Effective Mining Sequential Pattern by Last Position Induction

Zhenglu Yang and Masaru Kitsuregawa
The University of Tokyo
Institute of Industrial Science
4-6-1 Komaba, Meguro-Ku
Tokyo 153-8305, Japan
{yangzl,kitsure}@tkl.iis.u-tokyo.ac.jp

Abstract Sequence pattern mining is an important research problem because it is the basis of many other applications. Yet how to efficiently implement the mining is difficult due to the inherent characteristic of the problem - the large size of the data set. In this paper, by combining SPAM, we propose a new algorithm called LAst Position INduction Sequential PAttern Mining (abbreviated as LAPIN-SPAM), which can efficiently get all the frequent sequential patterns from a large database. The main difference between our strategy and the previous works is that when judging whether a sequence is a pattern or not, they use S-Matrix by scanning projected database (PrefixSpan) or count the number by joining (SPADE) or ANDing with the candidate item (SPAM). In contrast, LAPIN-SPAM can easily implement this process based on the following fact - if an item's last position is smaller than the current prefix position, the item can not appear behind the current prefix in the same customer sequence. When scanning the database for the first time, LAPIN-SPAM will construct the ITEM_IS_EXIST_TABLE. Then in every recursive, it only needs to check this table, avoid constructing S-Matrix, joining or ANDing. Because of the recursive characteristic in this research area, LAPIN_SPAM can save much time by improving efficiency for each recursive. The experimental results confirm that our algorithm outperforms previous algorithms.

Keywords Data Mining, General Mining, Knowledge Discovery.

1. Introduction

Sequential pattern mining is an important research theme because it is the basis of many applications. Consider the sales database of a store. If we know that "80% of the persons who buy television also buy video camera within a week", we can efficiently use the shelf space to convenient the customers. Another business example is that if we know that every time Microsoft stock drops 5%, IBM stock will also drops at least 4% within three days, we can properly decide what to do when economic problem happens. Sequential pattern mining is also used in biological data [12], telecommunication network analysis [1], web access click stream [9], and so on.

In the knowledge discovery and data mining research area, there exist some common methods. Basic level statistical analysis [3] is the most common way to extract knowledge. Clustering [10] is also used because it is very useful, especially when the customer data is merged with the demographic data to generate the customer segmentations. Classification [11] can also be used to categorize customers based on the properties and related demographic stamp is an information. Since the time important attribute of each dataset, sequence mining plays an important role in knowledge extraction from many databases.

Sequence discovery can essentially be thought of as association discovery over a temporal database. While association rules [13] discover only intra-event patterns (itemsets), sequential pattern mining discovers inter-event patterns (sequences). The set of all frequent sequences is a superset of the set of frequent itemsets.

1.1. Problem Definition

Based on the problem definition in [2], a *sequence database* S is a set of customer $\langle cid, s\rangle$, where cid is a customer_id and s is a customer sequence. A *sequence* s is denoted as $(s_1, s_2, \ldots, s_l)$, where s_i is an *itemset*, or can be called an *element*, which is a set of *items*. The number of items in a sequence is called the *length* of the sequence. A sequence with length *l* is called an $l-sequence$. A sequence $s_a = (a_1, a_2, \ldots, a_n)$ is contained in another sequence $s_b = (b_1, b_2, \ldots, b_m)$, if there exist integers $1 \leq i_1 < i_2 < \ldots < i_n \leq m$ such that $a_1 \subseteq b_{i_1}$, $a_2 \subseteq b_{i_2}$,..., $a_n \subseteq b_{i_n}$. We can call s_a a $subsequence$ of s_b and s_b a $supersequence$ of s_a. Given a sequence $s = (s_1, s_2, \ldots, s_l)$ and an item α, $s \diamond_s \alpha$ means s concatenates with α whose name is *sequence extension*, $s \diamond_s \alpha$=$(s_1, s_2, \ldots, s_l, \{\alpha\})$, denoted as $S-step$. $s \diamond_i \alpha$ means s concatenates with α whose name is *itemset extension*, $s \diamond_i \alpha$=$(s_1, s_2, \ldots, \{s_l, \alpha\})$, denoted as $I - step$.

A customer $\langle cid, s\rangle$ is said to contain a sequence α, if α is a subsequence of s. The support of a sequence α in a sequence database S is the number of customer sequences in the database containing α, denoted as $support(\alpha)$. Given a user specified positive integer ε, a sequence α is called a frequent sequential pattern if $support(\alpha) \geq \varepsilon$. In this paper, our objective is to find the complete set of sequential patterns in the database in an efficient way.

CID	Customer Sequence
10	ac(bd)c(ab)
20	b(cd)acd
30	a(bc)(acd)c

Table 1. Sequence Database

Example. Let our running database be sequence database S given in Table 1 and min_support=2. We will use this sample database throughout of this paper. We can see that the set of items in the database is {a,b,c,d}. A 2-sequence $\langle ac\rangle$ is contained in the customer sequence 10, 20 and 30, respectively. So the support of $\langle ac\rangle$ is 3, which is larger than the user specified minimum support, from where we can know that $\langle ac\rangle$ is a frequent pattern.

1.2. Related Works

The problem of mining sequential patterns was first introduced in [5], in which they presented three algorithms for solving sequential pattern mining problem. The AprioriAll algorithm was shown to perform better than the other two approaches. In [2], the same authors proposed the GSP algorithm. GSP strategy generates candidate k-sequences from frequent (k-1)-sequences in iteration based on the anti-monotone property that all the subsequences of a frequent sequence must be frequent. The authors also introduced maximum gap, minimum gap, and sliding window constraints on the discovered sequences.

Zaki [6] proposed SPADE to find frequent sequences using efficient lattice [8] search techniques and simple join operations. It divides the candidate sequences into groups by items such that each group can be completely stored in the main memory. At first the sequence database is transformed into a vertical ID-List database format, in which each id is associated with corresponding items and the time stamp. There are two methods used in SPADE to enumerate frequent sequences: Breadth First Search (BFS) and Depth First Search (DFS). It computes the support count of a candidate k-sequence generated by merging the ID-Lists of any two frequent (k-1)-sequences with the same (k-2)-prefix in each iteration. Ayres et al. [4] proposed SPAM algorithm, which is built based on SPADE's lattice concept but represents each ID-List as a vertical bitmap. The detail of SPAM will be described in Section 2.2.

GSP, SPADE and SPAM algorithms can be categorized as candidate-generate-test method. On the other hand, Pei et al. proposed a pattern growth algorithm named PrefixSpan [7], which employs the projection scheme to project the customer sequences into different groups called *projected databases*. All the customer sequences in each group have the same prefix. For the example database in Table 1, the PrefixSpan algorithm first scans the database to find the frequent 1-sequences, i.e. $\langle a\rangle$, $\langle b\rangle$, $\langle c\rangle$ and $\langle d\rangle$. Then the sequence database is divided into different partitions according to these frequent items, each partition is the projection of the sequence database that take the corresponding 1-sequence. For these projected databases, the PrefixSpan algorithm continues the discovery of frequent 1-sequences to form the frequent 2-sequences with the corresponding prefix. Recursively, the PrefixSpan algorithm generates the projected database for each frequent k-sequence to find frequent (k+1)-sequences. To get the sequence pattern, PrefixSpan constructs a S-Matrix in each recursive step, it spends much time on the support counting due to the inherent characteristic of this research area - large number of the iterations.

As proposed in [16] , all of these algorithms use some common strategies such as candidate sequence pruning, database partitioning and customer sequence reducing.

1.3. Overview of our algorithm

Instead of ANDing operation and comparison when testing the candidate items, we can directly accumulate the support of them by checking ITEM_IS_EXIST_TABLE.

In some customer sequence, if the last position of some item is smaller than the current position, it means that the item will not appear behind the current prefix in the same customer sequence. For example, we know that the first positions of item *a* in Table 1 are 10:1, 20:3, 30:1, where sid:eid represents the customer sequence ID and the element ID. We can see that the last position of item *b* in the customer sequence 20 is 1, smaller than the first position of item *a* which is 3. It means that item *b* cannot appear behind item *a* in the same customer sequence.

From the above analysis, we know that our proposed algorithm can implement support counting by avoiding to ANDing, compare or construct and scan the projected database to find all frequent patterns.

The main difference between our algorithm and all algorithms proposed before is that when judging a sequence is a pattern or not, they use comparison, ANDing or S-Matrix, which need to implement in every recursive step. But for the LAPIN_SPAM, we can directly accumulate the candidate's support by using ITEM_IS_EXIST_TABLE, which is constructed while scanning the database for the first time. Because the unavoidable large iteration when mining, our proposed algorithm can effectively reduce the cost used to test the candidates.

From the experiments we have done, which we will explain in detail in Section 3, it can be seen that our algorithm outperforms the current fastest algorithm, named SPAM, what confirms the efficiency of our algorithm.

The rest of this paper is organized as follows: In section 2 we introduce the detail of LAPIN_SPAM algorithm. In section 3 the experiment of comparing LAPIN_SPAM and SPAM is presented. Finally we make a conclusion and present future works in section 4.

2. LAPIN-SPAM (LAst Position INduction Sequential PAttern Mining)

2.1. Lexicographic Tree

Because our LAPIN_SPAM algorithm is built based on SPAM, we use the lexicographic tree, which was also shown in [4][15] as the search path of our algorithm. Furthermore, we apply the lexicographic order which is defined in the same way as in [4]. It uses the depth first search strategy. Fig. 1 shows a sample of the lexicographic tree. We obey the following rules based on DFS:

(a) if $\gamma^{'} = \gamma \diamond \theta$, then $\gamma < \gamma^{'}$; (Search the prefix first, then the sequence. For example, we first search $\langle a \rangle$, then $\langle (ab) \rangle$.)

(b) if $\gamma = \alpha \diamond_s \theta$ and $\gamma^{'} = \alpha \diamond_i \theta$, then $\gamma < \gamma^{'}$; (Search the sequence-extension first, then the itemset-extension. For example, we first search $\langle ab \rangle$, then $\langle (ab) \rangle$.)

(c) if $\gamma = \alpha \diamond \theta$ and $\gamma^{'} = \alpha \diamond \theta^{'}, \theta < \theta^{'}$ indicates $\gamma < \gamma^{'}$. (For two sequences which have the same prefix, search them based on the alphabetic order of the postfix. For example, we first search $\langle aa \rangle$, then $\langle ab \rangle$.)

Because our algorithm is built based on SPAM [4], first we will introduce SPAM.

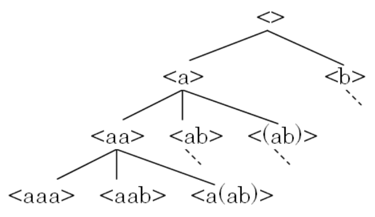


Figure 1. Lexicographic Tree

2.2. SPAM

Ayres et al. [4] proposed SPAM algorithm, which utilizes a bitmap representation of the database. While scanning the database for the first time, a vertical bitmap is constructed for each item in the database, and each bitmap has a bit corresponding to each element of the sequences in the database. If an item appears in an element, the bit corresponding to the element of the bitmap for the item is set to *one*; otherwise, the bit is set to *zero*. The size of a sequence is the number of elements contained in the sequence. A sequence in the database of size between $2^k + 1$ and 2^{k+1} is considered as a 2^{k+1}-bit sequence. The bitmap of a sequence will be constructed according to the bitmaps of items contained in it.

To generate and test the candidate sequences, SPAM uses two steps: S-step and I-step, based on the lattice concept. As a depth-first approach, the overall process starts from S-step and then I-step. To extend a sequence, the S-step appends an item to it as the new last element, and the I-step appends the item to its last element if possible. Each bitmap partition of a sequence to be extended is transformed first in the S-step, such that all bits after the first bit with value one are set to one. Then the resultant bitmap of the S-step can be obtained by doing ANDing operation for the transformed bitmap and the bitmap of the appended item, as shown in Fig. 2 based on the example database in Table 1. On the other hand, the I-step just uses the bitmaps of the sequence and the appended item to do ANDing operation to get the resultant bitmap, as shown in Fig. 3. The support counting becomes a simple check how many bitmap partitions not containing all zeros. Yet for the inherent characteristic existed in the sequential pattern mining problem, these ANDing operations cost a lot during the whole mining process, which should be reduce for efficiency improving.

According to the two processes existed in SPAM, it uses two pruning techniques: S-step pruning and I-step pruning, based on the Apriori heuristic to minimize the size of the candidate items.

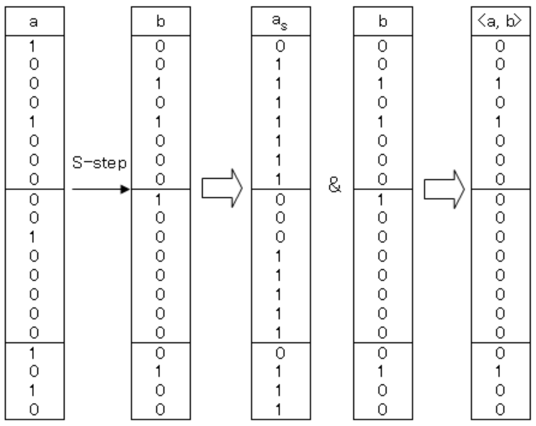


Figure 2. SPAM S-step join

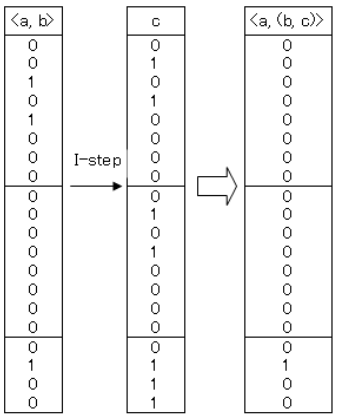


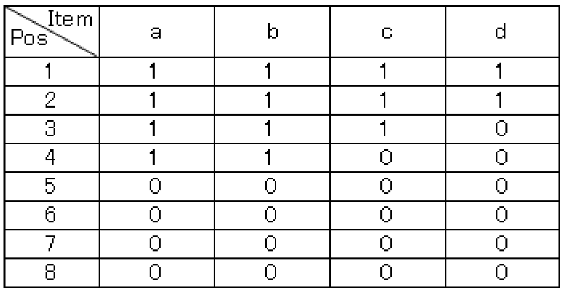
Figure 3. SPAM I-step join

2.3. LAPIN-SPAM

Although the authors of SPAM claim that they efficiently count the support of the candidate, we have found more efficiency improving space in this support counting process.

In SPAM, to judge a candidate is a pattern or not, it does as many ANDing operation as to the number of customers involved. For example, if there are 10000 customers in certain dataset, it will cost 10000 ANDing operation time for each candidate item testing. Consider the recursive characteristic in the implementation, this cost is too big. So how to avoid this ANDing operation becomes essential step.

As mentioned earlier, if given a current position in certain customer, we can know which items are behind current position and which are not based on the last position of them. So a naive method to judge a candidate is to compare the last position of it with the current position. This is in fact the same cost as ANDing operation in SPAM. To avoid this comparison or ANDing operation, we can construct a ITEM_IS_EXIST_TABLE when scanning the database for the first time. In each iteration, we only need to check this table to get information that a candidate is behind current position or not. By this way, we can save much time by avoiding ANDing operation or comparison.

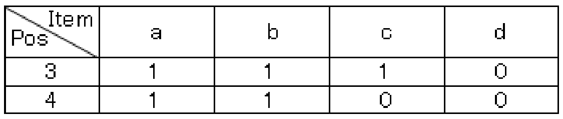


Item / Pos	a	b	c	d
1	1	1	1	1
2	1	1	1	1
3	1	1	1	0
4	1	1	0	0
5	0	0	0	0
6	0	0	0	0
7	0	0	0	0
8	0	0	0	0

Figure 4. ITEM_IS_EXIST_TABLE

Fig.4, which is built based on the example database in Table 1, shows one part of the ITEM_IS_EXIST_TABLE for the first customer. The left column is the position number and the top row is the item ID. In the table, we use bit vector to represent candidates existence for respective position. Bit value is 1 indicates the item existing, otherwise the item does not exist. The bit vector size is equal to the total number of the candidate items. For example, if the current position is 2, we can get its corresponding bit vector as 1111, which means that all candidates can be appear behind current prefix. When the current position is 4, we can get the bit vector as 1100, indicates that only item a and b exist in the same customer sequence after the current prefix. To accumulate the candidate sequence's support, we only need to check this table and add the corresponding item's vector value, avoiding comparison, ANDing operation or constructing S-Matrix in each recursive step, which largely improve efficiency during mining. Attention that here we only discuss the S-Step process, the reader can easily extend it to the I-Step process based on the same strategy.

2.3.1. Space Optimization SPAM assumes that the whole vector representation of the database should be filled in the main memory, yet the space necessary is always a key factor of an algorithm. As Fig. 4 shows, we can easily know that the main memory used in LAPIN_SPAM is no more than twice of that used in SPAM, because each item needs one bit for every transaction no matter it exists or not in the ITEM_IS_EXIST_TABLE.



Item / Pos	a	b	c	d
3	1	1	1	0
4	1	1	0	0

Figure 5. Optimized ITEM_IS_EXIST_TABLE

Symb.	Meaning
D	Number of customers in 000s
C	Average number of transactions per customer
T	Average number of items per transaction
N	Number of different items in 000s
S	Average length of maximal sequences

Table 2. Parameters used in datasets generation

After consideration, we find that only part of the table is useful and most are not. For example in Fig.4, when the current position is smaller than 3, all items exist and when the position is larger than 4, there is no item existing. So the useful information is store in some key positions' lines. We define *key position* as follows: Given a position, if its corresponding bit vector is different from that of the position one smaller than it (except the one whose bit vector is equal to 0), this position is called *key position*. For example, in Fig.4, the position 3 and 4 are key positions and others are not (position 5 is not because its bit vector is equal to 0). We can find that these key positions are indeed the last positions of the candidates items (except the last one). The optimized ITEM_IS_EXIST_TABLE is shown in Fig.5, which stores only two bit vectors instead of eight ones shown in Fig.4. For long pattern dataset, this space saving strategy is more efficient. Through thorough experiments what we will mention in section 3, the memory used to store the ITEM_IS_EXIST_TABLE is less than 10 percent of the one used in SPAM, which can be neglected when comparing LAPIN_SPAM and SPAM efficiency.

3. Experiment

We perform the experiments on a 1.6GHz Intel Pentium(R)M PC machine with 1G memory, running Microsoft Windows XP. The synthetic datasets are generated by the IBM data generator described in [2]. The meaning of the different parameters used to generate the datasets is shown in Table 2. As [4] shown, SPAM is by far the fastest algorithm when mining to get the whole set of the sequential patterns, nearly an order faster than PrefixSpan [7] and SPADE [6] for large datasets. So we only compare SPAM and LASIN_SPAM in this paper. All methods are implemented using Microsoft Visual C++ 6.0.

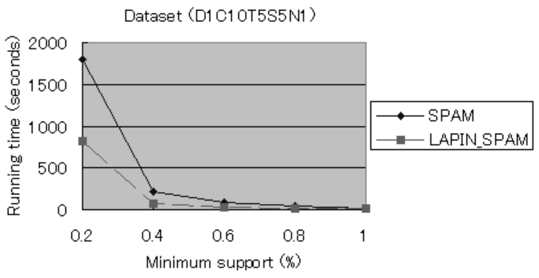


Figure 6. Varying support for small dataset

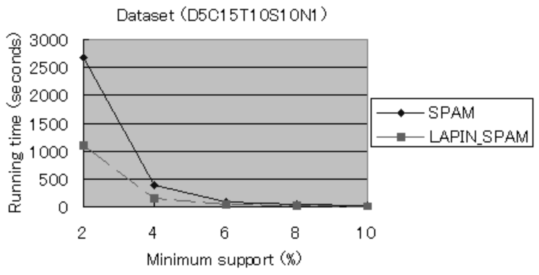


Figure 7. Varying support for medium-sized dataset

Because the dataset size plays a key role on the performance of the algorithm, we first compare SPAM and LAPIN_SPAM for different size of datasets, as Fig. 6, Fig. 7 and Fig. 8 shown. This set of tests presents that LASIN_SPAM outperforms SPAM by about 2 to 3 times for different size of the datasets.

In the second group of the experiments, we consider the different parameters used to generate the datasets on the effect of the performance. Fig. 9 shows the result when changing the number of the customers. Fig. 10 presents the effect when varying average number of transactions per customer. Fig. 11 shows the result when changing the average number of items per transaction parameter. Fig. 12 modifies the average length of maximal sequences and the variable in Fig. 13 is the number of different items in the datasets. We can see that no matter which parameter changes, LAPIN_SPAM is always faster than SPAM about 2 to 3 times. The primary reason that LAPIN_SPAM performs so well for all datasets is due to avoiding ANDing operation or comparison of the bitmap for efficient counting. This process is critical because it is performed many times at each recursive step, and LAPIN_SPAM can save much time compared with SPAM.

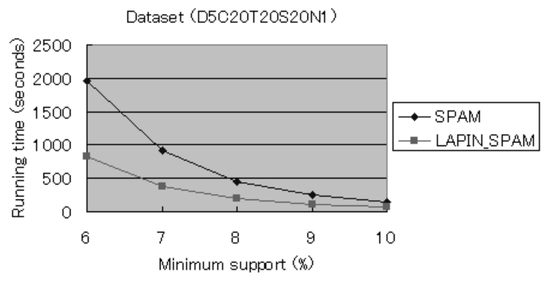


Figure 8. Varying support for large dataset

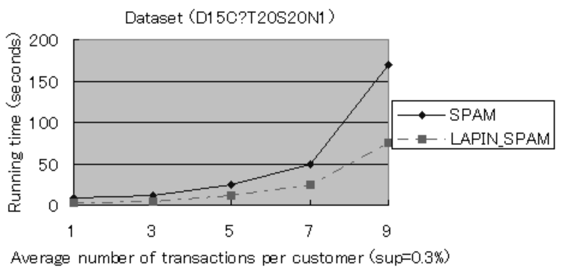


Figure 10. Varying num. of trans. per customer

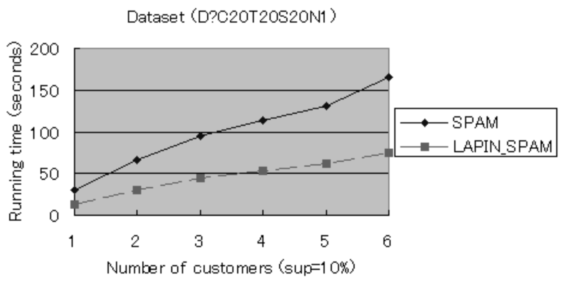


Figure 9. Varying number of customers

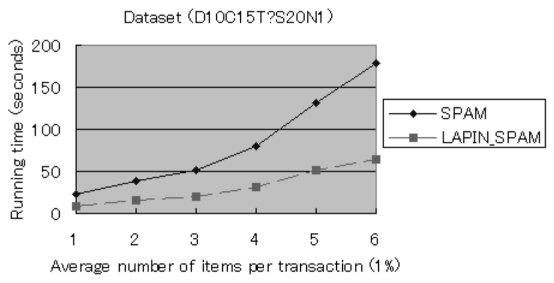


Figure 11. Varying num. of items per trans.

4. Conclusion and Future Works

In this paper, we have proposed a new strategy, named LAPIN_SPAM, to mine sequential patterns. The key idea is that if given the current position, we can immediately know which items should be appear behind current prefix items, based on the items' last positions. LAPIN_SPAM avoids ANDing operation or comparison in each iteration in the support counting process, which can largely improve the efficiency. In fact, for any time-series database, the last positions of different items should be paid more attention because they can be treated as the judgement for the items' existence at each recursive step. We also present a set of thorough experiments when comparing LASPIN_SPAM and SPAM on different parameters of the datasets. The result shows that our algorithm outperforms the current fastest one by about 2 to 3 times.

Currently we only implement S-Step process based on LAPIN strategy. We expect our algorithm will become more efficient after implement on I-Step in the near future. Yet there is a paid-off between space and time while mining [14]. To make it a practical tool, we need to find a balance through more experiments. We also will apply our strategy on other more applications, such as dynamic database(data stream, etc).

5. Acknowledgement

The authors are grateful to Jay Ayres for providing us the source code of SPAM.

References

[1] K.Hatonen, M.Klemettinen, H.Mannila, P.Ronkainen, and H.Toivonen, “Knowledge discovery from telecommunication network alarm databases”, In *12th Intl. Conf. Data Engineering*, 1996.

[2] R.Srikant and R.Agrawal, “Mining sequential patterns: Generalizations and performance improvements”, In *5th Intl. Conf. Extending Database Technology(EDBT)*, pp.13-17, Avignon, France, Mar.1996.

[3] M.Chen, A.Zheng, J.Lloyd, M.Jordan, and E.Brewer, ”A statistical learning approach to failure diagnosis”, In *International Conference on Autonomic Computing (ICAC-04)*, New York, May 1995.

[4] J.Ayres, J.Flannick, J.Gehrke, and T.Yiu, “Sequential Pattern Mining using A Bitmap Representation”, In *ACM SIGKDD Conference*, pp.429-435, 2002.

[5] R.Agrawal and R.Srikant, “Mining sequential patterns”, In *11th Int'l ICDE*, pp.3-14, 1995.

[6] M.J.Zaki, “SPADE:An Efficient Algorithm for Mining Frequent Sequences”, In *Machine Learning Journal*, Vol 42, No.1/2, 2001.

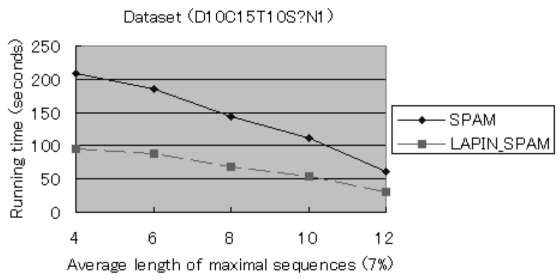


Figure 12. Varying number of different items

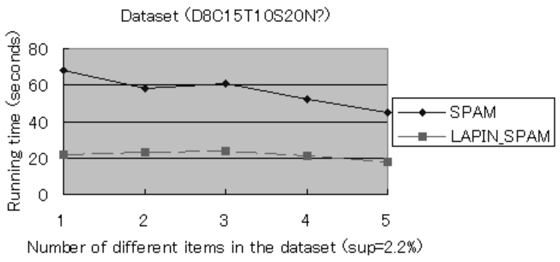


Figure 13. Varying average length of maximal sequences

[7] J.Pei, J.Han, M.A.Behzad, and H.Pinto, "PrefixSpan:Mining Sequential Patterns Efficiently by Prefix-Projected Pattern Growth", In *17th Int'l ICDE*, Heidelberg, Germany, 2001.

[8] B.A.Davey, and H.A.Priestley, "Introduction to Lattices and Order", In *Cambridge University Press*, 1990.

[9] Y.H.Wu and A.L.P.Chen, "Prediction of Web Page Accesses by Proxy Server log", In *World Wide Web: Internet and Web Information Systems*, 5(1):67-88, 2002.

[10] J.Han and M.Kamber, ”Data Mining: Concepts and Techniques”, *Morgan Kaufmann Publishers*, 2001.

[11] M.Mehta, R.Agrawal, and J.Rissanen, ”SLIQ: A fast scalable classifier for data mining”, *Proc. of the Fifth Int'l Conference on Extending Database Technology*, Avignon, France, 1996.

[12] J.K.Bonfield and R.Staden, "ZTR: A New Format for DNA Sequence Trace Data", In *Bioinformatics*, 18(1):3-10, 2002.

[13] R.Agrawal, H.Mannila, R.Srikant, H.Toivonen and A.I.Verkamo, ”Fast Discovery of Association Rules”, *Advances in Knowledge Discovery and Data Mining*, Chapter 12, AAAI/MIT Press, 1995.

[14] C.Antunes and A.L.Oliveira, ”Sequential Pattern Mining Algorithms: Trade-offs between Speed and Memory”, *Proc. of 2 nd Intfl Workshop on Mining Graphs, Trees and Sequences (MGTS 2004)*, Pisa, Italy, 2004.

[15] R.J.Bayardo, "Efficiently mining long patterns from databases", In *Proc. Int. Conf. Management of Data (SIGMOD'98)*, pp.85-93, Seattle, WA, June 1998.

[16] D.Chiu, Y.Wu, A.L.P.Chen, "An Efficient Algorithm for mining Frequent Sequences by a New Strategy without Support Counting", In *20th Intl. Conf. Data Engineering*, 2004.

[17] H.Mannila, H.Toivonen, and I.Verkamo ”Discovering frequent episodes in sequences”, In *1st Int. Conf. Knowledge Discovery and Data Mining*, 1995.