Mining Uncertain Data with Probabilistic Guarantees

Liwen Sun  Reynold Cheng  David W. Cheung  Jiefeng Cheng

Department of Computer Science
The University of Hong Kong
Pokfulam Road, Hong Kong
{lwsun, ckcheng, dcheung, jfcheng}@cs.hku.hk

ABSTRACT
Data uncertainty is inherent in applications such as sensor monitoring systems, location-based services, and biological databases. To manage this vast amount of imprecise information, probabilistic databases have been recently developed. In this paper, we study the discovery of frequent patterns and association rules from probabilistic data under the Possible World Semantics. This is technically challenging, since a probabilistic database can have an exponential number of possible worlds. We propose two efficient algorithms, which discover frequent patterns in bottom-up and top-down manners. Both algorithms can be easily extended to discover maximal frequent patterns. We also explain how to use these patterns to generate association rules. Extensive experiments, using real and synthetic datasets, were conducted to validate the performance of our methods.

Source codes and data are available at: http://www.cs.hku.hk/~lwsun/codes/kdd10/

Categories and Subject Descriptors
H.2.8 [Information Systems]: Database applications—Data Mining

General Terms
Algorithms, Experimentation, Performance, Theory

Keywords
uncertain data, frequent pattern, association rule

1. INTRODUCTION
The data managed in many emerging applications is often uncertain. Integration and record linkage tools, for example, 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 [28]. In habitat monitoring systems, data collected from sensors like temperature and humidity are noisy [1]. The locations of users obtained through RFID and GPS systems are also imprecise [25, 16]. To handle these problems, probabilistic databases have been recently proposed, where uncertainty is treated as a “first-class citizen” [8, 11, 21, 10, 15].

Due to its simplicity in database design and query semantics, the tuple-uncertainty model is commonly used in probabilistic databases. Conceptually, each tuple carries an existential probability attribute, which denotes the confidence that the tuple exists. Figure 1 illustrates this model, which records traffic violation events due to red-light running. The details of each event (e.g., location, and traffic volume) are captured by a red-light camera system, which contains sensors and cameras mounted in road intersections. Each tuple is annotated by a probability that a true violation happens. The probability that a violation occurs is determined by sensor measurement errors, as well as the uncertainty caused by automatic information extraction of the photographs taken by the system [30].

To interpret tuple uncertainty, 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 zero or more tuples. A possible world for Figure 1 consists of the tuples \{t_2, t_3, t_5\}, existing with a probability of \((1 - 0.1) \times 1.0 \times 0.5 \times (1 - 0.2) \times 1.0 = 0.036\). Any query evaluation algorithm for a probabilistic 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, evaluating queries under this notion is costly. This is because a probabilistic database has an exponential number of possible worlds. For example, the table in Figure 1 has \(2^3 = 8\) possible worlds. Performing query evaluation or data mining under PWS can thus be technically challenging. In fact, the mining of uncertain or probabilistic data has recently attracted research attention [3]. In [18], efficient clustering algorithms were developed to group uncertain objects that are close to each other.
other. Recently, a Naïve Bayes classifier has been developed [26]. The goals of this paper are: (1) propose a definition of frequent patterns and association rules for the tuple uncertainty model; and (2) develop efficient algorithms for mining patterns and rules that are correct under PWS.

The frequent patterns discovered from probabilistic data are also probabilistic, to reflect the confidence placed on the mining results. Figure 1 shows two probabilistic frequent patterns (or p-FPs) extracted from the database in Figure 1. A p-FP is a set of attribute values that occur frequently with sufficiently-high probabilities. In Figure 1, the support probability mass function (or support pmf) for each p-FP is shown. This is the pmf of the number of tuples (or support count) that contains a pattern. Under PWS, a database is a set of possible worlds, each of which records a (different) support of the same pattern. Hence, the support of a frequent pattern is a pmf. In Figure 1, if we consider all possible worlds where pattern \{location=x\} occurs three times, the pmf of \{location=x\} with a support of 3 is 0.49. For the p-FPs shown, Figure 3 displays their related probabilistic association rules (or p-ARs). Here, rule r_2 suggests that with a 0.49 probability, 1) red-light violations occur frequently at location x and 2) when this happens, the involved vehicle is likely driving at a high speed amid low traffic. We will later explain more about the semantics of p-FP and p-AR.

A simple way of finding p-FPs is to extract frequent patterns from every possible world. This is practically infeasible, since the number of possible worlds is exponentially large. We present simple and effective methods to prune infrequent patterns. We also adopt dynamic programming (DP) to compute the support pmf of a pattern. This method, also used by [31, 27] to derive p-FPs for other probabilistic models, has a complexity of \(O(n^3)\). We further develop a divide-and-conquer (DC) approach, which achieves \(O(n \log^2 n)\) time. Based on these methods, we propose the p-Apriori algorithm to retrieve p-FPs in a bottom-up manner. We further observe that, given patterns X and Y, if X is a subset of Y, the support pmf of X can be efficiently derived from that of Y. We realize this by proposing the TODIS algorithm, which extracts p-FPs in a top-down fashion. We extend p-Apriori and TODIS to retrieve maximal p-FPs, i.e., those that are not the subsets of other p-FPs. Our experiments on real and synthetic datasets showed that DC is more scalable than DP, and TODIS is generally faster than p-Apriori. When TODIS is used with DC together, it can outperform p-Apriori by an order of magnitude.

In exact databases, deriving association rules from frequent patterns is not difficult. Given two frequent patterns X and XY, the confidence of \(X \Rightarrow Y\) can be calculated with an arithmetic division on their supports. This is no longer true for probabilistic data. Here, the support of X and XY become correlated random variables. It is not clear how to define and compute the "confidence" of \(X \Rightarrow Y\). We propose the concept of p-AR, which naturally extends the association rule semantics. We also study an efficient method, which uses deconvolution operations, to evaluate the probability of p-AR based on the support pmfs of p-FPs. We further develop a new algorithm for generating p-ARs, which exploits the anti-monotonicity property of p-ARs, and attain a 80% performance gain in our experiments.

Prior work. [7] studied approximate frequent patterns on noisy data, while [19] examined association rules on fuzzy sets. The notion of a "vague association rule" was developed in [20]. These solutions were not developed on probabilistic data models. For probabilistic databases, [9, 2] derived patterns based on their expected support counts. [31, 27] found that the use of expected support may render important patterns missing. They discussed the computation of the probability that a pattern is frequent. While [31] handled the mining of single items, their solution can discover patterns with more than one item. The data model used in [27] assumes that for each tuple, each attribute value has a probability of being correct. This is different from the tuple-uncertainty model, which describes the joint probability of attribute values within a tuple. Moreover, their DC algorithm is asymptotically faster than the DP algorithms used in [31, 27], and is thus more scalable for large and dense datasets. To our best knowledge, none of the above works considered the important problem of generating association rules on probabilistic databases.

This paper is organized as follows. Section 2 introduces the notions of p-FPs and p-ARs. Sections 3 and 4 present two algorithms for mining p-FPs. In Section 5, we develop an algorithm for generating p-ARs. Section 6 presents our experimental results. We conclude in Section 7.

2. PROBLEM DEFINITION

We first review frequent patterns and association rules in Sections 2.1. Then, we discuss the uncertain data model in Section 2.2. We present the problems of mining p-FPs and p-ARs, in Sections 2.3 and 2.4.

2.1 Frequent Patterns and Association Rules

A transaction is a set of items (e.g., goods bought by a customer in a supermarket). A set of items is also called an itemset or a pattern. Given a transaction database of size n and a pattern X, we use \(\text{sup}(X)\) to denote the support of X, i.e., the number of times that X appears in the database. A pattern X is frequent if:

\[
\text{sup}(X) \geq \text{minsup}
\]

where \(\text{minsup} \in \mathcal{N} \cap [1, n]\) is the support threshold [4]. Given patterns X and Y (with \(X \cap Y = \emptyset\)), if pattern XY is frequent, then X is also frequent (called the anti-monotonicity property). Also, \(X \Rightarrow Y\) is an association rule if the follow-

\[
\text{support of the same pattern. Hence, the support of a pattern} \quad \text{is a set of possible worlds, each of which records a (different) support of the same pattern. Hence, the support of a frequent pattern is a pmf. In Figure 1, if we consider all possible worlds where pattern \{location=x\} occurs three times, the pmf of \{location=x\} with a support of 3 is 0.49. For the p-FPs shown, Figure 3 displays their related probabilistic association rules (or p-ARs). Here, rule r_2 suggests that with a 0.49 probability, 1) red-light violations occur frequently at location x and 2) when this happens, the involved vehicle is likely driving at a high speed amid low traffic. We will later explain more about the semantics of p-FP and p-AR.

A simple way of finding p-FPs is to extract frequent patterns from every possible world. This is practically infeasible, since the number of possible worlds is exponentially large. We present simple and effective methods to prune infrequent patterns. We also adopt dynamic programming (DP) to compute the support pmf of a pattern. This method, also used by [31, 27] to derive p-FPs for other probabilistic models, has a complexity of \(O(n^3)\). We further develop a divide-and-conquer (DC) approach, which achieves \(O(n \log^2 n)\) time. Based on these methods, we propose the p-Apriori algorithm to retrieve p-FPs in a bottom-up manner. We further observe that, given patterns X and Y, if X is a subset of Y, the support pmf of X can be efficiently derived from that of Y. We realize this by proposing the TODIS algorithm, which extracts p-FPs in a top-down fashion. We extend p-Apriori and TODIS to retrieve maximal p-FPs, i.e., those that are not the subsets of other p-FPs. Our experiments on real and synthetic datasets showed that DC is more scalable than DP, and TODIS is generally faster than p-Apriori. When TODIS is used with DC together, it can outperform p-Apriori by an order of magnitude.

In exact databases, deriving association rules from frequent patterns is not difficult. Given two frequent patterns X and XY, the confidence of \(X \Rightarrow Y\) can be calculated with an arithmetic division on their supports. This is no longer true for probabilistic data. Here, the support of X and XY become correlated random variables. It is not clear how to define and compute the “confidence” of \(X \Rightarrow Y\). We propose the concept of p-AR, which naturally extends the association rule semantics. We also study an efficient method, which uses deconvolution operations, to evaluate the probability of p-AR based on the support pmfs of p-FPs. We further develop a new algorithm for generating p-ARs, which exploits the anti-monotonicity property of p-ARs, and attain a 80% performance gain in our experiments.

Prior work. [7] studied approximate frequent patterns on noisy data, while [19] examined association rules on fuzzy sets. The notion of a “vague association rule” was developed in [20]. These solutions were not developed on probabilistic data models. For probabilistic databases, [9, 2] derived patterns based on their expected support counts. [31, 27] found that the use of expected support may render important patterns missing. They discussed the computation of the probability that a pattern is frequent. While [31] handled the mining of single items, their solution can discover patterns with more than one item. The data model used in [27] assumes that for each tuple, each attribute value has a probability of being correct. This is different from the tuple-uncertainty model, which describes the joint probability of attribute values within a tuple. Moreover, their DC algorithm is asymptotically faster than the DP algorithms used in [31, 27], and is thus more scalable for large and dense datasets. To our best knowledge, none of the above works considered the important problem of generating association rules on probabilistic databases.

This paper is organized as follows. Section 2 introduces the notions of p-FPs and p-ARs. Sections 3 and 4 present two algorithms for mining p-FPs. In Section 5, we develop an algorithm for generating p-ARs. Section 6 presents our experimental results. We conclude in Section 7.

2. PROBLEM DEFINITION

We first review frequent patterns and association rules in Sections 2.1. Then, we discuss the uncertain data model in Section 2.2. We present the problems of mining p-FPs and p-ARs, in Sections 2.3 and 2.4.

2.1 Frequent Patterns and Association Rules

A transaction is a set of items (e.g., goods bought by a customer in a supermarket). A set of items is also called an itemset or a pattern. Given a transaction database of size n and a pattern X, we use \(\text{sup}(X)\) to denote the support of X, i.e., the number of times that X appears in the database. A pattern X is frequent if:

\[
\text{sup}(X) \geq \text{minsup}
\]

where \(\text{minsup} \in \mathcal{N} \cap [1, n]\) is the support threshold [4]. Given patterns X and Y (with \(X \cap Y = \emptyset\)), if pattern XY is frequent, then X is also frequent (called the anti-monotonicity property). Also, \(X \Rightarrow Y\) is an association rule if the follow-
ing conditions hold:

\[ \text{sup}(XY) \geq \text{minsup} \tag{2} \]

\[ \frac{\text{sup}(XY)}{\text{sup}(X)} \geq \text{minconf} \tag{3} \]

where \(\frac{\text{sup}(XY)}{\text{sup}(X)}\), denoted by \(\text{conf}(X \Rightarrow Y)\), is the confidence of \(X \Rightarrow Y\), and \(\text{minconf} \in \mathbb{R} \cap (0, 1]\) is the confidence threshold. To verify Equation 3, the values of \(\text{sup}(XY)\) and \(\text{sup}(X)\) have to be found first.

We remark that a transaction database is essentially a relational table with asymmetric binary attribute values. For example, the existence of item “apple” in a transaction is considered by some mining algorithms (e.g., [4, 5]). To handle other attribute types (e.g., continuous and categorical), discretization and binarization techniques can be used to convert them to binary attributes [29].

### 2.2 The Possible World Semantics

We assume that each transaction has an existential probability, which specifies the chance that the transaction exists. Figure 4(a) illustrates this database, in which each transaction is a set of items represented by letters. This model has been used to capture uncertainty in many applications, including data streams [10] and geographical services [22].  

Now, let \(P(E)\) be the probability that an event \(E\) occurs, and \(\mathcal{PDB}\) be a probabilistic database of size \(n\). Also, let \(T_i\) (where \(i = 1, \ldots, n\)) be the ID of each tuple in \(\mathcal{PDB}\). Suppose \(T_i.S\) is the set of items contained in \(T_i\), and \(T_i.p\) is the existential probability of \(T_i\).

**Figure 4:** (a) \(\mathcal{PDB}\) example; and (b) \(f_X\)

### 2.3 Probabilistic Frequent Patterns

We first explain the concept of “support” for probabilistic data. Given a pattern \(X\), we denote its support in each world \(W_i\) as \(\text{sup}_i(X)\). Note that \(\text{sup}_i(X)\) is obtained by counting the number of times \(X\) appears in \(W_i\). Since each \(W_i\) exists with a probability, the support of \(X\) in \(\mathcal{PDB}\), i.e., \(\text{sup}(X)\), is a random variable. We denote by \(f_X(k)\) the probability mass function (pmf) of \(\text{sup}(X)\), where \(k\) is a non-negative integer that \(\text{sup}(X)\) can take. Specifically, \(f_X(k)\) is the probability that \(\text{sup}(X) = k\), and \(f_X(0) = 0\) for any \(k \notin [0, n]\). We use an array to store the non-zero values of \(f_X\), where \(f_X[k] = P(\text{sup}(X) = k)\). Figure 4(b) depicts the support pmf of \(\{a\}\) for Figure 5. The probability that \(\text{sup}(\{a\}) = 1\) is 0.29.

**Definition 1.** A pattern \(X\) is a probabilistic frequent pattern (or p-FP) in \(\mathcal{PDB}\) if

\[ P(\text{sup}(X) \geq \text{minsup} \geq \text{minprob} \tag{4} \]

where \(\text{minprob} \in \mathbb{R} \cap (0, 1]\) is the probability threshold.

**Problem 1 (p-FP Mining).** Given \(\mathcal{PDB}\), \text{minsup} and \text{minprob}, return a set of \(\{X, f_X(k)\}\), where \(X\) is a p-FP.

As we will discuss, the pmfs obtained with p-FPs are essential to generating probabilistic association rules. There are methods to approximating and compressing pmfs (e.g., see [12]). Here we assume that the pmf is exact, but our solutions can be extended to support these schemes. Next, we present a useful lemma.

**Lemma 1 (Anti-monotonicity).** If pattern \(X\) is a p-FP, then any pattern \(X' \subset X\) is also a p-FP.

**Proof.** Let \(X'\) be a sub-pattern of \(X\). Suppose that \(W_X^X\) and \(W_X^{X'}\) are the sets of possible worlds where \(X\) and \(X'\) are frequent respectively. For each possible world \(W_i \in W_X^{X'}\), if \(X\) is frequent in \(W_i\), then \(X'\) is also frequent. Hence, we have \(W_X^X \subseteq W_X^{X'}\). Then \(P(\text{sup}(X) \geq \text{minsup}) = \sum_{W_i \in W_X^{X'}} P(W_i)\),

---

**Table 1: Summary of Notations**

<table>
<thead>
<tr>
<th>Notation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\mathcal{PDB})</td>
<td>a probabilistic database of size (n)</td>
</tr>
<tr>
<td>(T_i)</td>
<td>ID of a tuple in (\mathcal{PDB}), where (i = 1, \ldots, n)</td>
</tr>
<tr>
<td>(T_i.S)</td>
<td>set of items contained in (T_i)</td>
</tr>
<tr>
<td>(T_i.p)</td>
<td>existential probability of (T_i)</td>
</tr>
<tr>
<td>(W)</td>
<td>set of possible worlds expanded from (\mathcal{PDB})</td>
</tr>
<tr>
<td>(W_i)</td>
<td>a possible world, where (W_i \in W)</td>
</tr>
<tr>
<td>(P(E))</td>
<td>probability of event (E)</td>
</tr>
<tr>
<td>(\text{sup})</td>
<td>support</td>
</tr>
<tr>
<td>(\text{minsup})</td>
<td>support threshold</td>
</tr>
<tr>
<td>(\text{minprob})</td>
<td>probability threshold</td>
</tr>
<tr>
<td>(f_X(k))</td>
<td>support pmf of (X) in (\mathcal{PDB})</td>
</tr>
<tr>
<td>(\text{cnt}_X)</td>
<td>No. of tuples for which (\mu_i &gt; 0)</td>
</tr>
<tr>
<td>(\text{exp}_{\mathcal{PDB}}(X))</td>
<td>expected support of pattern (X)</td>
</tr>
<tr>
<td>(P_X^X)</td>
<td>prob. that (X) occurs in (T_i)</td>
</tr>
<tr>
<td>(L_X^X)</td>
<td>inverted probability list of (X)</td>
</tr>
<tr>
<td>(X.\text{exItem})</td>
<td>exclusive item of (X)</td>
</tr>
<tr>
<td>(X.\text{cnt})</td>
<td>length of (L_X^X)</td>
</tr>
</tbody>
</table>

---

**Figure 5:** Possible worlds for Figure 4(a)

<table>
<thead>
<tr>
<th>(W)</th>
<th>Tuples in (W)</th>
<th>Prob.</th>
<th>(W)</th>
<th>Tuples in (W)</th>
<th>Prob.</th>
</tr>
</thead>
<tbody>
<tr>
<td>(W_1)</td>
<td>(T_1)</td>
<td>0.06</td>
<td>(W_5)</td>
<td>(T_1, T_2, T_3)</td>
<td>0.09</td>
</tr>
<tr>
<td>(W_2)</td>
<td>(T_4)</td>
<td>0.09</td>
<td>(W_6)</td>
<td>(T_1, T_5, T_4)</td>
<td>0.21</td>
</tr>
<tr>
<td>(W_3)</td>
<td>(T_5, T_4)</td>
<td>0.14</td>
<td>(W_7)</td>
<td>(T_1, T_2, T_3, T_4)</td>
<td>0.21</td>
</tr>
</tbody>
</table>

Under PWS, \(\mathcal{PDB}\) is a set of possible worlds \(W\). Figure 5 lists all possible worlds for Figure 4(a). Each world \(W_i \in W\) exists with probability \(P(W_i)\). For example, \(P(W_2) = T_2.p \times (1 - T_2.p) \times T_3.p \times (1 - T_3.p) \times T_5.p\) or 0.09. The sum of possible world probabilities is one. Also, the number of possible worlds is exponentially large, i.e., \(|W| = O(2^n)\). Our goal is to discover patterns and rules without expanding \(\mathcal{PDB}\) into possible worlds. Table 1 summarizes the symbols used.
and \( P(\text{sup}(X') \geq \text{minsup}) = \sum_{W \in \mathcal{N}X'} P(W) \). Since \( X \) is a p-FP, we have: \( P(\text{sup}(X') \geq \text{minsup}) \geq P(\text{sup}(X) \geq \text{minsup}) \geq \text{minprob} \). Therefore, \( X' \) is also a p-FP.

The anti-monotonicity property is true for frequent patterns in exact data [4]. Lemma 1 allows us to stop examining a pattern, if any of its sub-pattern is not a p-FP. A p-FP \( X \) is said to be maximal if we cannot find another p-FP \( Y \) such that \( X \subseteq Y \). A maximal p-FP can succinctly represent a set of p-FPs when their supports are not concerned. Since the mining of maximal frequent patterns is an important problem [5] for exact data, we also study maximal p-FPs:

**Problem 2 (Maximal p-FP Mining).** Given a database \( \mathcal{PD}_{\mathcal{B}} \), \text{minsup} and \text{minprob}, return all maximal p-FPs.

### 2.4 Probabilistic Association Rules

In a probabilistic database, the support counts of patterns are random variables. Let \( P(X \Rightarrow Y) \) be the probability that \( X \Rightarrow Y \) is an association rule. By Equations 2 and 3, we have:

\[
P(X \Rightarrow Y) = P[\text{sup}(XY) > \text{minsup} \land \text{conf}(X \Rightarrow Y) \geq \text{minconf}]
\]  
(5)

**Definition 2.** \( X \Rightarrow Y \) is a probabilistic association rule (p-AR in short) if

\[
P(X \Rightarrow Y) \geq \text{minprob}
\]  
(6)

The problem of p-AR mining is defined as follows.

**Problem 3 (p-AR Mining).** Given \text{minsup}, \text{minprob}, \text{minconf}, and the p-FPs and their support pmfs obtained from Problem 1, derive all p-ARs and their probabilities.

A simple way of solving Problems 1, 2 and 3 is to expand \( \mathcal{PD}_{\mathcal{B}} \) into all possible worlds, compute patterns and rules from each world, and then combine the results. If \text{minsup} = 2, \text{minconf} = 0.5, and \text{minprob} = 0.2, for Figure 4(a), \{a\} \Rightarrow \{c\} is an association rule only in worlds \( W_0 \) and \( W_8 \) (Figure 5), with \( P(\{a\} \Rightarrow \{c\}) = Pr(W_0) + Pr(W_8) = 0.09 + 0.21 = 0.3 \). Since this is larger than 0.2, \{a\} \Rightarrow \{c\} is a p-AR. This method is not practical, due to the large number of possible worlds. To tackle Problems 1 and 2, we propose two efficient algorithms, namely p-Apriori and TODIS, in respectively Sections 3 and 4. Then we address Problem 3 in Section 5.

### 3. THE P-APRIORI ALGORITHM

To solve Problem 1, we propose the p-Apriori algorithm, which is an adaptation of the Apriori algorithm [4] for probabilistic databases. Specifically, p-Apriori uses the bottom-up framework [4]: each item is tested to see whether it is a p-FP. All probabilistic frequent singletons then have their support pmfs computed, and are used to generate size-2 patterns (called candidate patterns). These patterns are examined to see which are frequent patterns. The size-2 p-FPs again have their support pmfs evaluated, and are used to create size-3 candidate patterns. The process is repeated until no more frequent patterns are found. In general, to create a size-\( (m+1) \) candidate patterns from size-\( m \) p-FPs, we use the anti-monotonicity property (Lemma 1), which implies that a size-\( (m+1) \) pattern can be a candidate only if all its size-\( m \) sub-patterns are frequent. Finally, the p-FPs and their support pmfs are returned.

The p-Apriori can also solve Problem 2, by returning only the maximal p-FPs that appear in p-Apriori’s mining result.

For p-Apriori to work well, we have to efficiently check whether a given pattern \( X \) is frequent. When the database is exact, we can scan the database once, find the support count of \( X \), and test it with Equation 1. For p-Apriori, we have to (1) find \( X \)’s support pmf; (2) derive \( P(\text{sup}(X) \geq \text{minsup}) \) and test against Equation 4. To find \( X \)’s support pmf, we may consider its support from all possible worlds; however, this is practically infeasible. In Sections 3.1, we discuss how to prune infrequent patterns without deriving support pmfs. For patterns that cannot be pruned, we discuss two efficient methods for generating support pmfs, in Sections 3.2 and 3.3. We present a data structure to improve our algorithms, in Section 3.4.

#### 3.1 Pruning Infrequent Patterns

Let \( \text{cnt}(X) \) be the number of tuples that contain pattern \( X \) regardless of the tuple probabilities (i.e., \( T_i, p \)). Also, let \( \text{esup}(X) \) be the expected support of \( X \), which can be found by summing up all \( T_i, p \)’s in \( \mathcal{PD}_{\mathcal{B}} \) [9]. The lemmas below describe how to prune \( X \) without knowing its support pmf.

**Lemma 2.** If \( \text{cnt}(X) < \text{minsup} \), then \( X \) is not a p-FP.

**Proof.** The maximum possible value of \( \text{sup}(X) \) is \( \text{cnt}(X) \), which happens when all tuples that contain \( X \) exist. Hence, \( P(\text{sup}(X) > \text{cnt}(X)) = 0 \). Since \( \text{cnt}(X) < \text{minsup} \), we have \( P(\text{sup}(X) \geq \text{minsup}) \leq Pr[\text{sup}(X) > \text{cnt}(X)] = 0 \). Therefore, for any \( \text{minprob} > 0 \), \( X \) cannot be a p-FP according to Equation 4.

**Lemma 3.** Let \( \mu = \text{esup}(X) \), and \( \sigma = \frac{\text{minsup} - \mu - 1}{\mu} \). Then \( X \) is not a p-FP if:

- \( \sigma \geq 2e - 1 \) and \( 2^{-\sigma \mu} < \text{minprob} \), or
- \( 0 < \sigma < 2e - 1 \) and \( e^{-\sigma \mu} < \text{minprob} \)

**Proof.** Notice that the probability \( P(\text{sup}(X) \geq \text{minsup}) \) is equal to \( Pr[\text{sup}(X) > \text{minsup} - 1] \), or \( Pr[\text{sup}(X) > (1 + \sigma)\mu] \). Using the Chernoff Bound [23], we have:

\[
Pr[\text{sup}(X) > (1 + \sigma)\mu] < \begin{cases} 
2^{-\sigma \mu}, & \text{if } \sigma \geq 2e - 1 \\
 e^{-\sigma \mu}, & \text{if } 0 < \sigma < 2e - 1 
\end{cases}
\]

which is less than \( \text{minprob} \). Hence, according to Equation 4, \( X \) is not a p-FP.

To prune \( X \), we first scan the database once and obtain the values of \( \text{cnt}(X) \) and \( \text{esup}(X) \), in \( O(n) \) time. Then, the lemmas are used to prune away \( X \), in \( O(1) \) time. For patterns that cannot be pruned, we next present two efficient techniques to derive their support pmfs.

#### 3.2 The Dynamic-Programming Algorithm

Algorithm 1 (DP) finds support pmf by dynamic programming. The pmf \( f_X \) of pattern \( X \) is initialized to \( \{1, 0, \ldots, 0\} \) in Step 2 (i.e., \( \text{sup}(X) \) is zero before \( \mathcal{PD}_{\mathcal{B}} \) is visited). Then, each \( f_X[k] \) is updated by the information of every tuple \( T_i \) (Steps 3-7). Here, \( p^{i+1}_X \) is the probability that \( X \) occurs in tuple \( T_i \), where \( p^{i+1}_X = T_i, p \) if \( X \subseteq T_i, S \), or zero otherwise. The array \( f_X \) is a temporary buffer. Step 6 is a recursive equation. It means that \( \text{sup}(X) \) can be i when either 1) \( X \) occurs in the current tuple and had \((i-1) \) occurrences in visited tuples, or 2) \( X \) does not occur in the current tuple.
but occurred in visited tuples for exactly $i$ times. This step is repeated until all tuples have been processed. The time and space complexity of \textsc{DP} are $O(n^2)$ and $O(n)$ respectively.

### 3.3 The Divide-and-Conquer Algorithm

**Algorithm 1: DP**

- **Input:** probabilistic database \( \mathcal{PDB} \), pattern \( X \)
- **Output:** support pmf \( f_X \)

1. Initialize \( f_X \leftarrow \{1,0,\ldots,0\} \)
2. for each tuple \( T_i \) in \( \mathcal{PDB} \) do
3. \( \frac{f_X}{k} \leftarrow 1 - p_X^k \times f_X[k] \)
4. for \( k \leftarrow 1 \) to \( n \) do
5. \( f_X[k] \leftarrow p_X^k \times f_X[k-1] + (1 - p_X^k) \times f_X[k] \)
6. return \( f_X \)

**Algorithm 2: DC**

- **Input:** probabilistic database \( \mathcal{PDB} \), pattern \( X \)
- **Output:** support pmf \( f_X \)

1. if \( n = 1 \) then
2. \( f_X[0] \leftarrow 1 - p_X^1 \times f_X[0] \)
3. return \( f_X \)
4. Horizontally partition \( \mathcal{PDB} \) into \( D_1 \) and \( D_2 \), where \( |D_1| = \left\lceil \frac{n}{2} \right\rceil \) and \( |D_2| = \left\lfloor \frac{n}{2} \right\rfloor \)
5. \( f_X^1 \leftarrow \text{DC}(D_1, X) \)
6. \( f_X^2 \leftarrow \text{DC}(D_2, X) \)
7. \( f_X \leftarrow \text{Convolution}(f_X^1, f_X^2) \)
8. return \( f_X \)

Algorithm 2 (DC) is another way of evaluating support pmf. Given a pattern \( X \), Steps 2-4 compute the pmf for the case where \( \mathcal{PDB} \) has only one tuple. Otherwise, \( \mathcal{PDB} \) is horizontally partitioned into two databases \( (D_1 \) and \( D_2 \) in Step 5. Then, DC is recursively invoked on \( D_1 \) and \( D_2 \) to obtain \( X \)'s pmf for each database (Steps 6-7). The two pmfs are used to generate the support pmf of \( X \) (Step 8).

To understand Step 8, let \( \text{sup}_{D_1}(X) \) and \( \text{sup}_{D_2}(X) \) be the support of \( X \) in \( D_1 \) and \( D_2 \) respectively. Since \( \text{sup}_{D_1}(X) \) and \( \text{sup}_{D_2}(X) \) are independent random variables, \( \text{sup}(X) = \text{sup}_{D_1}(X) + \text{sup}_{D_2}(X) \). Let \( f_X^1 \) and \( f_X^2 \) be the pmfs of \( \text{sup}_{D_1}(X) \) and \( \text{sup}_{D_2}(X) \) respectively. Then,

\[
f_X[k] = \sum_{i=0}^{k} f_X^1[i] \times f_X^2[k - i] \tag{7}
\]

In fact, \( f_X \) is the convolution of \( f_X^1 \) and \( f_X^2 \), denoted by \( f_X = f_X^1 \ast f_X^2 \) [14]. While a naïve way of evaluating Equation 7 requires \( O(n^2) \) time, this can be improved by applying Discrete Fourier Transform (DFT) on \( f_X^1 \) and \( f_X^2 \), computing the pairwise products of the two transformed sequences, and performing an inverse DFT on the product. With the use of the Fast Fourier Transform (FFT) algorithm, Equation 7 can be evaluated in \( O(n \log n) \) time [24].

**Complexity.** Let \( c(n) \) be the time cost of Algorithm 2 with database size \( n \). Steps 6 and 7 both need \( c(n/2) \) time, and Step 8 takes \( O(n \log n) \) time with the use of the FFT algorithm. Then, \( c(n) = 2c(n/2) + O(n \log n) \), which yields \( O(n \log^2 n) \). Thus, DC is more efficient and scalable than DP for large datasets. The space complexity is \( O(n) \).

---

3.4 Inverted Probability List

So far, all tuples in \( \mathcal{PDB} \) are used to prune a pattern \( X \), or to find \( X \)'s support pmf. This is not always necessary. Consider a tuple \( T_i \), where \( X \) does not appear in \( T_i \). Since \( T_i \) does not contribute to the values of \( \text{cnt}(X) \) and \( \text{sup}(X) \), the absence of \( T_i \) will not affect the correctness of Lemmas 2 and 3. Since \( p_i^X = 0 \), Algorithms 1 and 2 also work without \( T_i \). In fact, our pruning lemmas and support pmf algorithms can be used on only the list of tuples which contain \( X \). We call this the inverted probability list (ip-list) of \( X \), denoted by \( L^X \). Figure 6(a) shows \( L^a \) for pattern \( \{a\} \). Now let \( l = |L^X| \). If (1) \( \mathcal{PDB} \) is sparse, and (2) \( l \) is long, then \( l \) can be much smaller than \( n \). In this case, the algorithms executed on \( L^X \) can be faster than \( \mathcal{PDB} \). From now on, we assume that \( L^X \) is used in our algorithms.

Let us summarize the process of handling a pattern \( X \) in p-Apriori. We first scan the database to obtain \( L^X \), \( \text{cnt}(X) \) and \( \text{sup}(X) \). Then, Lemmas 2 and 3 are used to check whether \( X \) can be pruned. If not, either Algorithm 1 or 2 is used to find the support pmf, which is passed to Equation 4 for checking the frequency of \( X \). Finally, \( L^X \) is discarded, so that the memory can be reused to store other lists.

**Complexity.** Let \( l = |L^X| \). We can generate \( L^X \) by scanning the database in \( O(n) \) time. Notice that \( \text{cnt}(X) = 1 \), and \( \text{sup}(X) \) can be found by summing up all the tuple probabilities on \( L^X \). Afterwards, Lemmas 2 and 3 can be evaluated in \( O(l) \) time. If \( X \) cannot be pruned, and Algorithm 1 (DP) is used on \( L^X \), then \( O(l^2) \) operations are needed. If Algorithm 2 (DC) is used instead, a \( O(l \log^2 l) \) cost is required. Finally, we discard \( L^X \). Hence, the overall cost of evaluating a pattern in p-Apriori is \( O(n \log^2 l) \) when Algorithm DP is used, or \( O(n + l \log^2 l) \) when Algorithm DC is used. The space complexity is \( O(n) \).

While Lemmas 2 and 3 can prune infrequent patterns effectively, our experiments revealed that p-Apriori spends a lot of time to compute pmfs for the remaining patterns. Moreover, to extract maximal p-FPs, p-Apriori cannot avoid deriving the pmfs of all their sub-patterns. We next study a novel algorithm that alleviates these problems.

4. THE TODIS ALGORITHM

We call the second p-FP mining algorithm \( \text{TODIS} \), which stands for “Top-Down Inheritance of Support pmf”. It involves the execution of two phases: (1) extract patterns that are supersets of p-FPs; and (2) derive p-FPs in a top-down manner (i.e., in descending order of pattern size).

We first explain the intuition behind TODIS. Let \( X \) be a sub-pattern of \( X \). Then, any tuple that consists of \( X \)}
must also contain \( X' \). Hence, the ip-list of \( X \), i.e., \( L^X \), is a subset of \( L^{X'} \). In Figure 6(b), for instance, \( L^{(aeg)} \subseteq L^{(ae)} \subseteq L^{(a)} \). Notice that these ip-lists, with their common entries shadowed, are quite similar. Since ip-lists are used to generate support pmfs, these pmfs should be similar too. We exploit this observation by incrementally deriving support pmfs. For example, \( L^{(ae)} \) is first used to generate \( f_{(ae)} \). We next use \( f_{(aec)} \) to derive \( f_{(ae)} \), which then produces \( f_{(a)} \). This new approach, as we will explain, is faster than computing it from scratch.

**Inheriting from the “best” pmf.** A pattern may be able to inherit pmf from two or more patterns. In Figure 7, for instance, \( \{a\} \) can inherit the support pmf from either \( \{ae\} \) and \( \{ag\} \), since \( \{a\} \) is a sub-pattern of both of them. Which pmf should \( \{a\} \) inherit then? We make the decision based on the lengths of the ip-lists. To understand, let \( X'' \) be a parent of pattern \( X \). Then, \( X'' = X \cup X.\text{exit} \). Also, let \( L \) be the ip-list where \( X \subseteq T, S \) but \( X.\text{exit} \not\in T, S \). Then, \( L \) is used by the support pmf update operation to compute \( X \). For example, \( L = L^{(aeg)} \) in the previous paragraph. Notice that \( |L| = |L^X| - |L^{X''}| \). If \( X'' \) is longer, then \( L \) is shorter and the pmf update is more efficient. For example, suppose \( |L^{(aeg)}| > |L^{(ae)}| \), then \( \{a\} \) inherits \( f_{(ae)} \) instead of \( f_{(a)} \). Hence, a pattern should choose to inherit from the parent whose ip-list is the longest, in order to minimize the cost of updating its pmf.

**Algorithm Design.** Phase 1: Generate candidate patterns. We first swiftly identify a set of patterns which contains all p-FPs. This is done by slightly modifying A-Priori. Specifically, for patterns that cannot be pruned by lemmas 2 and 3, we assume that they are candidate p-FPs, without evaluating their exact support pmfs. All these candidate p-FPs generated are then passed to the next phase for verification.

![Figure 7: Illustrating the TODIS algorithm.](image_url)

**Phase 2:** Top-down support inheritance. To illustrate this step, we suppose that the patterns in Figure 7 are those generated by Phase 1. Since TODIS examines patterns in descending order of their lengths, the longest pattern, i.e., \( \{aeg\} \), is considered first. Specifically, the support pmf of \( \{a, e, g\} \) or \( f_{(aeg)} \), is created “from scratch”, by Algorithms DP or DC. Then \( f_{(aeg)} \) is inherited by its three sub-patterns: \( \{a\}, \{ag\} \) and \( \{eg\} \). That is to say, we use \( f_{(aeg)} \) to derive the pmf for these sub-patterns. For other size-2 patterns, \( \{ae\} \) and \( \{fh\} \), we also generate their pmfs from scratch. After all the support pmfs for size-2 patterns are generated, the pmfs are inherited by size-1 patterns that are subsets of them. For example, \( \{a\} \) inherits the pmf of \( \{ae\} \). All patterns that are true p-FPs (Equation 4) are returned. In this example, all patterns except \( \{ae\} \) and \( \{c\} \) are p-FPs.

How to use the “inherited support pmf” to generate a pmf of a given pattern \( X \)? We define the exclusive item of \( X \), denoted by \( X.\text{exit} \), to be the set difference between \( X \) and the pattern from which \( X \) inherits the pmf. In Figure 7, for example, \( \{ae\} \) is a sub-pattern of \( \{aeg\} \) and inherits \( f_{(aeg)} \). Then, \( \{ae\}.\text{exit} = \{aeg\} - \{ae\} \), or \( \{g\} \). To derive \( f_{(ae)} \), given \( f_{(aeg)} \) and \( \{g\} \), we first evaluate the pmf \( f_{(aeg)} \), with algorithms DP or DC. Since the sets of tuples that contain \( \{aeg\} \) and \( \{ae\}\{g\} \) are disjoint, \( \text{sup}(\{aeg\}) \) and \( \text{sup}(\{ae\}\{g\}) \) are independent. As discussed in Section 3.3, \( f_{(ae)} \) is the convolution \( f_{(aeg)} \) and \( f_{(ae)} \). We call the above procedure the update of support pmf. Observe that list \( L^{(aeg)} \) is empty, and is shorter than \( L^{(ae)} \) that contains \( T_1 \) and \( T_3 \) (Figure 6(b)). Hence, with the efficient convolution of \( f_{(aeg)} \) and \( f_{(ae)} \), updating \( f_{(ae)} \) is generally faster than computing it from scratch.

**Algorithms.** Based on the above discussions, we propose Algorithm 3. Let \( F_k \) be the set of p-FPs of length \( k \), where \( m \) is the size of the largest p-FP. Also, let \( F \) be the set of all \( F_k \)'s. For each pattern \( X \in F_{k-1} \), let \( X.\text{cnt} \) be the length of \( L^X \). In Step 2, we execute Phase 1 and store all candidate p-FPs in \( F \). Then, Steps 6-15 execute Phase 2, by examining each pattern in \( F_k \) in descending order of \( k \). Step 8 updates \( X \)'s support pmf, based on \( X.\text{exit} \) and the pmf inherited from its parent. Steps 11-13 select the best support pmf for \( X \) to inherit, where a pattern's inherited pmf is replaced by another one if there exists a longer ip-list. In Steps 14-15, \( X \) is removed from \( F_k \) if it is not a p-FP (Equation 4). The final result, \( F \), is returned in Step 16.

Algorithm 4 shows the routine for updating support pmf. Steps 3-5 generate \( L \), the ip-list used by the support pmf update operation. If \( X \) has no exclusive item, \( X \) is not a subset of any candidate p-FP. Then \( L = L^X \), and we evaluate \( f_X \) with the DP or DC algorithm (Steps 6-7). Otherwise, we compute the support pmf on \( L \), perform convolution with the inherited pmf of \( X \) (i.e., \( f' \)), in order to generate \( f_X \). Step 11 updates the length of \( L^X \). Finally, \( L \) is discarded and \( X \) is returned (Steps 12-13).

**Complexity.** In Algorithm 3, evaluating a maximal p-FP’s pmf requires executing DP/DC alone, and so the cost
we now explain how to efficiently evaluate
\(XY\) expanding possible worlds is not practical. Given that
check whether

5.1 Computing Association Rule Probability

p-ARs can be obtained.

5. PROBABILISTIC ASSOCIATION RULES

is the same as that of DP/DC. For a non-maximal p-FP \(X\),
its pmf is obtained by updating the inherited pmf, with
a cost of \(O(n + l')^2\) (for DP) or \(O(n + l' \log l')\) (for DC).
Here, \(l' = |L^n| - \max\{|L^{n-1}| \mid X \subseteq X'\}\). When \(l' \ll |L^n|\)
TODIS computes pmfs much faster than p-Apriori. To facilitate
searching of sub-patterns for inheritance (Step 9) and
deletion of patterns (Step 15), we build a trie [17] of depth \(k\)
for each \(F_k\), in which each non-leaf node uses a hash map
to store the addresses of its child nodes. The cost of a pattern
retrieval and deletion in \(F_k\) is \(O(k)\).

Since we only need the maximal candidates for top-down
generation in Phase 2, we remark that it is possible to adapt
existing algorithms of maximal FP mining (e.g., [6, 13]) to
achieve a faster candidate generation in Phase 1.

4.2 Finding Maximal p-FPs

To discover only maximal p-FPs, TODIS can be modified
as follows. First, the same Phase 1 is used to find the candidate
patterns. In Phase 2, once we have identified a p-FP,
we do not compute the support pmf of its sub-patterns.
Finding maximal p-FPs with TODIS is generally faster than
p-Apriori, since (1) long patterns that are potentially maximal
p-FPs can be quickly yielded in Phase 1; and (2) to
obtain a maximal pattern, there is no need to generate the
support pmf for any of its sub-patterns. Next, we examine
how association rules can be generated from p-FPs.

Conversely, \(f_{X \cap Y}\) is the deconvolution of \(f_X\) and \(f_{XY}\). By using
Fast Fourier Transform methods, deconvolution can also be
implemented in \(O(n \log n)\) time. The details can be found in
[24].

Algorithm 4: UpdateSupPMF

Input: \(FDB\), pattern \(X\), inherited pmf \(f\)
Output: support pmf of \(X\)
1 begin
2 | Set \(L\) as an empty list
3 | for each tuple \(T_i \in FDB\) do
4 | | if \(X \subseteq T_i.S \land X.exItem \not\in T_i.S\) then
5 | | | Add \(T_i.p\) to \(L\)
6 | if \(X.exItem = \text{NULL}\) then
7 | | \(f \leftarrow \text{CompSupPMF}(L)\) \(\triangleright\) invoke Algo. 1 or Algo. 2
8 | else
9 | | \(f' \leftarrow \text{CompSupPMF}(L)\) \(\triangleright\) invoke Algo. 1 or Algo. 2
10 | \(f_X \leftarrow \text{Convolution}(f, f')\)
11 | \(X.cmpt \leftarrow X.cmpt + |L|\)
12 | Discard \((L)\)
13 | return \(X\)
14 end

Hence, Lemma 4 is correct. \(\Box\)

Algorithm 5: Computing prob. of a p-AR

Input: support pmf \(f_{XY}\) and \(f_X\)
Output: \(Pr[\mathbf{X} \Rightarrow \mathbf{Y}]\)
1 begin
2 | \(f_{X \cap Y} \leftarrow \text{Deconvolution}(f_{XY}, f_X)\)
3 | \(\triangleright f_{XY} = \{f_{XY}[0], f_{XY}[1], \ldots, f_{XY}[h_1]\} (h_1 \leq n)\)
4 | \(\triangleright f_{X \cap Y} = \{f_{X \cap Y}[0], f_{X \cap Y}[1], \ldots, f_{X \cap Y}[h_2]\} (h_2 \leq n)\)
5 | Initialize \(prAR\) and \(prCum\) to be 0
6 | Initialize \(j\) to be 0
7 | for \(i \leftarrow \text{minsup} \) to \(h_1\) do
8 | | while \(j < h_2\) do
9 | | | if \(j > \frac{\text{minconf}}{\text{minsup}} \times i\) then
10 | | | | break loop
11 | | else
12 | | | \(prCum \leftarrow prCum + f_{X \cap Y}\)
13 | | | \(j \leftarrow j + 1\)
14 | | \(prAR \leftarrow prCum + prCum \times f_{XY}[i]\)
15 | return \(prAR\)
16 end

Algorithm 5 implements the above procedures. We first evaluate
the deconvolution \(f_{X \cap Y}\) (Step 2). Then, Steps 7-14 compute \(P(\mathbf{X} \Rightarrow \mathbf{Y})\) with Equation 8. While a basic implementation
of Equation 8 requires \(O(n^2)\) time, we use a variable, \(prCum\), to accumulate the temporary sum of
\(f_{X \cap Y}[j]\). Then Equation 8 can be evaluated in \(O(n)\) time.
Hence, Algorithm 5 has a \(O(n \log n)\) cost.
5.2 Deriving Association Rules

We now explain how to obtain the p-ARs from the p-FPs generated by p-FP mining algorithms. We first present the following lemma.

**Lemma 5 (Anti-monotonicity).** Let \( X \) be a p-FP, \( X' \) and \( X'' \) be non-empty patterns where \( X'' \subseteq X' \subseteq X \). If \( X - X' \Rightarrow X'' \) is a p-AR, then \( X - X'' \Rightarrow X' \) is a p-AR.

**Proof.** Let \( X \) be a p-FP, and \( X' \) and \( X'' \) be non-empty sub-patterns of \( X \), where \( X'' \subseteq X' \). Suppose that \( W' \) and \( W'' \) are the set of possible worlds where \( X - X' \Rightarrow X'' \) and \( X - X'' \Rightarrow X' \) are association rules respectively. For each possible world \( W_i \in W \), if \( X - X' \Rightarrow X'' \) is an association rule, then so is \( X - X'' \Rightarrow X' \), based on the anti-monotonicity property of association rules for exact data [4]. We know that \( P(X - X' \Rightarrow X') = \sum_{W_i \in W'} P(W_i) \), and \( P(X - X'' \Rightarrow X'') = \sum_{W_i \in W''} P(W_i) \). Since \( X - X' \Rightarrow X'' \) is an association rule, we have: \( P(X - X' \Rightarrow X') \geq P(X - X'' \Rightarrow X'') \geq \minprob \). By Equation 6, \( X - X'' \Rightarrow X' \) is also therefore a p-AR. □

This lemma implies that if \( Y \subseteq Z \) and \( X \Rightarrow Y \) is a p-AR, then so is \( X \Rightarrow Z \) and \( X \Rightarrow Y \).

To generate p-ARs, we adopt the framework of the Apriori algorithm [4], which was developed for exact data. Specifically, let \( X_1 \) be a size-1 sub-pattern of \( X \). For a p-FP \( X \), we enumerate all \( X_1 \)’s as the consequent of the rule. Then we check whether \( X - X_1 \Rightarrow X_1 \) is a p-AR by computing its probability with Algorithm 5. If \( X - X_1 \Rightarrow X_1 \) is not a p-AR, then for any \( X_i \) where \( X_1 \subseteq X_i \), \( X - X_i \Rightarrow X_i \) cannot be a p-AR (by Lemma 5). Hence, we can stop examining the rules that contain the superset of \( X_1 \) as the consequent. Otherwise, we check the validity of \( X - X_2 \Rightarrow X_2 \), for every \( X_2 \) such that \( X_1 \subseteq X_2 \). The checking goes on with larger \( X_i \)s, until we find that \( X - X_i \Rightarrow X_i \) is not a p-AR, or \( X_i = X \). We repeat this for every p-FP \( X \).

6. EXPERIMENTAL RESULTS

We evaluated the performance of our solutions on two datasets. The first one, called T25f10D500, is provided by the IBM data generator\(^1\). The average length of a transaction is 25, the average length of a frequent pattern is 10, and the dataset size \( n \) is 500k. The existential probability of each tuple is randomly drawn between \([0, 1] \). Since no item has support larger than 10% of \( n \), this dataset is sparse, and we set \( \minsup \) to be 0.65% \( \cdot \) \( n \).

The second dataset, called Accident, comes from the Frequent Itemset Mining (FIMI) Dataset Repository \(^2\). It contains 340k transactions, which record traffic accidents in the Flanders-Belgium region during 1991-2000. Each transaction stores the attributes (e.g., time, place and car type) of an accident. The average length of a frequent pattern is 45, and the number of items is 572. The existential probability of each transaction is a random variable, drawn from a Gaussian distribution with mean 0.5 and variance 0.02. The default value of \( \minsup \) is 35% of \( n \).

For both datasets, the default values of \( \minprob \) and \( \minconf \) are 0.5 and 0.9 respectively. All the experiments were carried out on the Windows XP operating system, on a machine with a 2.66 GHz Intel Core Duo processor and 2GB memory. The programs were written in C++ and compiled on Microsoft Visual Studio 2005. We next present the results for the two datasets in Sections 6.1 and 6.2.

6.1 Results on the Synthetic Dataset

(a) Basic solution. We compared the running time of p-Apriori and the “basic” solution, which finds p-FP by expanding possible worlds. For p-Apriori, we used DP to compute support pmf. Figure 8(a) shows that the performance of the basic solution degrades sharply with a slight increase of \( n \), due to the exponential growth of the number of possible worlds. p-Apriori does not expand possible worlds, and so its performance is much better. In fact, all our solutions (i.e., p-Apriori and TODIS) significantly outperform the basic solution. Next, we focus on our solutions.

(b) ip-list. We then studied the benefit of using the ip-list in p-Apriori. We used DP-n and DC-n to denote the versions of DP and DC that do not use the ip-list. Figure 8(b) shows that DP (DC) perform better than DP-n (correspondingly DC-n). Hence, it is worthwhile to build and maintain the ip-list. For example, when \( n = 8k \), DC is two order of magnitudes faster than DC-n. We next assume that the ip-list is used.

(c) Scalability. We compared p-Apriori and TODIS over a wide range of \( n \) in Figure 8(c). For both algorithms, we examined DP and DC. The four variants scale well with \( n \). Also, DC is always better than DP. When TODIS and DC are used together, the best result is yielded. At \( n = 1000k \), TODIS-DC is more than one order of magnitude faster than pApriori-DC.

(d) Effect of \( \minsup \). Figure 8(d) compared p-Apriori and TODIS under different \( \minsup \) values. When \( \minsup \) decreases, the performance gap between pApriori-DC and TODIS-DC increases. With a smaller \( \minsup \), longer patterns have to be tested, and so computing their pmf from scratch is more expensive. TODIS, which updates pmfs instead, benefits more in handling long patterns. The same can be said for pApriori-DC and TODIS-DC. Again TODIS-DC is the best.

(e) Analysis of TODIS. Recall that TODIS spends some effort (in Phase 1) to generate candidate patterns, so that they can be used to identify p-FPs in Phase 2. Over different values of \( \minsup \), the time spent on Phase 1 is a small fraction of that of Phase 2 (Figure 8(e)). The effort of finding candidate patterns in Phase 1 is thus worthwhile, since this facilitates the top-down pmf update in Phase 2. Hence TODIS outperforms p-Apriori. We also investigate the effectiveness of pruning lemmas used in Phase 1. Let \( r \) be the number of true p-FPs. The number of candidates generated in Phase 1 is less than \( 2r \) under different \( \minsup \) settings.

(f) Effect of \( \minprob \). Figure 8(f) showed that the running times of p-Apriori and TODIS increase with a lower \( \minprob \). A lower \( \minprob \) implies that more patterns can become p-FPs. Hence both algorithms spend more time to compute their pmfs. Again, TODIS outperforms p-Apriori.

(g) Maximal p-FPs. Figure 8(g) studied algorithms for finding maximal p-FPs. Since the results produced by TODIS and p-Apriori contain the maximal p-FPs, these p-FPs can be obtained from their results. We also see that TODIS-MAX (Section 4.2) is faster than TODIS, since not all p-FPs have their pmfs evaluated. At \( \minsup = 0.7\% \cdot n \), TODIS-MAX is 20% faster than TODIS.

(h) Finding p-ARs. In Figure 8(h), we examined the use of the anti-monotonicity property (Lemma 5) for generating p-ARs. The running time increases with smaller \( \minprob \).
since more rules become p-ARs. Under a wide range of minprob, Lemma 5 prunes a lot of rules without computing their probabilities. Hence, the speed of generating p-ARs increases significantly.

6.2 Results on the Real Dataset

Since most experiments on the real dataset show a similar trend as the synthetic data, we only present the most representative ones.

(i) Effect of minsup. Figure 8(i) compared p-Apriori and TODIS using DC. We do not show the results for DP, since they take more than three hours to complete. TODIS substantially outperforms p-Apriori; at minsup = 35% · n, TODIS is 75% faster than p-Apriori. This big difference is due to the fact that the real dataset is dense — regardless of the tuple probability, 11 singletons have a support of more than 80%. This results in long ip-lists, which benefits TODIS more than p-Apriori.

(j) Maximal p-FPs. As shown in Figure 8(j), TODIS-MAX is much better than TODIS. When minsup = 35% · n, TODIS-MAX is 70% faster than TODIS. The dense dataset implies that the maximal p-FPs can be large. A maximal p-FP contains many subpatterns. While TODIS has to compute pmls for all subpatterns, TODIS-MAX does not do so. Hence TODIS-MAX is better than TODIS.

(k) Finding p-ARs. From Figure 8(k), we observed that Lemma 5 effectively reduces its running time under different minprob values. Finally, Figure 8(l) showed that the p-AR algorithm needs more time to complete with a lower minconf. When minconf decreases, more rules can become p-ARs, and so more time is spent on computing their probabilities. The running time flattens when minconf ≤ 0.75. We found that most rules generated have confidence higher than 0.75, and so a decrease of minconf does not have much effect on the performance.

7. CONCLUSIONS

We studied efficient algorithms for extracting frequent patterns from probabilistic databases. The TODIS algorithm, when used together with DC, yields the best performance. We examined the efficient mining of p-ARs, which, to our knowledge, has not been studied before. We plan to study the discovery of association rules for other uncertain data models. We will also consider to extend existing mining methods on exact databases, to handle probabilistic data.

8. ACKNOWLEDGMENTS

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.

9. REFERENCES

[22] M. Yiu et al. Efficient evaluation of probabilistic advanced spatial queries on existentially uncertain data. IEEE Transactions on Knowledge and Data Engineering, 21(9), 2009.