A Comprehensive Survey on Sequential Pattern Mining

Irfan Khan[1]

Department of computer Application,
S.A.T.I. Vidisha, (M.P.), India

Anoop Jain[2]

Department of computer Application,
S.A.T.I. Vidisha, (M.P.), India

Abstract

Mining sequential patterns has been a focused theme in data mining research for over a decade. One of the promising approaches of SPM is mainly deal with finding the behaviour of a sequential pattern that can help in many analyzing applications like predicting next event. There are several efficient algorithms that cope with the computationally expensive task of sequential pattern mining. One of them is Generalized Sequential Pattern (GSP) mining algorithm which is an Apriori-based algorithm used for sequential pattern mining. The GSP algorithm has several deficiencies whenever the database size is large, too many scanning of database when seeking frequent sequences and very large amount of candidate sequences generated unnecessary. PrefixSpan which is pattern growth method outperforms GSP and solves all above problems. Its main idea is to examine only the prefix subsequences with minimum support. This paper investigates these algorithms by classifying study of SPM algorithms.

Keywords: - Sequential pattern mining, Apriori based Algorithms, Pattern growth based algorithms.

1. Introduction

SEQUENTIAL pattern mining [2], which discovers frequent subsequences as patterns in a sequence database, is an important data mining problem with broad applications, including the analysis of customer purchase patterns or Web access patterns, the analysis of sequencing or time related processes such as scientific experiments, natural disasters, and disease treatments, the analysis of DNA sequences, etc.

The sequential pattern mining problem was first introduced by Agrawal and Srikant in [2] Given a set of sequences, where each sequence consists of a list of elements and each element consists of a set of items, and given a user-specified min_support threshold, sequential pattern mining is to find all frequent subsequences, i.e., the subsequences whose occurrence frequency in the set of sequences is no less than min_support. Many previous studies contributed to the efficient mining of sequential patterns or other frequent patterns in time-related data. Srikant and Agrawal [3] generalized their definition of Sequential patterns in [2] to include time constraints, sliding time window, and user-defined taxonomy, and presented apriori-based improved algorithm GSP (i.e., generalized sequential patterns).

Almost all of the above proposed methods for mining sequential patterns and other time-related frequent patterns are apriori-like, i.e., based on the apriori principle, which states the fact that any super-pattern of an infrequent pattern cannot be frequent, and based on a candidate generation-and test paradigm proposed in association mining [1].

A typical apriori-like sequential pattern mining method, such as GSP [3], adopts a multiple-pass, candidate generation-and-test approach outlined as follows: The first scan finds all of the frequent items that form the set of single item frequent sequences. Each subsequent pass starts with a seed set of sequential patterns, which is the set of sequential patterns found in the previous pass. This seed set is used to generate new potential patterns, called candidate sequences, based on the apriori principle. Each candidate sequence contains one more item than a seed sequential pattern, where each element in the pattern may contain one item or multiple items. The number of items in a sequence is called the length of the sequence. So, all the candidate sequences in a pass will have the same length. The scan of the database in one pass finds the support for each candidate sequence. All the candidates with support no less than min_support in the database form the set of the newly found sequential patterns. This set is then used as the seed set for the next pass. The algorithm terminates when no new sequential pattern is found in a pass, or when no candidate sequence can be generated.

Second approach adopts a divide-and-conquers pattern-growth principle as follows: Sequence databases are recursively projected into a set of smaller projected databases based on the current sequential pattern, and sequential patterns are grown in each projected databases by exploring only locally frequent fragments. Based on this philosophy, we first proposed a straightforward pattern growth method, FreeSpan (Frequent pattern-projected Sequential pattern mining) [8], which reduce the efforts of candidate subsequence generation. In this paper, we introduce another and more efficient method, called PrefixSpan [10] (Prefix-projected Sequential pattern mining), which offers ordered growth and reduced

projected databases. To further improve the performance, a pseudoprojection technique is developed in PrefixSpan. A comprehensive performance study shows that PrefixSpan, in most cases, outperforms the apriori-based algorithm GSP, FreeSpan, and SPADE [5] (a sequential pattern mining algorithm that adopts vertical data format) and PrefixSpan integrated with pseudoprojection, is the fastest among all the tested algorithms. Furthermore, our experiments show that PrefixSpan consumes a much smaller memory space in comparison with GSP and SPADE. The Sequential pattern mining can be further extended to mining multidimensional sequential patterns; time-interval sequential pattern, closed sequential pattern and constraint based sequential patterns.

2. Mining Sequential Pattern by Apriori Based Algorithms

The Apriori [1] [Agrawal and Srikant 1994] and AprioriAll [Agrawal and Srikant 1995] set the basis for a breed of algorithms that depend largely on the apriori property and use the Apriori-generate join procedure to generate candidate sequences. The apriori property states that ―All nonempty subsets of a frequent itemset must also be frequent. It is also described as antimonotonic.

Key features of Apriori-based algorithm are:

I. Breadth-first search: Apriori-based algorithms are described as breath-first (level-wise) search algorithms because they construct all the k-sequences, in kth iteration of the algorithm, as they traverse the search space.

II. Generate-and-test: This feature is used by the very early algorithms in sequential pattern mining. Algorithms that depend on this feature only display an inefficient pruning method and generate an explosive number of candidate sequences and then test each one by one for satisfying some user specified constraints, consuming a lot of memory in the early stages of mining.

III. Multiple scans of the database: This feature entails scanning the original database to ascertain whether a long list of generated candidate sequences is frequent or not. It is a very undesirable characteristic of most apriori-based algorithms and requires a lot of processing time and I/O cost.

Classification of Apriori based mining algorithm

2.1 GSP: The GSP (Generalized Sequential Pattern) algorithm described by Agrawal and Shrikant [3] makes multiple passes over the data. This algorithm is not a main-memory algorithm. If the candidates do not fit in memory, the algorithm generates only as many candidates as will fit in memory and the data is scanned to count the support of these candidates. Frequent sequences resulting from these candidates are written to disk, while those candidates without minimum support are deleted. This procedure is repeated until all the candidates have been counted. As shown in Fig 1, first GSP algorithm finds all Frequent Sequence and orders them with respect to their support ignoring ones for which support < min_sup. Then for each level (i.e., sequences of length-k), the algorithm scans database to collect support count for each candidate sequence and generates candidate length (k+1) sequences from length-k frequent sequences using Apriori. This is repeated until no frequent sequence or no candidate can be found.

Frequent Sequence

Itemsets
< (1) (2) (3) >
< (1) (2) (5) >
< (1) (5) (3) >
< (2) (3) (4) >
<(2) (5) (3) >
< (3) (4) (5) >
< (5) (3) (4) >

Candidate Generation

Length-1 Itemsets
< (1) (2) (3) (4)>
< (1) (2) (5) (3)>
< (1) (5) (3 4) >
< (2) (3) (4) (5) >
< (2 5) (3 4) >

Candidate Pruning

Itemset
< (1) (2 5)(3)>

Fig. 1 Candidate generation and Candidate pruning in GSP

2.2. SPIRIT: The Novel idea of the SPIRIT algorithm is to use regular expressions as flexible constraint specification tool [4]. It involves a generic user-specified regular expression constraint on the mined patterns, thus enabling considerably versatile and powerful restrictions. In order to push the constraining inside the mining process, in practice the algorithm uses an appropriately relaxed, that is less restrictive, version of the constraint. There exist several versions of the algorithm, differing in the degree to which the constraints are enforced to prune the search space of pattern during computation. Choice of regular expressions (REs) as a constraint specification tool is motivated by two important factors. First, REs provide a simple, natural syntax for the succinct specification of families of sequential patterns. Second, REs possess sufficient expressive power for specifying a wide range of interesting, non-trivial pattern constraints.

2.3 SPADE: M. Zaki [5] introduce SPADE algorithm to divide the candidate sequences into groups by items such that each group can be completely stored in the main memory. In addition, this algorithm uses the *ID-List* technique to reduce the costs for computing support counts. An ID-list of a sequence keeps a list of pairs, which indicate the positions that it appears in the database. In a pair, the first value stands for a customer sequence and the second refers to a transaction in it, which contains the last itemset of the sequence. For the example database in Table:1 , the ID-list of sequence <(a, g)(b)> is <(1,2), (1,6), (4,3), (4,4)>, where the pair (1,2) means that this sequence appears in the first customer sequence and ends in the second transaction. Note that a sequence may appear more than once in the same customer sequence, and therefore more than one pair will be recorded

Table: 1 Example of SPADE

CID	*Customer Sequence*
1	(a,e,g) (b) (h) (f) (c) (b,f)
2	(b) (d,f) (e)
3	(b,f,g)
4	(f) (a,g) (b,f,h) (b,f)

This approach 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. Consider the same database in Table 1 .To compute the support count of sequence < (a, g) (h) (f)>, the SPADE algorithm merges the two ID-lists of sequences <(a, g)(h)> and <(a, g)(f)>, which are <(1,3), (4,3)> and<(1,4), (1,6), (4,3), (4,4)> respectively. As a result, the ID-list of sequence < (a, g) (h) (f)> is < (1, 4), (1, 6), (4, 4)>, indicating that this sequence appears in the first and the fourth customer sequences and therefore has a support count of 2. The SPADE algorithm costs a lot to repeatedly merge the ID-lists of frequent sequences for a large number of candidate sequences.

2.4 SPAM: To reduce cost of merging, Ayres et al. [6] adopt the lattice concept in the SPAM (Sequential Pattern Mining) algorithm but represent each ID-list as a vertical bitmap The SPAM uses a vertical bitmap data structure representation of the database as shown in table 2 and 3, which is similar to the ID-list in SPADE., which can be completely stored in the main memory. With the size of current main memories reaching gigabytes and growing many moderate-sized to large databases will soon become completely memory resident.

Table: 2 Dataset sorted by CID and TID

CID	*TID*	*Itemsets*
1	1	{a,b,d}
1	3	{b,c,d}
1	6	{b,c,d}
2	2	{b}
2	4	{a,b,c}

Table: 3 Sequence for each customer

CID	*Sequence*
1	({a,b,d} {b,c,d} {b,c,d})
2	({b } {a,b,c})

A database *D* is a set of tuples *(CID, TID, X)*, where *CID* is a customer-id, *TID* is a transaction-id based on the transaction time, and *X* is an itemset such that *X Subset of I.* each tuples in D referred to as a transaction. All the transactions with the same *cid* can be viewed as a sequence of itemsets ordered by increasing *TID*.
Table 2 shows the dataset consisting of tuples of (customer id, transaction id, itemset) for the transaction. It is sorted by customer id and then transaction id. Table 3 shows the database in its sequence representation. Consider the sequence of customer 2, the size of this sequence is 2, and the length of this sequence is 4.

3. Mining Sequential Pattern by Pattern Growth Based Algorithms

Soon after the apriori-based methods of the mid-1990s, the pattern growth-method [7] emerged in the early 2000s, as a solution to the problem of generate-and-test. The key idea is to avoid the candidate generation step altogether, and to focus the search on a restricted portion of the initial database. The search space partitioning feature plays an important role in pattern-growth. Almost every pattern-growth algorithm starts by building a representation of the database to be mined, then proposes a way to partition the search space, and generates as few candidate sequences as possible by growing on the already mined frequent sequences, and applying the apriori property as the search space is being traversed recursively looking for frequent sequences. The early algorithms started by using *projected databases*, for example, *FreeSpan* [Han et al. 2000], *PrefixSpan* [Pei et al. 2001], with the latter being the most influential.

Key features of pattern growth-based algorithm are:

I Search space partitioning: It allows partitioning of the generated search space of large candidate sequences for efficient memory management. There are different ways to partition the search space. Once the search space is partitioned, smaller partitions can be mined in parallel. Advanced techniques for search space partitioning include projected databases and conditional search, referred to as split-and-project techniques.

II Tree projection: Tree projection usually accompanies pattern-growth algorithms. Here, algorithms implement a physical tree data structure representation of the search space, which is then traversed breadth-first or depth-first in search of frequent sequences, and pruning is based on the apriori property.

III Depth-first traversal: That depth-first search of the search space makes a big difference in performance, and also helps in the early pruning of candidate sequences as well as mining of closed sequences [Wang and Han 2004]. The main reason for this performance is the fact that depth-first traversal utilizes far less memory, more directed search space, and thus less candidate sequence generation than breadth-first or post-order which are used by some early algorithms.

IV Candidate sequence pruning: Pattern-growth algorithms try to utilize a data structure that allows them to prune candidate sequences early in the mining process. This result in early display of smaller search space and maintain a more directed and narrower search procedure

Classification of Prefix Growth based mining algorithm:

3.1. FREESPAN: - FreeSpan [8] was developed to substantially reduce the expensive candidate generation and testing of Apriori, while maintaining its basic heuristic. In general, FreeSpan uses frequent items to recursively project the sequence database into projected databases while growing subsequence fragments in each projected database. Each projection partitions the database and confines further testing to progressively smaller and more manageable units. The trade-off is a considerable amount of sequence duplication as the same sequence could appear in more than one projected database. However, the size of each projected database usually (but not necessarily) decreases rapidly with recursion.

For a sequence $\alpha = (s1 \ldots\ldots sl)$ the itemset $s1U \ldots. Usl$ is called a's projected itemset. FreeSpan is based on the following property: If an itemset X is infrequent, any sequence whose projected itemset is a superset of X cannot be a sequential pattern. FreeSpan mines sequential patterns by partitioning the search space and projecting the sequence sub databases recursively based on the projected itemsets.

Let $f\_list = (x1 \ldots\ldots xn)$ be a list of all frequent items in sequence database S. Then, the complete set of sequential patterns in S can be divided into n disjoint subsets: 1) the set of sequential patterns containing only item x1, 2) those containing item $x2$ but no item in $\{x3; \ldots ; xn\}$ and so on. In general, the ith subset $(1 \leq i \leq n)$ is the set of sequential patterns containing item xi but no item in $\{x_{i+1}; \ldots ; xn\}$. Then, the database projection can be performed as follows: At the time of deriving p's projected database from DB, the set of frequent items X of DB is already known.

Only those items in X will need to be projected into p's projected database. This effectively discards irrelevant information and keeps the size of the projected database minimal. By recursively doing so, one can mine the projected databases and generate the complete set of sequential patterns in the given partition without duplication. The details are illustrated in the following example:

Table: 4 Example (Freespan):-

Sequence_id	*Sequence*
1	<a (abc) (ac) d (cf) >
2	< (ad) c (bc) (ae) >
3	< (ef) (ab) (df) cb >
4	< eg (af) cbc >

Let the sequence database be S given in Table 4 and min_support = 2 the set of items in the database is $\{a; b; c; d; e; f; g\}$. FreeSpan first scans S, collects the support for each item, and finds the set of frequent items. Frequent items are listed in support descending order (in the form of "item : support"), that is, $f\_list = a : 4; b : 4; c : 4; d : 3; e : 3; f : 3$. They form six length-one sequential patterns: *<a>:4; <b>:4; <c>:4; <d>:3; <e>:3; <f>:3:*

According to the *f_list*, the complete set of sequential patterns in S can be divided into six disjoint subsets:

1. The ones containing only item *a.*
2. The ones containing item *b* but no item after *b* in f list.
3. The ones containing item *c* but no item after *c* in f list, and so on, and, finally.
4. The ones containing item f.

The sequential patterns related to the six partitioned subsets can be mined by constructing six projected databases (obtained by one additional scan of the original database).

3.2. WAP-MINE: - It is a pattern growth and tree structure-mining technique with its WAP-tree structure. Here the sequence database is scanned only twice to build the WAP tree from frequent sequences along with their support; a ―*header table* is maintained to point at the first occurrence for each item in a frequent itemset, which is later tracked in a threaded way to mine the tree for frequent sequences, building on the suffix. The WAP-mine [9] algorithm is reported to have better scalability than GSP and to outperform it by a margin. Although it scans the database only twice and can avoid the problem of generating explosive candidates as in apriori-based methods, WAP-mine suffers from a memory consumption problem, as it recursively reconstructs numerous intermediate WAP-trees during mining, and in particular, as the number of mined frequent patterns increases. This problem was solved by the PLWAP algorithm [Lu and Ezeife 2003], which builds on the prefix using position-coded nodes.

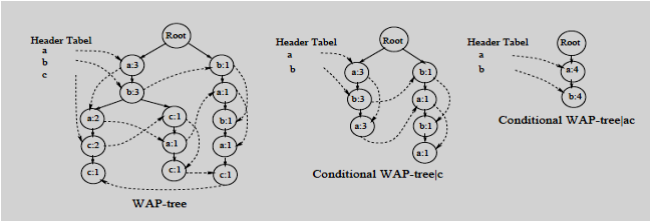

Fig. 2: Classification of Prefix Growth based mining algorithm

3.3. PREFIXSPAN: - Based on the analysis of the FreeSpan algorithm, one can see that one may still have to pay high cost at handling projected databases. Is it possible to reduce the size of projected database and the cost of checking at every possible position of a potential candidate sequence. To avoid checking every possible combination of a potential candidate sequence, one can first fix the order of items within each element. Since items within an element of a sequence can be listed in any order, without loss of generality, one can assume that they are always listed alphabetically.

Pei et al. [10] employ the projection scheme in the PrefixSpan algorithm to project the customer sequences into overlapping groups called *projected databases* such that all the customer sequences in each group have the same prefix which corresponds to a frequent sequence. For the example database in Table 5, assuming that the minimum support count is two, the PrefixSpan algorithm

first scans the database to find the frequent 1-sequences, i.e. <(a)>, <(b)>, <(e)>, <(f)>, <(g)>, and <(h)>. After that, this algorithm generates the projected database for each frequent 1-sequence. For instance, Table 5 shows the projected database of < (a)>. For this projected database, the PrefixSpan algorithm continues the discovery of frequent 1-sequences to form the frequent 2-sequences with prefix < (a)>. In this way, the PrefixSpan algorithm recursively generates the projected database for each frequent k-sequence to find frequent (k+1)-sequences. Obviously, the PrefixSpan algorithm costs a lot to recursively generate a large number of projected databases.

Table: 5 Customer Database

CID	*Customer Sequence*
1	(a,e,g) (b) (h) (f)(c)(b,f)
2	(b)(d,f)(e)
3	(b,f,g)
4	(f)(a,g)(b,f,h)(b,f)

Table: 6 Projected databases of <a>

CID	*Customer Sequences*
1	(_, e, g)(b)(h)(f)(c)(b, f)
4	(_, g)(b, f, h)(b, f)

4. Extension of Sequential Pattern Mining

Sequential pattern mining has been intensively studied during recent years; there exists a great diversity of algorithms for sequential pattern mining. Along with that Motivated by the potential applications for the sequential patterns, numerous extensions of the initial definition have been proposed which may be related to other types of time-related patterns or to the addition of time constraints. Some extensions of those algorithms for special purposes such as multidimensional, closed, time interval, and constraint based sequential pattern mining are discussed in following section.

I. Multidimensional Sequential Pattern Mining:

Mining sequential patterns with single dimension means that we only consider one attribute along with time stamps in pattern discovery process, while mining sequential patterns with multiple dimensions we can consider multiple attributes at the same time. In contrast to sequential pattern mining in single dimension, mining multiple dimensional sequential patterns introduced by Helen Pinto and Jiawei Han [11] can give us more informative and useful patterns. For example we may get a traditional sequential pattern from the supermarket database that after buying product *a* most people also buy product *b* in a defined time interval. However, using multiple dimensional sequential pattern mining we can further find different groups of people have different purchase patterns.

For example, M.E. students always buy product *b* after they buy product *a*, while this sequential rule weakens for other groups of students. Hence, we can see that multiple-dimensional sequential pattern mining can provide more accurate information for further decision support.

II. Discovering Time-interval Sequential Pattern:

Although sequential patterns can tell us what items are frequently bought together and in what order, they cannot provide information about the time span between items for further decision support. In other words, although we know which items will be bought after the preceding items, we have no idea when the next purchase will happen. Y. L. Chen, M. C. Chiang, and M. T. Kao [12] have given the solution of this problem that is to generalize the mining problem into discovering time-interval sequential patterns, which tells not only the order of items but also the time intervals between successive items. An example of time-interval sequential pattern is (*a*, *I1*, *b*, *I2*, *c*), meaning that we buy item *a* first, then after an interval of *I1* we buy item *b*, and finally after an interval of *I2* we buy item *c*. Similar type of work done by C. Antunes, A. L. Oliveira, [10] by presenting the concept of gap constraint. A gap constraint imposes a limit on the separation of two consecutive elements of an identified sequence. This type of constraints is critical for the applicability of these methods to a number of problems, especially those with long sequence.

III. Closed Sequential Pattern Mining:

The sequential pattern mining algorithms developed so far have good performance in databases consisting of short frequent sequences. Unfortunately, when mining long frequent sequences, or when using very low support thresholds, the performance of such algorithms often degrades dramatically. This is not surprising: Assume the database contains only one long frequent sequence < (*a*1) (*a*2) . . . (*a*100) >, it will generate $2^{100} - 1$ frequent subsequence if the minimum support is 1, although all of them except the longest one are redundant because they have the same support as that of < (*a*1) (*a*2) . . . (*a*100) > . So proposed an alternative but equally powerful solution instead of mining the complete set of frequent subsequence, we mine frequent *closed subsequence* only, i.e., those containing no super-sequence with the same support. This mining technique will generate a significant less number of discovered sequences than the traditional methods while preserving the same expressive power since the whole set of frequent subsequences together with their supports, can be derived easily from the mining results [13].

IV. Discovering Constraint Based Sequential Pattern:

Although efficiency of mining the complete set of sequential patterns has been improved substantially, in

many cases sequential pattern mining still faces tough challenges in both effectiveness and efficiency. On the one hand, there could be a large number of sequential patterns in a large database. A user is often interested in only a small subset of such patterns. 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 [15] have systematically presented the problem of pushing various constraints deep into sequential pattern mining using pattern growth methods.

5. Future Direction and Research Challenges:

Future Research in this area will be focus on improving the efficiency of the algorithms either with new structures, new representations or by managing the database in the main memory. So based on these criteria's sequential pattern mining is classified into two major groups, Apriori Based and Pattern Growth based algorithms. So, from the previous studies and comparative analysis of various mining algorithms, it is clear that PrefixSpan Algorithm more efficient with respect to running time, space utilization and scalability and it could be more efficient if we use DISC (Direct Sequent Comparison) Strategy [14] with PrefixSpan Algorithm in the pruning step its say we can remove nonfrequent sequences according to the other sequences with the same length. But still there are various research challenges in this field of data mining. Some of the research challenges are: –

- To find the complete set of patterns, when possible, satisfying the minimum support (Frequency) threshold.
- Algorithm should handle large search space.
- Algorithm should avoid repeated scanning of database during mining process.
- To use some method by which early candidate sequence pruning and search space partitioning will be possible for efficient mining of patterns.
- For large sequence database there can be a possibility of having distributed sequential pattern mining to provide scalability.

5. CONCLUSION: - From the study of various sequential pattern mining algorithms, we can say that PrefixSpan [9] is an efficient pattern growth method because it outperforms GSP [3], FreeSpan [7] and SPADE [5]. It explores prefix-projection which reduces the size of projected database and leads to efficient processing in sequential pattern mining. Also Bi-level projection and pseudo-projection may improve mining efficiency. It is clear that PrefixSpan Algorithm is more efficient with respect to running time, space utilization and scalability then Apriori based algorithms and FreeSpan algorithm, and PrefixSpan consumes a much smaller memory space in comparison with GSP and SPADE.

References:-

[1] R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules," Proc. 1994 Int'l Conf. Very Large Data Bases (VLDB '94), pp. 487-499, Sept. 1994.
[2] R. Agrawal and R. Srikant, "Mining Sequential Patterns," Proc.1995 Int'l Conf. Data Eng. (ICDE '95), pp. 3-14, Mar. 1995.
[3] Srikant R. and Agrawal R., ―Mining sequential patterns: Generalizations and performance improvements, Proceedings of the 5th International Conference Extending Database Technology, 1996, 1057, 3-17.
[4] M. Garofalakis, R. Rastogi, and K. Shim, "SPIRIT: Sequential pattern mining with regular expression constraints", VLDB'99, 1999.
[5] M. Zaki, "SPADE: An efficient algorithm for mining frequent sequences, Machine Learning, 2001.
[6] AYRES, J., FLANNICK, J., GEHRKE, J., AND YIU, T., ―Sequential pattern mining using a bitmap representation, In Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining-2002.
[7] J.Han, J.Pei and Y.Yin. "Mining freq. patt.without candidate Generation", in Proceeding of ACM SIGMOD International Conference Management of Data, 2000, pp.1-12.
[8] Han J., Dong G., Mortazavi-Asl B., Chen Q., Dayal U., Hsu M.-C., Freespan: Frequent pattern-projected sequential pattern mining, Proceedings 2000 Int. Conf. Knowledge Discovery and Data Mining (KDD'00), 2000, pp. 355-359.
[9] Han, J., Pei, J., Mortazavi-Asl, B. and Zhu, H., ―Mining access patterns efficiently from web logs", In Proceedings of the Pacific- Asia Conference on Knowledge Discovery and Data Mining (PAKDD'00) Kyoto Japan, 2000.
[10] J. Pei, J. Han, B. Mortazavi-Asi, H. Pino, "PrefixSpan: Mining Sequential Patterns Efficiently by Prefix- Projected Pattern Growth", ICDE'01, 2001.
[11] Helen Pinto Jiawei Han Jian Pei Ke Wang, ―Multidimensional Sequential Pattern Mining, In Proc. 2001 Int. Conf. Information and Knowledge Management (CIKM'01), Atlanta, GA, Nov. 2001 pp. 81–88.
[12] Chen, Y.L., Chiang, M.C. and Kao, M.T, ―Discovering time interval sequential patterns in sequence databases, Expert Syst. Appl., Vol. 25, No. 3, 2003, pp. 343–354.
[13] Yan, X., Han, J., and Afshar, R., ―CloSpan: Mining closed sequential patterns in large datasets, In Third SIAM International Conference on Data Mining (SDM), San Fransico, CA, 2003, pp. 166–177.
[14] Ding-Ying Chiu, Yi-Hung Wu, Arbee L.P. Chen "An Efficient Algorithm for Mining Frequent Sequences by a New Strategy without Support Counting" Data engineer, 2004, proc. of 20 International conference, pp.375-386, 2004.
[15] Jian Pei, Jiawei Han, Wei Wang, ―Constraint-based sequential pattern mining: the pattern growth methods, J Intell Inf Syst, Vol. 28, No.2, 2007, pp. 133 –160.