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 is 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 Aprioriall [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*** (Frequent Sequence MIning usiNg Itemset-baseD Extension AppRoach), 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.

II. PROBLEM STATEMENT

Let $I=\{i_1, i_2, \ldots, i_m\}$ be a set of m distinct items comprising the

Manuscript received November 20, 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, (phone: 86-13038799505; fax: 86-931-8912778; e-mail: mazhx@ lzu.edu.cn).

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

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

alphabet. An *itemset* $e = \{i_1, i_2, \dots, i_k\}$ 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 $(i_1 i_2 \dots i_k)$. A *sequence* $s = \{e_1, e_2, \dots, e_n\}$ is an ordered list of itemsets and denoted as $(e_1\text{-}e_2\text{-} \dots \text{-}e_n)$, 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 $l = ? |e_i|$ and $1 = i = n$. For example, C-AB-A is a 4-sequence.

A sequence $s_1 = (a_1\text{-}\ a_2\text{-} \dots \text{-}a_m)$ is said to *contained* in another sequence $s_2 = (b_1\text{-}b_2\text{-} \dots \text{-}b_n)$ if and only if $\exists\ i_1, i_2, \dots, i_m$, such that $1 = i_1 < i_2 < \dots < i_m = n$, and $a_1 \subseteq b_{i1}, a_2 \subseteq b_{i2}, \dots, a_m \subseteq b_{im}$. 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

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*.

From above definitions, it is obvious that each itemset in a frequent sequence is a frequent itemset, each frequent itemset is a frequent 1'-sequence, and the set of all frequentitemsets is a subset of the set of all frequent sequences, $FE \subseteq FS$.

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%. In this example we have eight frequent itemsets (1'-sequence), seven frequent 2'-sequences and three frequent 3'-sequences.

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] and CHARM [16]. This approach is extended to describe the framework of sequence lattice in SPAM [2]. Assume there is a partial ordering relationship, denoted as ≤ 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 ∅ and each node in tree represents a sequence. Each lower level *k* in tree contains all of k-sequences which are ordered lexicographically. If *n* is a node in the tree, the children of *n* are all nodes *n*' such that *n*=*n*' and $\forall \in \quad \leq \Rightarrow \leq$

straightforward than lexicographic sequence tree. It gives us a new and simple viewpoint to analyzing the problem of sequence mining. Assume that the database to be mined has n different items and the maximal size of sequence in database is m. In itemset-based tree, there would be 2^n-1 different itemsets in level 1 and $(2^n-1)^k$ different k'-sequences in level k. Because all nodes in tree compose the sequence lattice of database, the problem of mining frequent sequence can be considered as a process of traversing the itemset-based tree and finding a cut through this lattice such that all nodes above the cut are frequent sequences, and all nodes below are infrequent. Theoretically, the maximal search space size of mining frequent sequence problem is T given by equation (1).

generating and testing all k itemset-based extension sequences (candidate sequences) will waste a lot of computation. Extremely, if the database has plenty of frequent itemsets, our algorithm is still impractical.

Theorem 2. Given a node n and its $EL=\{e_1,e_2,...,e_k\}$. 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 $EL=\{e_1,e_2,...,e_k\}$. 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.

For example: given a frequent sequence (A-AC) and its EL={ A, AB, ABC, AC, B, BC,C}. If sequence (A-AC-AB) is frequent, (AB) is a frequent extension itemset of (A-AC). We scan the rest itemsets in EL and find (B) is subitemset of (AB). So, (A-AC-B) is a frequent sequence and can be added into FS without testing.

Theorem 3. Given a node n and its $EL=\{e_1,e_2,...,e_k\}$. 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 $EL=\{ e_1,e_2,...,e_k \}$. 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.

For example: given a fequent sequence (A-AB) and its EL={A, AB, ABC, AC, B, BC,C}. If (A-AB-A) is not frequent, (A) is an infrequent extension itemset of (A-AB). We scan the rest itemsets in EL and find all superitemsets of (A). So, itemsets (AB), (ABC) and (AC) are infrequent extension itemsets of (A-AB) and can be trimmed off from EL without testing.

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 $EL=\{e_1,e_2,...,e_k\}$. 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 $e_i \in EL$ do
(3)      if $support(n \oplus e_i) \geq min\_sup$ then
(4)          $FS=FS \cup \{n \oplus e_i\}$; $FEL=FEL \cup e_i$;
(5)          for each $e_j \in EL$ and $j>i$ do
(6)              if $e_j \subseteq e_i$ then
(7)                  $FS= FS \cup \{n \oplus e_i\}$;
                     $FEL= FEL \cup \{e_i\}$;
                     $EL=EL-(e_j\}$;
(8)      if $support(n \oplus e_i)<min\_sup$ then
(9)          for each $e_j \in EL$ and $j>i$ do
(10)             if $e_i \subseteq e_j$ then
(11)                 $EL=EL-(e_j\}$;
(12) for each $e_i \in FEL$ do
(14)     s= $n \oplus e_i$ ;
```

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.

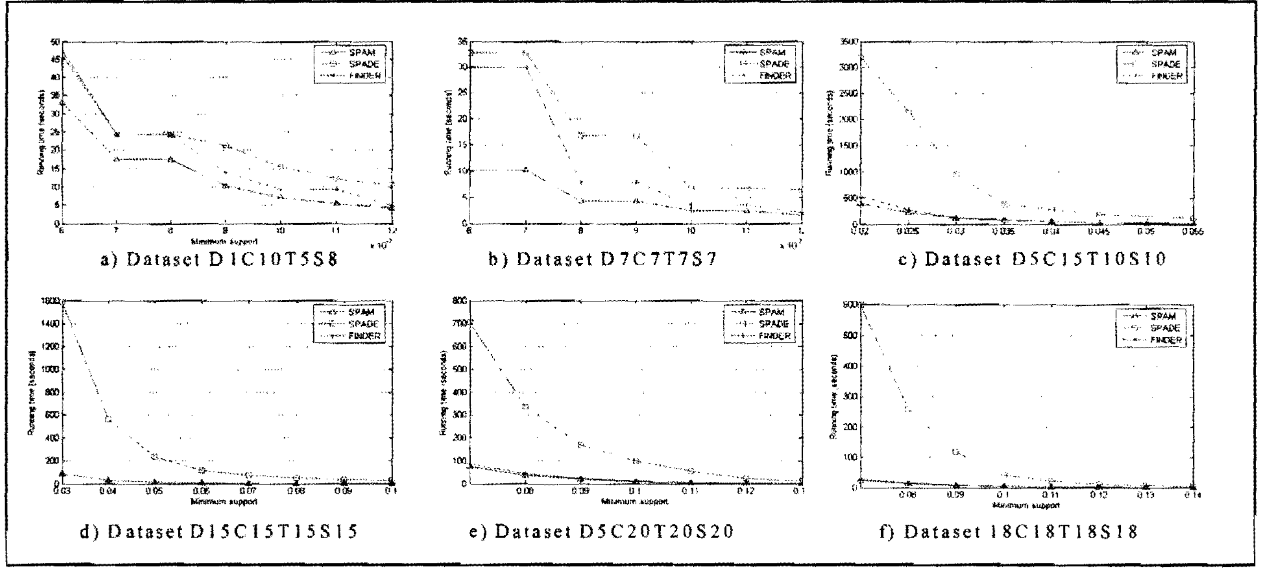

Figure 7: Execution times on different synthetic datasets for various minimum support values

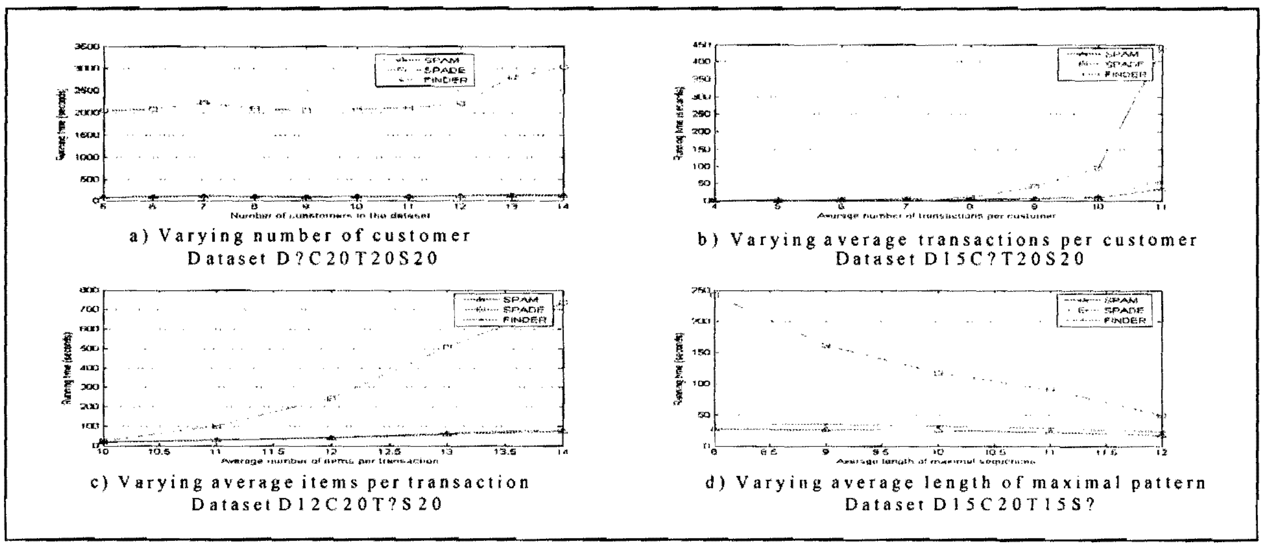

Figure 8: Scale-up with varying parameters of database

I. 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.

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. of3rd 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, L. Feng, E. Chang. Tree Model Guided Candidate Generation for Mining Frequent Patterns from XML Documents. ACS TOIS Journal(2005) (Submitted).

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

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

[16] M. J. Zaki and C. Hsiao. Efficient algorithm for mining closed itemsets and their lattice structure. IEEE Transactions on Knowledge and Data Engineering. Vol. 17(4), pp. 462~478, 2005.