Mining Frequent Sequences Using Itemset-Based Extension

Zhixin Ma, Yusheng Xu†, Tharam S. Dillon, Chen Xiaoyun

***Abstract*—In this paper, we systematically explore an itemset-based extension approach for generating candidate sequence which contributes to a better and more straightforward search space traversal performance than traditional item-based extension approach. Based on this candidate generation approach, we present FINDER, a novel algorithm for discovering the set of all frequent sequences. FINDER is composed of two separated steps. In the first step, all frequent itemsets are discovered and we can get great benefit from existing efficient itemset mining algorithms. In the second step, all frequent sequences with at least two frequent itemsets are detected by combining depth-first search and itemset-based extension candidate generation together. A vertical bitmap data representation is adopted for rapidly support counting reason. Several pruning strategies are used to reduce the search space and minimize cost of computation. An extensive set of experiments demonstrate the effectiveness and the linear scalability of proposed algorithm.**

***Index Terms*—Frequent sequence mining, data mining algorithms, frequent pattern, sequence database.**

I. Introduction

The sequences mining task, which discovers all frequent subsequences from a large sequence database, is an important data mining problem. It has attracted considerable attention from database practitioners and researches because of its broad applications in many areas such as analysis of sales data, discovering of Web access patterns in Web-log dataset, extraction of Motifs from DNA sequence, analysis of medical database, identifying network alarm patterns, etc.

In the last decade, a number of algorithms have been proposed to deal with the problem of mining sequential patterns from sequence database. Most of them are based on Apriori property which states that any sub-pattern of a frequent pattern must be frequent. These Apriori-like algorithms utilize a bottom-up candidate generation-and-test method and a breadth-fist search space traverse strategy. In each candidate generation step, algorithm iteratively generate all candidate k-sequences from all frequent (k-1)-sequences. Because each candidate k-sequences has one more item than a frequent (k-1)-sequences, this candidate generation method can be considered as an item-based extension approach. In other words, all these algorithms deal with the problem of mining sequential patterns using an item-based viewpoint. The main bottleneck of these algorithms is that huge number of candidate sequences could be generated and the cost of candidate generation, test and support counting is very expensive. In fact, a lot of candidate sequences are infrequent or not exist in database. Furthermore, some algorithms require multiple full database-scans as the longest frequent sequence and the cost of I/O is very expensive, some approaches use very complicated internal data structures to maintain database in memory which add great space and computation overhead.

In this paper, we systematically explore an itemset-based extension approach for generating candidate sequence which contributes to a better and more straightforward search space traversal performance than traditional item-based extension approach. The general idea is outlined as follow: A candidate sequence can be generated by adding one frequent itemset into the end of a frequent sequence instead of adding one item into a frequent sequence each time. Since any candidates with infrequent itemsets are not generated, the number of candidates is reduced efficiently. This idea is derived from Apriorial1 [1], the first sequence mining algorithm which uses itemset, not item, to generate candidate sequence.

Based on this candidate generation approach, we present a novel algorithm, called ***FINDER*** (**F**requent Sequence M**I**ning usi**N**g Itemset-base**D** **E**xtension App**R**oach), for discovering the set of all frequent sequences. FINDER is composed of two separated steps. In the first step, all frequent itemsets are discovered and we can get great benefit from existing efficient itemset mining algorithms [3][5]. In the second step, all frequent sequences with at least two frequent itemsets are detected by combining depth-first search and itemset-based extension candidate generation together. For rapidly support counting reason, we adopt vertical bitmap data representation proposed in SPAM [2]. In addition, FINDER can reduce the search space and minimize cost of computation efficiently by using several pruning strategies.

The rest of the paper is organized as follows: Section 2 introduces the basic concepts related to the sequence mining problem. Section 3 discusses the related work. Section 4 presents our itemset-based extension approach in detail. In Section 5, we describe FINDER algorithm with pruning strategies and vertical bitmap data representation. An experimental study is presented in Section 6. We conclude in Section 7 with a discussion of future works.

Manuscript received November 21, 2007. This work was supported in part by National Natural Science Foundation of China under grant No. 90612016 and grant No. 60473095.

Ma Zhixin is with the School of Information Science and Technology, Lanzhou University, Lanzhou, 730000, China, (e-mail: mazhx@lzu.edu.cn).

† Corresponding author. Xu Yusheng is with the School of Information Science and Technology, Lanzhou University, Lanzhou, 730000, China, (phone: 86-13389310278; fax: 86-931-8912778; e-mail: xuyusheng@ lzu.edu.cn).

Tharam S. Dillon is with School of Information System, Curtin University, Perth, Australia, (e-mail: tharam.dillon@cbs.curtin.edu.au)

Chen Xiaoyun is with the School of Information Science and Technology, Lanzhou University, Lanzhou, 730000, China, (e-mail: chenxy@lzu.edu.cn)

II. Problem Statement

Let $I=\{i_1, i_2, \ldots, i_m\}$ be a set of m distinct items comprising the alphabet. An *itemset* $e = \{i_1, i_2, \ldots, i_k\}$ is a non-empty unordered collection of items. Without loss of generality, we assume that items of an itemset are sorted in lexicographic order and denoted as $(i_1 i_2 \ldots i_k)$. A *sequence* $s = \{e_1, e_2, \ldots, e_n\}$ is an ordered list of itemsets and denoted as $(e_1 - e_2 - \ldots - e_n)$, where e_i is an itemset. An item can occur at most once in an itemset of a sequence, but can occur multiple times in different itemsets of a sequence. The number of instances of items in a sequence is called the *length* of sequence. Let $|e_i|$ refer to the number of items in itemset e_i, a sequence with length l is called *l-sequence*, where $l = \sum |e_i|$ and $1 = i = n$. For example, C-AB-A is a 4-sequence.

A sequence $s_1=(a_1 - a_2 - \ldots - a_m)$ is said to *contained* in another sequence $s_2=(b_1 - b_2 - \ldots - b_n)$ if and only if $\exists\ i_1, i_2, \ldots, i_m$, such that $1=i_1<i_2<\ldots<i_m=n$, and $a_1 \subseteq b_{i1}, a_2 \subseteq b_{i2}, \ldots, a_m \subseteq b_{im}$. If s_1 is contained in s_2, s_1 is a *subsequence* of s_2 and s_2 is a *supersequence* of s_1. This relationship is denoted by $s_1 \subseteq s_2$. For example, the sequence A-C is a subsequence of (AB-CD). On the other hand, the sequences (C-A) and (AC) are not subsequence of (AB-CD).

The database D for sequence mining consists of a collection of input-sequences. Each input-sequence has a unique identifier called *sequence-id* (*sid*) and each itemset in a given input-sequence also have an unique identifier called *itemset-id* (*eid*).

Given a sequence database D, the *support count* of a sequence s, denoted as $\delta(s,D)$, is the total number of input-sequences in D which contain s. The *support* of s, denoted as *support(s)*, is the fraction of sequences in D that contain s. If the symbol $|D|$ denotes the number of sequences in D, $support(s) = \delta(s,D) / |D|$. Given a user-specified threshold *min_sup*, we say that a sequence s is *frequent* if *support*(s) is greater than or equal to min_sup. A *maximal* frequent sequence is a frequent sequence and none of its supersequences is frequent.

Given a database D of input-sequences and a user-specified threshold min_sup, the problem of sequence mining is to find all the frequent sequences in the database.

III. Related Works

Since the problem of frequent sequence mining was first introduced in [1], a large amount of studies have been done toward the development of efficient algorithms for solving this problem and its variations. Agrawal and Srikant [1] presented three algorithms: AprioriAll, AprioriSome and DynamicSome for solving this problem. Note that these three algorithms utilize itemset to generate candidate sequence, but this idea is not adopted in later sequence mining algorithms. In [11], the same authors generalized definitions of sequence mining to include time constrains, sliding time window, and user defined taxonomy. They also proposed GSP algorithms which outperformed AprioriAll by up to 20 times. GSP is a multi-phase iterative algorithm and requires multiple passes over database. At pass k, the set of candidate k-sequences are checked against the database and frequent k-sequences are determined. Then the set of candidate $(k+1)$-sequences for next pass are generated by joining the set of candidate k-sequences with itself. This process will continue until no candidate is generated. GSP is seemed as one of most important algorithms for mining sequential patterns.

Mannila et al. [7] presented a problem of mining frequent episodes which are essentially mining frequent subsequences in a single long input sequence where each itemset of sequence consists of one item. They further extended their framework in [8] to discover generalized episodes. Kao et al. [6] proposed MFS which combine sampling technique and maximal sequence together. Since a set of long frequent sequences (local maximal sequences) is found in a small sample database early, MFS can mine frequent sequences in original database efficiently by applying supersequence frequency based pruning. Tan and Dillon et al. [13] presented SEQUEST which uses a Direct Memory Access Strips (DMA-Strips) structure to efficiently enumerate candidate subsequence. This structure or model guided method is derived from TMG approach used for tree mining [4][12].

In [14], Zaki proposed SPADE algorithm which uses a vertical id-list database format for efficient joining operation and a lattice-theoretic approach to decompose the original search into small pieces so that all working id-list can be load into memory. Pei et al. [9] proposed PrefixSpan which utilizes a pattern-growth approach instead of refinement of the candidate generation-and-test approach. PrefixSpan recursively projects a sequence database into a set of smaller projected sequence databases and grows sequential patterns in each projected database by exploring only locally frequent fragment. A memory-based pseudo-projection technique is applied to reduce the number of physical projected databases to be generated. Ayres et al. [2] presented SPAM which integrates a depth-first traversal of the search space with some efficient pruning mechanisms. In addition, SPAM utilizes vertical bitmap representation for candidate generation and rapid support counting. Spade, PrefixSpan and SPAM are considered as the three fastest algorithms that mine sequential patterns.

IV. Sequence Enumeration: An Itemset-based Extension Approach

In this section, some notations are defined to simplify our discussion. Then, we describe the itemset-based extension approach and the itemset-based lexicographic tree of sequence lattice upon which our algorithms is based.

A. Some Notations and Lexicographic Tree

Definition 1. Let e_1 and e_2 be two itemsets. If e_1 is a subset of e_2, then e_1 is a *subitemset* of e_2 and e_2 is a *superitemset* of e_1.

For example, itemset (ABC) is a subitemset of (ABCD), itemset (ABC) is superitemset of (AB).

Definition 2. The number of itemsets in a sequence is called the *size* of sequence. A sequence with k itemsets is called *k'-sequence*.

For example, each itemset is a *1*'-sequence because its size is 1, sequence (AC-CD) is a *2*'-sequence because its size is 2.

Note that the *size* of a sequence is different from the *length* of a sequence.

Definition 3. Given sequence database *D*, a user-specified threshold min_sup, we say that an itemset *e* is *frequent* if *support*(*e*) is greater than or equal to min_sup. The set of all frequent itemsets is denoted as *FE* .

Definition 4. A frequent sequence of size *k* is called a *frequent k'-sequence*.

As an example, consider the database shown in figure 1 which has four items (A to D) and four input-sequences. The figure also shows all the frequent sequences with a min_sup of 50%.

Example database

s-id	sequences
001	ABC-ABD-AD
002	ABD
003	AB-CD
004	A-BD-D

Frequent sequences (min_sup=50%)

Frequent 1'-sequence	A, AB, ABD, AD, B, BD, C, D
Frequent 2'-sequence	A-B, A-BD, A-D, AB-D, B-D, BD-D, D-D
Frequent 3'-sequence	A-B-D,A-BD-D, A-D-D

Figure 1: Example database and frequent sequences

Lexicographic Tree for sequences. The *lexicographic subset tree* is presented originally by Rymon [10] and adopted to describe the itemset lattice in most of well-known frequent itemset mining algorithms such as MAFIA [3]. This approach is extended to describe the framework of sequence lattice in SPAM [2]. Assume there is a partial ordering relationship, denoted as $\leq$, on sequences. Let s_1 and s_2 are two sequences, if s1 is a subsequence of s2 then $s_1 \leq s_2$. If s_1 is not a subsequence of s_2, then there is no relationship in this order. All sequences can be arranged in a *lexicographic sequence tree* whose root is null sequence labeled with $\varnothing$ and each node in tree represents a sequence. Each lower level *k* in tree contains all of k-sequences which are ordered lexicographically. Each node is recursively generated from its parent node by using a sequence-extension step or an itemset-extension step. The sequence-extension step is the process of generating a sequence-extended sequence which is generated by adding a new itemset consisting of a single item to the end of its parent's sequence. The itemset-extension step is the process of generating itemset-extended sequence which is a sequence generated by adding an item into the last itemset in the parent's sequence.

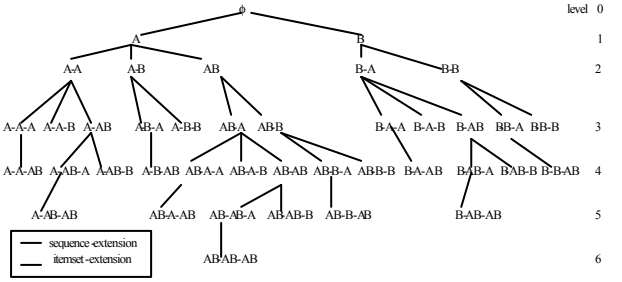

Figure 2: example of the lexicographic sequence tree

For example, Figure 2 shows the complete lexicographic sequence tree for two items, *A* and *B*, given that the maximum size of a sequence is three.

B. Enumeration sequences using itemset-based extension approach

Definition 5 . Given a *k*'-sequence s_1 and a $(k+1)$'-sequence s_2. If s_2 can be generated by adding an itemset *e* to the end of sequence s_1, we say that s_2 is an *itemset-based extension sequence* of s_1, denoted as $s_2 = s_1 \oplus e$. For example, sequence (AB-C-BD) is an itemset-based extension sequence of (AB-C).

From this definition, each sequence can be considered as an itemset-based extension sequence. So, we can enumerate all sequences in lattice and organize lexicographic sequence tree by using an *itemset-based extension approach*. First, all itemsets are generated from the set of items and kept in an *itemset-list* by lexicographical order. The root of tree is null sequence labeled with *f* and each node in tree represents a sequence. Each lower level *k* contains nodes of all *k*'-sequences. The level 1 of tree contains nodes of all itemsets (1'-sequences) in itemset-list. In level *k*, each node is an itemset-based extension sequence of its parent node in level (*k-1*). All these nodes of *k*'-sequence are generated by iteratively adding an itemset from itemset-list to the end of its parent node in level (*k-1*). We refer to lexicographic sequence tree organized in this itemset-based extension manner as *itemset-based lexicographic sequence tree* (abbr. *itemset-based tree*). Theoretically, an itemset-based tree is infinite. But in practice, it is finite because the maximal size of sequences in an input database is limited.

For example, given the maximum size of a sequence is three, figure 3 shows the complete itemset-based tree for two item A and B. The root is null sequence and the itemset list is {A, AB, B}. The level 1 of tree contains all *1'*-sequences: (A), (AB) and (B). Node of sequence (A-AB) in level 2 is generated by adding itemset (AB) to the end of its parent node (A) in level 1, node of sequence (A-AB-B) in level 3 is generated by adding itemset (B) to the end of its parent node (A-AB) in level 2.

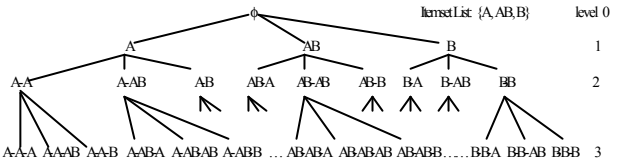

Figure 3: example of the lexicographic sequence tree

Contrasting figure 2 with figure 1, itemset-based tree is more straightforward than lexicographic sequence tree. It gives us a new and simple viewpoint to analyzing the problem of sequence mining.

V. THE FINDER ALGORITHM

A. Basic Idea of FINDER

Since all frequent sequences in a database can be considered as two types: frequent itemset (frequent 1'-sequenc) and frequent sequence with at least two frequent itemsets (frequent *k*'-sequence, k>1), the problem of frequent sequences mining can be divided into two sub-problems: one is to find all frequent itemsets, the other is to find all frequent

k'-sequences where *k*>1. The first sub-problem is equal to the problem of mining frequent itemset and can be solved by using existing efficient frequent itemset mining approaches.

Like most of exiting algorithms, *FINDER* also uses the candidate generation and test approach to solve the second sub-problem. If all frequent itemsets are known, the itemset-based extension approach can be used to enumerate candidate sequences. The general idea is outlined as follow: A candidate *k*'-sequence is generated by adding one frequent itemset into the end of a frequent (*k-1*)'- sequence. We refer to this approach as *itemset-based extension candidate generation*. Based on this approach, the second sub-problem can be seemed as a process of generating and test candidate sequences by traversing the itemset-based tree discussed above.

Figure 4 shows the high level structure of FINDER algorithm which is composed of three main steps: 1) Finding the set of all frequent items (*1*-sequences) F_1. 2) Finding the set of all frequent itemsets (*1*'-sequences) *FE*. 3) Finding all frequent *k*'-sequences, where *k*>1, by using procedure *DFS*.

```
FINDER (min_sup, D)

(1) Finding F1;
//  F1 is the set of all frequent items;
(2) Finding EL;
//  EL is the set of all frequent events;
(3) FS=EL;
//  FS is the set of all frequent sequences;
(4) for each event e_i ∈ EL do
(5)     DFS(e_i, EL)
```

Figure 4: pseudo-code of FINDER

The pseudo-code of procedure *DFS* with no pruning is shown in figure 5. It repeats depth-first search recursively on *each n*'s itemset-based extension sequence. Notice that this recursive process is finite because the maximal size of sequences in an input database is limited.

```
DFS(n, EL)
// without pruning strategies

(1) for each event e_i ∈ EL do
(2)     s= n ⊕ e_i;
(3)     if support(s) ≥ min_sup then
(4)         FS= FS ∪ {s};
(5)     DFS(s, EL)
```

Figure 5: pseudo-code of procedure *DFS* with no pruning

B. Pruning Strategies

Assume that the database to be mined has *k* frequent itemsets and the maximal size of sequence in database is *m*. *DFS* without pruning must generate and test all m^k candidate sequences. It is obvious that *DFS* without pruning is not practical. So, we must explore some pruning strategies to reduce the search space and generate as small a set of nodes containing all frequent sequences as possible while searching the itemset-based tree.

Definition 6. In the itemset-based tree, each node *n* is associated with one *itemset list*, denoted by *EL*, which is the set of frequent itemsets that are considered for a possible itemset-based extension of node *n*. All itemsets in *EL* are kept in lexicographic order.

Definition 7. Given a node *n* and its itemset list $EL=\{e_1,e_2,\ldots,e_k\}$. An itemset e_i is said to be *a frequent extension itemset* of *n* if $e_i \in EL$ and $support(n \oplus e_i)=min\_sup$. An itemset e_j is said to be *an infrequent extension itemset* of *n* if $e_j \in EL$ and $support(n \oplus e_j)<min\_sup$. The *frequent extension itemsets list* (abbr. *FEL)* of *n* is the set of all *n*'s frequent extension itemsets.

Theorem 1. Given a node *n* and its itemset list $EL=\{e_1,e_2,\ldots,e_k\}$. If *n* is not frequent, then all *n*'s itemset-based extension sequences $(n \oplus e_i)$ are not frequent.

Proof. Note that each *n*'s itemset-based extension sequence is a supersequences of n. since all supersequences of an infrequent sequence are not frequent, all *n*'s itemset-based extension sequences are not frequent.

Pruning strategy 1 (abbr. *PS1*). In the itemset-based tree, if a node *n* is not frequent, then all its children are not frequent and can be trimmed off.

Theorem 2. Given a node *n* and its $EL=\{e_1,e_2,\ldots,e_k\}$. If e_i is a frequent extension itemset of *n* and $e_j \subseteq e_i$, then e_j is a frequent extension itemset of *n*.

Proof. If $e_j \subseteq e_i$, then $(n \oplus e_j)$ is a subsequence of $(n \oplus e_i)$. Since all subsequences of a frequent sequence are frequent, $(n \oplus e_j)$ is a frequent sequence and e_j is a frequent extension itemset of *n*.

Pruning strategy 2 (abbr. *PS2*). Given a node *n* and its $EL=\{e_1,e_2,\ldots,e_k\}$. Each $e_i \in EL$ is checked iteratively. If one frequent extension itemset e_i is found, we scan the rest itemsets in *EL* and find each e_j which is a subset of e_i. Since e_j is a frequent extension itemset, sequence $(n \oplus e_j)$ can be inserted into the set of all frequent sequences directly without further testing.

Theorem 3. Given a node *n* and its $EL=\{e_1,e_2,\ldots,e_k\}$. If e_i is an infrequent extension itemset of *n* and $e_i \subseteq e_j$, then e_j is an infrequent extension itemset of *n*.

Proof. If $e_i \subseteq e_j$, then $(n \oplus e_j)$ is a supersequence of $(n \oplus e_i)$. Since all supersequences of an infrequent sequence are not frequent, $(n \oplus e_j)$ is an infrequent sequence and e_j is an infrequent extension itemset of *n*.

Pruning strategy 3 (abbr. *PS3*). Given a node *n* and its $EL=\{e_1,e_2,\ldots,e_k\}$. Each $e_i \in EL$ is checked iteratively. If one infrequent extension itemset e_i is found, we scan the rest itemsets in EL and trim off all itemsets which are superitemsets of e_i.

Theorem 4. Given node *m* and node *n*, if *n* is an itemset-based extension sequence of *m*, then the frequent extension itemsets list of *n* is the subset of the frequent extension itemsets list of *m*.

Proof. Let FLL_m and FEL_n be the frequent extension itemsets list of *m* and *n* respectively. Note that *m* is a subsequence of *n* since *n* is an itemset-based extension sequence of *m*. For each $e_i \in FEL_n$, $(n \oplus e_i)$ is a frequent sequence. Since *m* is a subsequence of *n*, then $(m \oplus e_i)$ is a subsequence of $(n \oplus e_i)$. Since all subsequences of a frequent sequence are frequent, $(m \oplus e_i)$ is a frequent sequence and $e_i \in FEL_m$. Thus, if *n* is an itemset-based extension sequence of *m*, $FEL_n \subseteq FEL_m$.

Pruning strategy 4 (abbr. *PS4*). Given a node *n* and its itemset list $EL=\{e_1,e_2,\ldots,e_k\}$. Each $e_i \in EL$ is checked iteratively and the frequent extension itemsets list *FEL* is

generated. We can use the *FEL* as *n*'s children's *EL*.

Since each lower node's *EL* is its parent node's *FEL*, the lower node's *EL* is reduced and the total search space is pruned efficiently. In practice, the benefit of using PS4 is significant.

Figure 6 shows the pseudo-code of procedure *DFS* with all pruning strategies discussed above. At each node *n*, every $e_i \in EL$ is checked iteratively. If $support(n \oplus e_i) = min\_sup$, then use *PS2* to trim the itemsets in *EL*. If $support(n \oplus e_i) < min\_sup$, then use *PS3* to trim the itemsets in *EL*. We use frequent extension itemsets list *FEL* to perform *PS1*. Because *FEL* contains only frequent extension itemsets of *n*, we do not repeat depth-first search on *n*'s infrequent children which can be seemed as being pruned by using *PS1*. At last, the frequent extension itemsets list *FEL* is transferred to *n*'s frequent children as their *EL*.

```
DFS(n, EL)
// with pruning strategies

(1)  FEL={}
(2)  for each event e_i ∈ EL do
(3)      if support(n⊕e_i) ≥ min_sup then
(4)          FS=FS ∪ {n⊕e_i}; FEL=FEL ∪ e_i;
(5)          for each e_j ∈ EL and j>i do
(6)              if e_j ⊆ e_i then
(7)                  FS= FS ∪ {n⊕e_i};
                     FEL= FEL ∪ {e_j};
                     EL=EL-(e_j};
(8)      if support(n⊕e_i)<min_sup then
(9)          for each e_j ∈ EL and j>i do
(10)             if e_i ⊆ e_j then
(11)                 EL=EL-(e_j};
(12) for each e_i ∈ FEL do
(14)     s= n⊕e_i ;
```

Figure 6: Pseudeo-code of DFS with pruning

C. Data representation

For efficient support courting reason, FINDER adopts the vertical bitmap data representation which is first presented by Ayres et al. We refer the reader to [2] for additional detail on the vertical bitmap.

VI. Experimental Results

In this section, we study the performance of proposed FINDER algorithms by comparing it with SPADE and SPAM. The experiments were performed on a 1.7GHz Pentium 4 PC with 512MB main memory, running Microsoft Windows 2003 server. We obtained the source code of SPADE and SPAM from their authors' websites. All three algorithms are written in C++, and compiled using g++ with option -03. Same as SPAM, all synthetic datasets are generated by using the IBM AssocGen program [1] which takes the parameters listed in table 1.

Option	Description
D	Number of customers
C	Average transactions per customer
T	Average items per transaction
S	Average length of maximal pattern

Table 1: Parameters used in dataset generation

A. Comparison with SPADE and SPAM

We compared FINDER with SPADE and SPAM on several synthetic datasets for various minimum support values. The results of these tests are shown in Figures 8.

The figures clearly show that FINDER outperforms SPADE by about a factor of average 1.5 on small datasets and better than an order of magnitude for reasonably large datasets. There are several reasons why FINDER outperforms SPADE: 1) FINDER uses itemset-based extension approach for generating candidate sequence which insures no candidate with infrequent itemsets is generated, the number of candidates is reduced efficiently. 2) Since FINDER discovers all frequent itemsets in the first step, we can get great benefit from existing efficient itemset mining algorithms. 3) FINDER adopts vertical bitmap representation of data structure which performs counting process in an extremely efficient manner.

The Figures 8 also shows that SPAM outperforms FINDER. For each dataset, SPAM is about twice as fast as FINDER at lower values of support and two algorithms have nearly equal performance at higher values of support. The primary reason is due to space requirement problem of FINDER. Assume that the database to be mined has *n* different items, there would be 2^n-1 different possible frequent itemsets in database. It is obvious that keeping all bitmaps of frequent itemsets in memory is not practical. In implementation of FINDER, only bitmaps of each item are kept in main memory, each bitmap of frequent itemset is generated and released dynamically. Because same bitmap of a frequent itemset should be generated several times, the costs of runtime are increased accordingly.

B. Scale-up

We study the scale-up performance of algorithms as several parameters in dataset generation were varied. For each test, one parameter was varied and the others were kept fixed. The parameters that we varied were number of customers, average transactions per customer, average items per transaction and average length of maximal pattern. The results of tests are shown in Figure 9. It can be easily observed that the FINDER scales linearly with four varying parameters.

VII. Conclusion

In this paper, we systematically explore an itemset-based extension approach for generating candidate sequence. Based on this approach, a novel algorithm for discovering the set of all frequent sequences is presented which can reduce the search space and minimize cost of computation efficiently by using several efficient pruning strategies.

The itemset-based extension approach opens several research opportunities and future work will be done in various directions. First, we are studying how to discover maximal or closed sequential patterns by using proposed approach. Second, we are investigating how to apply this approach to incremental mining of sequential patterns. In addition, extending FINDER for parallel sequence mining is also considered.

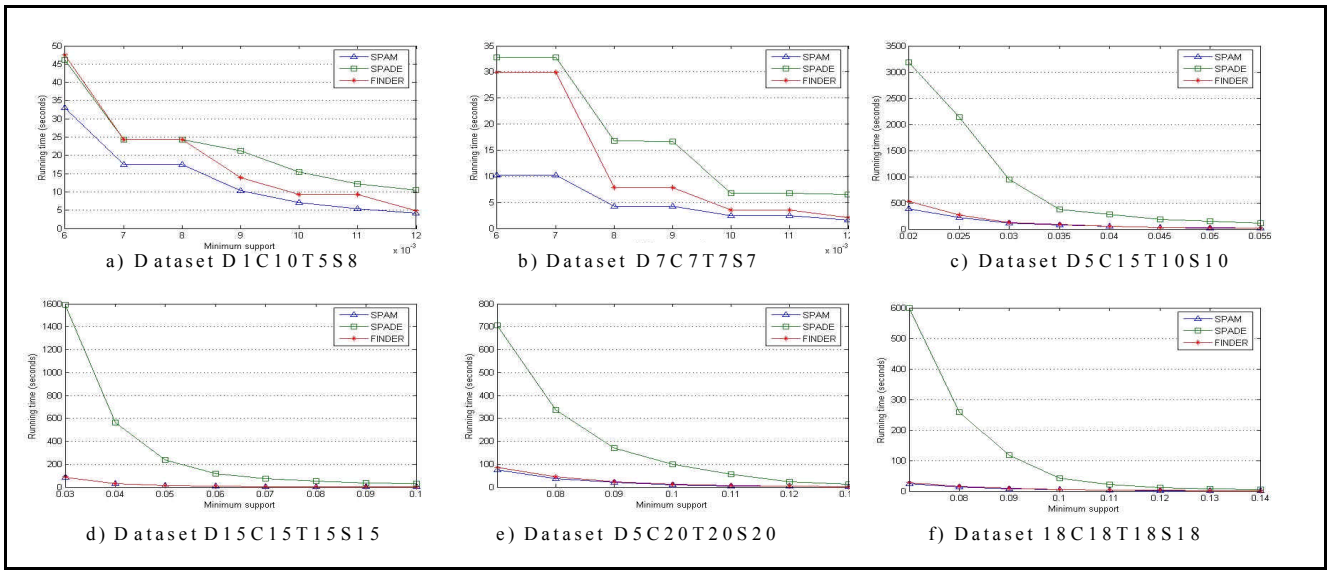

Figure 7: Execution times on different synthetic datasets for various minimum support values

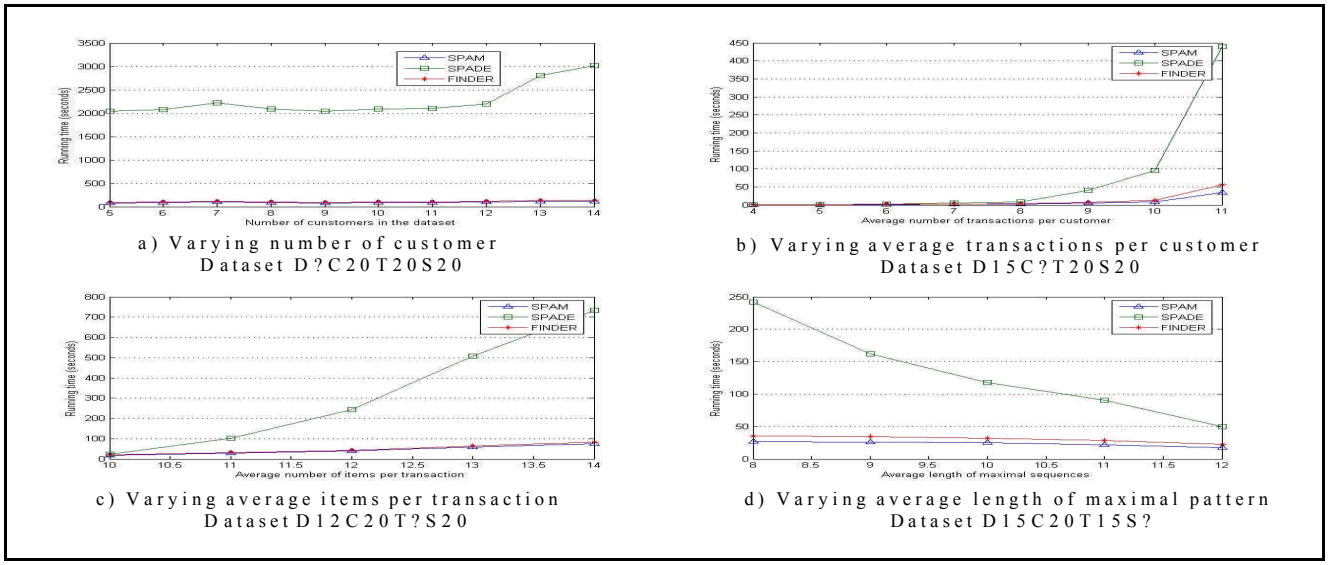

Figure 8: Scale-up with varying parameters of database

REFERENCES

[1] R. Agrawal and R. Srikant. Mining Sequential Patterns. In Proc. of 11th Int'l Conf. on Data Engineering, pp. 3–14, Mar. 1995.

[2] J. Ayres, J. Gehrke, T. Yiu, and J. Flannick. Sequential Pattern Mining Using a Bitmap Representation. In Proc. of ACM SIGKDD Conf. on Knowledge Discovery and Data Mining, pp. 429-435, 2002.

[3] D. Burdick, M. Calimlim, J. Gehrke. MAFIA: A maximal frequent itemset algorithm for transactional databases. In Proc. of 17th Int'l Conf. on Data Engineering, pp. 443-452, 2001.

[4] L. Feng, T. DILLON. Mining XML-Enabled association rule with templates. In Proc. of 3rd Int'l workshop on Knowledge Discovery in Inductive Databases, pp. 66-88, 2004.

[5] J. Han, J. Pei, Y. Yin. Mining frequent patterns without candidate generation. In Proc. of the 2000 ACM SIGMOD Int'l Conf on Management of Data, pp. 1~12, 2000.

[6] B. Kao, M. Zhang, C. Yip, and D.W. Cheung. Efficient Algorithms for Mining and Incremental Update of Maximal Frequent Sequences. Data Mining and Knowledge Discovery. Vol. 10, pp. 87-116, 2005.

[7] H. Mannila, H. Toivonen, and A.I. Verkamo. Discovery of Frequent Episodes in Itemset Sequences. In Proc. of 1st Int'l Conf. on Knowledge Discovery and Data Mining. Vol. 1, pp. 210-215, 1995.

[8] H. Mannila and H. Toivonen, Discovering Generalized Episodes Using Minimal Occurrences. In Proc. of 2nd Int'l Conf. on Knowledge Discovery and Data Mining. 1996.

[9] J. Pei, J. Han, B. Mortazavi-Asi, J. Wang, H.Pinto, and Q. Chen. Mining Sequential Patterns by Pattern-growth: The PrefixSpan Approach. IEEE Transactions on Knowlede and Data Engineering. Vol. 16, pp. 1-17, 2004.

[10] R. Rymon. Search through systematic set enumeration. In Proc. of 3rd Int'l Conf. on Principles of Knowledge Representation and Reasoning, pp. 539–550, 1992.

[11] R. Srikant and R. Agrawal. Mining Sequential Patterns: Generalizations and Performance Improvements. In Proc. of 15th Int'l Conf. on Extending Database Technology, pp. 3-17, 1996.

[12] H. Tan, T. Dillon, F. Hadzic, L. Feng, E. Chang. IMB3-Miner: Mining Induced/Embedded Subtrees by Constraining the Level of Embedding. In Proc. of Pacific-Asia Conference on Knowledge Discovery and Data Mining, 2006.

[13] H. Tan, T. Dillon, F. Hadzic, E. Chang. SEQUEST: Mining frequent subsequences using DMA Strips. In Proc. of Data Mining & Information Engineering'06, 2006.

[14] M.J. Zaki. SPADE: An Efficient Algorithms for Mining Frequent Sequences. Machine Learning. Vol. 40, pp.31-60, 2001.