Improved Sequential Pattern Mining using an Extended Bitmap Representation

Chien-Liang Wu, Jia-Ling Koh*, and Pao-Ying An

Department of Information and Computer Education
National Taiwan Normal University
Taipei, Taiwan 106, R.O.C
*Email: jlkoh@ice.ntnu.edu.tw

Abstract. The main challenge of mining sequential patterns is the high processing cost of support counting for large amount of candidate patterns. For solving this problem, SPAM algorithm was proposed in SIGKDD'2002, which utilized a depth-first traversal on the search space combined with a vertical bitmap representation to provide efficient support counting. According to its experimental results, SPAM outperformed the previous works SPADE and PrefixSpan algorithms on large datasets. However, the SPAM algorithm is efficient under the assumption that a huge amount of main memory is available such that its practicability is in question. In this paper, an Improved-version of SPAM algorithm, called I-SPAM, is proposed. By extending the structures of data representation, several heuristic mechanisms are proposed to speed up the efficiency of support counting further. Moreover, the required memory size for storing temporal data during mining process of our method is less than the one needed by SPAM. The experimental results show that I-SPAM can achieve the same magnitude efficiency and even better than SPAM on execution time under about half the maximum memory requirement of SPAM.

1. Introduction

The problem of mining sequential patterns was first introduced by Agrawal and Srikant in [2]: Given a database of data-sequences, the problem is to find all sequential patterns with a user-defined minimum support, also named frequent sequential patterns. The main challenge of mining sequential patterns is the high processing cost of support counting for large amount of candidate patterns.

Many studies have proposed methods for solving this problem [2, 3, 5, 6, 7]. Among the related works, Apriori-ALL[1], GSP[6], and SPADE[7] algorithms all belong to Apriori-like algorithms. An Apriori-like method finds all frequent items first. By adopting multi-pass approach, the candidate patterns with length l are generated from the frequent patterns with length (l-1) in each iteration. Then the supports of these candidate patterns are checked to discover frequent patterns with length l. The Apriori-like sequential pattern mining methods suffer from the costs to handle a potentially huge set of candidate patterns and scan the database repeatedly. For solving these problems, PrefixSpan algorithm, originated from FreeSpan [4], was proposed in [5]. PrefixSpan was designed based on divide-and-conquer

scheme. An elegant recursive method was presented to create projected databases where each one has the same prefix subsequence. By growing local frequent prefix subsequences in each projected database recursively, all the sequential patterns were discovered. Although PrefixSpan prevented from generating unnecessary candidate patterns, the cost of constructing projected databases recursively was a burden when processing large databases.

To further speed up the efficiency of support counting, SPAM algorithm was proposed in [3]. In SPAM, a vertical bitmap representation was created for each item to record its appearing information in a sequence. Then, a depth-first traversal strategy was adopted for generating candidate patterns. By performing bitwise operations on the bitmaps, the supports of candidate patterns were obtained quickly. In addition, an effective pruning mechanism was employed in SPAM to reduce the number of generated candidates. According to its experimental results, SPAM outperformed not only SPADE but also PrefixSpan for large databases. However, the SPAM algorithm is efficient under the assumption that a huge amount of main memory is available such that its practicability is in question.

In this paper, an Improved-version of SPAM algorithm, called I-SPAM, is proposed for mining frequent sequential patterns efficiently. By extending the structures of bitmap data representation, an appearing sequence table is constructed additionally. Based on the modified data representation, several heuristic mechanisms are proposed to speed up the efficiency of support counting further. Moreover, the required memory size for storing temporal data during performing depth-first traversal on the search space is less than the one of SPAM. The experimental results show that I-SPAM can achieve the same magnitude efficiency and even better than SPAM on execution time under about half the maximum memory requirement of SPAM.

The remaining of this paper is organized as follows. We define the related terms for the problem of mining sequential patterns in Section 2. The designed structures of data representation are introduced in Section 3. Then I-SPAM algorithm is developed in Section 4. In Section 5, the experimental results of performance evaluation by comparing I-SPAM with SPAM are reported. Finally, we summarize the contribution of the proposed method and discuss further research issues in Section 6.

2. Preliminaries

The problem of mining sequential patterns was originally proposed by [2]. The following definitions refer to [2, 3, 5, 6].

Let $I=\{i_1, i_2,\ldots, i_n\}$ be the set of all possible **items** in a specific domain. A subset of I is called an **itemset**. A sequence $\alpha = \langle s_1 s_2 \ldots s_l \rangle$ is an ordered list of itemsets, where s_j is an itemset. Each s_j in a sequence is called an **element** of the sequence and denoted as $(x_1 x_2 \ldots x_m)$, where x_k is an item. For brevity, the brackets are omitted if an element has only one item, i.e., (x) is written as x. The number of instances of items in a sequence α is called the length of the sequence and denoted as $|\alpha|$. A sequence with length l is called an ***l*–sequence**. A sequence $\alpha = \langle a_1 a_2 \ldots a_n \rangle$ is called a **subsequence** of another sequence $\beta = \langle b_1 b_2 \ldots b_m \rangle$ and β a **supersequence** of α, denoted as $\alpha \sqsubseteq \beta$, if there exist integers $1 \le j_1 < j_2 < \ldots < j_n \le m$ such that $a_1 \subseteq b_{j1}$, $a_2 \subseteq b_{j2}$, …, and $a_n \subseteq b_{jn}$.

A **sequence database** *S* is a set of tuples, where each tuple: [*sid*, *s*] consists of a sequence, *s*, and the identification of sequence, *sid*. A tuple [*sid*, *s*] is said to contain a sequence α, if α is a subsequence of *s*. |*S*| denotes the number of sequences in sequence database *S*. The **support** of a sequence α in database *S* is the number of tuples in the database containing α, denoted as $sup_S(\alpha)$. Given a positive integer, *min_sup*, as the **support threshold**, a sequence α is called a **frequent sequential pattern** in database *S* if $sup_S(\alpha) \geq min\_sup$. Otherwise, the pattern is **infrequent**. The problem of mining sequential patterns is to find all frequent sequential patterns from a sequential database *S*.

3. Bit Sequence Representation

In our approach, for each sequence α in a sequence database *S*, a bit sequence table is constructed. In the table, each item X contained in sequence α has a corresponding bit sequence, denoted as $Bit_X(\alpha)$. The length of $Bit_X(\alpha)$ equals the number of elements in α. If item X is in the *j*-th element of α, the *j*-th bit of $Bit_X(\alpha)$ is set to be 1; otherwise, it is set to be 0. The bit sequence tables of all the sequences in sequence database *S* collectively represent the contents of sequences in the database.

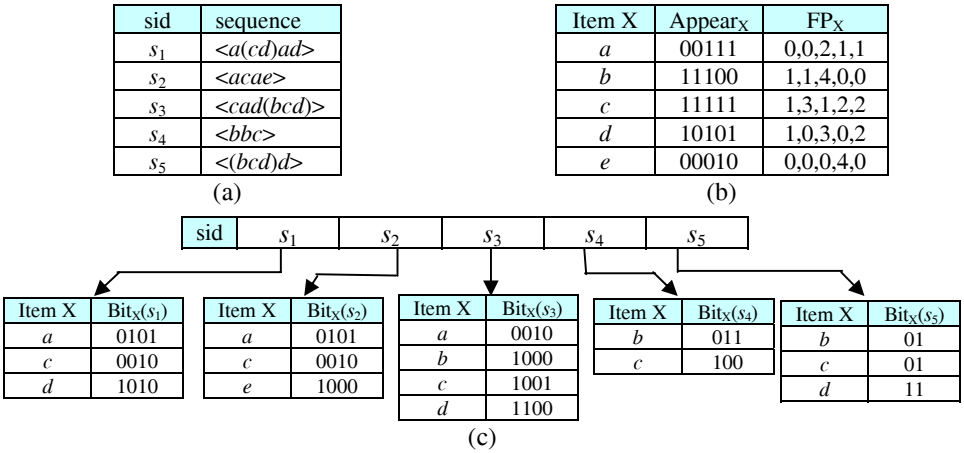

Fig. 1. (a) the sample database *S*, (b) the appearing sequence table of *S*, and (c) the bit sequence structure of *S*.

[Example 1] Consider sequence s_1=<*a*(*cd*)*ad*> shown in Fig. 1(a), which consists of four elements: *a*, (*cd*), *a*, and *d*. Because item *a* appears in the 1st and the 3rd elements of s_1, the bit sequence of *a* in s_1, denoted as $Bit_a(s_1)$, is 0101. Similarly, $Bit_c(s_1)$= 0010 and $Bit_d(s_1)$= 1010 are obtained. The bit sequence tables of all the five sequences in *S* are constructed as shown in Fig. 1(c).

[Definition 1] Given a sequence α=<$a_1a_2\ldots a_n$>. If α is contained in a tuple [s_i, β] of the sequence database where β=<$b_1b_2\ldots b_m$>, there must exist integers $1 \leq j_1 < j_2 < \ldots < j_n \leq m$ such that $a_1 \subseteq b_{j1}$, $a_2 \subseteq b_{j2}$, …, and $a_n \subseteq b_{jn}$. The integer j_n is named a **sequential position** of α in sequence s_i. The **first sequential position** of α in s_i is defined to be the minimum value among all the sequential positions of α in s_i. Otherwise, if α is not contained in sequence s_i, the first sequential position of α in s_i is 0.

In order to reduce the cost of checking bit sequence tables, an appearing sequence table is constructed in our approach, which is composed of three fields: **item name**, **appearing sequence**, and **first_position sequence**. The appearing sequence of an item X is a bit sequence with length $|S|$, denoted as Appear_X, which is used to record whether item X appears in the sequences of database S. If X appears in the i-th sequence of S, the i-th bit in Appear_X, denoted as $\text{Appear}_X(i)$, is set to be 1; otherwise, it is set to be 0. In addition, for each item X, an integer sequence called first_position sequence, is constructed. The sequence is denoted as FP_X, which consists of $|S|$ nonnegative integers, from right to left, used to record the first sequential positions of <X> in every sequence of the database.

[Example 2] Consider the example shown in Fig. 1(a). Item d appearing in sequences s_1, s_3, and s_5, thus, Appear_d ="10101" and FP_d = "1,0,3,0,2" are constructed. The whole appearing sequence table of S is shown as Fig. 1(b).

The representation of appearing sequence and first position sequence are applicable to represent the distribution of a sequential pattern contained in the database. For example, pattern P=<ad> is contained in sequences s_1 and s_3. Therefore, Appear_P="00101" and FP_P="0,0,3,0,2". Accordingly, if the appearing sequence of a pattern Q is known, the number of bits with 1 in Appear_Q, denoted as **1_count**(Appear_Q), implies the support of Q.

4. I-SPAM Algorithm

In this section, based on the representations of appearing sequences, the strategy for computing the supports of candidate patterns efficiently is introduced. Then the mining process of the proposed I-SPAM algorithm is described.

4.1 Candidate Patterns Generation

According to the monotonic property of frequent patterns, a pattern is possible frequent only if all its subsequences are frequent. Therefore, a candidate pattern is generated by inserting a data item into a pre-known frequent pattern.

Given a data item T in the database, the S-extended method generates a candidate sequence by appending a new element containing itemset {T} after the last element of a sequence α. The generated pattern is named a S-extended sequence of α. On the other hand, an I-extended sequence of α is obtained by inserting a data item T to the last element X of α. These two patterns are named the S-extended and I-extended sequences of α by T, respectively. For example, suppose sequence α=<aa> and b denotes a data item. Then <aab> is the S-extended sequence and <a(ab)> is the I-extended sequence of α by b.

4.2 Support Counting Strategies

The appearing and first_position sequences, introduced in the previous section, are used to speed up the support counting of candidate patterns.

1) Checking S-extended Sequences

Let β denote a S-extended sequence of α by appending a new element containing itemset {T} to α. Suppose $Appear_\alpha$ and FP_α are given and the appearing sequence table of the database is constructed. The appearing sequence of the new pattern β, $Appear_\beta$, must have the properties that if bit *j* has value 1, then the corresponding sequence s_j must contain sequence α and {T}, and there exists α before T in the sequence. After getting $Appear_\beta$, the support of β is obtained easily. The following strategies are designed to get the appearing sequence of β efficiently by avoiding the non-necessary checking on bit sequence tables as far as possible.

First, the approximation of $Appear_\beta$, denoted as A_Appear_β, is obtained by performing an AND operation on $Appear_\alpha$ and $Appear_T$. Because $1\_count(A\_Appear_\beta)$ is larger than or equal to $1\_count(Appear_\beta)$, β is not possible a frequent pattern if $1\_count(A\_Appear_\beta) <$ *min_sup* and it is pruned without needing further checking.

On the other hand, if $1\_count(A\_Appear_\beta) \geq$ *min_sup*, for those bits in A_Appear_β with value 1, the corresponding sequences have to be checked whether they contain β actually to get $Appear_\beta$. For each bit *k* in A_Appear_β, if its value is 0, both $Appear_\beta(k)$ and $FP_\beta(k)$ are set to be 0. Otherwise, the values in $FP_\alpha(k)$ and $FP_T(k)$ are compared. If $FP_\alpha(k)$ is less than $FP_T(k)$, it implies the first sequential position of sequence α appearing is before all the occurring of item T in the *k*-th sequence. In other words, the *k*-th sequence of *S* contains the new pattern β and $FP_T(k)$ is the first sequential position that β occurring in this sequence. Therefore, $Appear_\beta(k)$ is set to be 1 and $FP_\beta(k)$ is set to be $FP_T(k)$. Although, it is not necessary that β does not occur in the *k*-th sequence if $FP_\alpha(k)$ is larger than or equal to $FP_T(k)$. Therefore, the following detailed checking on the bit sequence $Bit_T(k)$ is executed.

A right-shift operation is performed on $Bit_T(k)$ by $FP_\alpha(k)$ bits. If the resultant sequence is non-zero, it means there existing a position where item T located after the first sequential position of α in the *k*-th sequence. That is, β is contained in the *k*-th sequence. Let bit *h* denote the first bit in the resultant sequence with value 1, it indicates the first sequential position of β in the sequence is located *h* positions after the first sequential position of α. Therefore, $Appear_\beta(k)$ is set to be 1 and $FP_\beta(k)$ is set to be $FP_\alpha(k)+h$. Otherwise, sequence β is not contained in the *k*-th sequence of *S*, and both of $Appear_\beta(k)$ and $FP_\beta(k)$ are set to be 0.

The appearing sequence of β, $Appear_\beta$, is obtained after performing the checking for all the bits in A_Appear_β with value 1. Finally, β is certified to be a frequent pattern if $1\_Count(Appear_\beta)$ is larger than or equal to *min_sup*.

2) Checking I-extended Sequences

Let sequence α be represented as <α'X>, where X denotes the last element of α. Besides, let γ denote an I-extended sequence of α by inserting item T to the last element of α. For the appearing sequence of γ, $Appear_\gamma$, if its *j*-th bit has value 1, sequence α and {T} must be contained in the corresponding sequence s_j. Moreover, there exists an element containing both X and {T}, which is located after α' in the sequence.

Similarly, the approximation of $Appear_\gamma$, denoted as A_Appear_γ, is obtained by performing an AND operation on $Appear_\alpha$ and $Appear_T$. The candidate pattern γ is not possible frequent if $1\_count(A\_Appear_\gamma)<min\_sup$ and it is pruned.

On the other hand, if $1\_count(A\_Appear_\gamma)\geq min\_sup$, for those bits in A_Appear_γ with value 1, the corresponding sequences have to be checked whether they contain γ actually to get $Appear_\gamma$. For each bit k in A_Appear_γ, if its value is 0, $Appear_\gamma$ is set to be 0. Otherwise, the values in $FP_\alpha(k)$ and $FP_T(k)$ are compared. If $FP_\alpha(k)$ is equal to $FP_T(k)$, it implies there exists an element within the k-th sequence which contains both {T} and the last element X of α. Besides, the element is located after α' because $FP_{\alpha'}(k) < FP_\alpha(k)$. In other words, the new pattern γ is contained in the k-th sequence of S and $FP_\alpha(k)$ is the first sequential position of γ in this sequence. Therefore, $Appear_\gamma(k)$ is set to be 1 and $FP_\gamma(k)$ is set to be $FP_\alpha(k)$.

However, it is not necessary that γ does not occur in the k-th sequence if $FP_\alpha(k)$ is larger or less than $FP_T(k)$. Therefore, the following detailed checking on the bit sequences $Bit_X(k)$ and $Bit_T(k)$ is executed. First, $Bit_X(k)$ is obtained by performing AND operations on $Bit_{x1}(k)$, $Bit_{x2}(k)$, …, and $Bit_{xi}(k)$,where x_i is an item in X.

Then, another AND operation is performed on $Bit_X(k)$ and $Bit_T(k)$ to get $Bit_{(X\cup\{T\})}(k)$. If the resultant sequence is non-zero, it indicates that both X and {T} appear in certain element in the k-th sequence at the same time. To make sure there existing such an element located after α', the similar strategy adopted for checking S-extended sequences is applied.

The first sequential position of α (i.e. <α'X>) in the k-th sequence is $FP_\alpha(k)$. It implies, after α' appears, $FP_\alpha(k)$ is the smallest sequential position of X in sequence k. If γ is contained in the sequence, there must exist an element containing both X and {T} whose sequential location is no less than $FP_\alpha(k)$. Therefore, a right-shift operation is performed on $Bit_{(X\cup\{T\})}(k)$ by $(FP_\alpha(k)-1)$ bits. If the resultant sequence is non-zero, it implies that such an element exists which is located after α'. That is, γ is contained in the k-th sequence. Let bit h denote the first bit in the resultant sequence with value 1, it indicates the first sequential position of γ in the sequence is located h positions after position $(FP_\alpha(k)-1)$. Therefore, $Appear_\gamma(k)$ is set to be 1 and $FP_\gamma(k)$ is set to be $(FP_\alpha(k)-1)+h$. Otherwise, sequence γ is not contained in the k-th sequence of S, and both of $Appear_\gamma(k)$ and $FP_\gamma(k)$ are set to be 0.

The appearing sequence of γ, $Appear_\gamma$, is obtained after performing the checking for all the bits in A_Appear_γ with value 1. Finally, γ is certified to be a frequent pattern if $1\_Count(Appear_\gamma)$ is larger or equal to min_sup.

[Example 3] Following the running example shown in Fig. 1, suppose min_sup is set to be 2. A sequence α=<ac> is given, and $Appear_{<ac>}$=00111 and $FP_{<ac>}$="0,0,4,2,2" are known. The process for checking whether the S-extended sequence <acd> and I-extended sequence <$a(cd)$> of <ac> being frequent is described as following.

- Checking S-extended sequence <acd>:

(1) $A\_Appear_{<acd>}=Appear_{<ac>} \wedge Appear_d$=00101; $1\_Count(A\_Appear_{<acd>}) \geq 2$, continue.

(2) For each bit k in $A\_Appear_{<acd>}$

(2-1) k=1: $A\_Appear_{<acd>}(1)\neq 0$; $FP_{<ac>}(1)=2$ is not less than $FP_d(1)=2$;
Get $Bit_d(1)$=1010; Right-shift($Bit_d(1)$) by 2 bits→0010(non-zero);
The first bit in the resultant sequence with value 1 is bit 2;

Set Appear$_{<acd>}$(1)=1, and FP$_{<acd>}$(1)=2+2=4.

(2-2) k=2: Check A_Appear$_{<acd>}$(2)=0; Set Appear$_{<acd>}$(2)=0, and FP$_{<acd>}$(2)=0.

(2-3) k=3: Check A_Appear$_{<acd>}$(3)≠0; FP$_{<ac>}$(3)=4 is not less than FP$_d$(3)=3;
Get Bit$_d$(3)=1100; Right-shift(Bit$_d$(3)) by 4 bits→0000(zero);
Set Appear$_{<acd>}$(3)=0, and FP$_{<acd>}$(3)=0.

(2-4) k=4: Check A_Appear$_{<acd>}$(4)=0; Set Appear$_{<acd>}$(4)=0, and FP$_{<acd>}$(4)=0.

(2-5) k=5: Check A_Appear$_{<acd>}$(5)=0; Set Appear$_{<acd>}$(5)=0, and FP$_{<acd>}$(5)=0.

(3) Appear$_{<acd>}$=00001, 1_Count(Appear$_{<acd>}$)<2; <*acd*> is not a frequent pattern.

- Checking I-extended sequence <*a*(*cd*)>:

(1) A_Appear$_{<a(cd)>}$= Appear$_{<ac>}$ ∧ Appear$_d$= 00101;1_Count(A_Appear$_{<a(cd)>}$) ≥ 2, continue.

(2) For each bit k in A_Appear$_{<a(cd)>}$

(2-1) k=1: Check A_Appear$_{<a(cd)>}$ (1)≠0; FP$_{<ac>}$(1)=2 is equal to FP$_d$(1)=2;
Set Appear$_{<a(cd)>}$(1)=1, and FP$_{<a(cd)>}$(1)=2.

(2-2) k=2: Check A_Appear$_{<a(cd)>}$(2)=0; Set Appear$_{<a(cd)>}$(2)=0, and FP$_{<a(cd)>}$(2)=0.

(2-3) k=3: Check A_Appear$_{<a(cd)>}$(3)≠0; FP$_{<ac>}$(3)=4 is not equal to FP$_d$(3)=3;
Get Bit$_c$(3)=1001 and Bit$_d$(3)=1100; Bit$_{(cd)}$(3)= Bit$_c$(3) ∧ Bit$_d$(3)=1000;
Right-shift(Bit$_{(cd)}$(3)) by (4-1) bits→0001(non-zero);
Set Appear$_{<a(cd)>}$(3)=1, and FP$_{<a(cd)>}$(3)=(4-1)+1=4.

(2-4) k=4: Check A_Appear$_{<a(cd)>}$(4)=0; Set Appear$_{<a(cd)>}$(4)=0, and FP$_{<a(cd)>}$(4)=0.

(2-5) k=5: Check A_Appear$_{<a(cd)>}$(5)=0; Set Appear$_{<a(cd)>}$(5)=0, and FP$_{<a(cd)>}$(5)=0.

(3) Appear$_{<a(cd)>}$=00101, FP$_{<a(cd)>}$= "0,0,4,0,2",
Check 1_Count(Appear$_{<a(cd)>}$) ≥ 2, <*a*(*cd*)> is a frequent pattern.

4.3 I-SPAM Algorithm

The whole process of I-SPAM Algorithm is described as the pseudo codes shown below, which are similar to the ones of SPAM algorithm. The modified parts include the codes for constructing the bit sequence table and appearing sequence table, and removing infrequent items from the tables. The S-temp$_{<\alpha>}$ /I-temp$_{<\alpha>}$ is used to store the candidate items which are possible to construct frequent S-extended/I-extended sequences from sequence α according to the pruning strategy adopted in SPAM algorithm [3]. Initially, for each item T∈L_1, S-temp$_{<T>}$ is set to be L_1, and I-temp$_{<T>}$ is assigned the set of items in L_1 and greater than T, respectively. Then procedure M_DFS() is called recursively to perform the process of generating candidates from T and discovering frequent patterns in a depth-first manner.

The significant difference between SPAM and I-SPAM is that the appearing sequences and first_position sequences are used for more efficient support counting to avoid checking bit sequence tables as possible. Moreover, the memory size to retain the appearing and first_position sequences of patterns temporally during executing I-SPAM is less than the one to retain the bitmap sequences of patterns while executing SPAM.

```
Algorithm I-SPAM (Sequence Database S, min_sup)
  Initialize bit sequences, appearing sequences, and
  first_position sequences as zeros;
  For each [sid_i, s] ∈ S  /* construct the tables */
```

```
  For each element s_j of s
    For each item k∈s_j
      If Appear_k(i)==0, set Appear_k(i)= 1;
      Set the j-th bit in Bit_k(i) to be 1;
      If FP_k(i)==0, set FP_k(i)= j;
L_1=φ;
For each item k in appearing sequence table
  If 1_Count(Appear_k)< min_sup, /* remove from the tables */
     For i=1 to |S|
       If Appear_k(i)==1,
          Remove Bit_k(i) from the bit sequence table;
     Remove the tuple [k, Appear_k, FP_k] from appearing
     sequence table;
  Else L_1= L_1 ∪ {k};
For each item T∈L_1
  S-temp_<T>= L_1;
  I-temp_<T>= {x | x∈ L_1 ∧ x > T by lexicographic order};
  Call M_DFS(T, Appear_T, FP_T, S-temp_<T>, I-temp_<T>).
```

5. Performance Evaluation

In this section, the experimental results on the performance of I-SPAM in comparison with SPAM [3] are reported. The proposed I-SPAM algorithm was implemented in Microsoft Visual C++ 6.0 programming language and the source code of SPAM was obtained from the web site with http://himalaya-tools.sourceforge.net/Spam/ provided by the author. All the experiments are performed on a personal computer with 2.4GHz Intel Pentium 4 CPU, 512MB main memory, and running Microsoft Windows XP.

The experiments were performed on synthetic data generated by the IBM synthetic market-basket data generator AssociGen[2]. The inputted parameters AssociGen: D(number of sequences in the dataset), C(average number of elements per sequence), T(average number of items per element), S(average length of potentially frequent sequential patterns), and I(average length of itemsets in maximal potentially frequent patterns) are considered the factors while comparing I-SPAM against SPAM.

5.1 Comparison with Spam on execution time

The experimental results on execution time are shown in Fig. 2, where the *min_sup* setting and the parameters used for generating data set are controlled individually in each experiment. For SPAM algorithm, some experimental results are missing from the figures. It means, under the parameter setting, the SPAM algorithm could not be executed properly in the running environment.

First, by varying *min_sup* setting, the execution times of these two algorithms are evaluated on two datasets with various scales(one is 1.4MB and another is 7.9MB). The experimental results are shown in Fig. 2(a) and 2(b), respectively. The results show that I-SPAM can achieve the same magnitude efficiency and even better than SPAM on execution time. The primary reason is due to the representation of appearing and first_position sequences, which are used for more efficient support counting to avoid checking the bit

sequences as possible. When the *min_sup* setting becomes larger, fewer candidate patterns are generated such that the benefit gained by I-SPAM is reduced. Additionally, the cost for pruning infrequent items from bits sequence and appearing sequence tables by I-SPAM is increasing. Therefore, the execution time of I-SPAM approaches the one of SPAM when *min_sup* is larger than 0.025. However, when *min_sup* is small enough, the execution efficiency of I-SPAM outperforms SPAM about a factor of 1.5. Moreover, SPAM is not executable when performing on large dataset with *min_sup* less than 0.08.

Agreeing with the previous experimental results, the results shown in Fig. 2(c) indicate that I-SPAM outperforms SPAM when the numbers of sequences in the datasets(D) are larger than 5K. For small datasets, checking the bitmap representation directly could be performed very quickly. Therefore, in some cases of small datasets(when D is 3K), the overhead for processing the appearing and first_position sequences outweighs the benefits achieved by these structures, and SPAM runs slightly faster in these situations.

Among the parameters used in AssociGen, as the average number of elements per sequence(C) and the average number of items per element(T) increase, the size of generated synthetic datasets will increase. Therefore, the experimental results shown in Fig. 2(d) and 2(e) indicate the coincident result shown in Fig. 2(c) due to the similar reasons as in the case of increasing the number of sequences in the dataset. Due to page limit, the experimental results on parameters (S) and (I) setting are omitted here.

5.2 Comparison with SPAM on Maximal Memory Usage

The estimated results on maximal memory usage are shown in Fig. 2(f) and 2(g). In the experiment performed on the 7.9MB dataset, Fig. 2(f) shows that the maximal memory requirement of I-SPAM is about half of the one required by SPAM. The primary reason is due to the size of required memory for storing the appearing and first_position sequences of a pattern temporally during executing I-SPAM is less than the one for storing the bitmap sequence of a pattern while executing SPAM. Fig. 2(g) shows the maximal memory requirement of I-SPAM and SPAM by varying the average number of elements per sequence in the datasets, which indicates the similar outcomes.

To summarize the experimental results, in general, I-SPAM has better scalability than SPAM for larger datasets and less *min_sup* setting under the same running environment.

6. Conclusion and Future Works

In this paper, an improved-version of SPAM algorithm, called I-SPAM, for mining frequent sequential patterns is proposed. With the aid of appearing sequence table, more efficient support counting is achieved by avoiding checking the bit sequences as possible.

Moreover, the required memory size for storing the temporal variables is reduced effectively to be less than the one needed by SPAM. The experimental results demonstrate that I-SPAM outperforms SPAM on execution time especially when performed on larger datasets and with smaller *min_sup* setting. Furthermore, the maximal memory requirement is reduced effectively to be about half of the one required for executing SPAM in most cases.

Constraints are essential for many sequential pattern mining applications. In the future, it is worthy our studying on pushing constraints in the mining process of I-SPAM to reduce the explored portion of the search space dramatically.

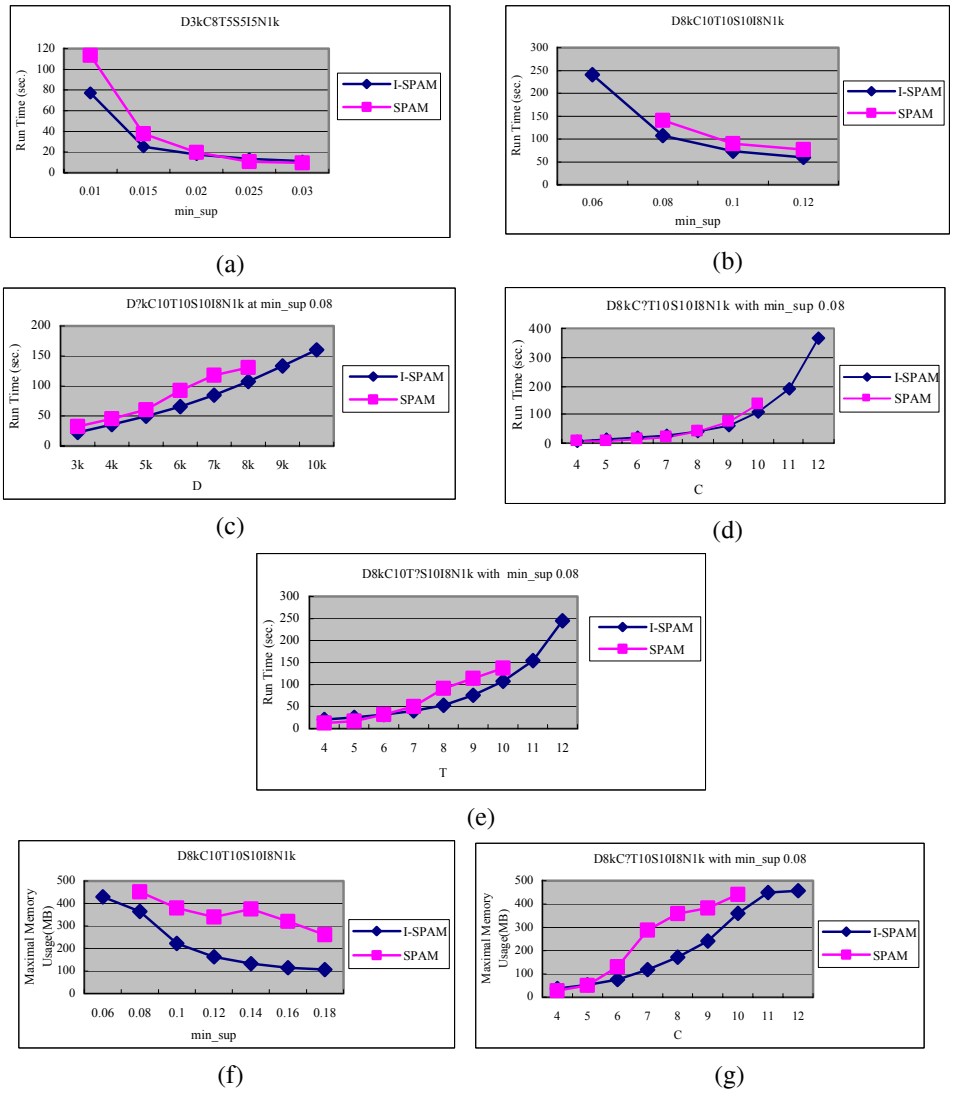

Fig. 2. Experimental results

References

[1] R. Agarwal and R. Srikant. Fast Algorithm for Mining Association Rule in Large Databases. In Proc. 1994 Int. Conf. Very Large DataBases, pp. 487-499, 1994.

[2] R. Agarwal and R. Srikant. Mining Sequential Pattern. In Proc. 1995 Int. Conf. Data Engineering, pages 3-10, 1995.

[3] J. Ayres, J. Flannick, J. Gehrke, and T. Yiu. Sequential PAttern Mining Using A Bitmap Representation. In Proc. 2002 Int. Conf. Knowledge Discovery and Data Mining, 2002.

[4] J. Pei, J. Han, Q. Chen, U. Dayal, and H. Pinto. FreeSpan: Frequent Pattern-Projected Sequential Pattern Mining. In Proc. 2000 Int. Conf. Knowledge Discovery and Data Mining, 2000.

[5] J. Pei, J. Han, B. Mortazavi-Asi and H. Pinto. PrefixSpan：Mining Sequential Patterns Efficiently by Prefix-Projected Pattern Growth. In Proc. 2001 Int. Conf. on Data Engineering , 2001.

[6] R. Srikant and R. Agrawal. Mining Sequential Patterns: Generalizations and Performance Improvements. In Proc. 5th Int. Conf. Extending Database Technology, 1996.

[7] M.J. Zaki. SPADE: An Efficient Algorithm for Mining Frequent Sequences. In Machine Learning Journal, 42(1/2): 31-60, 2001.