A PROJECTION-BASED HYBRID SEQUENTIAL PATTERNS MINING ALGORITHM

Chichang Jou
Department of Information Management, Tamkang University, 151 Ying-Chuan Road, Tamsui, Taipei, Taiwan

Keywords: Hybrid sequential pattern, Pattern growth, Projected position array, Projected support array, Projected database.

Abstract: Sequential pattern mining finds frequently occurring patterns of item sequences from serial orders of items in the transaction database. The set of frequent hybrid sequential patterns obtained by previous researches either is incomplete or does not scale with growing database sizes. We design and implement a Projection-based Hybrid Sequential Pattern Mining algorithm, PHSPAM, to remedy these problems. PHSPAM first builds Supplemented Frequent One Sequence itemset to collect items that may appear in frequent hybrid sequential patterns. The mining procedure is then performed recursively in the pattern growth manner to calculate the support of patterns through projected position arrays, projected support arrays, and projected databases. We compare the results and performances of PHSPAM with those of other hybrid sequential pattern mining algorithms, GFP2 and CHSPAM.

1 INTRODUCTION

Data mining aims to extract implicit information inside voluminous electronic data, and has become more and more important for science, engineering, business, etc. Frequent sequential pattern mining obtains item sequence patterns satisfying the condition that the number of their occurrences in the database, called support, is greater than or equal to a given minimum threshold. It has been applied to applications like stock trend, genome sequencing, web page traversal, customer behavior analysis, etc.

Based on requirement of items’ adjacency in a pattern, sequential patterns could be classified into three categories (Chen et al., 2002): (1) Continuous patterns, where the adjacent items in a pattern must be matched by consecutive items in the transactions. (2) Discontinuous patterns, where adjacent items in a pattern are matched by items separated by zero or more items. (3) Hybrid patterns, where discontinuity is represented by a special symbol ‘*’ so that continuous and discontinuous adjacency requirements could be specified together. For example, for a transaction T to match the hybrid pattern <AB*C>, its matching items for A and B must be adjacent, and its matching items for B and C are not necessarily consecutive.

As far as we know, the most efficient hybrid sequential pattern mining algorithm is GFP2 (Chen et al., 2002). Its basic concept is the Apriori principle--candidate frequent patterns of length n could be generated by joining frequent patterns of length n-1. However, with repetitive items in a transaction, some frequent patterns might be missed by this technique. For example, suppose the database has only one transaction $T=<ABCD>$, and the minimum threshold is 2. The only frequent pattern of length 1 is $<B>$. By joining frequent patterns of length 1, the candidate frequent patterns of length 2 are $<B*B>$ and $<BB>$. Supports for patterns $<A*B>$ and $<B*D>$ are 2, but they are not obtainable by filtering $<B*B>$ and $<BB>$.

Jou (2006) proposed an algorithm, CHSPAM, to obtain the complete hybrid sequential patterns. However, CHSPAM’s performance is no better than GFP2. We utilize the pattern growth methodology and propose Projection-Based Hybrid Sequential Pattern Mining algorithm, PHSPAM, to tackle this performance problem.

The rest of the paper is organized as follows: Section 2 surveys related work. Problem definition and PHSPAM are illustrated in Section 3. Experimentations and their results are discussed in Section 4. Section 5 concludes the paper and points out future work directions.
2 RELATED WORK

Agrawal and Srikant (1995) extended the frequent itemset mining algorithm (Agrawal and Srikant, 1994; Agrawal et al., 1993) for non-serial transactions to discontinuous sequential pattern mining for serial transactions using the Apriori principle for sequential patterns. Pei et al. (2000) obtained discontinuous sequential web traversal patterns by constructing the access pattern tree from web access logs. Chen et al. (1998) extended the Apriori principle to obtain frequent continuous sequential web path traversal patterns.

Based on the approach to attack the sequential pattern mining problem, its solutions could be classified into two categories: 1. Candidate pattern generation and filtering; 2. Pattern growth.

The methods in the candidate pattern generation and filtering category extend the Apriori principle. The whole database needs to be scanned in each candidate pattern generation step. For large databases, this approach needs huge memory space to store the candidate patterns. GSP (Agrawal and Srikant, 1996), GFP2 (Chen et al., 2002), and SPADE (Zaki, 2001) all belong to this category. GSP added time constraints between adjacent items in a pattern, relaxed the restriction that elements of a pattern must be from the same transaction, and allowed items to cross taxonomy levels. GFP2 was the first to obtain hybrid sequential patterns. It utilized the containment relationship of supports between the patterns with '*' and those without '*' to reduce the number of database scans. SPADE utilized subset relationship of patterns in calculating the support of candidate patterns by checking the positions of each item in the transactions.

In the pattern growth methodology, to skip the time-consuming candidate patterns generation step, a candidate pattern is built by concatenating a pattern \( p \) with patterns obtained from its projected databases, which are collections of after-\( p \) sub-transactions. Supports of patterns with prefix \( p \) are counted by summing supports of patterns supported by sub-transactions in \( p \)'s projected database. FreeSpan (Han et al., 2000a; Pei et al., 2004) first utilized the pattern growth concept from FP-Growth (Han et al., 2000b). PrefixSpan (Pei et al., 2001; Pei et al., 2004) used bi-level projection to reduce the number of projected databases. It also recorded the transaction id and the position of sequential patterns, instead of the real data, to reduce the memory requirement. Jou (2006) discovered that some frequent patterns could not be obtained by the Apriori principle. He proposed the first complete hybrid sequential pattern mining algorithm, CHSPAM. Its mining procedure is performed recursively in the pattern-growth manner by the backward support counting method.

3 THE PHSPAM ALGORITHM

Our definition of hybrid sequential pattern follows that of GFP2 (Chen et al., 2002): Let \( I = \{i_1, i_2, ..., i_n\} \) denotes the set of items in a database. A transaction \( T=<t_1 t_2 ... t_k> \) is an item sequence, where for all \( i \) between \( l \) and \( k, t_i \in I \). A database consists of a set of transactions. A special symbol '*' not in \( I \) is used in the pattern specification to denote the occurrence of 0 or more items. A sequence \( <x_1 x_2 ... x_n> \) is called a hybrid sequential pattern if:

\[
\begin{align*}
(1) & \quad x_i \in I \cup \{*\} \text{ for all } 1 \leq i \leq n \\
(2) & \quad x_1 \in I \text{ and } x_n \in I; \text{ and} \\
(3) & \quad \text{ for all } 1 < i < n, \text{ if } x_i = '*' \text{ then } x_{i-1} \in I \text{ and } x_{i+1} \in I.
\end{align*}
\]

Condition (2) requires a pattern to start and end with an item. Condition (3) exclude consecutive '*'s in a pattern. Hybrid sequential pattern will be abbreviated as pattern henceforth when no confusion arises. Thus, \( <ABC>, <A*BC> \) and \( <A*B*C> \) are legitimate patterns, while \( <AB> \) and \( <AB**C> \) are not.

A pattern \( p \) is said to be contained in a transaction \( T \), denoted as \( p \subseteq T \), if:

\[
\begin{align*}
(1) & \quad \text{for all items } x \in p, \text{ there is a matching item } x' \in T; \\
(2) & \quad \text{for all items } x, y \in p, \text{ if } x \text{ precedes } y \text{ in } p, \text{ then } x' \text{ 's matching item in } T \text{ also precedes } y' \text{ 's; and} \\
(3) & \quad \text{ for all items } x, y \in p, \text{ if } x \text{ is adjacent to } y \text{ in } p, \text{ then } x' \text{ 's matching item in } T \text{ is also adjacent to } y' \text{ 's.}
\end{align*}
\]

The above conditions require that a matching instantiation of \( p \) in \( T \) must respect the serial order of items in \( p \).

For each pattern \( p \) and transaction \( T \), the number of different matching instantiations for \( p \subseteq T \) is called the support of \( p \) in \( T \), denoted by \( \text{sup}_{p,T} \). For example, suppose \( p=<A*BC> \) and \( T=<ABC> \). Then \( \text{sup}_{p,T} = 2 \), where the first instantiation matches \( A, B, C \) to positions 2, 5, 6 of \( T \), and the second matches them to positions 4, 5, 6. For a given minimum support threshold \( \text{minSupp} \), we call pattern \( p \) a frequent pattern if \( p \) satisfies the condition:

\[
\sum_{T} \text{sup}_{p,T} \geq \text{minSupp}.
\]

From short to long patterns, PHSAMP generates for each pattern \( p \) a projected position array, which
keeps for all matching transactions the positions of the items matching p’s last item. Corresponding to each position in the position array, their contribution to the support for p is stored in p’s projected support array. The mining procedure is recursively applied to the projected databases by summing supports from their projected support arrays.

PHSPAM has four steps: Step 1 scans the database and constructs the Supplemented Frequent One Sequence itemset, denoted as SFOS, to collect items that may appear in frequent patterns. Step 2 transforms items not in SFOS into ‘+’ to generate the reduced database, which helps decrease the number of comparisons in step 4. Step 3 generates the projected position arrays, projected support arrays, and projected databases for each SFOS item. Step 4 performs recursive support counting in the projected databases, and finally checks whether SFOS items are frequent patterns. We discuss each step in the following subsections.

3.1 Generating the SFOS Itemset

The length of a pattern p is defined as the number of items occurring in p. In hybrid sequential pattern mining, frequent patterns of length 1 are the same as the set of items with support greater than or equal to minSupp. As illustrated in Section 1, items with support less than minSupp could appear in frequent patterns of length longer than 1. PHSPAM remedies the problem by constructing SFOS in the first step as follows:

1. All items with their total number of occurrence greater than or equal to minSupp. These are the traditional frequent one sequence items;
2. All repetitive items, which are items occurring more than once in a single transaction;
3. All items before and after any pair of repetitive items in a transaction.

For the example in Section 1, Items (3) are the repetitive items in a transaction. All repetitive items, which are items occurring more than once in a single transaction, for repetitive items in a transaction, we will have a sub-transaction for each occurrence of the item. These are the traditional frequent one sequence items.

Table 1: The sample database.

<table>
<thead>
<tr>
<th>trans. id</th>
<th>serial items</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ABCED</td>
</tr>
<tr>
<td>2</td>
<td>BACBAD</td>
</tr>
<tr>
<td>3</td>
<td>BCABD</td>
</tr>
</tbody>
</table>

Table 2: Reduced sample database.

<table>
<thead>
<tr>
<th>trans. id</th>
<th>serial items</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>AB+D</td>
</tr>
<tr>
<td>2</td>
<td>BA+BAD</td>
</tr>
<tr>
<td>3</td>
<td>B+ABD</td>
</tr>
</tbody>
</table>

3.2 Database Reduction

The second step transforms each series of consecutive non-SFOS items in the original database DB into a special symbol ‘+’, to obtain DB’, the reduced database. Since the first and last items of a pattern are in SFOS, ‘+’s appearing in the prefix or suffix of a reduced transaction are deleted in this reduction. For example, suppose the sample database DB contains the 3 transactions in Table 1, and minSupp is 4. By scanning DB once, its SFOS could be constructed as {A, B, D} with supports 4, 5, 3 respectively. Note that the support of item D is less than minSupp. D’s inclusion in SFOS is because D is after the repetitive items B in transactions 2 and 3. By scanning DB once more, the reduced database DB’ in Table 2 could be constructed.

Table 3: Projected position array, projected support array, projected database of pattern <B>.

<table>
<thead>
<tr>
<th>subtr. id</th>
<th>trans. id</th>
<th>PPA_B</th>
<th>PSA_B</th>
<th>PDB_B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>+D</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>A+BAD</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>AD</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>+ABD</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>D</td>
</tr>
</tbody>
</table>

To simplify the explanation, a tabular form of PPA_B, PSA_B, PDB_B will be indexed by the same sub-transaction id. Table 3 illustrates the tabular form of PPA_B, PSA_B, and PDB_B for the reduced sample database. These projected data structures for SFOS items are stored in the hard disk. Since the support counting procedure uses one projected database for
an SFOS item at a time, at any time PHSPAM loads only one set of these projected data structures into memory.

3.4 Recursive Traverse and Count

Figure 1 demonstrates the fourth step, TraverseAndCount, for a pattern \( p \). Let \( I_p \) denote the set of items in the projected database \( PDB_p \). This step first calls GetSupport\((p)\) to obtain the support count mapping, \( M_p \), which maps, for all items \( i \) in \( I_p \), patterns \( <p*i> \) and \( <pi> \) to their supports. Details of GetSupport will be explained in the next paragraph. The supports \( M_p (<p*i>) \) and \( M_p (<pi>) \) are compared to \( minSupp \) to decide whether \( <p*i> \) and \( <pi> \) are frequent patterns. Then, it calls BuildProjectedArrays\((p,i)\) to build projected position arrays \( (PPA_p) \) and \( PPA_p \), and projected support arrays \( (PSA_p) \). Suppose the first item of \( p \) is \( x \). The projected database \( PDB_x \) (similarly, \( PDB_p \)) is the set of frequent itemsets. \( PDB_x \) could be obtained from the fly from \( PDB \), and \( PPA_p \) (similarly, \( PDB \) and \( PPA_p \)). TraverseAndCount will then be recursively applied to \( <p*i> \) and \( <pi> \).

```plaintext
/* \( p \) denotes a pattern */
/* PDB_p is the projected database of \( p \) */
/* \( IP \) is the set of frequent patterns */
Procedure TraverseAndCount(p)
Begin
    \( M_p = GetSupport(p) \)
    /* \( M_p \) maps \( <p*i> \) and \( <pi> \). \( i \) being one of the items in \( PDB_p \) to their supports */
    for each item \( i \in I_p \)
        if \( M_p(<p*i>) \geq minSupp \)
            add \( <p*i> \) to FP
        endif
        if \( M_p(<pi>) \geq minSupp \)
            add \( <pi> \) to FP
        endif
    BuildProjectedArrays\((p,i)\)
    TraverseAndCount\((<p*i>)\) TraverseAndCount\((<pi>)\)
endFor
End Procedure
```

Figure 1: Pseudo code of TraverseAndCount.

For a pattern \( p \), Figure 2 shows the construction of the support count mapping \( M_p \) in GetSupport\((p)\) by scanning the sub-transactions in \( PDB_p \) as follows: Suppose the index of sub-transaction \( T_j \) in \( PDB_j \) is \( j \). Out of all matching instantiations of \( <p> \) in \( T_j \)'s original transaction \( T \), \( PSA_p(T)[j] \) matching instantiations satisfy the condition that the last element of \( p \) in the matching instantiation is adjacent to \( i \), verified by checking whether the position of \( i \) is equal to \( PPA_p(T)[j]+1 \), each occurrence of \( i \) would increase the support count mapping of \( <p*i> \) by \( PSA_p(T)[j] \). Similarly, if an item \( i \) in \( T \) satisfies the condition that the last element of \( p \) in the matching instantiation is adjacent to \( i \), verified by checking whether the position of \( i \) is equal to \( PPA_p(T)[j]+1 \), each occurrence of \( i \) would increase the support count mapping of \( <p*i> \) by \( PSA_p(T)[j] \). In \( M_p \), the support counts for \( <p*i> \) and \( <pi> \) are the accumulated sum of supports of \( <p*i> \) and \( <pi> \) over all sub-transactions in \( PDB_p \). It is clear that \( M_p \) maps each pattern to their support count in the original database.

From projected database \( PDB_p \) of pattern \( <B> \) in Table 3, \( I_B \) is \( \{A, B, D\} \). Take item \( A \) as an example. The detailed support counting of the mapping of \( <B*A> \) in \( M_p \) is demonstrated in Table 4. Note that the support of \( <B*A> \) contributed from transaction 2 position 5 is 2, where both subtransactions 2 and 3 in \( PDB_B \) contribute one support. Since sum of supports of \( <B*A> \) in \( PDB_B \) is 4, \( <B*A> \) is added into the set of frequent patterns. However, since only 2 of the 4, both from transaction 2, satisfy the condition that item \( A \) is immediately after \( B \), support of \( <BA> \) is 2, and \( <BA> \) is not a frequent pattern. The function BuildProjectedArrays\((<B>,A)\) then builds projected position arrays \( (PPA_B) \) and \( PSA_B \), and projected support arrays \( (PSA_B) \). TraverseAndCount is then recursively performed on \( <B*A> \) and \( <BA> \) separately.

For the projected database \( PDB_B \) in Table 4, item \( D \) occurs once in sub-transactions 1, 2 and 3.
4 EXPERIMENTS

We implement GFP2, CHSPAM and PHSPAM in VB.NET 2005. The first experiment compares numbers of obtained frequent patterns and execution time with respect to database size and average length of frequent patterns. The second experiment checks the sensitivity of PHSPAM regarding the minimum support. The experiments are performed in the Windows Server operating system, Intel Xeon 3100 Dual-Core CPU, and 1024MB DDR memory. The databases, generated by IBM Quest Synthetic Data Generator, are stored in Microsoft SQL Server 2005. The fixed parameters for generating the databases are: the number of distinguished items is 10000; the number of items in the frequent one sequence itemset is 1500; the average transactions length is 15. The varied parameters are: |D|: the number of transactions in the database, and |I|: the average length of frequent patterns.

The first experiment is with minSupp set as 0.015% of |D|. The parameters of |D| are 100K, 200K, 300K, 400K and 500K, and the parameters of |I| are 2 and 4. Number of frequent patterns and execution time results are displayed in Table 5. In all 10 settings, the numbers of frequent patterns obtained by CHSPAM and PHSPAM are the same. They are 0.08% to 14.67% more than those by GFP2, but the percentage fluctuates with the database size. This shows that the behavior of PHSPAM is dependent on the probabilistic population of the repetitive items. Figure 3 displays the execution time in these 10 settings for GFP2 and PHSPAM only. With the projected data structures to accumulate support counts, PHSPAM runs faster than GFP2 and CHSPAM.

The second experiment runs PHSPAM with minSupp set as 0.0100%, 0.0125%, 0.0150%, 0.0175% and 0.0200% of |D|. The parameters of |D| are 100K, 200K, and 300K, and the parameters of |I| are 2 and 4. Number of frequent patterns and execution time results are displayed in Table 6. Both consistently grow with the decrease of the minimum support. Figure 4 shows the trend of number of frequent patterns regarding to the minimum support.

<table>
<thead>
<tr>
<th>subtr. id</th>
<th>trans. id</th>
<th>PPA_B*A</th>
<th>PSA_B*A</th>
<th>PDB_B*A</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>+BAD</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>5</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>BD</td>
</tr>
</tbody>
</table>

The supports for <B*A*D> contributed by sub-transaction 1, 2, and 3 are 1, 2, and 1, respectively. Thus, MB*A(<B*A*D>)=4. Since only the case in sub-transaction 2 satisfies the condition that D is immediately after A, MB*A(<B*A*D>)=2. The support count accumulation details of <B*A*i> and <B*Ai> for all items i in IB*A could be computed in the same fashion. Thus, the sample database has the following frequent patterns: <A>, <B>, <A*D>, <B*A>, <B*D>, and <B*A*D>.
Note that the growth rate is sharper for the number of frequent patterns than that of the execution time. Thus, by decreasing the minimum support, the extra execution time is worthy of the insights gained from the extra frequent patterns.

Table 6: Execution time for \( \text{minSupp} \) equal to 0.0100%, 0.0125%, 0.0150%, 0.0175%, 0.0200% of |D|.

<table>
<thead>
<tr>
<th>database</th>
<th>no. of freq patterns / execution time (second)</th>
</tr>
</thead>
<tbody>
<tr>
<td>I2D100K</td>
<td>1633/125 1295/114 973/102 714/95 579/88</td>
</tr>
<tr>
<td>I2D200K</td>
<td>2920/394 2235/371 1509/352 1287/327 1133/304</td>
</tr>
<tr>
<td>I2D300K</td>
<td>1843/557 1340/511 1062/467 859/426 692/391</td>
</tr>
<tr>
<td>I4D100K</td>
<td>1902/237 1688/211 1326/200 1163/193 747/180</td>
</tr>
<tr>
<td>I4D200K</td>
<td>7460/794 3429/586 2117/502 1648/453 1381/425</td>
</tr>
<tr>
<td>I4D300K</td>
<td>7185/1206 3280/826 1760/687 937/452 589/415</td>
</tr>
</tbody>
</table>

5 CONCLUSIONS

We designed the PHSPAM algorithm to remedy the problems that the set of frequent hybrid sequential patterns obtained by previous researches is incomplete and that the execution time does not scale with growing database sizes. PHSPAM obtains the complete set by first collecting items that might appear in the frequent patterns. PHSPAM then uses the pattern growth techniques to calculate the support of patterns. PHSPAM was implemented and compared with GFP2 and CHSPAM. The experiments demonstrated that PHSPAM indeed obtained more frequent patterns than GFP2. In addition, achieving the same completeness result, the execution time of PHSPAM is better than GFP2 and much better than CHSPAM, due to the accumulated counts preserved in the projected data structures.

Our future research regarding hybrid sequential pattern mining includes:

1. Apply PHSPAM in real world applications, like web page traversal paths mining through web logs.
2. Examine the effect of replacing the support definition such that a transaction could contribute at most one in the support counting of a pattern.
3. Consider application specific constraint of the patterns, like timing limitations.

ACKNOWLEDGEMENTS

This work was supported in part by Taiwan’s National Science Council under Grant NSC : 97-2221-E-032-046.

REFERENCES


