Novel Tree based Approach for Mining Sequential Pattern in Progressive Database

S. Daniel Rajkumar
PG Scholar Dept of CSE
Velammal Engineering College
Chennai

T.K.S. Rathish Babu
Asst. Professor Dept of CSE
Velammal Engineering College
Chennai

Dr. N. Sankar Ram
Professor & Head Dept of CSE
Velammal Engineering College
Chennai

ABSTRACT
The sequential pattern mining on progressive databases is comparatively very new, in which progressively find out the sequential patterns in time of interest. Time of interest is a sliding window which is continuously move forwards as the time goes by. As the focus of sliding window changes, the new items are added to the dataset of interest and obsolete items are removed from it and become up to date. In previous pattern mining techniques sequential patterns are generated, the newly arriving patterns may not be identified as frequent sequential patterns due to the existence of old data and sequences. Progressive databases have posed new challenges because of the following innate characteristics such as it should not only add new items to the existing database but also removes the obsolete items from the database. The proposed tree based approach efficiently overcomes the inconsistencies in the existing methodologies and the execution time also prominent good for huge databases.

General Terms
Data Mining, Sequential Database, Apriori All, Incremental Database, Time of Interest, Progressive Database.

Keywords
Progressive Database, Time of Interest, PS-Tree, Fast Pisa Algorithm.

1. INTRODUCTION
Data mining is the process of extracting exciting information or patterns from large information repositories such as relational database, data warehouses, XML repository, etc. Also data mining is known as one of the core processes of Knowledge Discovery in Database (KDD). Many people take data mining as a synonym for another popular term, Knowledge Discovery in Database (KDD). Otherwise other people treat Data Mining as the core process of KDD. Commonly there are three processes. One is called preprocessing, which is executed before data mining techniques are applied to the correct data. The pre processing includes data cleaning, integration, selection and transformation. The main process of KDD is the data mining process, in this process different algorithm are applied to produce hidden knowledge. After that another process called post processing, this evaluates the mining result according to users' requirements.

First clean and integrate the databases. Ever since the data source may come from different databases, which may have some inconsistencies and duplications, clean the data source by removing those noises or make some compromises.

Suppose it have two different databases, different words are used to refer the similar thing in their schema. When incorporate the two sources only choose one of them, if they denote the same thing. And also real world data tend to be incomplete and noisy due to the manual input mistakes. The incorporated data sources can be stored in a database, data warehouse or other repositories.

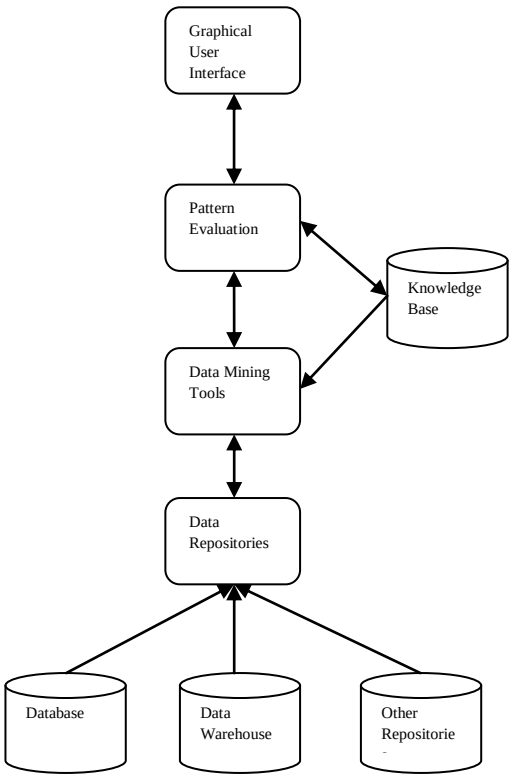

Figure 1: Knowledge Discovery Process

A variety of data mining techniques are applied to the data source, different knowledge comes out as the mining result. That knowledge is evaluated by certain rules, such as the domain knowledge or concepts. After the evaluation, as shown in Figure 1, if the result does not satisfy the requirements or contradicts with the domain knowledge, redo some processes until getting the right results. Depending on the evaluation result we may have to redo the mining or the user may modify his requirements. After the knowledge, the final step is to visualize the results. They can be displayed as raw data, tables, decision trees, rules, charts, data cubs or 3D graphics. This process is try to make the data mining results easier to be used and more understandable.

2. SEQUENTIAL PATTERN MINING
Sequential pattern is a sequence of item sets that frequently occurred in a specific order, all items in the same item sets are supposed to have the same transaction time value or within a time gap. Usually all the transactions of a customer are together viewed as a sequence, usually called customer-

sequence, where each transaction is represented as an item sets in that sequence, all the transactions are list in a certain order with regard to the transaction time.

Support is defined as follows if *s* is contained in the corresponding customer sequence; the support of sequence *s* is defined as the fraction of customers who support this sequence.

$$\text{Support}(S) = \frac{\text{Number of support customers}}{\text{Total number of customers}}$$

Sequential pattern mining is the process of extracting certain sequential patterns whose support exceed a predefined minimal support threshold. Since the number of sequences can be very large, and users have different interests and requirements, to get the most interesting sequential patterns, usually a minimum support is predefined by users. By using the minimum support we can prune out those sequential patterns of no interest, consequently make the mining process more efficient. Obviously a higher support of sequential pattern is desired for more useful and interesting sequential patterns. However some sequential patterns that do not satisfy the support threshold are still interesting. The Table 1 is the example for sequential database.

Table 1. Sequential Database

Customer id	Transaction time	Purchased items
1	Jan 03'2012	30
1	Jan 05'2012	90
2	Dec 27'2011	10,20
2	Dec 21'2011	40,50,80
3	Jan 04'2012	30,50,10

2.1 Sequential Pattern Mining in Static Database

There are many researches about mining sequential patterns in a static database. It was first addressed by Agarwal and Srikant [1]. In general sequential pattern mining algorithms can be classically categorized into three classes. (i) Apriori based horizontal partitioning methods such as Generalized Sequential Pattern mining [4], which adopts multiple-pass candidate generation and test approach in sequential pattern mining. (ii) Apriori based vertical partitioning methods such as Sequential Pattern Discovery using Equivalent classes [5], utilizes combinatorial properties to decompose the original problem into smaller sub-problems that can be independently solved in main memory using efficient lattice search and simple join operations.(iii) Projection based pattern growth algorithms such as prefix-projected sequential pattern mining algorithms [2], which represents the pattern growth methodology and finds the frequent items after scanning database once. In addition to the traditional algorithms there are many which include closed sequential pattern mining [4], maximal sequential pattern mining [5] and constraint sequential pattern mining [3].

2.2 Sequential Pattern Mining in Incremental Database

The incremental sequential pattern mining algorithms resolve major drawback of the sequential pattern mining algorithms such as mining the patterns from up-to-date database without deleting the obsolete. The key algorithms of incremental sequential pattern mining are: Parthasarathy et al. [1], developed an incremental mining algorithm ISM by maintaining a sequence lattice of an old database. Sequence lattice includes all the frequent sequences and all the sequences in the negative border. Later Masseglia et al. [6], proposed another incremental algorithm ISE for incremental mining of sequential patterns when new transactions are added to the database. This algorithm adopts candidate generation and test approach. Hang Cheng et al. [3], presented Incspan algorithm which mines sequential pattern over an incremental databases. The limitation of these algorithms is its inability to delete the obsolete data.

2.3 Sequential Pattern Mining in Progressive Database

Progressive sequential pattern mining is a generalized pattern mining methodology that brings out the most recent frequent sequential patterns. This algorithm works both static as well as dynamic changing databases and is unaffected by the presence of obsolete data. The patterns are not affected by the old data. This algorithm uses the sliding window to progressively update sequences in the database and accumulate the frequencies of candidate sequential patterns as time progresses. The sliding window called time of interest determines the time stamp over which the algorithm is currently working.

3. EXISTING SEQUENTIAL PATTERN MINING APPROACHES

Sequential pattern mining has been intensively studied during recent years, there exists a great diversity of algorithms for sequential pattern mining. In this Section first introduce some general and basic algorithms for sequential pattern mining, extensions of those algorithms for special purposes, such as multi-dimensional sequential pattern mining and incremental mining are covered later on. Also periodical pattern mining is elaborated as an extension of sequential pattern mining.

The sequential pattern mining with a static database and with an incremental database are two special cases of the progressive sequential pattern mining. In the following, we introduce the previous works on the static sequential pattern mining, the incremental sequential pattern mining, and the progressive sequential pattern mining. Previous researchers have developed various methods to find frequent sequential patterns with a static database. AprioriAll and GSP are the milestones of sequential pattern mining algorithms based on traditional association rule mining technique, Apriori. SPADE, illustrated by Zaki, systematically searched the sequence lattice spanned by the subsequence relation. Han et al. and Pei et al. brought up FreeSpan and PrefixSpan, which found sequential patterns by constructing subdatabases of the entire database. Ayres et al. then proposed SPAM to search a lexicographic sequence tree in depth-first manner and use a vertical bitmap data layout to support simple and efficient counting process. Aseervatham et al. presented bitSPADE using a lattice-based bitmap representation for sequential pattern mining. In addition, there are also several works on adding constraints to find sequential patterns, closed sequential patterns, maximal sequential pattern mining,

spatiotemporal sequential pattern mining, sequential pattern mining on specific data domain, sequential pattern mining on stream data, frequent episode mining, and path traversal pattern mining. The assumption of having a static database may not hold in many applications. The data in real world usually change on the fly. When we deal with an incremental database, it is not feasible to refine the whole sequential patterns every time when the database increases because the refining process is costly. To handle the incremental database, Parthasarathy et al. presented the algorithm ISM using a lattice framework to incrementally update the support of each sequential pattern in equivalent classes. Masseglia et al. derived the algorithm ISE to join candidate sequential patterns in original database with the newly increasing database. Cheng et al. introduced algorithm IncSpan, which utilized a special data structure named sequential pattern tree to store the projection of database. Then, the improvements of IncSpan were made in .The incremental update technique of implicit merging and efficient counting methods. Additionally, Chen et al. utilized prior knowledge of the data distribution into the mining process in algorithm MILE. However, the incremental mining algorithms can only handle the incremental parts of the database. Because of the limitation of data structures maintained in their algorithms, they can only create new candidates but cannot delete the obsolete data in a progressive database. The deletion of an item from the database results in the reconstruction of all candidate item sets, which induces incredible amount of computing.

3.1 AprioriAll

AprioriAll is based on the Apriori algorithm in association rule mining, similarly there are two sub process. The first is to generate those sequences that may be frequent, which is also called candidate sequences. Then the sequential database is scanned to check the support of each candidate to determine frequent sequential patterns according to minimal support. Since the time cost of the second process is determined by the number of passes over the database and number of candidates, most researchers mainly concern about the candidate generation process and the passes over the database.

AprioriAll was the first algorithm for sequential pattern mining, it is based on the naive approach of Apriori association rule mining. The main drawback of AprioriAll is that too many passes over the database is required and too many candidates are generated.

3.2 MEMISP

All those aforementioned sequential pattern mining algorithms either require many passes over the databases as in GSP, or generate many intermediate projected databases as in Prefix Span. Another approach called MEMory Indexing for Sequential Pattern mining (MEMISP) requires one pass over the database, at most two passes for very large database, and it avoids generation of candidates and projection of intermediate database as well. In this approach, MEMISP uses a recursive searching and indexing strategy to generate all the sequential patterns from the data sequences stored in memory. Some terms are defined in this algorithm. Given a pattern *p* and a frequent item *x* in the sequence database, *p* ' is a type-1 pattern if it can be formed by appending the item sets(*x*) as a new element to *p*, *p* ' is a type-2 pattern if it can be formed by extending the last element of *p* with item sets(*x*). The frequent item set *x* is called a *stem*, while the sequential pattern *p* is the Prefix pattern (P-pat) of *p* '. The MEMISP algorithm works as follows. The first phase is to scan the database and write it into memory to form the MDB (Memory Database). During this process the support counts of those length 1 sequences are recorded to get the frequent 1-sequences. Those frequent 1-sequences will be used as stem of type-1 pattern with respect to P-pat=< >. The second phase is to output the sequential pattern ½ formed by current P-pat and stem *x* and construct the index set *p* -idx. Index set *p* -idx is the collection of these(ptr ds, pos) pairs. A (ptr ds, *pos*) pair for each data sequence *ds* in MDB is allocated, if *ds* contains *x*, where prt ds is a pointer to *ds* and *pos* is the first occurring position of *x* in *ds*. The third phase is using index set *p* -idx and MDB to find stems with respect to P-pat= *p*. To find any sequential patterns that having current pattern *p* as its P-pat, first they find those *p* -idx pairs whose ptr ps points to the data sequences that contain *p*, those items that occur after the corresponding *pos* positions are taken as potential *stem*. The count of these items increase by one every time when they appear after the *pos* position, with the support count the *stem* can be determined frequent or not. Then we turn to the second phase. The second and third phases are executed recursively until no further *stem* can be found. During the whole process no further scan of the database is needed, the index is recursively updated. This consequently makes MEMISP more efficient than others. With more and more memory installed in the computer, most databases can be easily fitted into the main memory. However some very large databases still can not resident in the main memory. For those databases, they are partitioned into small ones that can be stored in memory and apply MEMISP to each of them to get the sequential patterns, the candidate sequential patterns of the whole database are the collection of patterns outputted from each partitions. To determine final frequent sequential patterns another scanning of the whole database is needed to check the actual support, only two passes over the whole database is needed for those large databases. Experiment results showed that MEMISP is more efficient than GSP and PrefixSpan, it also has a good linear scalability to the size of database and the number of data sequences.

3.3 SPIRIT (Sequential Pattern mIning with Regular expressIon consTraints)

Many different algorithms for sequential pattern mining in time series database have been designed and implemented, most of those algorithms aims to improve the efficiency of the ad-hoc algorithms. In reality, users are interested in different sequential patterns even for the same database, users are interested in some specific patterns rather than the whole possible sequential patterns. SPIRIT is a method of mining user-specified sequential patterns by using regular expression constraints. This method avoid wastage of computing effort for mining patterns that users are not interested in, it also avoids overwhelming users with potentially useless patterns. In this approach, user's specific requirements are stated in regular expression, denoted by C, four algorithms of SPIRIT are introduced. They proposed a regular expression and a series of operators that can read and interpret the regular expression of constraints. SPIRIT (N) is the most naive approach with regular expression constraints, it uses only the including constraints to pruning the candidate sequences, other processes is the same as in GSP. However in SPIRIT (L) and SPIRIT (V), every candidate k-sequence are checked according to the regular expression by using different operator such as legal and valid. With those operators the pruning techniques become more efficient. SPIRIT(R) is totally different in the candidate generation process, rather than join the sequences of less lengths the candidate are generated by enumerate the possible combination of path traverse of the regular expression. Detail and example of the four algorithms

are available in the original paper. The difference between those four algorithms is the extent to which the constraints are pushed into the candidate generation and pruning processes. Experiments have been conducted with synthetic and real-life data sets, the results showed that when the constraints are highly selective SPIRIT(R) outperforms the other three algorithms, while in most cases SPIRIT (V) is the overall winner over the entire range of regular expression constraints. SPIRIT (N) is the most straightforward and heuristic method since it enforces only limited constraint to the mining process.

However, the existing algorithms cannot cope with sequential pattern mining with a progressive database efficiently. To remedy this problem, in this paper an efficient algorithm Fast Pisa, this stands for Fast Progressive mIning of Sequential pAtterns, corresponding to the mining in a progressive database. Fast Pisa takes the concept of Time of interest (TOI) into consideration.

4. PROPOSED WORK

4.1 Progressive sequential pattern mining problem

Given a user-specified length of TOI and a user defined minimum support threshold, find the complete set of frequent sub sequences whose occurrence frequencies are greater than or equal to the minimum support times the number of sequences in the recent TOI of a progressive database.

S01	A	B		C	AD		B	C		BD
S02	AD	B		A						
S03	A	BC	B	C		A		C	D	
S04		D	C		A		BC			D
S05			D	B	A		C		D	
S06								A	C	
SID	t1	t2	t3	t4	t5	t6	t7	t8	t9	t10

Db1,5
Db2,6
Db3,7
Db4,8
Db5,9
Db6,10

Figure 2: Progressive Database

Figure 2 is the example for progressive database. S_{01}, S_{02} . . . S_n represents different sequence IDs. A, B, C, and D are different items in the database and t_1, t_2 . . . t_k represent timestamps.

4.2 Time of Interest

TOI is a sliding window, whose length is a user specified time interval, continuously advancing as the time goes by. The sequences having elements whose timestamps fall into this period, TOI, contribute to the |Db| for current sequential patterns. On the other hand, the sequences having only elements with timestamps older than TOI should be pruned away from the sequence database immediately and will not contribute to the |Db| thereafter.

To solve the progressive sequential pattern mining algorithm, a progressive mining algorithm Fast Pisa is proposed. Fast Pisa maintains a PS-tree to keep the information of the progressive database and up-to-date sequential patterns in each TOI.

4.3 PS-Tree

The algorithm Fast Pisa is PS-tree. PS-tree not only contains the information of all sequences in a progressive database but also helps Fast Pisa to produce frequent sequential patterns in each TOI. The nodes in PS-tree can be separated into two different types. They are root node and common nodes. Root node is the root of PS-tree containing nothing but a list of common nodes as its children. Each common node stores two information, say node label and a sequence list. The label is the same as the element in a sequence. The sequence list stores a list of sequence IDs to represent the sequences containing this element. Each sequence ID in the sequence list is marked by a corresponding timestamp. It is worth to mention that only the nodes in the first and the second levels have to preserve the corresponding timestamps for the sequence IDs. The timestamps for the sequence IDs in the nodes below the third level are the same as the timestamps for the same sequence IDs in the second level. Therefore, it is not necessary to store duplicate information. Whenever there are a series of elements appearing in the same sequence, there will be a series of nodes labeled by each element, respectively, with the same sequence IDs in their sequence lists. Then, the first node will be connected to the root node and the second node representing the following element will be connected to the first node. The other nodes will be connected analogously. Note that, in such a way, the path from root node to any other node will represent the candidate sequential pattern appearing in this sequence. The appearing timestamp for each candidate sequential pattern will be marked in the node labeled by the last element. If there is another sequence having the same pattern, the sequence ID will be inserted into the sequence lists of the same nodes labeled by these elements on the path. On the other hand, if an element appearing in a sequence is obsolete, the corresponding sequence ID will be removed from the sequence list of the node. In addition, if a node has no sequence in the sequence list, it will be pruned away from PS-tree.

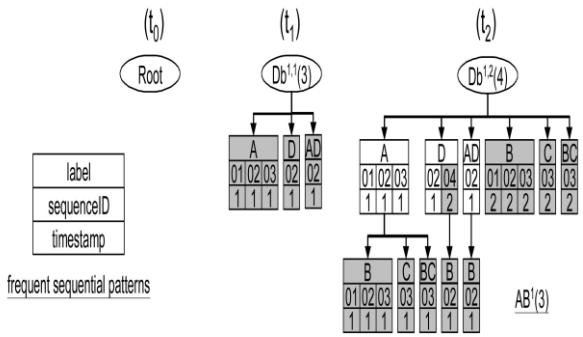

Figure 3: PS-tree of the example database (t_0-t_2)

Figure 3 gives the sample structure for PS-tree and it explains what would happen in the timestamps t_0, t_1, t_2.

4.4 Fast Pisa Algorithm

The main concept of Fast Pisa is to progressively update the information of each sequence and each candidate sequential pattern in the database. To achieve this goal, Fast Pisa utilizes PS-tree to store all sequences from one TOI to another. When receiving the elements at the arriving timestamp, say t +1, Fast Pisa traverses the original PS-tree of timestamp t in post order (children first, then the node itself) and updates the PS-tree of timestamp t. While traversing PS-tree, Fast Pisa 1) deletes the obsolete elements from, 2) updates current

sequences in, and 3) inserts newly arriving elements into the PS-tree of timestamp t. From line 4 to line 7, Fast Pisa gets the elements of all sequences at current timestamp and traverses the PS-tree. Then, Fast Pisa moves forward to the next timestamp until there is no newly arriving element in a progressive database. The main procedure, traverse, is used to traverse the PS-tree of timestamp t and transform it to the new PS-tree of timestamp t+1.

Algorithm Fast Pisa(Support,TOI)

{

1. Var PS;// PS-Tree
2. Var curtime;// timestamp now
3. Var eleSet;//used to store elements ele
4. while(there is still new transaction)
5. elset=read all ele at curtime;
6. navigate(currentTime,PS);
7. curtime++;
8. End

}

In this way, Fast Pisa can easily generate all frequent sequential patterns of timestamp t+1. After the traverse, the original PS-tree of timestamp t becomes a new PS-tree of timestamp t+1 and all information needed for the following timestamps is updated. It is noted that Fast Pisa can simply view repeated items as different items so that Fast Pisa is able to output sequential patterns containing repeated elements. In addition, since all sequential patterns are stored in PS-tree, Fast Pisa is capable of reporting sequential patterns with only a single element or maximal sequential patterns. To output sequential patterns with only a single element, Fast Pisa needs only to traverse first level nodes connected to root node. To output maximal sequential patterns, Fast Pisa reports those paths which are from root node to leaf nodes.

5. EXPERIMENTAL DESIGN

The previous works about incremental sequential pattern mining append the newly arriving elements of all sequences directly to the end of the original sequence database. They, in essence, do not concern themselves with the TOI, but instead, take the whole database of all elements into consideration. In our work, the obsolete elements which exceed the TOI will be deleted from the sequence database. For this reason, each element should be designated an arriving timestamp. Note that each timestamp can be viewed as an arbitrarily small time interval in real world such as an hour, a day, a month, and so on. Then, the items in this interval are combined as an element at a timestamp. We, thus, transform the format of the generated data sets. First, we divide the target data set into n timestamps. According to the input parameter TOI, the first m timestamps (m=TOI and m < n) are viewed as the original database and the rest of elements in the data set are received by the system incrementally. The length of TOI is inevitably smaller than n, and the overall timestamps must be longer than the maximal number of elements that one sequence produces. The first run of the experiments mines the first TOI from the beginning m timestamps of the data set (m=TOI). After that, we shift the TOI one timestamp forward for the following runs. In this way, the elements in the up-to-date timestamp stand for the incremental part of the sequence database, and the obsolete elements are deleted.

6. EXPECTED RESULTS

The execution time of Fast Pisa is about 10 to 100 times of original Pisa over different data sets. This is because the number of nodes stored in PS-tree can be reduced significantly with the slight modification of the procedure traverse of Fast Pisa. Therefore, Fast Pisa consumes less memory than Pisa by 20 to 250 times. Calculate the total number of sequential patterns generated by Pisa and Fast Pisa in every timestamp. Then, the information loss rate is defined by the {[1- (Number of patterns by Fast Pisa / Number of patterns by Pisa)]*100}percent. The information loss rate is less than 12 percent no matter which support value is chosen. In real data sets, there are many candidate sequential patterns containing elements with more than one item. However, these candidate sequential patterns are seldom recognized as frequent sequential patterns, because there are few sequences having exactly the same elements with more than one items. Therefore, the information loss is marginal.

7. CONCLUSION

The proposed system that have to improve the time efficiency factor better than the existing approach. Using that Fast Pisa approach the efficiency is much improved compared to the existing methodologies. Thus with this idea, an enhanced sequential pattern technique is proposed in this paper. In general the proposed approach Fast Pisa reduces the processing time. Thus the time efficiency is improved much effectively. In future the work proposed in this thesis would be extended by jointly memory utilization. Fast Pisa is more advantageous when the minimum support threshold is small. Furthermore, Fast Pisa possesses good scalability and outperforms competitive algorithms significantly on real data sets.

8. ACKNOWLEDGMENTS

Thanks to all Faculty members of Computer Science and Engineering Department for their support to complete this work.

9. REFERENCES

[1] J. Han and M. Kamber, Data Mining: Concepts and Techniques.Morgan Kaufmann, 2000.

[2] F. Masseglia, P. Poncelet, and M. Teisseire, “Incremental Mining of Sequential Patterns in Large Databases,” Data and Knowledge Eng., vol. 46, pp. 97-121, July 2003.

[3] Jen-Wei Huang, Chi-Yao Tseng, Jian Chih Ou, and Ming Syan Chen, ”A General Model for Sequential Pattern Mining with a Progressive Database” March 2011

[4] A.Mhatre,M.Verma,D.Toshniwal”Extracting Sequential Patterns from Progressive Databases: A Weighted Approach” 2009

[5] Jiawei Han, Jian Pei. PrefixSpan: Mining Sequential Patterns Efficiently by Prefix-Projected Pattern Growth[J]. IEEE Transactions on Knowledge and Data Engineering, 2004.1~17.

[6] J.Han, J.Pei, Y.Yin and R.Mao: Mining frequent patterns without candidate generation: A Frequent pattern tree approach. Data Mining and knowledge Discovery 8 (2004), 53~87.

[7] M. Zhang, B. Kao, D.W.-L. Cheung, and C.L. Yip, “Efficient Algorithms for Incremental Update of Frequent Sequences,” Proc. Sixth Pacific-Asia Conf. Knowledge Discovery and Data Mining, 2002.

[8] C. Romero, S. Ventura, J.A. Delgado, and P.D. Bra, "Personalized Links Recommendation Based on Data Mining. In Adaptive Educational Hypermedia Systems," Proc. Second European Conf.Technology Enhanced Learning (EC-TEL '07), Sept. 2007.

[9] S. Nguyen, X. Sun, and M. Orlowska, "Improvements of INCSPAN: Incremental Mining of Sequential Patterns in Large Database," Proc. Ninth Pacific-Asia Conf. Knowledge Discovery and Data Mining (PAKDD), 2005.

[10] J.Z. Ouh, P. Wu, and M.-S. Chen, "Constrained Based Sequential Pattern Mining," Proc. Int'l Workshop Web Technology, Dec. 2001.

[11] A.B. Pandey, J. Srivastava, and S. Shekhar, "Web Proxy Server with Intelligent Prefetcher for Dynamic Pages Using Association Rules," Technical Report 01-004, Univ. of Minnesota, Jan. 2001.

[12] A.B. Pandey, R.R. Vatsavai, X. Ma, J. Srivastava, and S. Shekhar, "Data Mining for Intelligent Web Prefetching," Proc. Workshop Mining Data Across Multiple Customer Touchpoints for CRM (MDCRM '02), May 2002.

[13] S. Parthasarathy, M.J. Zaki, M. Ogihara, and S. Dwarkadas, "Incremental and Interactive Sequence Mining," Proc. Eighth ACM Int'1999.