

# Mining Complete Hybrid Sequential Patterns

Chichang Jou

**Abstract**—We discovered that the set of frequent hybrid sequential patterns obtained by previous researches is incomplete, due to the inapplicability of the Apriori principle. We design and implement the CHSPAM algorithm to remedy the problem. CHSPAM first builds the Supplemented Frequent One Sequence itemset (SFOS) to collect items that may appear in a frequent hybrid sequential pattern. It then constructs the projected databases for each item in the SFOS. Its mining procedure is performed recursively in the pattern-growth manner through the projected databases to calculate the support of patterns by backward support counting. We prove the completeness of CHSPAM, and compare the results and performances of CHSPAM with those of GFP2, the most efficient hybrid sequential pattern mining algorithm as far as we know.

## I. INTRODUCTION

WITH the universal deployment of internet and computer technologies, the amount of accumulated electronic data has been growing exponentially. Since data mining techniques aim to extract implicit information inside these data, they have become more and more important in many applications. Sequential pattern mining is one of the successful data mining endeavors. It obtains frequent sequential patterns of items satisfying the condition that the number of their occurrences, called *support*, in the item sequence, called *transaction*, database is greater than or equal to a given threshold, called *minimum support*. The obtained frequent patterns could be applied to analysis and decision making in applications like time-series stock trend, web page traversal, customer purchasing behavior, content signature of network applications, etc.

Based on the different requirements about the adjacency condition of items in a sequential pattern, sequential pattern mining could be classified into the following three categories: The first is to find *continuous patterns* [1, 6, 9], where the adjacent items in a pattern must be consecutive in the transactions. The second is to find *discontinuous patterns* [4, 14], where adjacent items in a pattern are not necessarily consecutive in the transactions. The third is to find *hybrid patterns* [7, 8], where discontinuous patterns are represented by a special wildcard symbol ‘\*’ so that both continuous and discontinuous requirements could be specified in one pattern. We focus on the hybrid sequential pattern mining in this article.

As defined in Chen *et al.* [8], the support of a hybrid

sequential pattern, abbreviated as *pattern* henceforth, is incremented once for each matching instantiation of the pattern in the transactions. Thus, the support of a pattern in a transaction could be greater than 1, and the support of a pattern in a database does not represent the percentage of transactions having the pattern. Suppose the set of all distinct items is of size  $n$ , and the maximal number of items in a transaction is  $m$ . The number of possible patterns without ‘\*’ is  $n^m$ . For each pair of adjacent items in the above patterns, there is a choice of whether to place a ‘\*’ between them. Thus, the total number of patterns that we need to check their support is in the order of  $n^m * 2^{n-1}$ . The problem is computationally complex.

As far as we know, the most efficient hybrid sequential pattern mining algorithm is GFP2 [8]. The basic concept of GFP2 is the Apriori principle that longer frequent patterns could be obtained by filtering candidate frequent patterns of their length, which are generated by shorter frequent patterns. However, we discovered that some frequent patterns could not be obtained by this induction. For example, suppose the database has only one transaction  $T = \langle ABCDBE \rangle$ , and the minimum support is set as 2. Then the only frequent pattern with length 1 is  $\langle B \rangle$ . According to GFP2, the generated candidate frequent patterns from  $\langle B \rangle$  with length 2 are  $\langle B*B \rangle$  and  $\langle BB \rangle$ . However, the supports for patterns  $\langle A*B \rangle$  and  $\langle B*E \rangle$  are 2, and they could not be obtained from filtering  $\langle B*B \rangle$  and  $\langle BB \rangle$ .

We propose a Complete Hybrid Sequential Pattern Mining algorithm, CHSPAM, which utilizes the pattern growth [11] and backward support counting methods. In the pattern growth method, a candidate pattern is built by concatenating a pattern with those patterns obtained from its projected databases. In the backward support counting method, supports of patterns are accumulated segment by segment from the back to the front of a transaction. From shorter patterns to longer patterns, CHSPAM generates for each pattern a projected database of the transactions having the pattern and the array of the pattern’s occurring positions in these transactions. The mining procedure is recursively applied to the projected databases to calculate the support of patterns by backward support counting.

The rest of the paper is organized as follows: Section 2 surveys related work. Problem definition and CHSPAM are illustrated in Section 3. Section 4 proves the completeness of CHSPAM. Results of experimentations are discussed in Section 5. Section 6 concludes the paper and points out future work.

This work was supported in part by Taiwan’s National Science Council under Grant 94-2213-E-032-021.

C. Jou is with the Information Management Department, Tamkang University, Tamsui, Taipei 25137, Taiwan (email:cjou@mail.tku.edu.tw).

## II. RELATED WORK

Agrawal and Srikant [4] extended the frequent itemset mining algorithm [3, 2] for non-serial transactions to discontinuous sequential pattern mining for serial transactions. For non-serial transactions, the Apriori principle states that candidate larger frequent itemsets could be generated by smaller frequent itemsets. Similar to the Apriori principle, they utilized the fact that longer frequent patterns could be quickly obtained from shorter frequent patterns. Pei *et al.* [14] obtained discontinuous sequential web traversal patterns by constructing the access pattern tree from web access logs.

Agrawal *et al.* [1] and Faloutsos *et al.* [9] proposed techniques of Discrete Fourier Transform and Minimum Binding Rectangle, respectively, to obtain frequent similar continuous sequential patterns in a time-series database. Chen *et al.* [6] extended the Apriori principle to obtain frequent continuous sequential web path traversal patterns.

Based on the approach to attack the problem, recent researches about sequential pattern mining could be grossly classified into the following 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 will be scanned in each candidate pattern generation. When the database is very large, this approach needs huge memory space to store the candidate patterns. GSP [5], GFP2 [8], SPADE [15], and ESPM [7] all belong to this category. GSP adds time constraints between adjacent items in a pattern, relaxes the restriction that elements of a pattern must be from the same transaction, and allows items to cross taxonomy levels. GFP2 is the first algorithm to obtain hybrid sequential patterns. It constructs candidate trees and compensation lists to utilize the containment relationship of supports between the patterns with '\*' and those without '\*' to reduce the number of database scans. To avoid repetitive database scanning, SPADE calculates the support of candidate patterns by checking the positions of each item in the transactions. Based on GFP2 and SPADE, ESPM integrates the hash-based DHP algorithm [12] to reduce the number of candidate patterns with length 2.

To skip the time-consuming candidate patterns generation step in Apriori-based algorithms, the methods in the pattern growth category, like FreeSpan [10], utilize the concept from FP-Growth [11] to partition the database and to generate projected databases. For each item  $i$  in the frequent itemset, these methods scan the database to obtain  $i$ -projected-database that consists of after- $i$  sub-transactions for all transactions containing  $i$ . The use of projected databases reduces the memory requirement in scanning the whole database. Pei *et al.* [13] proposed PrefixSpan, which additionally uses bi-level projection to reduce the number of projected databases. PrefixSpan also uses pseudo-projection

to record the transaction id and the position of sequential patterns, instead of the real data, to reduce the memory requirement.

## III. THE CHSPAM ALGORITHM

Our definition of hybrid sequential pattern follows that of GFP2 [8]: Suppose  $I = \{i_1, i_2, \dots, i_n\}$  denotes the set of all items in a database. A transaction  $T = \langle e_1, e_2, \dots, e_k \rangle$  is a sequence of items, where for all  $1 \leq i \leq k$ ,  $e_i \in I$ . A database consists of a set of transactions. The special symbol '\*', not in  $I$ , is used in the definition of patterns to denote the occurrence of 0 or more items.  $X = \langle x_1, x_2, \dots, x_n \rangle$  is called a *hybrid sequential pattern*, abbreviated as *pattern* henceforth when no confusion arises, if :

- 1) for all  $1 \leq i \leq n$ ,  $x_i \in I \cup \{'*\}$ ;
- 2)  $x_1 \in I$  and  $x_n \in I$ ;
- 3) for all  $1 < i < n$ , if  $x_i = '*'$  then  $x_{i-1} \in I$  and  $x_{i+1} \in I$ .

According to the above definition,  $\langle ABC \rangle$ ,  $\langle A*BC \rangle$  and  $\langle A*B*C \rangle$  are legitimate patterns, while  $\langle *AB \rangle$ ,  $\langle AB**C \rangle$  are not. A pattern  $X$  is said to be *contained in a transaction*  $T$ , denoted as  $X \subset T$ , if:

- 1) for all items  $x \in X$ , there is a matching item  $x \in T$ ;
- 2) for all items  $x, y \in X$ , if  $x$  precedes  $y$  in  $X$ , then  $x$ 's matching item in  $T$  also precedes  $y$ 's matching item in  $T$ .
- 3) for all items  $x, y \in X$ , if  $x$  is adjacent to  $y$  in  $X$ , then  $x$ 's matching item in  $T$  is also adjacent to  $y$ 's matching item in  $T$ .

For each pattern  $X$  and transaction  $T$ , the number of different matching instantiations of  $X \subset T$  is called the *support* of  $X$  in  $T$ , and is denoted by  $supp_{X,T}$ . For example, suppose  $X = \langle A*BC \rangle$  and  $T = \langle BACABCC \rangle$ . Then  $supp_{X,T} = 2$ , where the first instantiation matches  $A, B, C$  to the second, fifth, and sixth items in  $T$ , and the second instantiation matches them to the fourth, fifth, and sixth items. For a given minimum support threshold  $minSupp$ , we call  $X$  a *frequent pattern with respect to minSupp*, if  $X$  satisfies the condition:

$$\sum_{\forall T} supp_{X,T} \geq minSupp$$

As illustrated in Section 1, by generating candidate frequent patterns iteratively from short to long patterns, GFP2 might miss frequent patterns containing items not in the short patterns. We propose CHSPAM to remedy the problem. Figure 1 displays the pseudo code of CHSPAM. Especially note that CHSPAM constructs the *Supplemented Frequent One Sequence* itemset, denoted as *SFOS*, to collect items that may appear in a frequent hybrid sequential pattern. It then performs recursive backward support counting for all possible patterns through the reduced database, which contains only the SFOS items. CHSPAM has four major

steps: Step 1 scans the database and generates the SFOS. Step 2 transforms items not in the SFOS into '+' to reduce database, which helps decrease the number of comparisons later. Step 3 generates the projected databases for each SFOS item. Step 4 performs recursive backward support counting in the projected databases, and finally checks whether the SFOS items are frequent patterns. We will discuss each step in the following subsections.

```

/* D is the original database */
1) S = GenerateSFOS(D)
   /* S is the supplemented frequent one sequence itemset of D */
2) D' = ReduceDataBase(S,D) /* D' is the reduced database of D */
3) ConstructProjectedDatabases(S, D')
   /* Construct projected database for each item in S */
4) for each item i ∈ S
   ni = TraverseAndCount(i)
       /* recursively obtain frequent patterns starting with <i>
       and returns the support for <i> */
   if ni ≥ minSupp then
       add <i> to the set of frequent patterns
   endfor
    
```

Fig. 1. Pseudo Code of CHSPAM

### A. Generating the Supplemented Frequent One Sequence Itemset

The length of a pattern  $X$  is defined to be the number of items occurring in both  $X$  and  $I$ . For example, the length of  $\langle A * B * C \rangle$  is 3. In hybrid sequential pattern mining, frequent patterns with length 1 are actually 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 with length longer than 1. CHSPAM remedies the problem by constructing the SFOS  $S$  in the first step  $GenerateSFOS(D)$ , where  $D$  is the original database.  $S$  includes the following items:

- 1) traditional frequent one sequence items, as defined in the frequent itemset mining algorithms [3, 2];
- 2) all items occurring more than once in a single transaction;
- 3) and all items before and after the pair of repetitive items.

For example, suppose  $minSupp$  is set as 2. In the database with exactly one transaction  $T = \langle ABCDBE \rangle$ ,  $B$  is the only frequent one sequence item. Items  $A$  and  $E$  are added into the SFOS since  $A$  occurs before the first  $B$  and  $E$  occurs after the second  $B$ . According to the definition of SFOS, if an item occurs more than two times in a transaction, then all items in that transaction will be added into the SFOS.

### B. Database Reduction

Before counting the support for patterns, the second step

of CHSPAM,  $ReduceDataBase(S,D)$ , transforms each serial non-SFOS items in the original database  $D$  into a '+', to obtain  $D'$ , the reduced database of  $D$ . Since frequent patterns only contain items in the SFOS, which will be proved in Section 4, this reduction does not change the result of frequent pattern mining. Since '+' only appears in the middle of a pattern, '+'s appearing in the prefix or suffix of a reduced transaction would also be deleted. For example, suppose the sample database is as displayed in Table I, and  $minSupp$  is set as 2. By scanning the database once, the SFOS in Table II is constructed. By scanning the database once more, the reduced sample database in Table III is constructed. This transformation will reduce the number of database scanning in generating projected databases.

tid	Item
1	AENCAG
2	CGBAGD
3	KBGAC
4	BIGHLMG
5	BGACAFJ

Item	Support
A	6
B	4
C	4
D	1
F	1
G	7
I	1
J	1

tid	Item
1	A+CAG
2	CGBAGD
3	BGAC
4	BIG+G
5	BGACAFJ

### C. Generating Projected Databases for Each SFOS Item

$ConstructProjectedDatabases(S, D')$ , the third step of CHSPAM, constructs a projected database for each item in  $S$  from the reduced database  $D'$ . For each item  $i$  in the SFOS  $S$ , if  $i$  occurs in a transaction  $T$ , this procedure pushes the pair of  $T$ 's transaction id and the array of  $i$ 's positions in  $T$  into  $i$ 's projected database. Since for each pattern, each distinct matching instantiation of items in a transaction is counted as one support, we need to scan the whole transaction in  $D'$ . For example, Figure 2 displays the projected databases of the reduced sample database in Table III. Since the support counting procedure is independent for each SFOS item, CHSPAM keeps only one projected database in the memory at one time and saves all the other projected databases in the hard disk.

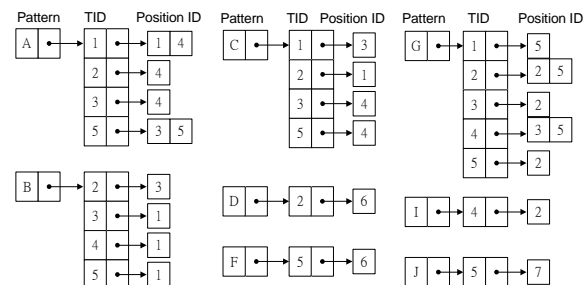


Fig. 2. Projected Databases for each SFOS Item

### D. Recursive Traverse and Count

For a pattern  $\langle i \rangle$ , Figure 3 demonstrates the pseudo code

of the fourth step of CHSPAM,  $TraverseAndCount(i)$ . It first computes by  $GetSupport(i)$ , which will be explained in the next paragraph, the support count mapping  $F_i$  for items  $f$  in  $i$ 's projected database  $P_i$  to the supports of  $\langle i^*f \rangle$  and  $\langle if \rangle$  respectively. Then for all items  $f$  in the domain of  $F_i$ , the function  $BuildProjectedDB(i,f)$  builds projected databases for patterns  $\langle if \rangle$  and  $\langle i^*f \rangle$  from  $i$ 's projected database  $P_i$ . The support for  $\langle i^*f \rangle$  and  $\langle if \rangle$  will be checked to decide whether  $\langle i^*f \rangle$  and  $\langle if \rangle$  are frequent patterns.  $TraverseAndCount$  will then be recursively applied to  $\langle i^*f \rangle$ . Since the projected database of  $\langle if \rangle$  is a subset of the projected database of  $\langle i^*f \rangle$ , if  $TraverseAndCount(\langle i^*f \rangle)$  returns 0, then  $TraverseAndCount(\langle if \rangle)$  will return 0 too. Thus,  $TraverseAndCount$  is applied to  $\langle if \rangle$  only if  $TraverseAndCount(\langle i^*f \rangle)$  returns a value greater than 0. It finally returns the number of frequent items in  $F_i$ .

```

/* i denotes a pattern <i> */
/* Pi is the projected database of <i> */
/* itemcount is the recursively returned value for TraverseAndCount */
1) Function TraverseAndCount(i) as Integer
2)   Fi = GetSupport(i)
   /* Fi is the mapping of the items in Pi to its support */
3)   if Fi is empty then return 0
4)   for each item f ∈ Fi
       BuildProjectedDB(i, f)
       /* build projected databases Pi*f and Pif */
       if support of <i*f> ≥ minSupp then
           add <i*f> to the set of frequent patterns
       endif
       if support of <if> ≥ minSupp then
           add <if> to the set of frequent patterns
       endif
       itemcount = TraverseAndCount(<i*f>)
       if itemcount > 0 then
           TraverseAndCount(<if>)
       endif
   endfor
   return the number of frequent items in Fi
End Function

```

Fig. 3. Pseudo Code of TraverseAndCount

Since the scanning process for a projected database will encounter repetitive items in a transaction, to construct the support count mapping  $F_i$ , we scan the transactions segment by segment backwardly in  $GetSupport(i)$  as follows:

For each projected transaction  $T_p$  in the projected database  $P_i$ , let the two variables  $sp$  and  $ep$  denote the starting and ending positions of the to-be-matched item segment of  $T_p$ . Suppose the position array  $PA$  of  $T_p$  is of size  $j$ , and the value of the last element of  $PA$ ,  $PA[j]$ , is  $v$ . The initial value of  $sp$  is set as  $v+1$ , and the initial value of  $ep$  is set as the position of the last item of  $T_p$ . Thus, the first to-be-matched segment is

the last segment of  $T_p$ . Since the values in  $PA$  are monotonically increasing, the items in this segment of  $T_p$  would be matched  $j$  times differently, each by a distinct matching of the last item of  $\langle i \rangle$  to the item in the  $PA[1]$ -th,  $PA[2]$ -th, ...,  $PA[j]$ -th position of  $T_p$ . Thus, for items  $x$  in this last segment, the supports of  $x$  are multiplied by  $j$ , and then are added to  $F_i$ 's final supports of  $x$ . After calculating supports for all items in the last segment, the computation is then moved backward to be applied to the next-to-last segment by setting  $ep$  as the current value of  $sp$ , and setting  $sp$  as the value of  $PA[j-1]+1$ . The same backward support counting procedure continues until the projected transaction  $T_p$  is scanned completely. In  $F_i$ , the support count for an item  $x$  is the sum of supports of  $x$  over all projected transactions, which is equal to the support of the pattern  $\langle i^*x \rangle$  in the original database. The same method is used to compute the support for pattern  $\langle ix \rangle$ , where a support for  $x$  would be added to the support count mapping if it satisfies the additional requirement that in the matching instantiation the last element of  $\langle i \rangle$  be adjacent to  $x$ .

For the example in Figure 2, item  $A$  occurs twice in positions 3 and 5 in the projected transaction 5, which has 7 items, of the reduced database. In calculating the support count backward for the items in the projected database of pattern  $\langle B^*A \rangle$ ,  $sp$  and  $ep$  of the first to-be-matched segment are set as 6 and 7, respectively. Items in this last segment are  $F$  and  $J$ . Their support will be multiplied by 2. After that,  $sp$  and  $ep$  for the next to-be-matched segment is updated as 4 and 5. The support for items in this segment, which are  $C$  and  $A$ , will be multiplied by 1. Take item  $J$  as an example in constructing the support count mapping  $F_{\langle B^*A \rangle}$ . The support of  $J$  in  $F_{\langle B^*A \rangle}$  is summed over all projected transactions to obtain 2. Thus, the support of pattern  $\langle B^*A^*J \rangle$  is 2. Since  $A$  is not adjacent to  $J$  in the projected transactions, the support of  $\langle B^*A^*J \rangle$  is 0.

For pattern  $\langle i \rangle$ , in  $TraverseAndCount(i)$ , for an item  $f$  from the projected database  $P_i$ ,  $BuildProjectedDB(i, f)$  builds the projected databases  $P_{i^*f}$  and  $P_{if}$ . Note that the occurring positions associated with the items in these projected databases are their positions in the projected databases constructed in Step 3. Therefore, this procedure could be performed by projecting all items after the pattern  $\langle i \rangle$  for all transactions in  $P_i$ . If  $f$  occurs in the projected database  $P_i$ , then add the matched transaction id and the corresponding position into  $P_{i^*f}$ . If the position of  $f$  is one larger than the position of  $i$  in the transaction, then add the matched transaction id and the corresponding position into  $P_{if}$ . It continues to build up projected databases for longer patterns in the recursive call until no more frequent patterns could be found in the projected databases.

#### IV. PROOF OF THE COMPLETENESS OF CHSPAM

We use induction to prove that the set of frequent patterns

obtained by CHSPAM is complete.

**Lemma 1:** All items occurring in any frequent pattern with length 1 or 2 are in the SFOS obtained in CHSPAM.

**Proof:** As stated in Section III.A, frequent patterns with length 1 are the frequent itemset with length 1. Since the SFOS by definition is a superset of the frequent itemset with length 1, the lemma for frequent items with length one is clear. For frequent patterns with length equal to 2, we use contradiction to prove the two possible patterns  $\langle x_1 * x_2 \rangle$  and  $\langle x_1 x_2 \rangle$  for some items  $x_1$  and  $x_2$ .

Suppose there exists one frequent pattern  $X = \langle x_1 * x_2 \rangle$  such that at least one of  $x_1$  and  $x_2$  are not in the SFOS. There are three cases to be discussed:

- 1) Neither  $x_1$  nor  $x_2$  is in the SFOS: Then by the definition of SFOS, neither of them occurs in transactions with duplicate items. Therefore, only transactions containing both  $x_1$  and  $x_2$  would contribute one in the support counting. Since  $X$  is frequent pattern, both  $x_1$  and  $x_2$  are frequent patterns, which is a contradiction with the fact that neither  $x_1$  nor  $x_2$  is in the SFOS.
- 2)  $x_1$  is in the SFOS and  $x_2$  is not in the SFOS: From the definition SFOS,  $x_2$  is not in the frequent one sequence itemset. That means

$$\sum_{\forall T} \text{supp}_{\langle x_2 \rangle, T} < \text{minSupp}$$

Since  $\langle x_1 * x_2 \rangle$  is a frequent pattern, we have

$$\sum_{\forall T} \text{supp}_{\langle x_1 * x_2 \rangle, T} < \text{minSupp}$$

There must exist at least one transaction  $T = \langle y_1 \dots y_n \rangle$  contributing more than 1 in the support counting for  $\langle x_1 * x_2 \rangle$ . Then, there must exist  $n_1$ ,  $n_2$ , and  $n_3$  satisfying the conditions that  $n_1 < n_2 < n_3$ , and

$$y_{n_1} = x_1, y_{n_2} = x_1, y_{n_3} = x_2.$$

By definition,  $x_2$  is in the SFOS. This is a contradiction with  $x_2$  is not in the SFOS.

- 3)  $x_2$  is in the SFOS and  $x_1$  is not in the SFOS: This case can be proved by the same reasoning as case 2.

Therefore, there does not exist any frequent pattern  $X = \langle x_1 * x_2 \rangle$  such that at least one of  $x_1$  and  $x_2$  are not in the SFOS. In other words, if  $\langle x_1 * x_2 \rangle$  is a frequent pattern, then both  $x_1$  and  $x_2$  would be in the SFOS. The proof for the patterns  $\langle x_1 x_2 \rangle$  is similar.  $\square$

**Theorem 1:** CHSPAM finds all frequent patterns.

**Proof:** With Lemma 1, this theorem could be proved by induction on the length of the frequent patterns. Detailed proof is omitted.  $\square$

## V. EXPERIMENTATION

We implement both the GFP2 and CHSPAM algorithms,

and compare the number of obtained frequent patterns and the execution time for each experiment. The experiments are all performed in a desktop computer with the Windows XP operating system, AMD Athlon XP-A 2500 CPU, and 1024MB DDR memory. The programs of GFP2 and CHSPAM are both written in VB.NET 2003. The simulated databases, where transaction items are generated according to the Poisson distribution, are constructed following the method used in [3]. The databases are stored in the MS SQL Server 2000. The parameters used in generating the databases are as follows:

- 1)  $|T|$ : the average number of items in the transactions.
- 2)  $|I|$ : the average length of frequent patterns.
- 3)  $|D|$ : the number of transactions in the database.
- 4)  $|L|$ : the number of items in the frequent one sequence itemset.
- 5)  $|N|$ : the number of distinguished items.

The experiments are performed for 12 different databases, all with fixed settings of  $|N|=10000$ ,  $|L|=1500$ ,  $|T|=15$ , and  $\text{minSupp}$  equal to 0.015% of  $|D|$ . The parameters for  $|D|$  are 10K, 30K, 50K, 100K, 200K, and 300K. The parameters for  $|I|$  are 2 and 4. The number of obtained frequent patterns and their execution time for each database setting are displayed in Table IV. In all 12 settings, the number of frequent patterns obtained by CHSPAM is greater than or equal to that by GFP2. By examining the transactions containing the extra frequent patterns generated from CHSPAM, we confirm that added items in the SFOS indeed help the finding of extra frequent patterns. Note that the number of frequent patterns by CHSPAM in T1514D300K is 50% more than that by GFP2. There is no difference in T1512D30K, while the

TABLE IV  
RESULTS OF GFP2 AND CHSPAM FOR DATABASES WITH  $|N|=10000$ ,  
 $|L|=1500$ ,  $|T|=15$ , AND  $\text{MINSUPP}$  EQUAL TO 0.015% OF  $|D|$

Database	Number of Frequent Patterns		Execution Time (second)	
	GFP2	CHSPAM	GFP2	CHSPAM
T1512D10K	969	982	24	154
T1512D30K	959	959	64	445
T1512D50K	972	977	106	878
T1512D100K	968	973	266	2285
T1512D200K	1506	1509	654	3162
T1512D300K	1059	1062	903	7160
T1514D10K	1234	1247	21	160
T1514D30K	1389	1411	58	1533
T1514D50K	1325	1335	97	1079
T1514D100K	1315	1326	222	3356
T1514D200K	2108	2117	571	5390
T1514D300K	1535	2331	816	11059

differences in the other cases are very little. This shows that the behavior of the hybrid sequential pattern mining is probabilistic. Figure 4 displays the execution time in these 12 parameter settings. The execution time of CHSPAM grows exponentially with respect to the database size.

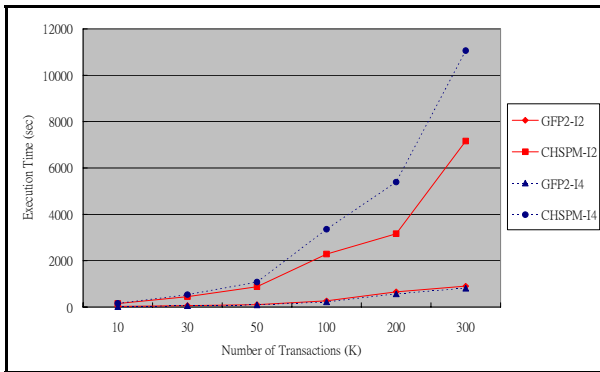


Fig. 4. Execution Time of GFP2 and CHSPAM

## VI. CONCLUSION

We discovered that the set of frequent hybrid sequential patterns obtained by previous researches is incomplete, due to the inapplicability of the Apriori principle. We designed the CHSPAM algorithm to remedy the problem. CHSPAM obtains the complete set by first constructing the supplemented frequent one sequence itemset to collect items that might appear in the frequent patterns. CHSPAM then uses the techniques of pattern growth and backward support counting to calculate the support of patterns. We proved by contradiction and by induction that the set of frequent patterns obtained by CHSPAM is complete. CHSPAM was implemented and compared with GFP2, the most efficient hybrid sequential pattern mining algorithm as far as we know. The experiments demonstrated that CHSPAM indeed obtained more frequent patterns than GFP2 in most cases. However, to achieve the completeness result, the execution time of CHSPAM decays exponentially with respect to the database size.

Our future research regarding hybrid sequential pattern mining includes:

- 1) Identify the characteristics of the extra frequent patterns obtained by CHSPAM to explore the possibilities of making use of the results of GFP2 to obtain better execution time performance.
- 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. This definition would be useful in applications focusing on the percentage of transactions satisfying a pattern.
- 3) Allow the user to specify the pattern or the constraint of the expected pattern to make the system more interactive.

- 4) Explore the possibility of reducing the workload of CHSPAM through the technologies of distributed systems.

## ACKNOWLEDGMENT

The author would like to thank Hsiao-Ren Yuan for implementing part of the GFP2 and CHSPAM algorithms.

## REFERENCES

- [1] R. Agrawal, C. Faloutsos, and A. Swami, "Efficient similarity search in sequence databases," in *Proc. of the 4th International Conference on Foundations of Data Organization and Algorithms*, Chicago, U.S.A., pp. 69-84, 1993.
- [2] R. Agrawal, T. Imielinski, and A. Swami, "Mining association rules between sets of items in large databases," in *Proc. of the 1993 ACM SIGMOD International Conference on Management of Data*, Washington D.C., U.S.A., pp. 207-216, 1993.
- [3] R. Agrawal and R. Srikant, "Fast algorithm for mining association rules," in *Proc. of the 20th International Conference on VLDB*, Santiago, pp. 487-499, 1994.
- [4] R. Agrawal and R. Srikant, "Mining sequential patterns," in *Proc. of the 11th International Conference on Data Engineering*, Taipei, Taiwan, pp. 3-14, 1995.
- [5] R. Agrawal and R. Srikant, "Mining sequential patterns: generalizations and performance improvements," In: P. Apers, M. Bouzeghoub, G. Gardarin (eds.): *Lecture Notes in Computer Science*, Vol. 1057, pp. 3-17, 1996.
- [6] M. Chen, J. S. Park, and P. S. Yu, "Efficient data mining for path traversal patterns," *IEEE Trans. Knowledge Data Engineering*, Vol. 10(2), pp. 209-221, 1998.
- [7] Y. Chen, "An efficient sequential pattern mining system," Tamkang University, Taiwan Master Thesis, 2004.
- [8] Y. L. Chen, S. S. Chen, P. Y. Hsu, "Mining hybrid sequential patterns and sequential rules," *Information Systems*, Vol. 27(5) pp. 345-362, 2002.
- [9] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos, "Fast subsequence matching in time-series databases," in *Proc. of the 1994 ACM SIGMOD International Conference on Management of Data*, Minneapolis, U.S.A., pp. 419-429, 1994.
- [10] J. Han, J. Pei, B. Mortazavi-Asl, Q. Chen, U. Dayal, and M-C. Hsu, "Freespan: frequent pattern-projected sequential pattern mining," in *Proc. of the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Boston, U.S.A., pp. 355-359, 2000.
- [11] J. Han, J. Pei, and Y. W. Yin, "Mining frequent patterns without candidate generation," in *Proc. of the 2000 ACM SIGMOD International Conference on Management of Data*, New York, U.S.A. pp. 1-12, 2000.
- [12] J. S. Park, M. Chen, and P. S. Yu, "An effective hash based algorithm for mining association rules," in *Proc. of the 1995 ACM SIGMOD International Conference on Management of Data*, San Jose, U.S.A., pp. 175-186, 1995.
- [13] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and M. C. Hsu, "PrefixSpan: mining sequential patterns efficiently by prefix projected pattern growth," in *Proc. of the 17th International Conference on Data Engineering*, Heidelberg, Germany, pp. 106-115, 2001.
- [14] J. Pei, J. Han, B. Mortazavi-Asl, and H. Zhu, "Mining access patterns efficiently from web logs," in *Proc. of the 4th Pacific-Asia Conference on Knowledge Discovery and Data Mining*, Kyoto, Japan, pp. 396-407, 2000.
- [15] M. J. Zaki, "SPADE: an efficient algorithm for mining frequent sequences," *Machine Learning*, Special Issue on Unsupervised Learning, Vol. 42(1-2), pp. 31-60, 2001.