Report of presentation about “Mining Uncertain Data with Probabilistic Guarantees”

Disclaimer
All algorithms and ideas presented in this report belong to Liwen Sun, Reynold Cheng, David W. Cheung and Jiefeng Cheng and were presented in their paper “Mining Uncertain Data with Probabilistic Guarantees”.

Abstract
This report discusses some of the algorithms presented in the publication “Mining Uncertain Data with Probabilistic Guarantees”, in particular a Divide-And-Conquer and a dynamic programming algorithm to decide whether a certain pattern is a p-FP and a top down as well as bottom up strategy to extract all p-FPs of a probabilistic database. [1] The mentioned paper introduces top-down and bottom-up algorithms for p-FP (probabilistic frequent pattern) mining and p-AR (probabilistic association rules) mining in uncertain data models. The concepts will be presented by example of a probabilistic database with passed exams and the different study techniques used.

Introduction
In some applications the managed data is uncertain. This can be due to errors in measurements or uncertainty because of non-determinism or unpredictable factors. A possible solution to this problem are probabilistic databases that treat uncertainty as a first-class citizen. One way to implement probabilistic databases is the tuple-uncertainty model. The latter introduces an additional attribute that expresses the probability that the combination of the other attributes in this tuple occur in a real scenario in the database (existential probability). [1]

<table>
<thead>
<tr>
<th>ID</th>
<th>exercises</th>
<th>study time</th>
<th>old exams solved</th>
<th>lectures</th>
<th>probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a</td>
<td>g</td>
<td>k</td>
<td>m</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>a</td>
<td>e</td>
<td>l</td>
<td>n</td>
<td>0.5</td>
</tr>
<tr>
<td>3</td>
<td>a</td>
<td>f</td>
<td>k</td>
<td>n</td>
<td>0.7</td>
</tr>
<tr>
<td>4</td>
<td>a</td>
<td>d</td>
<td>h</td>
<td>m</td>
<td>0.4</td>
</tr>
<tr>
<td>5</td>
<td>c</td>
<td>d</td>
<td>j</td>
<td>l</td>
<td>0.1</td>
</tr>
<tr>
<td>6</td>
<td>b</td>
<td>e</td>
<td>i.</td>
<td>n</td>
<td>0.35</td>
</tr>
<tr>
<td>7</td>
<td>c</td>
<td>g</td>
<td>h</td>
<td>m</td>
<td>0.25</td>
</tr>
<tr>
<td>8</td>
<td>a</td>
<td>f</td>
<td>j</td>
<td>l</td>
<td>0.85</td>
</tr>
<tr>
<td>9</td>
<td>b</td>
<td>f</td>
<td>j</td>
<td>n</td>
<td>0.6</td>
</tr>
</tbody>
</table>

Illustration 2: Probabilistic database for passed exams
In Illustration 2 an example for a probabilistic database is given. The meaning of the letters that are used in Illustration 1 is explained in Illustration 2. The first column of Illustration 1, namely the ID, is the unique identifier for each tuple. The different attributes like study time, exercises and old exams solved describe the behaviour of a students regarding a lecture with exercise sessions and an exam. In this example it is assumed that each study behaviour is shown by exactly one student (it could be imagined as an experiment where each student behaves according to an assigned study behaviour). The probability attribute describes how likely it is that a student with the behaviour described in the tuple passes an exam. The probability is determined by an expected performance of students with these study behaviours. As an example, tuple 3 is observed: A student that has solved all exercises, studied 20-40h for the exam, solved at least 6 old exams and sometimes attended the lectures will pass the exam with a probability of 70%.

One possibility to compute the solution for a specific data mining problem is to list all possible scenarios, the Possible World Semantics, and determine the result from that.\[1\] This method does not scale well, so new algorithms have to be introduced for each individual problem. In this report the problem of p-FP mining will be explored.

In the following chapters some algorithms for data mining in probabilistic data bases are presented. First, the used concept will be presented, followed by two approaches to solve the problem of deciding whether a certain pattern is a p-FP, then a bottom up as well as top down algorithm for p-FP data mining is presented.

**Definitions**

In this Chapter, the vocabulary that is used in this report as well as the problem definition will be introduced and applied to the example.

### Abbreviations [1]

<table>
<thead>
<tr>
<th>PDB</th>
<th>Probabilistic database</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_i, S$</td>
<td>The itemset of tuple with ID i</td>
</tr>
<tr>
<td>$t_i, p$</td>
<td>The probability of tuple with ID i</td>
</tr>
<tr>
<td>$p_i^X$</td>
<td>The probability that X is subset of tuple with ID i. It is calculated by $p_i^X = \begin{cases} t_i, p &amp; \text{if } X \subseteq t_i, S \ 0 &amp; \text{else} \end{cases}$</td>
</tr>
<tr>
<td>$L^X$</td>
<td>The inverted probability list (ip-list) of X whose elements are all tuples that contain pattern X.</td>
</tr>
</tbody>
</table>

### PWS (Possible World Semantics)

PWS is a way to interpret tuple uncertainty. It lists all possible worlds (combinations of existing tuples) and their probability.\[1\] A possible world for the example is $W_1 = \{t_1, t_4, t_5\}$ with probability $p_{t_1} \cdot p_{t_4} \cdot (1 - p_{t_6}) \cdot (1 - p_{t_7}) \cdot (1 - p_{t_8}) \cdot (1 - p_{t_9})$. The probability is calculated as $P[W_1] = p_{t_1} \cdot (1 - p_{t_6}) \cdot (1 - p_{t_7}) \cdot (1 - p_{t_8}) \cdot (1 - p_{t_9}) = 0.0001755$. 

Course: “Algorithms for Database Systems”, SS15  
Supervisor: Prof. Michael Böhlen  
2/10
It is important to note that $t_1$ in our example has to be present in every possible world because it occurs with probability 100%, so every world that does not include $t_1$ cannot exist.

Unfortunately, PWS does not scale well. There are $O(2^n)$ possible worlds in a probabilistic database.[1] This can be seen in Illustration 3 where all possible worlds for a subset of the example are listed.

The probability for a certain possible world $W_i$ can be calculated by $P[W_i] = \prod_{t_k \in W_i} P[t_k] \prod_{t_k \notin W_i} (1 - P[t_k])$.

The sum of all world probabilities has to be 1. [1]

Illustration 3: PWS for first 4 tuples of example

Pattern
A pattern is a set of items in a database, namely a set of attribute values (in the example for instance $\{a, d, h\}$). It is also called transaction or itemset. [1]

Support of X
In a specific transaction database the support of X is the number of occurrences of the pattern X or number of tuples in the database that contain all items in X. It is written as $\text{sup}(X)$. [1]

Support pmf

Illustration 4: Support pmf for pattern $\{a\}$

The support probability mass function (pmf) describes the distribution of the probability for the support count of a certain pattern. The notation $p = f_X(k)$ means that the pmf function f takes the pattern X and the requested number of occurrences k as input and outputs the probability p that the pattern occurs the requested number of times in the
Illustrations 4 and 5 are examples for the probability mass function of patterns \{a\} and \{a, n\} of the example. In Illustration 4, the most likely support values of \{a\} are 3 and 4. In Illustration 5, it is shown that with a probability of 50% \( \text{sup}(\{a, n\}) = 1 \).

**Count of X**

The count of X, or \(\text{cnt}(X)\) as it is also called, is the number of tuples in a probabilistic database that contain pattern X. [1]

**Expected support**

The expected support of an itemset X (abbreviation \(\text{esup}(X)\)) is an estimation of the support of X in a probabilistic database. It is calculated – according to the rules of expected mean value evaluation - like followed:

\[
\text{esup}(X) = \sum_{t: X \subseteq t} P[t, p]
\]  

**p-FP**

A probabilistic frequent pattern (p-FP) is a pattern that occurs frequently (in a sufficiently large number of tuples) with a sufficiently high probability. Formally, a pattern is frequent when \( \text{sup}(X) \geq \text{minsup} \) with a specified support threshold \(\text{minsup}\). Sufficiently high probability can be expressed by \( P[\text{sup}(X) \geq \text{minsup}] \geq \text{minprob} \) with a specified probability threshold \(\text{minprob}\). [1]

A maximal p-FP is a p-FP that is not a subset of another p-FP in the same probabilistic database. [1] Formally, X is a maximal p-FP if and only if the following formula holds:

\[
\forall X', X \subseteq X : X' \not\subseteq Y | Y \text{ is p-FP of } PDB | \land X \text{ is p-FP}
\]

**p-FP Mining**

p-FP mining is a problem in probabilistic databases that calculates a set of p-FPs X with their respective support pmfs \( f_X(k) \) for given thresholds \(\text{minsup}\) and \(\text{minprob}\) and a probabilistic database. [1]

p-FP mining can be useful to find correlations between different attributes in a database. In the publication "Mining Uncertain Data with Probabilistic Guarantees" [1] the concept of p-AR (probabilistic association rules) is presented as well. p-FP help to find those association rules. [1] Concretely, in the example could be a correlation between students that attend lectures and solve exercises and there is – with a high probability – a correlation between students that solved all exercises (a) and passed the exam and therefore their study behaviour occurs in the database (\( P[\text{sup}((a)) \geq 3] = 0.86 \)).

**Anti-Monotonicity**

Anti-Monotonicity is a property that holds for p-FPs. It means that for any pattern X that is a p-FP all its subsets X’ are also p-FPs. [1]

Proof [1]:

---

Course: “Algorithms for Database Systems”, SS15  
Supervisor: Prof. Michael Böhlen  
4/10
the set of all possible worlds $W^X$ where $X$ is a frequent pattern is a subset of all possible worlds $W^{X'}$ where $X'$ is a frequent pattern

- the probability of a frequent pattern can be calculated by $f_X (k) = \sum_{W_i \in W^{X'}} P[W_i]$

From the first point follows $\forall W_i \in W^{X}: X' \in W_i$ and therefore $f_{X'} (k) = \sum_{W_i \in W^{X'}} P[W_i] \geq \sum_{W_i \in W^X} P[W_i]$, so the probability for a frequent pattern $X'$ is at least the probability for a frequent pattern of $X$ that is sufficiently high ($X$ is p-FP). Therefore $X'$ is p-FP.

### Algorithms on filtering p-FPs

To determine whether a candidate pattern $X$ is actually a p-FP there are several possibilities. One is to calculate all possible worlds and add all probabilities of possible worlds that contain pattern $X$, but this is exponentially expensive - as mentioned in the definition of PWS. In this chapter, some algorithms to decide whether pattern $X$ is a p-FP are presented.

#### Pruning

In this part some criteria to rule out some patterns $X$ directly are listed.

Pattern $X$ is not a p-FP when $cnt (X) < minsup$. Count of $X$ is the maximal number of supports of $X$, namely when all tuples in a PDB that contain $X$ are in the transaction database then $sup(X) = cnt(X)$. So when the count of $X$ is too small, $X$ cannot be frequent. [1]

Another method to exclude patterns $X$ that are definitely not p-FPs is to use the expected support. Lets define $\mu = \text{esup} (X), \sigma = \frac{\text{minsup} - \mu - 1}{\mu}$. $X$ is not a p-FP when one of the following criteria holds [1]:

- $\sigma \geq 2e - 1 \land 2^{-\sigma \mu} < \text{minprob}$
- $0 < \sigma < 2e - 1 \land e^{-\frac{\sigma \mu}{4}} < \text{minprob}$

Proof: The probability that a pattern is frequent can be transformed according to the following formula $P[\sup (X) \geq \text{minsup}] = P[\sup (X) > \text{minsup} - 1] = P[\sup (X) > (1 + \sigma) \mu]$. This term can be bounded by the Chernoff Bound and with the criteria from above it yields

$$Pr[\sup (X) > (1 + \sigma) \mu] \leq \begin{cases} 2^{-\sigma \mu} & \text{for } \sigma \geq 2e - 1 \\ \frac{-\sigma \mu}{e^4} & \text{for } 0 < \sigma < 2e - 1 \end{cases} < \text{minprob}$$. That leads to the conclusion that $X$ is not a p-FP. [1]

#### Inverted Probability List (ip-List)

An inverted probability list, or ip-list for short, of a pattern $X$ is the list of all tuples in a PDB that contain $X$. The notation for an inverted probability list of $X$ is $L^X$. [1]
Some examples of ip-lists for the example are:

- \( L_{\{a\}} = \{t_1, t_2, t_3, t_4, t_8\} \)
- \( L_{\{a, n\}} = \{t_2, t_3\} \)
- \( L_{\{f\}} = \{t_3, t_8, t_9\} \)

Inverted probability lists are used to reduce the complexity of the presented algorithms. For a pattern \( X \), every tuple that is not in \( L^X \) does not contribute anything to the existence of a p-FP. The probability that \( \text{sup}(X) = k \) is the same whether these tuples are present in the possible world or not. So it is sufficient to check all tuples in \( L^X \) for the p-FP property, therefore the complexity of the algorithms is now dependent on \( l = |L^X| \) rather than PDB size \( n \). It is notable that the PDB has to be traversed once to obtain the inverted probability list.

**Algorithm with Dynamic Programming (DP)**

One way to check whether pattern \( X \) is a p-FP is to use a dynamic programming algorithm. In this algorithm only tuples that are in \( L^X \) are considered, all tuples that are not in \( L^X \) are just a copy of the row above. The DP table has two dimensions, one for each tuple in \( L^X \) and one for each support value \( k \). The last row of the table contains \( f_X[k] \) for all support values \( k \) between 0 and \( n \) (size of PDB) or 0 and \( l \) (size of \( L^X \)) respectively.

The DP-table is constructed with the following rules [1]:

- **Meaning of different columns**: different integer values for \( \text{sup}(X) \) ranging from 0 to \( l \)
- **Meaning of different rows**: different tuple IDs of all tuples in \( L^X \). Row 0 is initialized with \([1, 0, ..., 0]\).
- **Meaning of entry \( p \) at position \((i, j)\)**: \( p \) is probability that \( \text{sup}(X) = j \) when only considering all tuples up to \( t_i \)
- **Calculation of entries**: first column in row \( i \) with formula \( E[i, 0] = \prod_{k=1}^{i} p^X_k \) and entries for \( i > 0, j > 0 \) with formula \( E[i, j] = E[i - 1, j - 1] * p^X_i + E[i - 1, j] * (1 - p^X_i) \)

Illustration 6 displays the DP table for pattern \( \{a, n\} \). Here the first column is non-zero because \( t_1 \) does not contain itemset \( \{a, n\} \). The last row represents all support values with their probabilities, their sum is 1. The corresponding chart is Illustration 5.

<table>
<thead>
<tr>
<th>Tuple</th>
<th>Support</th>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>init</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>( t_2 )</td>
<td>0.5</td>
<td>0.5</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>( t_3 )</td>
<td>0.15</td>
<td>0.5</td>
<td>0.35</td>
<td></td>
</tr>
</tbody>
</table>

**Illustration 6: DP table for \( X = \{a, n\} \) of example**

In Illustration 7, the DP table for pattern \( \{a\} \) is constructed. The first column is 0 for all tuples because the first tuple, \( t_1 \), occurs with probability 100% in the database, so there is always a support of at least 1 for patterns that are contained in \( t_1 \). Only tuples that contain pattern \( \{a\} \) are listed in the table. The last row represents the probabilities of each
support value. This distribution is also represented in Illustration 4.

<table>
<thead>
<tr>
<th>Tuple</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>init</td>
<td>0</td>
</tr>
<tr>
<td>t₁</td>
<td>1</td>
</tr>
<tr>
<td>t₂</td>
<td>0</td>
</tr>
<tr>
<td>t₃</td>
<td>0.5</td>
</tr>
<tr>
<td>t₄</td>
<td>0.15</td>
</tr>
<tr>
<td>t₅</td>
<td>0.09</td>
</tr>
<tr>
<td>t₆</td>
<td>0.0135</td>
</tr>
</tbody>
</table>

**Illustration 7: DP table for X = {a} of example**

The complexity of this algorithm is $O(n^2)$ when all tuples are listed and $O(n+l^2)$ with $|L_X| = l$ when only the inverted probability list is used (only tuples that contain X). This is so because the table has n (respectively l) rows as well as n (respectively l) columns and therefore has $n^2$ entries that have to be calculated in $O(1)$ time each. [1]

**Algorithm with Divide-And-Conquer (DC)**

Another way to evaluate the support pmf of pattern X is to use a divide-and-conquer algorithm where the problem is split into smaller subproblems and then their results are combined. Here the database is split up until only one tuple remains in the database.

In the following definitions the array $f_X$ contains at position i the pmf value for support i, the array only lists the first support values, all remaining array position that are not explicitly mentioned are 0.

For a divide-and-conquer algorithm the base case, as well as the split, recursion and merge functions have to be defined. For this specific algorithm those are defined as follows [1]:

- **Base case:** when the database contains only one tuple, namely $n = 1$, then $f_X[0] = 1 - p_i^X$, $f_X[1] = p_i^X$, $\forall k > 1: f_X[k] = 0$ when database $D = \{t_i\}$
- **Split:** Horizontally split PDB into two sub-databases $D_1, D_2$ with $|D_1| = \lfloor \frac{n}{2} \rfloor$, $|D_2| = \lceil \frac{n}{2} \rceil$
- **Recursion:** Call $f_X^1 = DC(D_1, X)$ and $f_X^2 = DC(D_2, X)$ with DC(D, X) as the recursive funtion of the divide-and-conquer algorithm
- **Merge:** Convolution of $f_X^1$ and $f_X^2$ with the formula $f_X[k] = \sum_{i=0}^{k} f_X^1[i] \cdot f_X^2[k-i]$

In this paragraph, the derivation for the merge step is presented. The support of sub-database $D_1$ and the support of sub-database $D_2$ are independent because their
tuple sets are disjoint. So \( P[\sup(X)=k] = \sum_{i=0}^{k} P[\sup_1(X)=i] \cdot P[\sup_2(X)=k-i] \) which is the same as \( f_X[k] = \sum_{i=0}^{k} f_X^1[i] \cdot f_X^2[k-i] \) according to the definition of \( f_X \). [1]

**Drawing 1: DC algorithm for pattern \{a\} in example**

Drawing 1 illustrates the application of the DC algorithm for pattern \{a\}. The gray box means that this is a function call. The downward arrows signify the recursive function calls and the upward arrows mean the return of the result of those. The content of the gray boxes are the programming steps that are executed. The elements of the arrays \([p_1, p_2, \ldots, p_i]\) imply that \( p_i = P[\sup(X)=i] \), probabilities for all other support values are 0.

As an example for the application of the merge step, this paragraph calculates the merging step of the left child of the root: \( f_X = [0.3, 0.7] \cdot [0.09, 0.57, 0.34] \)
• $f_X[0] = 0.3 \cdot 0.09 = 0.027$
• $f_X[1] = 0.3 \cdot 0.57 + 0.7 \cdot 0.09 = 0.234$
• $f_X[2] = 0.3 \cdot 0.34 + 0.7 \cdot 0.57 = 0.501$
• $f_X[3] = 0.7 \cdot 0.34 = 0.238$

The complexity of this algorithm is $O(n \log^2 n)$ (respectively $O(n + l \log^2 l)$ in case the inverted probability list is used) when the convolution is done by FFT (fast Fourier transformation). The recursive function call is executed $O(\log n)$ times and the convolution calculated by FFT takes $O(n \log n)$ time. [1]

**p-Apriori algorithm**

The p-Apriori algorithm is a bottom-up algorithm to solve p-FP mining. Bottom-up means that first patterns of size 1 are evaluated. When two patterns of size 1 are p-FP then their combination, a pattern of size 2, is a candidate pattern. Next, candidate patterns of size 3 are evaluated and so on. This method uses the anti-monotonicity property. Each subset of itemset $X$ has to be a p-FP when $X$ is a p-FP, so only combinations of p-FPs can be p-FPs themselves. [1] The decision whether a pattern is a p-FP is made by using one of the algorithms mentioned in the chapter above.

**The TODIS algorithm**

The TODIS algorithm (Top-down inheritance of support pmf) uses a top-down approach to find the support pmfs to all possible p-FPs. The general idea is to use the result pmfs of larger sets to calculate the support pmfs of their subsets. This can be done because there is a dependency between the support pmfs of a superset and its subsets. A subset $X'$ of superset $X$ shares many tuples in $L_{X'}$ with $X$, so this algorithm calculates the pmf for all the tuples in $L_{X'}$ that are not in $L_X$ and combines this with the pmf already calculated for $X$ by using convolution (This is basically the merge step in the DC algorithm presented above). This is faster than calculating the pmf for $X'$ from scratch, because the tuples that are not shared with $X$ are less than all tuples in $L_{X'}$ and therefore its ip-list is smaller.

The TODIS algorithm works in the following way: In phase 1, the candidate patterns are identified, in particular every pattern that can not be pruned is a candidate pattern. All candidate patterns are then connected in the inheritance graph where a smaller pattern is connected to one of its superpatterns. In phase 2, first the support pmfs of the longest candidate patterns are calculated by the DC or DP algorithms explained in earlier chapters. Then the probabilistic mass functions of the longest patterns are inherited by shorter ones according to the inheritance graph. This means that the calculation of shorter patterns is not done from scratch but uses the results from longer candidate patterns.

The inheritance graph for candidate patterns of the example is shown in Drawing 2. [1] Here, only the count pruning rule was used to find candidate patterns. As seen in this Drawing, an inheritance graph is a forest of trees with the longest candidate patterns as roots and single attribute patterns as leaves.
Conclusion

The presented algorithms are a lot more efficient than the naïve approach with PWS. The used methods are common and well understandable. In the p-Apriori as well as the TODIS algorithm it is more efficient to use divide-and-conquer rather than dynamic programming. [1] This is because of the efficient calculation of convolution by using FFT.

Unfortunately, there seems to be a redundancy in both the top down and bottom up approach by calculating the support pmf for every pattern X separately. Two patterns that are not independent (\( X \cap Y \neq \emptyset \)) share some steps of the calculation of the support pmfs.

The TODIS algorithm is more efficient than the p-Apriori algorithm because it takes advantage of the dependencies between some patterns. Additionally, when only the maximal p-FPs are required then the TODIS algorithm can stop after a p-FP is found, which means that no sub-patterns have to be inspected. [1] In the p-Apriori algorithm all p-FPs have to be calculated to find maximal p-FPs.

References