Effective Pruning Strategies for Sequential Pattern Mining

Xu Yusheng Ma Zhixin Li Lian
School of Information Science and Technology
Lanzhou University, China
e-mail: { xuyusheng, mazhx,lil}@ lzu.edu.cn

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

***Abstract*— In this paper, we systematically explore the search space of frequent sequence mining and present two novel pruning strategies, SEP (Sequence Extension Pruning) and IEP (Item Extension Pruning), which can be used in all Apriori-like sequence mining algorithms or lattice-theoretic approaches. With a little more memory overhead, proposed pruning strategies can prune invalidated search space and decrease the total cost of frequency counting effectively. For effectiveness testing reason, we optimize SPAM [2] and present the improved algorithm, $SPAM_{\text{SEPIEP}}$, which uses SEP and IEP to prune the search space by sharing the frequent 2-sequences lists. A set of comprehensive performance experiments study shows that $SPAM_{\text{SEPIEP}}$ outperforms SPAM by a factor of 10 on small datasets and better than 30% to 50% on reasonably large dataset.**

I. Introduction

Frequent sequence mining is a task of discovering frequent patterns shared across time among a large database objects [1]. 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, extraction of Motifs from DNA sequence, etc.

In the last decade, a number of algorithms have been proposed to deal with the problem of mining sequential patterns in sequence databases. Most of them are Apriori-like algorithms which utilize a bottom-up candidate generation-and-test method. Unlike frequent itemset mining, the order of items in a sequence is very important for mining sequential patterns. Due to this difference, the task of discovering all frequent sequences in large database is quite challenging. For example, with n different items there are $O(n^k)$ potentially frequent sequences of length k. Can we find some effective pruning strategies that help algorithms to generate as fewer candidates as possible? This is the motivation of this paper.

In this paper, we systematically explore the search space of frequent sequence mining and present two novel pruning strategies, SEP (Sequence Extension Pruning) and IEP (Item Extension Pruning), which can be used in all Apriori-like sequence mining algorithms or lattice-theoretic approaches. With a little more memory overhead, proposed strategies can prune invalidated search space effectively. For effectiveness testing reason, we optimize SPAM [2] by proposed pruning strategies and present the improved algorithm, $SPAM_{\text{SEPIEP}}$, which adopts a two steps mining process. In the first step, $SPAM_{\text{SEPIEP}}$ generates all frequent 2-sequences. In the second step, $SPAM_{\text{SEPIEP}}$ uses SEP and IEP to prune the search space of SPAM by sharing the frequent 2-sequences lists. A comprehensive performance study shows that $SPAM_{\text{SEPIEP}}$ outperforms SPAM by a factor of 10 on small datasets and better than 30% to 50% for reasonably large dataset.

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 works. The proposed pruning strategies are discussed in section 4. A comprehensive experimental study is presented in Section 5. Finally, conclusions can found in section 6.

II. Problem Statement

Let $I = \{i_1, i_2, \ldots, i_{\text{m}}\}$ be a set of m distinct items comprising the alphabet. An itemset $e = \{i_1, i_2, \ldots, i_{\text{k}}\}$ 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 $(i_1, i_2, \ldots, i_{\text{k}})$. A sequence s is an ordered list of itemsets, denoted as $(e_1 \rightarrow e_2 \rightarrow \ldots \rightarrow e_{\text{n}})$, where e_{i} is an itemset.

The number of instances of items in a sequence is called the length of sequence. Let $|e_{\text{i}}|$ refer to the number of items in itemset e_{i}, a sequence with length l is called l-sequence, where $l = \sum |e_{\text{i}}|$ and $1 \leq i \leq n$. For example, $(a \rightarrow ab \rightarrow a)$ is a 4-sequence.

The size of a sequence, denoted as $size(s)$, is the number of events that it contains. For example, for $s = (e_1 \rightarrow e_2 \rightarrow \ldots \rightarrow e_{\text{m}})$, size of s is m.

A sequence $s_1 = (a_1 \rightarrow a_2 \rightarrow \ldots \rightarrow a_{\text{m}})$ is said to be contained in another sequence $s_2 = (b_1 \rightarrow b_2 \rightarrow \ldots \rightarrow b_{\text{n}})$ if and only if $\exists i_1, i_2, \ldots, i_{\text{m}}$, such that $1 \leq i_1 < i_2 < \ldots < i_{\text{m}} \leq n$, and $a_1 \subset b_1$, $a_2 \subset b_2$, ..., $a_{\text{m}} \subset b_{\text{m}}$. If s_1 is contained in s_2, s_1 is a subsequence of s_2. This relationship is denoted by $s_1 \leq s_1$. For example, the sequence $(a \rightarrow c)$ is a subsequence of $(ab \rightarrow cd)$.

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 also have an unique identifier called itemset-id (eid). Given a sequence database D, the support of a sequence s, denoted as $\delta(s, D)$, is the fraction of sequences in D that contain s. 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. The problem of sequence mining is to find all the frequent sequences in the database.

E-FORENSICS 2008, January 21-23, Adelaide, Australia

DOI 10.4108/wkdd.2008.2682

As an example, consider the database shown in Fig.1 which has five items (a *to* e) and five input-sequences. The figure also shows all frequent sequences with a min_sup of 40%. In this example database D has 22 frequent sequences.

Input database D	Frequent sequences (40%)
Sid input sequences	F1={a, b, c, d, e}
1, a→b→c→d→e	F2={bc, a→b, a→c, a→d, a→e, b→c,
2, a→bc→d	b→d, b→e, c→d, c→e, d→e}
3, a→b→e	F3={a→b→d, a→b→e, b→c→d,
4, dc→e	b→c→e, bc→d}
5, b→bc→d→e	F4={b→c→d→e}

Fig. 1. Frequent sequence mining example.

III. Related Works

The sequential pattern mining problem was first proposed in [1] and three mining algorithms, AprioriSome, AprioriAll and DynamicSome were presented. All of these algorithms adopted a similar process: litemsets, transformation and sequence stage. AprioriAll performs better than the other two as the literature reported. In subsequent work, GSP [6] algorithm was proposed by the same authors. GSP is a multi-phase iterative algorithm and requires multiple passes of database scanning. Independently, by introducing complex data structure, several more mining algorithms were proposed, such as PLWAP [3], SPADE [8], SPAM [2], SEQUEST [7], PrefixSpan [4], etc. PLWAP is based on the concept of WAPTree and has a desirable scalability. SPADE uses a vertical id-list structure and a lattice-theoretic approach to decompose the original search space into smaller oness. SPAM employs vertical bitmap for frequency counting and depth-first tree traversal strategy for candidate generation. SEQUEST generates candidate sequences efficiently based upon a DMA-Strips structure. Among these algorithms, SPAM is to our best of knowledge the fastest sequential patterns mining algorithm.

Several search space pruning strategies were proposed in algorithms discussed above. For example, in SPADE [8], before generating the id-list for a k-sequence, all its $(k-1)$ subsequences must be frequent. Based on Apriori principle, full pruning [7], s-step pruning and i-step pruning [2] are used to trim invalid computing or searching. Unfortunately, unlike our pruning strategies, these strategies are based on their own algorithm and data structure and can not be shared by other algorithms.

A. The SPAM algorithm

In this paper, we will take SPAM as an example for demonstrating the proposed pruning strategies and comparing performance. This section describes SPAM in more details.

1) SPAM algorithm

SPAM algorithm is based on the lexicographic sequence tree and can make either depth or width first traversal. The pseudo-code of its depth first traversal version is shown in Fig.2. Taking current frequent sub-sequent node $n = (s_1 \rightarrow \ldots \rightarrow s_k)$, s-step extension candidate items list S_n and i-step extension candidate items list I_n as input parameters, SPAM generates a new sub-sequence by either s-step or i-step. And this process recursively goes on until no more extension can be performed

2) s-step and i-step pruning

```
01) DFS-SPAM(node n = (s1→...→sk), Sn, In)
02)   Stemp = ; Itemp = ;
03)   For each (i in Sn)
04)     if (s1→...→sk→i) is frequent ) Stemp = Stemp ∪ {i};
05)   For each (i in Stemp)
06)     DFS-SPAM((s1→...→sk→i), Stemp, all elements in Stemp greater than i)
07)   For each (i in In)
08)     if ((s1→...→ski) is frequent) Itemp = Itemp ∪ {i};
09)   For each (i in Itemp)
10)     DFS-SPAM((s1→...→ski), Stemp, all elements in Itemp greater than i)
11) END of DFS-SPAM
```

Fig. 2. Pseudo-code of SPAM algorithm.

s-step pruning: Let a and b are two s-step candidate items for sequence s, and candidates $(s \rightarrow a)$ is frequent while $(s \rightarrow b)$ is infrequent. By the Apriori principle, both $(s \rightarrow a \rightarrow b)$ and $(s \rightarrow ab)$ must not be frequent, thus both of them can be pruned.

i-step pruning: Let a and b are two i-step candidate items for sequence $(s \rightarrow i_1 i_2 \ldots i_n)$, and $(s \rightarrow i_1 i_2 \ldots i_n a)$ is frequent while $(s \rightarrow i_1 i_2 \ldots i_n b)$ is not. By the Apriori principle, $(s \rightarrow i_1 i_2 \ldots i_n ab)$ must not be frequent and can be pruned.

IV. Proposed Pruning Strategies

A. The lexicographic sequence tree

The lexicographic subset tree T is presented originally by Rymon [5] and adopted to describe the itemset lattice in most of well-known frequent itemset mining algorithms such as MAFIA, CHARM. This approach is extended to describe the framework of sequence lattice in SPADE[8] and SPAM [2]. Let s_1 and s_2 are two sequences and s_1 is a subsequence of s_2, then s_2 is a descendant node of s_1. By this way, all sequences can be arranged in a lexicographic sequence tree whose root is null. 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 one s-extension step or i-extension step.

B. Candidate item sets for extensions

A sequence may be generated in many ways by s-extension or i-extension. For example, sequence $(a \rightarrow ab)$ can either be generated from $(a \rightarrow a)$ or $(a \rightarrow b)$, and the two according paths are $\emptyset \leq (a) \leq (a \rightarrow a) \leq (a \rightarrow ab)$ and $\emptyset \leq (a) \leq (a \rightarrow b) \leq (a \rightarrow ab)$. In order to void sequence duplicated generation, each node n in the tree can be associated with two sets, denoted as C_{Sn} and C_{In}. C_{Sn} and C_{In} are the set of candidate items that can be used to generate next level of sequences of n by s-extension, i-extension respectively. For example, let the C_{In} of sequence $(a \rightarrow b)$ be $\emptyset$, then $(a \rightarrow ab)$ can only be generated from $(a \rightarrow a)$.

In the Apriori-like sequence mining algorithms or lattice-theoretic approaches, if the candidate sets C_{Sn} or C_{In} may be reduced, i.e. the search space is reduced, thus the performance of these algorithms can be effectively improved.

C. Proposed pruning strategies

LEMMA 1: Given a frequent sequence s and its s-extension candidate set C_{Sn}. For each pair of items a, b in C_{Sn}, if $(a \rightarrow b)$ is infrequent, then b must not be an s-extension item for sequence $(s \rightarrow a)$.

Proof: Suppose b is an s-extension item for sequence $(s \rightarrow a)$, i.e. $(s \rightarrow a \rightarrow \ldots \rightarrow b)$ is a frequent sequence. According to Apriori Principle, each sub-sequence of $(s \rightarrow a \rightarrow \ldots \rightarrow b)$ must be frequent. Then $(a \rightarrow b)$ is frequent. This conflict with that $(a \rightarrow b)$ is infrequent. Thus, b is not an s-extension item for sequence $(s \rightarrow a)$ and can be pruned.

LEMMA 2: Given a frequent sequence $s = (s' \rightarrow i_{\text{s1}} i_{\text{s2}} \ldots i_{\text{sn}})$ and its i-extension candidate set C_{In}. For each pair of items a, b in C_{In}, if (ab) is an infrequent sequence, then b must not be an i-extension item for sequence $(s' \rightarrow i_{\text{s1}} i_{\text{s2}} \ldots i_{\text{sn}} a)$.

Proof: Suppose b is an i-extension item for sequence $(s' \rightarrow i_{\text{s1}} i_{\text{s2}} \ldots i_{\text{sn}} a)$, i.e. $(s' \rightarrow i_{\text{s1}} i_{\text{s2}} \ldots i_{\text{sn}} a \ldots b)$ is a frequent sequence. According to Apriori Principle, each sub-sequence of $(s' \rightarrow i_{\text{s1}} i_{\text{s2}} \ldots i_{\text{sn}} a \ldots b)$ must be frequent. Then (ab) is frequent, too. This conflict with that (ab) is infrequent. Thus, b is not a i-extension item for sequence $(s' \rightarrow i_{\text{s1}} i_{\text{s2}} \ldots i_{\text{sn}} a)$and can be pruned.

Suppose there are two sequence lists, denoted as $S-list$, $I-list$, which contains all frequent 2-sequences with size of 2 and size of 1 respectively. By sharing $S-list$ and $I-list$, we have pruning strategies SEP and IEP, which are as follows.

Sequence extension Pruning (abbr. SEP): Given a frequent sequence s and its according s-extension candidate set C_{Sn}, and $s1$ is a not null subset of C_{Sn}. Let $SEP(s1) = \bigcap\{b | a \in s1 \; and \; (a \rightarrow b) \in S-list\}$, according to Lemma 1, the new s-extension candidate set of sequence $(s \rightarrow s1)$ is $C^*_{\text{Sn}} = C_{\text{Sn}} \bigcap SEP(s1)$.

Item extension Pruning (abbr. IEP): Given a frequent sequence $s = (s' \rightarrow i_{\text{s1}} i_{\text{s2}} \ldots i_{\text{sn}})$ and its according i-extension candidate item set C_{In}, and $i1$ is a not null subset of C_{In}. Let $IEP(i1) = \bigcap\{b | a \in i1 \; and \; (ab) \in I-list\}$, according to Lemma 2, the new i-extension candidate set for sequence $(s' \rightarrow i_{\text{s1}} i_{\text{s2}} \ldots i_{\text{sn}} i1)$ is $C^*_{\text{In}} = I_{\text{Sn}} \bigcap IEP(i1)$.

Algorithm 2(a): SEP pruning

```
1) SEP_Prune(s1, CSn,S-list)
2)  temp = CSn;
3)  for (each a in s1)
4)    {t = null;
5)     for all ((a→b) in S-list)
6)       t = t ∪ {b};
7)     temp = temp ∩ t; }
8) Return temp;
```

Algorithm 2(b): IEP pruning

```
1) IEP_Prune(i1, CIn,I-list)
2)  temp = CIn;
3)  for (a in i1)
4)    {t = null ;
5)     for all ((ab) in I-list)
6)       t = t ∪ {b};
7)     temp = temp ∩ t; }
8) Return temp;
```

Fig. 3. Pseudo-code of SEP and IEP pruning algorithms.

D. Example

Considering the database D and minimal support threshold in Fig. 1, the $S-list$ of database is $\{a \rightarrow b, a \rightarrow c, a \rightarrow d, a \rightarrow e, b \rightarrow c, b \rightarrow d, b \rightarrow e, c \rightarrow d, c \rightarrow e, d \rightarrow e\}$ and the $I-list$ is $\{bc\}$. Suppose we are at node $(a \rightarrow b)$ in the tree, the node's C_{Sn} is $\{c, d, e\}$ and $C_{\text{In}} = \{c\}$. Without pruning, the possible s-extended sequences are $(a \rightarrow b \rightarrow a)$, $(a \rightarrow b \rightarrow b)$, $(a \rightarrow b \rightarrow c)$, $(a \rightarrow b \rightarrow d)$ and $(a \rightarrow b \rightarrow e)$. Because $(b \rightarrow a)$ and $(b \rightarrow b)$ are not in S-list, sequence $(a \rightarrow b \rightarrow a)$ and $(a \rightarrow b \rightarrow b)$ must be infrequent. Item a and b should be trimmed off from node's C_{Sn}. Hence, we do not have to perform these s-extensions. Similarly, item d and e should be trimmed off from node's C_{In}.

E. $SPAM_{SEPIEP}$: Using SEP and IEP to optimize SPAM

In order to test effectiveness of SEP and IEP, we optimize SPAM by using proposed pruning strategies and present the improved algorithm, $SPAM_{\text{SEPIEP}}$, shown in Fig.4. The mining process is divided into two steps. In the first step, $SPAM_{\text{SEPIEP}}$ generates frequent 2-sequences and stores them in $S-list$ and $I-list$ respectively. In the second step, the algorithm recursively traverses the search space in a depth-first manner. At each node, SEP and IEP are used to trim node's s-extension candidate set C_{Sn} and i-extension candidate set C_{In}. The rest part of algorithms is as same as that in SPAM.

Stage 1

```
1) Start-DFS-SPAMSPIP ()
2)  S-list = {successful first s-extensions in lexicographic tree};
3)  I-list = {successful first i-extensions in lexicographic tree};
4)  For each ((a→b) in S-list)
5)     CSn = SEP_Prune ({a}, I,S-list) ∩ SEP_Prune ({b}, I,S-list) ;
6)     CIn = SEP_Prune ({a}, I,S-list) ∩ IEP_Prune ({b}, I,I-list);
7)     DFS-SPAMSPIP((a→b), CSn, CIn, b);
8)  For each ((ab) in I-list)
9)     CSn = SEP_Prune ({a}, I,S-list) ∩ SEP_Prune ({b}, I,S-list) ;
10)    CIn = IEP_Prune ({a}, I,I-list) ∩ IEP_Prune ({b}, I,I-list);
11)    DFS-SPAMSPIP((ab), CSn, CIn, b);
12) End of Start-DFS-SPAMSPIP
```

Stage 2

```
1) DFS-SPAMSPIP(node n = (s1→…→sk), CSn, CIn, ia)
2)  Sn'= SEP_Prune ({ia}, CSn,S-list);
3)  For each (i in Sn')
4)     If ((s1→…→sk→i) is frequent )
5)        Stemp = Stemp ∪ {i};
6)  For each (i in Stemp)
7)     DFS-SPAMSPIP((s1→…→sk→i), Stemp, all
                elements in Stemp greater than i, i)
8)  In'=Prune-IEP({ia}, CIn,I-list);
9)  For each (i in In')
10)    if ((s1→…→ski) is frequent)
11)       Itemp = Itemp ∪ {i};
12)  For each (i in Itemp)
13)     DFS-SPAMSPIP ((s1→…→ski), Stemp, all
                elements in Itemp greater than i, i)
14) END of DFS-SPAMSPIP
```

Fig. 4. Pseudo-code of $SPAM_{\text{SEPIEP}}$.

V. Experimental Evaluation

To test the performance improvement of SEP and IEP strategies, an extensive set of experiments were performed upon an Intel Pentium 4 CPU 1.7GHz PC with 512MB main memory, running Microsoft Windows 2003 server. Source code of SPAM was downloaded from the (http://himalaya-tools.sourceforge.net/) and modified with proposed strategies. Same as SPAM, all synthetic datasets are generated by using the IBM AssocGen program [1].

A. Performance comparison with SPAM

Firstly, a set of experiments were performed for studying the performance of the $SPAM_{\text{SEPIEP}}$ by compare it with SPAM. The generated datasets include one small database (Fig.5(a)), two medium-sized ones (Fig.5(b) and (c)) and a large one (Fig.5(d)). The experiments show that $SPAM_{\text{SEPIEP}}$ outperforms SPAM by a factor of about 10 on small dataset. Although, there is no distinct magnitude difference between $SPAM_{\text{SEPIEP}}$ and SPAM on reasonably large dataset, the running time of $SPAM_{\text{SEPIEP}}$ is decreased by 30% to 50% than that of SPAM.

In small sequence databases, frequent 2-sequences are only a small portion in all 2-sequences. As $SPAM_{\text{SEPIEP}}$ traverses the lexicographic sequence tree, a mass of infrequent extensions are pruned. On the other side, in large database, a majority of 2-sequences are frequent ones. When $SPAM_{\text{SEPIEP}}$ traverses the lexicographic sequence tree, only few branches are pruned. This is the primary reason why SEP and IEP strategies perform effectively in small database while not so effectively in large ones.

Secondly, we study the scale-up performance of

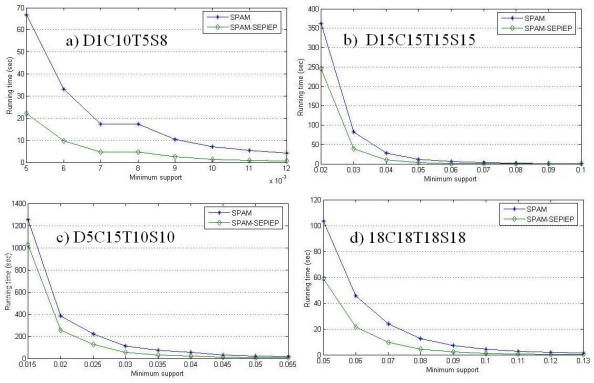

Fig. 5. Performance comparison: Varying support for synthetic datasets.

$SPAM_{\text{SEPIEP}}$ as several parameters in dataset generation were varied and min_sup is kept fixed. For each test, only one parameter was varied and 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 are shown in Fig.6. It can be easily observed that: (1) the proposed prune strategies can effectively improve the performance of SPAM. (2) The trend of curves of $SPAM_{\text{SEPIEP}}$ is same as that of SPAM, so the proposed prune strategies are independent of SPAM and can be used for other algorithms.

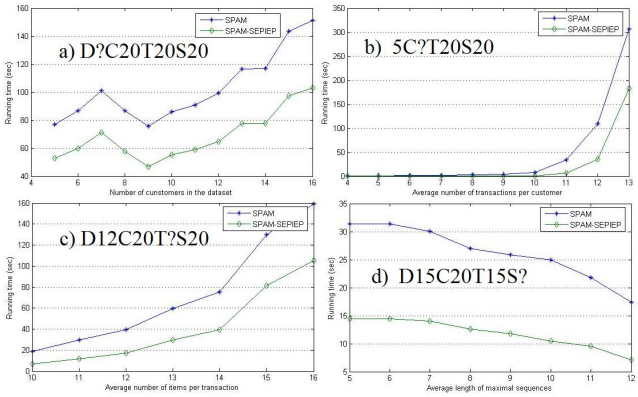

Fig. 6. Performance comparison: Varying datasets generation parameters.

VI. Conclusion

In this paper, by systematically exploring the search space of frequent sequence mining, we propose two novel pruning strategies, SEP and IEP, which can be used in all Apriori-like sequence mining algorithms or lattice-theoretic approaches. With a little more memory overhead, proposed pruning strategies can prune invalid search space. These strategies are independent of underlying enumerating methods and data structures of sequence mining algorithms, thus can be shared among different algorithms. For effectiveness testing reason, we optimize SPAM, one of the fastest algorithms for sequential pattern mining, by using proposed pruning strategies and present the improved algorithm, $SPAM_{\text{SEPIEP}}$. The experimental results show that $SPAM_{\text{SEPIEP}}$ outperforms SPAM on every dataset.

References

[1] R. Agrawal, R. Srikant. Mining Sequential Patterns. In ICDE 1995, Volume 6, pp.3-14, 1995.

[2] J. Ayres, J. Gehrke, T. Yiu, and J. Flannick. Sequential PAttern Mining using A Bitmap Representation, In ACM SIGKDD 2002, pp. 429-435, 2002.

[3] C.I. Ezeife, and Y. Lu. Mining Web Log Sequential Patterns with Position Coded Pre-Order Linked WAP-Tree, The International Journal of Data Mining and Knowledge Discovery(DMKD), Volume 10, pp.5-38, 2005.

[4] 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 ICDE 2001, pp.215-226, 2001.

[5] 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.

[6] R. Srikant, R. Agrawal. Mining Sequential Patterns: Generalizations and Performance Improvements, In Proc. 5th Int. Conf. Extending Database Technology (EDBT'96), Volume 1057, pp.3-17, 1996.

[7] H. Tan, T.S. Dillon, F. Hadzic, and E. Chang. SEQUEST: mining frequent subsequences using DMA Strips, In Proceedings of the Seventh International Conference on Data Mining and Information Engineering 2006, pp.35-328, 2006.

[8] M.J. Zaki. SPADE: An Efficient Algorithm for Mining Frequent Sequences. Machine Learning, Volume 0, pp. 1-31, 2000.