Fast Sequential Pattern Mining With UpDown Directed Acyclic Graph

Iswarya. T[1] and Sivakumar. E[2]

[1,2] Computer Science and Engineering, Sri Venkateswara College of Engineering, Sriperumbudur, Tamil Nadu, India

Abstract

Sequential Pattern Mining is a technique used to discover frequent patterns in a sequence database. An UpDown Directed Acyclic Graph (UDDAG) is a data structure used for sequential pattern mining which derives patterns based on the bidirectional pattern growth approach. In existing, to derive a pattern, a database projection method was used. This method projects the database according to their prefix and suffix recursively and makes the database much smaller to further levels. UDDAG derives length-(k+1) patterns based on the projected databases of length-k patterns recursively. However this method takes time to build the database and to find the support count. To overcome the problem, an approach to represent the item sets in database as vertical bitmap representation is proposed. It efficiently stores the database as vertical bitmaps, where each bitmap represents an item set in the database. Efficiently counting the support of the item set is one of the main advantages of the vertical bitmap representation of the data.

Keywords: *Sequential Pattern Mining, Database Projection, Vertical Bitmaps, UDDAG.*

1. Introduction

Data Mining is the process which helps to extracting interesting and hidden information or patterns from large information repositories such as relational database, data warehouses, XML repository, etc. Sequential Pattern Mining is an important problem in data mining.

Sequential Pattern Mining is the process of extracting certain sequential patterns from the sequence database whose support exceeds a predefined minimal support threshold. A minimum support is defined by users because the number of sequences can be very large, and users have different interests and requirements to get the most interesting sequential patterns. By using the minimum support we can prune out those sequential patterns, consequently making the mining process more efficient. Sequential pattern can be widely used in different areas.

For example, from a sequence database, we can find the frequent sequential purchasing patterns, for example if a customer buys the computer typically the same customer will buy the pen drive within few weeks. 2. Basic Concepts

Let I = { $x_1,\ldots,x_n$ } be a set of items. A non-empty subset of items is referred an item set. The number of items in an item set is called the length of an item set. A list of item sets, $\alpha = < X_1,\ldots,X_l >$ is called sequence. An item set X_i ($1 \le i \le l$) in a sequence is called a transaction. The number of transactions in a sequence is called the length of the sequence. A sequence $\alpha = < X_1,\ldots,X_n >$ is called a subsequence of another sequence $\beta = < Y_1,\ldots,Y_m >$ where ($n \le m$), and β a super-sequence of α, if there exist integers $1 \le i_1 <\ldots< i_n \le m$ such that $X_1 Y_{i1},\ldots, X_n Y_{in}$. A sequence database is a set of 2-tuples (sid, α), where sid is a sequence-id and α is a sequence.

2.1 Sequential Pattern

Sequential Pattern is a sequence of item sets that frequently occurs in a specific order, all items in the same item sets are supposed to have the same transaction time. Consider the database D is sorted, with customer-id as major key and transaction-time as minor key. Usually all the transactions of a customer are together viewed as a sequence, usually called customer-sequence as shown in Table 1.

Table 1: Customer sequence database

Seq.id	***Sequence***
s1	<(1,2)(4)(3,5)(7)>
s2	<(1,2,3)(3)(5,7)>
s3	<(6)(5,6)(4,7)>
s4	<(4,5)(5,6)(1)(3,5)>
s5	<(4)(5)(2,3)(1,6)>

2.2 Support

A customer supports a sequence s if s is contained in their corresponding sequences in the sequence database; the support of sequence s is defined as the fraction of customers who support this sequence which is given in Eq. (1).

$$Support(s) = \frac{Number\ of\ customers\ support\ s}{Total\ number\ of\ customers} \quad (1)$$

2.3 Problem Definition

Given (i) a set of sequential records (called sequences) representing a sequence database D; (ii) a minimum support threshold called min_sup; and (iii) a set of k unique items or events I = { $i_1, i_2, \ldots, i_k$ }. The problem of mining sequential patterns is to find the set of all frequent subsequences S of items I in D which satisfies the given min_sup.

3. Related Work

The AprioriAll algorithm described by Agrawal and Srikant [1] that states that "All nonempty subsets of a frequent item set must also be frequent". The main drawback of AprioriAll is that too many passes over the database is required and too many candidates are generated.

The GSP algorithm described by Agrawal and Srikant [2] is an improvement over AprioriAll. To reduce candidates, GSP only creates a new length-k candidate when there are two frequent length-(k-1) sequences with the prefix of one equal to the suffix of the other. In AprioriAll it is easy to get the support counts of all those candidate sequences but in GSP it is difficult to get the support counts of candidate sequences.

The SPIRIT algorithm uses regular expressions as constraint [4]. It allows the users to specify the regular expression constraint on the mined patterns. This method avoids wastage of computing effort for mining patterns that users are not interested in. Although it needs a systematic method to push various constraints into the process.

The PrefixSpan(Prefix-projected Sequential pattern mining) algorithm presented by Jian Pei [7] representing the pattern-growth methodology. PrefixSpan mainly employs the method of database projection and also in PrefixSpan there is no need for candidate generation only recursively project the database according to their prefix. Although the PrefixSpan algorithm successfully discovered patterns, the cost of memory space might be high due to the creation and processing of huge number of projected sub-databases.

SPADE (Sequential Pattern Discovery using Equivalence classes) algorithm presented by M.J.Jaki [9] is proposed to find frequent sequences using efficient lattice search techniques and simple joins. However in SPADE, candidates are generated and tested on the fly to avoid storing candidates, which costs a lot to merge the id-lists of frequent sequences for a large number of candidates.

Another approach called MEMory Indexing for Sequential Pattern mining (MEMISP) was introduced [6]. MEMISP uses a recursive searching and indexing strategy to generate all the sequential patterns from the data sequences stored in memory. MEMISP is more efficient than GSP and PrefixSpan but slower when pseudo projection technique is used in PrefixSpan.

By combining SPAM, a new algorithm called LAst Position INduction Sequential PAttern Mining (abbreviated as LAPIN-SPAM) has been proposed [10]. When judging whether a sequence is a pattern or not, the previous techniques 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 cannot appear behind the current prefix in the same customer sequence. LAPIN-SPAM could largely reduce the search space during mining process. But it consumes much more memory than PrefixSpan.

Although presenting the complete set of sequential patterns may make the mining result hard to understand and hard to use. To overcome this problem Jian Pei, Jiawei Han and Wei Wang [8] have presented the pushing of various constraints deep into sequential pattern mining using pattern growth methods. Constraint-based mining may overcome the difficulties of effectiveness and efficiency since constraints usually represent user's interest, which limits the patterns to be found.

4. Existing System

An UpDown Directed Acyclic Graph (UDDAG) is a data structure used for sequential pattern mining which derive patterns based on the bidirectional pattern growth approach along both ends of detected patterns recursively [3]. At each level of recursion, the length of detected patterns grows bidirectionally along its prefix and suffix of detected patterns. To derive a pattern, a method called database projection was used. This method makes the database much smaller to further levels and consequently make the algorithm more speedy, also there is no need for candidates generation only recursively project the database according to their prefix and suffix. UDDAG derives length-(k+1) patterns based on the projected databases of length-k patterns recursively.

4.1 UDDAG

UDDAG efficiently finds and represents the relationship between patterns. It represents patterns as vertexes and relationship of patterns as directed edges. Each vertex represents a pattern with occurrence information, i.e., the sequence-ids of tuples containing the pattern. By representing the relationship of patterns from prefix with a DAG (Up DAG) and the relationship of patterns from suffix with another DAG (Down DAG), we can decrease the number of candidates by using these DAGs.

5. Proposed Work

In existing, at each level of pattern mining, a prefix and suffix projected database needs to be formed, which consumes more memory if the number of data sequences increase. Also it takes more time to build the database and to find the support count of the items.

To overcome the problem of time, a vertical bitmap representation is proposed. The basic idea is that it completely eliminates the need of database projection and finds the patterns from the bitmap representation. The design of the sequential pattern mining is shown in Fig. 1.

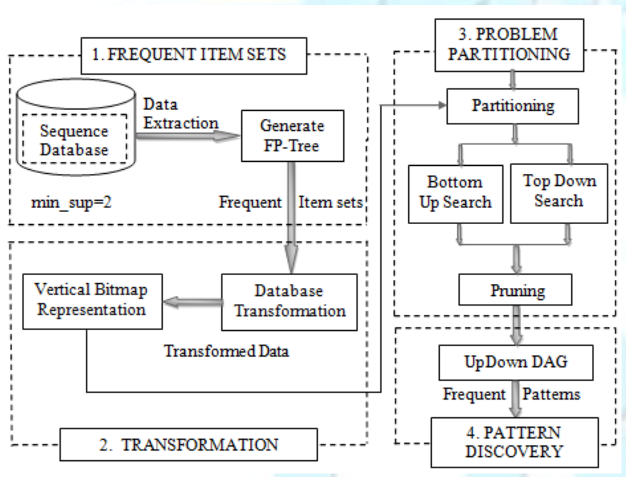


Fig. 1 Design of sequential pattern mining

Algorithm 1: Sequential Pattern Mining

```
1  Input: D, sequence database
2         min_sup, minimum support
3  Output: P , set of patterns in D
4  Procedure: pattern(D,min_sup)
5  begin
6      P = Φ
7      FISet = fpgrowth(D,min_sup)
8      D' = transform(D,FISet)
9      B = bitmap(D', id)
10     foreach FI x in FISet do
11         vertex root = new vertex(x)
12         prefixpattern(B,root,x,min_sup,tid1,tid2)
13         suffixpattern(B,root,x,min_sup,tid2,tid1)
14         UDDAG(root)
15         P =P ∪ root.getAllPatterns()
16     end
17 end
```

Algorithm 1 first finds the frequent item sets. Then transform the database and presents the bitmap representation for the transformed database. For each item set, a new vertex is created and finds the frequent item sets in the prefix and the suffix of the item set and then finds long subsequences by combining prefix and suffix item sets.

5.1 Frequent Item Sets

It refers to the item set whose number of occurrences satisfies the minimum support. To detect the frequent item sets, FP-growth algorithm [5] is used. It comprises of two steps:

1) First to construct the FP-tree that represents the data set in the form of a tree. Each transaction is read and then mapped onto a path in the FP-tree. This is done until all transactions have been read.

2) Secondly FP-Growth extracts frequent item sets from the FP-tree in a bottom-up fashion from the leaves towards the root. Using a divide and conquer approach, find the frequent item sets ending in a particular suffix.

For e.g., for the database in Table 1, the FIs are: (1), (2), (3), (4), (5), (6), (7), (1,2), (2,3), (3,5), (5,6).

5.2 Transformation

Based on frequent item sets, we transform each sequence in a database into an alternative representation. It works as follows:

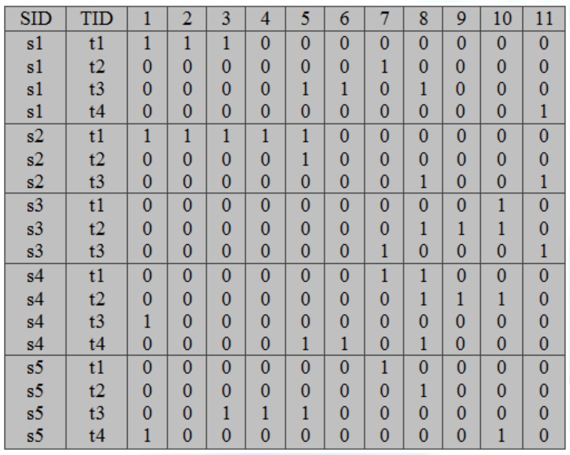
Database Transformation: First, we assign a unique id to each Frequent Item set (FI). We then replace each item set in each sequence with the ids of all the FIs contained in the item set. For e.g., (1)-1, (1,2)-2, (2)-3, (2,3)-4, (3)-5, (3,5)-6, (4)-7,(5)-8, (5,6)-9, (6)-10, (7)-11. We then replace the database as shown in Table 2.

Table 2: Transformed database

Seq.id	***Sequence***
s1	<(1,2,3)(7)(5,6,8)(11)>
s2	<(1,2,3,4,5)(5)(8,11)>
s3	<(10)(8,9,10)(7,11)>

s4	<(7,8)(8,9,10)(1)(5,6,8)>
s5	<(7)(8)(3,4,5)(1,10)>

Vertical Bitmap Representation: It stores the transformed database in vertical bitmaps, where each bitmap represents an item set in the database and a bit in each bitmap represents whether or not a given customer has the corresponding item set. If item set i appear in transaction j, then the bit corresponding to transaction j of the bitmap for item set i is set to one; otherwise, the bit is set to zero. For e.g., the vertical bitmap representation for the transformed database(shown in Table 2) is shown in Fig. 2.



SID	TID	1	2	3	4	5	6	7	8	9	10	11
s1	t1	1	1	1	0	0	0	0	0	0	0	0
s1	t2	0	0	0	0	0	0	1	0	0	0	0
s1	t3	0	0	0	0	1	1	0	1	0	0	0
s1	t4	0	0	0	0	0	0	0	0	0	0	1
s2	t1	1	1	1	1	1	0	0	0	0	0	0
s2	t2	0	0	0	0	1	0	0	0	0	0	0
s2	t3	0	0	0	0	0	0	0	1	0	0	1
s3	t1	0	0	0	0	0	0	0	0	0	1	0
s3	t2	0	0	0	0	0	0	0	1	1	1	0
s3	t3	0	0	0	0	0	0	1	0	0	0	1
s4	t1	0	0	0	0	0	0	1	1	0	0	0
s4	t2	0	0	0	0	0	0	0	1	1	1	0
s4	t3	1	0	0	0	0	0	0	0	0	0	0
s4	t4	0	0	0	0	1	1	0	1	0	0	0
s5	t1	0	0	0	0	0	0	1	0	0	0	0
s5	t2	0	0	0	0	0	0	0	1	0	0	0
s5	t3	0	0	1	1	1	0	0	0	0	0	0
s5	t4	1	0	0	0	0	0	0	0	0	1	0

Fig. 2 Bitmap representation

Algorithm 2: Patterns in the Prefix

```
1 Input: B, the bitmap representation of D'
2       root, root vertex
3       x, frequent item set
4       min_sup, minimum support
5       tid1, start transaction id
6       tid2, end transaction id
7 Output: PFISet, set of prefix frequent item set for x
8 Procedure: prefixpattern(B,root,x,min_sup,tid1,tid2)
9 begin
10  foreach sequence s in B do
11      $tid1_s$=upsearch(x,s,$tid1_s$,$tid2_s$)
12      $PISet_s$=pprune(B,x,s,$tid1_s$,$tid2_s$)
13  end
14  PFISet=count(PISet,min_sup)
15  foreach FI y in PFISet do
16      vertex ver = new vertex(y,root)
17      root.addUpChild(ver)
18      prefixpattern(B,ver,y,min_sup,tid1,tid2)
19      suffixpattern(B,ver,y,min_sup,tid2,tid1)
20      UDDAG(ver)
21  end
22 end
```

5.3 Problem Partitioning

Let { x_1,x_2,…,x_t } be the frequent item sets in a database D, where $x_1 < x_2 <\ldots< x_t$. D can be divided into t disjoint subsets. The ith subset (denoted by P_{xi} ,$1 < i < t$) is the set of patterns that contains x_i and FIs smaller than x_i.

P is the complete set of patterns in D. P_x is the set of patterns with respect to item set x. To detect P_x, we first detect patterns in prefix of x and suffix of x and combine them to derive P_x. This is a recursive process, we perform the same action until reaching the base case, where there is no frequent item set is found.
Algorithm 2 and Algorithm 3 are used to find the frequent item sets in the prefix and the suffix of each item set.

Algorithm 3: Patterns in the Suffix

```
1 Input: B, the bitmap representation of D'
2       root, root vertex
3       x, frequent item set
4       min_sup, minimum support
5       tid2, start transaction id
6       tid1, end transaction id
7 Output: SFISet, set of suffix frequent item set for x
8 Procedure: suffixpattern(B,root,x,min_sup,tid2,tid1)
9 begin
10  foreach sequence s in B do
11      $tid2_s$=downsearch(x,s,$tid2_s$,$tid1_s$)
12      $SISet_s$=sprune(B,x,s,$tid2_s$,$tid1_s$)
13  end
14  SFISet=count(SISet,min_sup)
15  foreach FI y in SFISet do
16      vertex ver=new vertex(y,root)
17      root.addDownChild(ver)
18      prefixpattern(B,ver,y,min_sup,tid1,tid2)
19      suffixpattern(B,ver,y,min_sup,tid2,tid1)
20      UDDAG(ver)
21  end
22 end
```

Searching: First we need to perform Bottom Up Search. It scan each customer transactions individually along x from the bottom to find the first existence of item set x in the transaction and return their corresponding tid. For the Fig. 2, consider item set 8, in sequence 1-first exists in the transaction 3, sequence 2-3, sequence 3-2, sequence 4-4, sequence 5-2. Similarly we need to perform Top Down Search. It scan each customer transactions individually along x from the top to find the first existence of item set x in the transaction and return their corresponding tid. For the Fig. 2, consider item set 8, in sequence 1-first exists in the transaction 3, sequence 2-3, sequence 3-2, sequence 4-1, sequence 5-2.

Pruning: Algorithm 4 describes the pruning function for prefix item set. Once searching has been completed, pruning is done for two cases:

Case 1: With the result of bottom up search, at each sequence perform bitwise OR operation for the item sets before the tid to find the item sets that the each customer bought in their purchase before x then count the support of each item set (total no.of customers who purchased the item set before x) from the output of OR operation. Those item sets satisfies the threshold (min_sup) are considered as prefix FIs. Then the FIs are projected in DAG.

Case 2: With the result of top down search, at each sequence perform bitwise OR operation for the item sets after the tid to find the item sets that the each customer bought in their purchase after x then count the support of each item set (total no.of customers who purchased the item set after x) from the output of OR operation. Those item sets satisfies the threshold (min_sup) are considered as suffix FIs. Then the FIs are projected in DAG.

For e.g., item set 8 have 1, 2, 3, 7 as prefix FIs and 1, 5 as suffix FIs.

Algorithm 4: Pruning of Prefix Frequent Item Set

```
1 Input: B, the bitmap representation of D'
2        root, root vertex
3        s, sequence
4        stid, start transaction id
5        etid, end transaction id
6 Output: ISet, set of item set
7 Procedure: pprune(B,x,s,stid,etid)
8 begin
9    while x!=0 do
10      ISet_s[x]=0
11      foreach transaction tidx from stid to etid in
        sequence s do
12         if B_{s,tidx} ==1 then
13            ISet_s[x]=1
14            break
15         end
16         else
17            continue
18         end
19      end
20      decrement x
21   end
22   return ISet_s
23 end
```

5.4 Pattern Discovery

UDDAG data structure is used represents the relationship between patterns. To represent the relationship of patterns from prefix with a DAG, Up DAG is used and the relationship of patterns from suffix with another DAG, Down DAG is used. For e.g., item set 8 have 1, 2, 3, 7 as prefix FIs which are projected using Up DAG and 1, 5 as suffix FIs which are projected using Down DAG. Fig. 3 shows the Up DAGs for the prefix of item set 8 and Fig. 4 shows the Down DAGs for the suffix of item set 8.

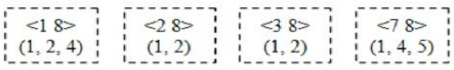


Fig. 3 Up DAGs for item set 8

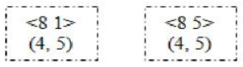


Fig. 4 Down DAGs for item set 8

After the Up and Down DAGs of x are represented then should detect P_x by evaluating each candidate vertex pair. It is done by combining each up vertex with each down vertex and finds the intersection of the occurrence sets. If the corresponding support satisfies the min sup then it is considered as a pattern.

For e.g., we detect P_8 based on the Up and Down DAGs of 8 (shown in Fig. 3 and 4). First, we detect the VDVSs for length-1 pattern in prefix of item set 8, i.e., up vertexes 1, 2, 3, and 7. For vertex 1, first, we check its combination with down vertex 1, the intersection of the occurrence sets is <4>. Thus, the corresponding support is at most 1, which is not a valid combination. Similarly, up vertex 1 and down vertex 5 are also invalid combination. Therefore, $VDVS_1 = \Phi$; $VDVS_2 = \Phi$; $VDVS_3 = \Phi$; $VDVS_7 = \{<1>,<5>\}$

Since no length-2 pattern exists in prefix, the detection stops. UDDAG for item set 8 is shown in Fig. 5.

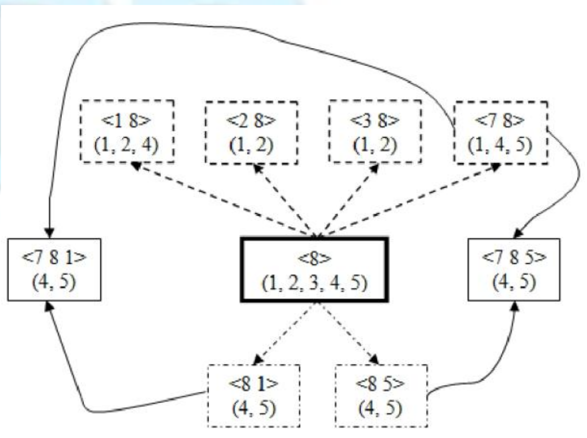


Fig. 5. UDDAG for item set 8

Hence the patterns are

P11 = { <11>,<1 11>,<2 11>,<3 11>,<5 11>,
<1 5 11>,<2 5 11>,<3 5 11>,<8 11> }

P10 = { <10>,<7 10>,<8 10> }

P9 = { <9> }

P8 = { <8>,<1 8>,<2 8>,<3 8>,<7 8>,<8 1>,
<8 5>,<7 8 1 >,<7 8 5> }

P7 = { <7>,<7 1>,<7 5>,<7 6> }

P6 = { <6>,<1 6> }

P5 = { <5>,<1 5>,<2 5>,<3 5> }

P4 = { <4> }

P3 = { <3> }

P2 = { <2> }

P1 = { <1> }

The complete set of patterns is the union of all the subsets of patterns detected above.

6. Complexity Analysis

To count the support of an item in a projected database requires scanning the entire items in a transaction in each sequence. However in a vertical bitmap representation, it is relatively having less number of scan and no need of projection.

Let C is the number of sequences, S is the average number of transactions in a sequence, T is the average number of items in a transaction and L is the average length of a sequential patterns then for database projection method, to detect a pattern with length L requires O(LCST) to scan the total items. However in a vertical bitmap representation, it requires O(LCS).

6.1 Performance Evaluation

Here we examine the performance of the algorithms based on support count calculation with two methods: Database Projection and Vertical Bitmap Representation. Figure 6 show the performance of the algorithms with different number of sequences (C) under S=5, T=3, L=4.

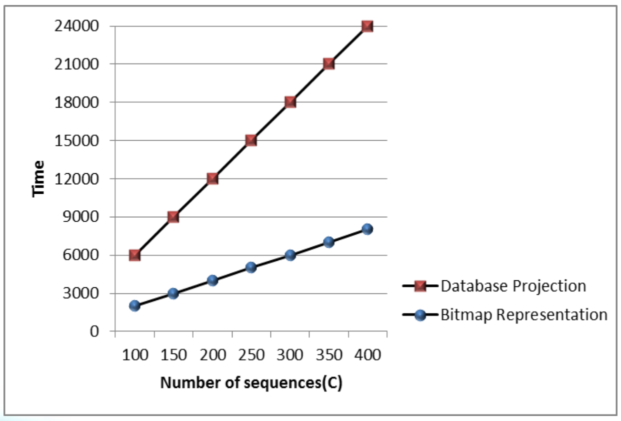


Fig. 6. Time usage on different number of sequences

7. Conclusion and Future Work

For efficient pattern mining, UDDAG data structure is used which supports bidirectional pattern growth approach. This allows faster pattern mining and efficient pruning of invalid candidates. In existing, to derive a pattern, database projection method was used. This method requires to build many intermediate databases. In order to eliminate the time taken to build the projected databases, a vertical bitmap representation is used. It efficiently stores the database as vertical bitmaps, where each bitmap represents an item set in the database. Efficiently counting the support of the item set is one of the main advantages of the vertical bitmap representation of the data and it reduces the scanning time of the item. In the future, we expect to form the strong association rules from the detected patterns.

References

[1] R. Agrawal and R. Srikant, “Mining sequential patterns”, Proc. International Conference on Data Engineering, 1995, pp. 3-14.

[2] R. Agrawal and R. Srikant, “Mining Sequential Patterns: Generalizations and Performance Improvements,” Proc. International Conference on Extending Database Technology, 1996, pp. 3-17.

[3] J. Chen, “An UpDown Directed Acyclic Graph Approach for Sequential Pattern Mining”, Proc. of IEEE Transactions on Knowledge and Data Engineering, Vol. 22, No.7, 2010, pp. 913-928.

[4] M. N. Garofalakis, R. Rastogi and K. Shim, “Spirit: Sequential pattern mining with regular expression constraints”, Proc. International Conference on Very Large Data Bases, 1999, pp. 223-234.

[5] G. Grahne and J. Zhu, “Efficiently Using Prefix-Trees in Mining Frequent Itemsets”, Proc. Workshop Frequent Itemset Mining Implementations (FIMI ’03), 2003.

[6] M. Y. Lin and S. Y. Lee, ”Fast Discovery of Sequential Patterns through Memory Indexing and Database Partitioning,” Proc. Journal of Information Science and Engineering, vol. 21, 2005, pp. 109-128.

[7] 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,” Proc. International Conference on Data Engineering, 2001, pp. 215-224.

[8] J. Pei, J. Han, W. Wang, “Constraint-based sequential pattern mining: the pattern growth methods”, Proc. Journal of intelligent information systems, Vol. 28, No.2, 2007, pp. 133 -160.

[9] M. J. Zaki, ”SPADE: An efficient algorithm for mining frequent sequences”, Proc. Machine Learning, Vol. 42, 2001, pp. 31-60.

[10] Z. Zhang and M. Kitsuregawa, ”LAPIN-SPAM: An Improved Algorithm for Mining Sequential Pattern,” Proc. International Special Workshop Databases for Next Generation Researchers, 2005, pp. 8-11.