1)$-PFIs, I and I', if there is only one item that differentiates I from I', a candidate item set $I \cup I'$ is produced. Using Lemma 1 (antimonotonicity), we can easily show that `GenerateCandidate` produces all size-k candidates.

Next, we examine how some of the candidates generated in this phase can be pruned.

6.3 Phase 2: Candidate Pruning

The goal of this phase is to remove infrequent item sets from a set of size-k candidates. In Line 5 of Algorithm 2, `Prune` is used to remove item sets from C_k^+. To understand how `Prune` works, we first present the following.

Lemma 4. *Any item set I in D^+ satisfies*

$$\begin{aligned} &Pr\big[s^{D^+}(I) < msc(D^+)\big] \\ &\quad \geq Pr\big[s^D(I) < msc(D)\big] \times Pr\big[s^d(I) \leq msc(d)\big]. \end{aligned} \tag{19}$$

This lemma, which relates the s-pmf of I in D^+ to those in D and d, is used to prove Lemma 5. The detailed proof of Lemma 4 can be found in Appendix B, available in the online supplemental material.

Let the number of tuples that contain I in d be $cnt^d(I)$. We use the following lemma for candidate pruning.

Lemma 5. *For any item set $I \notin F^D$, if $cnt^d(I) \leq msc(d)$, then $I \notin F^+$.*

Proof. Since I is not a PFI in D, we have

$$Pr\big[s^D(I) < msc(D)\big] > 1 - minprob. \tag{20}$$

If $cnt^d(I) \leq msc(d)$, then

$$Pr\big[s^d(I) \leq msc(d)\big] = 1. \tag{21}$$

Using Lemma 4, as well as (20) and (21), we have

$$\begin{aligned} &Pr\big[s^{D^+}(I) \geq msc(D^+)\big] \\ &\quad = 1 - Pr\big[s^{D^+}(I) < msc(D^+)\big] \\ &\quad \leq 1 - Pr\big[s^D(I) < msc(D)\big] \times Pr\big[s^d(I) \leq msc(d)\big] \\ &\quad < 1 - (1 - minprob) = minprob. \end{aligned}$$

Thus, I is not a PFI in D^+. □

Given an item set $I \in C_k^+$, `Prune` first checks if I is a frequent item set in D (i.e., $I \in F_k^D$). If this is false, and if $cnt^d(I)$ does not exceed $msc(d)$, then I cannot be a PFI in D^+ (Lemma 5), and I can be pruned. Notice that `Prune` does not test I on D^+, which can be expensive. Instead, it only computes $cnt^d(I)$, which can be obtained by scanning d once. If n', the size of d, is small, then getting $cnt^d(I)$ incurs a low cost. In this phase, pruning an item set not in F_k^D costs $O(n')$ times.

6.4 Phase 3: PFI Testing

Given a set of candidate item sets in C_k^+ not pruned in Phase 2, the objective of this phase is to verify whether these candidates are really k-PFIs. In particular, the subroutine `Test` (Line 7, Algorithm 2) is invoked to compute the s-pmfs of these item sets on D^+. Once this is obtained, we can easily verify whether these candidates are true k-PFIs, as discussed in the previous sections.

Although the approach of [6] can be used to compute the s-pmf of an item set I, this can be expensive, especially if the size of D^+ is large. However, if we know that I is a PFI in D, as well as its s-pmf in D, it is possible to derive the s-pmf of I in D^+ *without* computing it from scratch. The main idea is to modify the approach of [6], as outlined below: for every tuple t_j scanned from d, we evaluate the probability $Pr(I \subseteq t_j)$, and then use this to update the s-pmf of I through the use of the dynamic programming method in [6]. This process goes on, until all tuples in d are examined. Hence, the s-pmf of any item set $I \in F_k^D$ can be obtained by scanning d once. This method is effective, since if I is a PFI of D, it is highly likely that I will also be a PFI of D^+. Although the time complexity of this phase is still upper bounded by the algorithm in [6] (i.e., $O(n^{+2})$), its performance is practically improved, since the s-pmfs of some candidates can be obtained faster.

6.5 Discussions

The uFUP algorithm supports both tuple and attribute uncertainty models. First, the solutions presented in Phases 1 and 2 are not designed for any specific uncertainty model. Second, Phase 3 computes the s-pmf of an item set I by using the probability value $Pr(I \subseteq t_j)$. As explained before, this quantity can be obtained through (3) (for attribute uncertainty) and (4) (for tuple uncertainty). Hence, uFUP can be used in both models.

Our experiments reveal that for mining exact PFIs on evolving data, uFUP outperforms the algorithm mentioned in [6]. However, testing an item set in uFUP still requires $O(n^{+2})$ time. Moreover, Phase 3 needs the s-pmf information of all PFIs found in D. Since storing a s-pmf needs a cost of $O(n)$, the space cost consumed by Phase 3 can be enormous if there are many PFIs in D. We next examine how these problems can be alleviated.

7 Approximate Incremental Mining

As discussed before, our model-based algorithm enables PFIs to be accurately and quickly discovered. We now investigate how to extend it to retrieve PFIs from evolving data. We call this extension the **app**roximate **u**ncertain **F**ast **UP**date algorithm (or uFUP$_{\text{app}}$ in short).

The uFUP$_{\text{app}}$ algorithm adopts the framework of uFUP, as illustrated in Fig. 5. Algorithm 3 describes the details. In Line 3, the candidates in C_1^+ are generated (Phase 1). In Lines 5-7, the parameter values used for pruning are computed. (We will explain this later.) In the kth iteration (Lines 8-15), some candidates in the set C_k^+ are pruned (Phase 2; Line 9), while the remaining ones are tested (Phase 3; Line 11). In Line 14, size-$(k+1)$ candidates are generated by using the k-PFIs found. When no more candidates are left (Line 8), the algorithm outputs F^+, which contains PFIs of different sizes (Line 16).

Algorithm 3. uFUP$_{\text{app}}$

```
Input: $D$, $d$, $F^D$, $minsup$, $minprob$
Output: Approximate PFIs in $D$: $F^+ = \{F_1^+, F_2^+, ...F_m^+\}$
1  begin
2      $F^+ = \emptyset$;
3      $C_1^+$.GenerateSingleton($d, F_1^D$);
4      $k = 1$;
5      $\mu_m(D^+)$ = MinExpSup($minsup$, $minprob$, $D^+$);
6      $\mu_m(D)$ = MinExpSup($minsup$, $minprob$, $D$);
7      $\mu_m^- = \mu_m(D^+) - \mu_m(D)$;
8      while $|C_k^+| \neq 0$ do
9          $C_k^+$.Prune($d$, $F_k^D$, $\mu_m^-$);
10         if $|C_k^+| \neq 0$ then
11             $F_k^+ \leftarrow C_k^+$.Test($D$, $d$, $F_k^D$, $\mu_m(D^+)$);
12         else
13             break;
14         $C_{k+1}^+$.GenerateCandidate($F_k^+$);
15         $k = k + 1$;
16     return $F^+ = \{F_1^+, F_2^+, ..., F_{k-1}^+\}$;
17 end
```

Phase 1 of uFUP$_{\text{app}}$ is the same as that of uFUP; particularly, the details of GenerateSingleton and GenerateCandidate can be found in Section 6.2. In the rest of this section, we focus on Phase 2 (candidate pruning) and Phase 3 (PFI testing). Sections 7.1 and 7.2 present the details of these phases. We address other issues of uFUP$_{\text{app}}$ in Section 7.3.

7.1 Phase 2: Candidate Pruning

To facilitate our discussions, let $\mu^I(DB)$ be the expected value of random variable X^I in DB, where DB is any of the database D, d, or D^+. Also, let $\mu_m(DB)$ be a real value that satisfies (11) in DB. We first present the following theorem.

Theorem 3. *Consider an item set I that is not a PFI in D. Then I is a PFI in D^+ only if $\mu^I(d) > \mu_m^-$, where $\mu_m^- = \mu_m(D^+) - \mu_m(D)$.*

Proof. Since I is not a PFI, we have

$$\mu^I(D) < \mu_m(D), \text{ i.e., } \mu_m(D) - \mu^I(D) > 0.$$

From (9), we have

$$\mu^I(D^+) = \sum_{j=1}^{n^+} p_j^I = \sum_{j=1}^{n} p_j^I + \sum_{j=(n+1)}^{n^+} p_j^I = \mu^I(D) + \mu^I(d). \quad (22)$$

So, if I is a PFI in D^+, then

$$\begin{aligned} \mu^I(D^+) &\geq \mu_m(D^+) \\ \mu^I(D) + \mu^I(d) &\geq \mu_m^- + \mu_m(D) \\ \mu^I(d) - \mu_m^- &\geq \mu_m(D) - \mu^I(D) > 0 \\ \text{Therefore,} \quad \mu^I(d) &> \mu_m^-. \end{aligned}$$

□

This theorem is used by Phase 2. In Algorithm 3, lines 5-7 compute the value of μ_m^-. (The subroutine MinExpSup evaluates (11)). Then, in Line 9, subroutine Prune uses Theorem 3 to remove candidates that are not PFIs in D, and whose $\mu^I(d)$ values do not exceed μ_m^-. Since Prune needs to scan d once to obtain $\mu^I(d)$, the cost of pruning an item set is $O(n')$.

7.2 Phase 3: PFI Testing

The objective of this phase is to verify whether an item set in C_k^+ is a true k-PFI. In particular, subroutine Test (Line 11, Algorithm 3) is invoked to perform this task: for each item set I, it first computes $\mu^I(D^+)$. If this value is not less than $\mu_m(D^+)$, I is judged to be a PFI of D^+. The rationale behind this process can be found in Section 5.1.

A simple way of computing $\mu^I(D^+)$ is to scan the tuples in D^+ once. This can be costly, if many candidates need to be tested. Similar to the Phase 3 of uFUP, it is possible to improve the performance of this process, by using the PFI information of D. Suppose we know the $\mu^I(D)$ value of an item set I, which is a PFI of D. We first evaluate $\mu^I(d)$, by scanning d once. The value of $\mu^I(D^+)$ can be then obtained by adding these two values together (based on (22)). If d is small, scanning tuples in d is fast, and so computing $\mu^I(D^+)$ can be more efficient. In uFUP$_{\text{app}}$, we save the $\mu^I(D)$ values of all the PFIs discovered in D, so that they can later be used to derive PFIs for D^+.

7.3 Discussions

Since the model-based approach supports both tuple and attribute uncertainty (Section 4), the uFUP$_{\text{app}}$ algorithm, which adopts the model-based approach, can also be used in both data models. We also remark that uFUP$_{\text{app}}$ is generally faster than uFUP, since less time is needed to test approximate PFIs than exact PFIs. Moreover, in Phase 3, while `uFUP` has to store the complete s-pmf for every PFI found from D, $\texttt{uFUP}_{\text{app}}$ only stores a single value, $\mu^I(D)$, for every PFI I. Hence, $\texttt{uFUP}_{\text{app}}$ needs less space than `uFUP`. Our experiments, described next, show that $\texttt{uFUP}_{\text{app}}$ is highly efficient and accurate.

Tuple deletion. We now discuss briefly how tuple deletion can be handled in evolving databases. Suppose that a set of tuples $\delta \subseteq D$ is removed from D, resulting in database D^-. Inspired by Cheung et al. [12], we notice that an analogy to Theorem 3 can be deduced, with a similar proof.

Theorem 4. *Consider an item set I that is not a PFI in D. Then, I is a PFI in D^- only if $\mu^I(\delta) < \mu_m^+$, where $\mu_m^+ = \mu_m(D) - \mu_m(D^-)$.*

This can be used to handle tuple deletions efficiently, in a way analogous to the application of Theorem 3 in algorithm $\texttt{uFUP}_{\text{app}}$.

8 Results

We now present the experimental results on two data sets. The first one, called *accidents*, comes from the Frequent Item set Mining (FIMI) Data Set Repository.[1] This data set 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 data set contains 3,40,184 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 data set. The default value of $minsup$ is 20 percent. To test the incremental mining algorithms, we use the first $10k$ tuples as the old database D, and the subsequent tuples as the delta database d. The default size of d is 5 percent of D.

The second data set, called *T10I4D100k*, is produced by the IBM data generator.[2] The data set has a size n of 100k transactions. On average, each transaction has 10 items, and a frequent item set has four items. Since this data set is relatively sparse, we set $minsup$ to 1 percent. For the experiments on incremental mining algorithms, we use the first $90k$ tuples as D, and the remaining $10k$ tuples as d.

For both data sets, 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 value of $minprob$ is 0.4. In the results presented, $minsup$ is shown as a percentage of the data set 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 2 GB memory. The programs were written in C and compiled with Microsoft Visual Studio 2008.

1. http://fimi.cs.helsinki.fi/.
2. http://www.almaden.ibm.com/cs/disciplines/iis/.

TABLE 4
Recall and Precision of `MB`

$minsup$	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

Std. Dev.	0.125	0.25	$\sqrt{1/12}$	0.5	1.0
Recall	1	1	1	1	1
Precision	0.986	0.986	1	1	1

(d) Recall & Precision vs. Standard Deviation

We first present the results on the real data set. Section 8.1 describes the results for mining threshold-based PFIs for attribute-uncertain data. In Section 8.2, we present the results for incremental mining algorithms. We summarize the results for tuple-uncertain data and synthetic data, in Section 8.3.

8.1 Results on Threshold-Based PFI Mining

We now compare the performance of three PFI mining algorithms mentioned in this paper: 1) `DP`, the Apriori algorithm used in [6]; 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).

1. 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 [7], 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`. The recall and the precision of `MB`, relative to `DP`, are defined as follows:

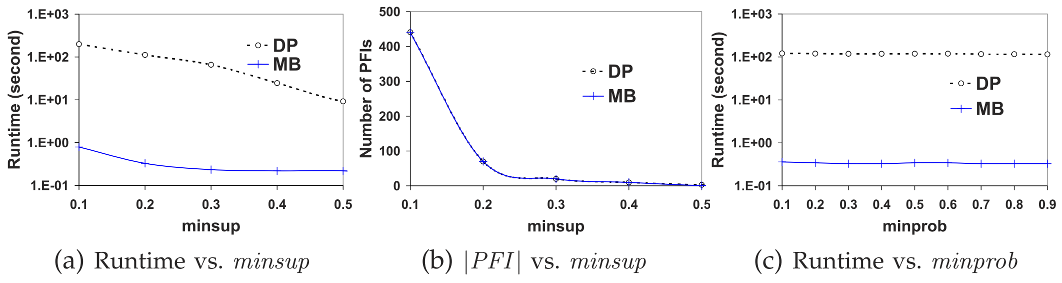
$$recall = \frac{|F_{DP} \cap F_{MB}|}{|F_{DP}|}, \tag{22}$$

$$precision = \frac{|F_{DP} \cap F_{MB}|}{|F_{MB}|}. \tag{23}$$

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

Table 4 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 percent. 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.

2. `MB` versus `DP`. Next, we compare the performance (in log scale) of `MB` and `DP`, in Fig. 6a. 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



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

Fig. 6. Efficiency of MB versus DP.

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 *minsup*. This is explained by Fig. 6b, 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 item sets 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.

Fig. 6c examines the performance of MB and DP (in log scale) over different *minprob* values. Their execution times drop by about 6 percent 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.

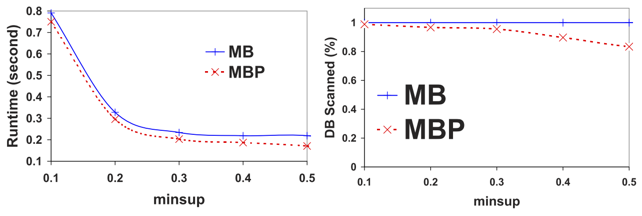
3. MB versus MBP. We then examine the benefit of using the improved PFI testing method (MBP) over the basic one (MB). Fig. 7a shows that MBP runs faster than MB over different *minsup* values. For instance, when $minsup = 0.5$, MBP addresses an improvement of 25 percent. Moreover, as *minsup* increases, the performance gap increases. This can be explained by Fig. 7b, 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 percent of the database. This reduces the I/O cost and the effort for interpreting the retrieved tuples. Thus, MBP performs better than MB.

4. Scalability. Fig. 8a 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 and 657.7 seconds, respectively; at $n = 100k$, MB finished in 3.1 seconds while DP spends 10 hours. Hence, the scalability of our approaches is better than that of DP.

5. Existential probability. We also examine the effect of using different distributions to characterize an attribute's probability, in Fig. 8b. We use Un to denote a uniform distribution, and G_i $(i = 0, \ldots, 5)$ to represent a Gaussian distribution. The details of these distributions are shown in Table 5. We observe that MB and MBP perform consistently better than DP over different distributions. All algorithms run comparatively slower on G_0. This is because G_0 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. Table 4d gives the accuracy for $G_1, \ldots, G_5$, which are Gaussian distributions with mean 0.5 and various standard deviations. We see that MB shows little variation in accuracy, which remains high (>98%), over the various distributions. We also found that the precision and 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.

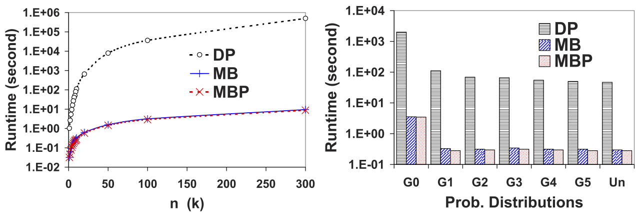
8.2 Results on Incremental PFI Mining

We now examine the performance of our incremental mining algorithms on attribute-uncertain data. For these algorithms, we assume that the PFIs for the old database D have already been obtained by some PFI mining algorithm, which can be used to discover the PFIs for the new database D^+. We compare them with the nonincremental counterparts, DP and MB. These algorithms are run directly on D^+ to obtain PFIs.



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

Fig. 7. Efficiency of MBP versus MB.

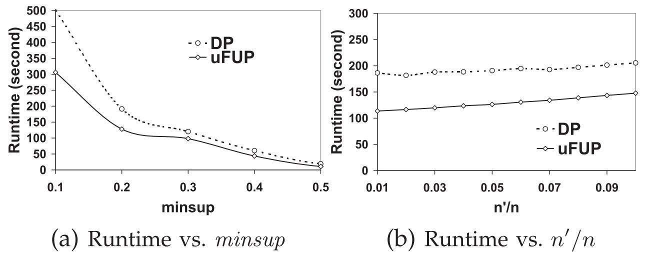


(a) Scalability (b) Prob. Distributions

Fig. 8. Other results for threshold-based PFIs.

TABLE 5
Existential Probability (Experiment (5))

Distribution	Mean	Standard Deviation
G_0	0.8	0.125
G_1(default)	0.5	0.125
G_2	0.5	0.25
G_3	0.5	$\sqrt{1/12} \approx 0.289$
G_4	0.5	0.5
G_5	0.5	1.0
Un	0.5	$\sqrt{1/12} \approx 0.289$

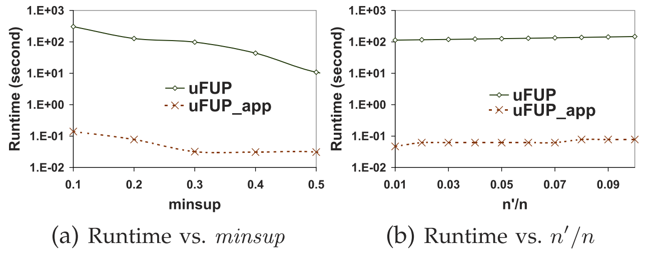


(a) Runtime vs. *minsup* (b) Runtime vs. n'/n

Fig. 9. Efficiency of uFUP versus DP.

6. uFUP versus DP. We first compare the performance of uFUP and DP. Notice that both methods produce exact threshold-based PFIs on D^+. Fig. 9a illustrates the result over different *minsup* values. We observe that uFUP is faster than DP. For example, at $minsup = 0.2$, the improvement is 37.5 percent. This is because uFUP does not generate PFIs from scratch; instead, it uses the PFIs of D to derive the new PFIs in D^+. Since most of the PFIs for D and D^+ are similar, only a few candidates need to be tested. Fig. 9b shows the performance of these algorithms over different sizes of the delta database (d), from 1 to 10 percent of the size of D. Their running times increase with d, since more effort needs to be spent on retrieving candidates from d. As we can see, uFUP is consistently better than DP; for instance, when n' is 5 percent of n, the improvement is 33 percent.

7. uFUP$_{app}$ versus MB. We next compare uFUP$_{app}$ and MB, which both yield approximate PFIs. Fig. 10a shows that uFUP$_{app}$ is faster than MB over different *minsup* values. For instance, at $minsup = 0.2$, uFUP$_{app}$ finished in only 0.078 seconds, giving an almost fivefold improvement over that of MB, which completes in 0.378 seconds. As we can see in Fig. 10b, uFUP$_{app}$ outperforms MB over different sizes of d. Fig. 10c examines the algorithms under a wide range of *minprob* values. Again, uFUP$_{app}$ runs faster than MB. Fig. 10d examines the effect of using different probability distributions on the attribute uncertainty model. The details are of these distributions are listed in Table 5. We can see that uFUP$_{app}$ performs better than MB over different types of distributions. The consistently high performance gain demonstrated by uFUP$_{app}$ can be explained by 1) the pruning method used by uFUP$_{app}$ removes many candidate item sets, so that only a few of them need to be tested; and 2) the old PFI results of D are effectively used, so that the time for scanning D^+ is significantly reduced.



(a) Runtime vs. *minsup* (b) Runtime vs. n'/n

Fig. 11. Efficiency of uFUP$_{app}$ versus uFUP.

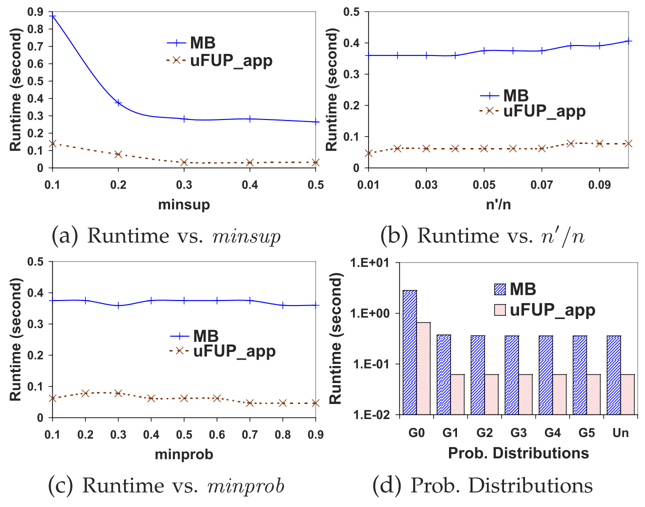
8. uFUP$_{app}$ versus uFUP. Fig. 11 compares uFUP$_{app}$ and uFUP over different values of *minsup* and n'/n. We can see that uFUP$_{app}$ performs better than uFUP, by two to three orders of magnitude. This shows that our way of adapting MB to devise an incremental mining algorithm (uFUP$_{app}$) is highly effective.

9. Accuracy. Table 6 compares the recall and the precision of uFUP$_{app}$ relative to that of uFUP. Here, we use (23) and (24); in particular, DP and MB are substituted by uFUP and uFUP$_{app}$, respectively. We can see that the recall and the precision values are always higher than 98 percent. We have also compared the accuracies for different existential probability distributions given in Table 5 and found that the standard deviation of Gaussian distribution has little effect on the accuracy. Hence, uFUP$_{app}$ can accurately maintain PFIs for evolving data.

8.3 Other Experiments

We have also performed experiments on the tuple uncertainty model and the synthetic data set. 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 data sets are still higher than 98 percent. Thus, our model-based approaches can return accurate results.

Tuple uncertainty. We compare the performance of DP, TODIS, MB, and MBP in Fig. 12a. Here, TODIS is proposed



(a) Runtime vs. *minsup* (b) Runtime vs. n'/n

(c) Runtime vs. *minprob* (d) Prob. Distributions

Fig. 10. Efficiency of uFUP$_{app}$ versus MB.

TABLE 6
Recall and Precision of uFUP$_{app}$

minsup	0.1	0.2	0.3	0.4	0.5
Recall	1	1	1	1	1
Precision	0.998	1	1	1	1

(a) Recall & Precision vs. *minsup*

minprob	0.1	0.3	0.5	0.7	0.9
Recall	1	1	1	0.970	1
Precision	0.986	1	1	1	1

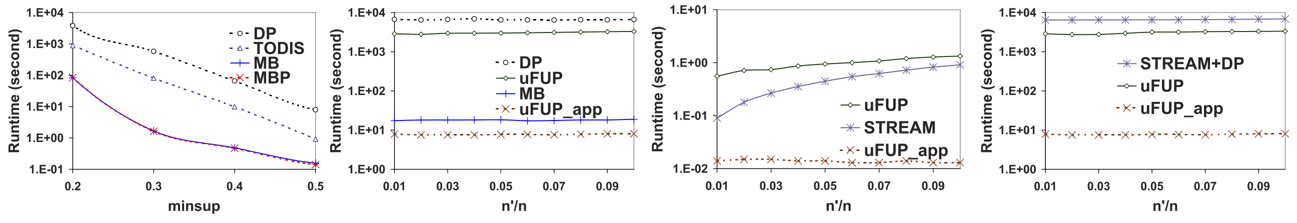
(b) Recall & Precision vs. *minprob*

n	1k	5k	10k	50k	100k
Recall	1	1	1	1	1
Precision	1	1	1	1	0.985

(c) Recall & Precision vs. n

n'/n	0.01	0.03	0.05	0.07	0.09
Recall	1	1	1	1	1
Precision	1	1	1	1	1

(d) Recall & Precision vs. n'



(a) Runtime vs. $minsup$ (b) Runtime vs. n'/n (c) Mining 1-PFIs vs. STREAM (d) Comparison with STREAM+DP

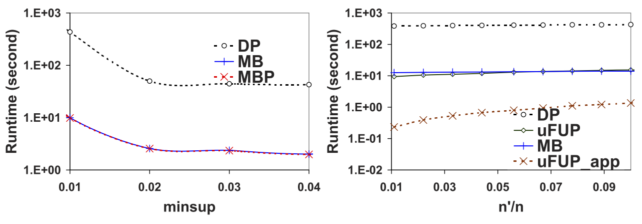
Fig. 12. Tuple uncertainty.

in [30], for retrieving exact threshold-based PFIs from tuple-uncertain data. We can see that both MB and MBP perform much better than DP and TODIS, under different $minsup$ values. When $minsup = 0.3$, MB needs 1.6 seconds, but DP and TODIS complete in 581 and 81 seconds, respectively. Fig. 12b compares the algorithms under different sizes of d. Similar to the results for attribute uncertainty, uFUP (uFUP_{app}) performs better than DP (respectively MB). Moreover, uFUP_{app} outperforms uFUP by more than three orders of magnitude. Hence, our algorithms also work well for tuple-uncertain databases.

In [35], an algorithm for finding *heavy hitters* from probabilistic data streams was proposed. We develop a variant of that algorithm, which we call STREAM, as another exact incremental mining algorithm for finding 1-PFIs. Fig. 12c studies the performance of incremental mining algorithms for finding 1-PFIs. STREAM performs better than uFUP, because STREAM maintains the s-pmfs for all candidate 1-PFIs, which can be updated easily upon the arrival of new transactions. On the other hand, uFUP only keeps the s-pmfs of 1-PFIs of D. For new candidate PFIs that appear in D^+ but not D, uFUP has to compute their s-pmfs by scanning D^+, which can be costly. Observe that uFUP_{app} is much faster than STREAM, since it does not compute the exact s-pmf information. We further found that the 1-PFIs returned by uFUP_{app} are the same as those generated by STREAM. We remark that while STREAM only returns 1-PFIs, both uFUP and uFUP_{app} can generate PFIs of any size.

Therefore, we have designed STREAM+DP, which first finds 1-PFIs with STREAM and then feeds the 1-PFIs to DP to find all other PFIs. Fig. 12d shows that STREAM+DP is not as efficient as uFUP nor uFUP_{app}. The reason is that finding 1-PFIs constitutes only a small portion of time in finding all PFIs. Although STREAM performs well in finding 1-PFIs, having to find the remaining PFIs with DP makes STREAM+DP inefficient.

Synthetic data set. Finally, we test our algorithms on a synthetic data set. Fig. 13a compares the performance of MB, MBP, and DP, for the attribute uncertainty model. We found that MB and MBP outperform DP. Fig. 13b compares the performance of DP, MB, uFUP, and uFUP_{app} for tuple uncertainty. We can see that the incremental mining algorithms perform better than their nonincremental counterparts. We also observe that uFUP_{app} runs faster than uFUP, by more than one order of magnitude. Hence, our model-based incremental mining algorithm also works well for this data set.



(a) Runtime vs. $minsup$ (b) Runtime vs. n'/n

Fig. 13. Synthetic data.

9 Conclusions

In this paper, we propose a model-based approach to extract threshold-based PFIs from large uncertain databases. Its main idea is to approximate the s-pmf of a PFI by some common probability model, so that a PFI can be verified quickly. We also study two incremental mining algorithms for retrieving PFIs from evolving databases. Our experimental results show that these algorithms are highly efficient and accurate. They support both attribute- and tuple-uncertain data. We will examine how to use the model-based approach to develop other mining algorithms (e.g., clustering and classification) on uncertain data. It is also interesting to study efficient mining algorithms for handling tuple updates and deletion. Another interesting work is to investigate PFI mining algorithms for probability models that capture correlation among attributes and tuples.

References

[1] A. Veloso, W. Meira Jr., M. de Carvalho, B. Pôssas, S. Parthasarathy, and M.J. Zaki, "Mining Frequent Itemsets in Evolving Databases," *Proc. Second SIAM Int'l Conf. Data Mining (SDM)*, 2002.
[2] C. Aggarwal, Y. Li, J. Wang, and J. Wang, "Frequent Pattern Mining with Uncertain Data," *Proc. 15th ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining (KDD)*, 2009.
[3] C. Aggarwal and P. Yu, "A Survey of Uncertain Data Algorithms and Applications," *IEEE Trans Knowledge and Data Eng.*, vol. 21, no. 5, pp. 609-623, May 2009.
[4] R. Agrawal, T. Imieliński, and A. Swami, "Mining Association Rules between Sets of Items in Large Databases," *Proc. ACM SIGMOD Int'l Conf. Management of Data*, 1993.
[5] O. Benjelloun, A.D. Sarma, A. Halevy, and J. Widom, "ULDBs: Databases with Uncertainty and Lineage," *Proc. 32nd Int'l Conf. Very Large Data Bases (VLDB)*, 2006.
[6] T. Bernecker, H. Kriegel, M. Renz, F. Verhein, and A. Zuefle, "Probabilistic Frequent Itemset Mining in Uncertain Databases," *Proc. 15th ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining (KDD)*, 2009.
[7] C.J. van Rijsbergen, *Information Retrieval.* Butterworth, 1979.
[8] L.L. Cam, "An Approximation Theorem for the Poisson Binomial Distribution," *Pacific J. Math.*, vol. 10, pp. 1181-1197, 1960.
[9] H. Cheng, P. Yu, and J. Han, "Approximate Frequent Itemset Mining in the Presence of Random Noise," *Proc. Soft Computing for Knowledge Discovery and Data Mining*, pp. 363-389, 2008.
[10] R. Cheng, D. Kalashnikov, and S. Prabhakar, "Evaluating Probabilistic Queries over Imprecise Data," *Proc. ACM SIGMOD Int'l Conf. Management of Data*, 2003.

[11] D. Cheung, J. Han, V. Ng, and C. Wong, "Maintenance of Discovered Association Rules in Large Databases: An Incremental Updating Technique," *Proc. 12th Int'l Conf. Data Eng. (ICDE)*, 1996.

[12] D. Cheung, S.D. Lee, and B. Kao, "A General Incremental Technique for Maintaining Discovered Association Rules," *Proc. Fifth Int'l Conf. Database Systems for Advanced Applications (DASFAA)*, 1997.

[13] W. Cheung and O.R. Zaïane, "Incremental Mining of Frequent Patterns without Candidate Generation or Support Constraint," *Proc. Seventh Int'l Database Eng. and Applications Symp. (IDEAS)*, 2003.

[14] C.K. Chui, B. Kao, and E. Hung, "Mining Frequent Itemsets from Uncertain Data," *Proc. 11th Pacific-Asia Conf. Advances in Knowledge Discovery and Data Mining (PAKDD)*, 2007.

[15] G. Cormode and M. Garofalakis, "Sketching Probabilistic Data Streams," *Proc. ACM SIGMOD Int'l Conf. Management of Data*, 2007.

[16] N. Dalvi and D. Suciu, "Efficient Query Evaluation on Probabilistic Databases," *Proc. 13th Int'l Conf. Very Large Data Bases (VLDB)*, 2004.

[17] A. Deshpande, C. Guestrin, S. Madden, J. Hellerstein, and W. Hong, "Model-Driven Data Acquisition in Sensor Networks," *Proc. 13th Int'l Conf. Very Large Data Bases (VLDB)*, 2004.

[18] J. Han, J. Pei, and Y. Yin, "Mining Frequent Patterns without Candidate Generation," *Proc. ACM SIGMOD Int'l Conf. Management of Data*, 2000.

[19] J. Huang, "MayBMS: A Probabilistic Database Management System," *Proc. 35th ACM SIGMOD Int'l Conf. Management of Data*, 2009.

[20] R. Jampani, L. Perez, M. Wu, F. Xu, C. Jermaine, and P. Haas, "MCDB: A Monte Carlo Approach to Managing Uncertain Data," *Proc. ACM SIGMOD Int'l Conf. Management of Data*, 2008.

[21] J. Ren, S.D. Lee, X. Chen, B. Kao, R. Cheng, and D.W. Cheung, "Naive Bayes Classification of Uncertain Data," *Proc. IEEE Ninth Int'l Conf. Data Mining (ICDM)*, 2009.

[22] N. Khoussainova, M. Balazinska, and D. Suciu, "Towards Correcting Input Data Errors Probabilistically Using Integrity Constraints," *Proc. Fifth ACM Int'l Workshop Data Eng. for Wireless and Mobile Access (MobiDE)*, 2006.

[23] H. Kriegel and M. Pfeifle, "Density-Based Clustering of Uncertain Data," *Proc. ACM SIGKDD Int'l Conf. Knowledge Discovery in Data Mining (KDD)*, 2005.

[24] C. Kuok, A. Fu, and M. Wong, "Mining Fuzzy Association Rules in Databases," *SIGMOD Record*, vol. 27, no. 1, pp. 41-46, 1998.

[25] C.K.-S. Leung, Q.I. Khan, and T. Hoque, "Cantree: A Tree Structure for Efficient Incremental Mining of Frequent Patterns," *Proc. IEEE Fifth Int'l Conf. Data Mining (ICDM)*, 2005.

[26] A. Lu, Y. Ke, J. Cheng, and W. Ng, "Mining Vague Association Rules," *Proc. 12th Int'l Conf. Database Systems for Advanced Applications (DASFAA)*, 2007.

[27] M. Mutsuzaki, "Trio-One: Layering Uncertainty and Lineage on a Conventional DBMS," *Proc. Third Biennial Conf. Innovative Data Systems Research (CIDR)*, 2007.

[28] P. Sistla, O. Wolfson, S. Chamberlain, and S. Dao, "Querying the Uncertain Position of Moving Objects," *Temporal Databases: Research and Practice*, Springer Verlag, 1998.

[29] C. Stein, *Approximate Computation of Expectations*, Lecture Notes - Monograph Series, vol. 7, Inst. of Math. Statistics, 1986.

[30] L. Sun, R. Cheng, D.W. Cheung, and J. Cheng, "Mining Uncertain Data with Probabilistic Guarantees," *Proc. 16th ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining*, 2010.

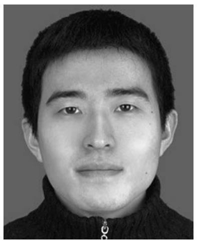
[31] T. Jayram et al., "Avatar Information Extraction System," *IEEE Data Eng. Bull.*, vol. 29, no. 1, pp. 40-48, Mar. 2006.

[32] S. Tsang, B. Kao, K.Y. Yip, W.-S. Ho, and S.D. Lee., "Decision Trees for Uncertain Data," *Proc. IEEE Int'l Conf. Data Eng. (ICDE)*, 2009.

[33] L. Wang, R. Cheng, S.D. Lee, and D. Cheung, "Accelerating Probabilistic Frequent Itemset Mining: A Model-Based Approach," *Proc. 19th ACM Int'l Conf. Information and Knowledge Management (CIKM)*, 2010.

[34] M. Yiu, N. Mamoulis, X. Dai, Y. Tao, and M. Vaitis, "Efficient Evaluation of Probabilistic Advanced Spatial Queries on Existentially Uncertain Data," *IEEE Trans Knowledge and Data Eng.*, vol. 21, no. 9, pp. 108-122, Jan. 2009.

[35] Q. Zhang, F. Li, and K. Yi, "Finding Frequent Items in Probabilistic Data," *Proc. ACM SIGMOD Int'l Conf. Management of Data*, 2008.

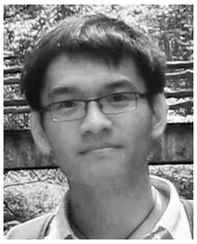


Liang Wang received the BEng degree in computer science from Shanghai Jiaotong University and the MPhil degree majoring in computer science from the University of Hong Kong in 2008 and 2011, respectively. His research interests include uncertain database, data mining, and data management. Now, he works as a software engineer at Microsoft Corporation.

David Wai-Lok Cheung received the MSc and PhD degrees in computer science from Simon Fraser University, Canada, in 1985 and 1989, respectively. Since 1994, he has been a faculty member in the Department of Computer Science at the University of Hong Kong. His research interests include database, data mining, database security and privacy. He was the program committee chairman of PAKDD 2001, program cochair of PAKDD 2005, conference chair of PAKDD 2007 and 2011, conference cochair of CIKM 2009, and conference cochair of PAKDD 2011.

Reynold Cheng received the BEng degree in computer engineering and the MPhil degree in computer science and information systems from the University of Hong Kong (HKU), in 1998 and 2000, respectively, and the MSc and PhD degrees from the Department of Computer Science, Purdue University, in 2003 and 2005, respectively. He is an assistant professor in the Department of Computer Science at HKU. He was the recipient of the 2010 Research Output Prize in the Department of Computer Science at HKU. From 2005 to 2008, he was an assistant professor in the Department of Computing at Hong Kong Polytechnic University, where he received two Performance Awards. He has served on the program committees and review panels for leading database conferences and journals. He is also a guest editor for a special issue in *IEEE Transactions on Knowledge and Data Engineering*. His research interests include database management as well as querying and mining of uncertain data. He is a member of the IEEE, the ACM, ACM SIGMOD, and UPE.

Sau Dan Lee received the BSc and MPhil degrees from the University of Hong Kong, in 1995 and 1998, respectively, and the PhD degree from the University of Freiburg, Germany, in 2006. He is a postdoctoral fellow at the University of Hong Kong. He is interested in the research areas of data mining, machine learning, uncertain data management and information management on the WWW. He has also designed and developed backend software systems for e-Business and investment banking.



Xuan S. Yang received the BSc degree in computer science from Fudan University in 2009. Currently, he is working toward the PhD degree at the University of Hong Kong under the supervision of Dr. Reynold Cheng and Prof. David Cheung. His research interests include uncertain data management, data cleaning, and web data mining.