Efficient Sequential Pattern Mining Algorithms

RENATA IVANCSY, ISTVAN VAJK
Department of Automation and Applied Informatics and HAS-BUTE Control Research Group
Budapest University of Technology and Economics
H-1111, Goldmann Gy. ter 3., Budapest
HUNGARY

Abstract: - Sequential pattern mining is a heavily researched area in the field of data mining with wide variety of applications. The task of discovering frequent sequences is challenging, because the algorithm needs to process a combinatorially explosive number of possible sequences. Most of the methods dealing with the sequential pattern mining problem are based on the approach of the traditional task of itemset mining, because the former can be interpreted as the generalization of the latter. Several algorithms use a level-wise "candidate generate and test" approach, while others use projected databases to discover the frequent sequences. In this paper a classification of the well-known sequence mining algorithm is presented. Because each algorithm has its own advantages and drawbacks regarding the execution time and the memory requirements, and the exact aim of the algorithms differs as well, thus an exact ranking of the methods is omitted. A basic level-wise algorithm, the GSP is described in detail. Because the level-wise algorithms need less memory in general than the projection-based ones, an efficient implementation of the GSP algorithm is also suggested. Two novel methods, the Bitmap-based GSP (BGSP) and the SM-Tree (State Machine-Tree) algorithms are presented as an enhancement of the GSP-based sequential pattern mining approach.

Key-Words: - Data mining, Sequential pattern mining, GSP algorithm, Itemset discovering, Apriori algorithm

1 Introduction

The aim of sequential pattern mining is to discover frequent recurring subsequences in a large sequence database as patterns. It is an important data mining task with broad applications like Web usage analysis, DNA sequences and customer sequences and many others.

The problem of sequence mining can be interpreted as a generalization of the itemset mining, which was first introduced in [1]. Consider a database of a supermarket which stores the transactions of the customers. The transaction is defined as the set of items bought by a customer at a time. If the customers are not distinguished in the database, i.e. no data is stored about the customers; the task is to find frequent recurring itemsets in the transactions. This is the itemset mining problem commonly known as market basket analysis. However, if the database differentiates the customers from each other, for example by collecting the customer's information by using a card for frequent buyer discounts, then a sequence of bought items can be created for each customer. In this way frequent recurring events, i.e. subsequences can be discovered for example for advertising purpose.

The problem of sequence pattern mining is more complex than the problem of frequent itemset discovering because of the following reason. While in frequent itemset mining "only" the combinations of the items has to be generated, in case of sequence mining the variations of the combinations of the items has to be generated.

There are several algorithms dealing with the problem of sequential pattern mining, and they differ in several ways. Most of them discover all the frequent patterns in the sequence database, while others discover only the frequent closed patterns or the maximal frequent patterns. Another aspect of differentiating the algorithms could be the representation of the transactions. All approaches have their advantages and disadvantages, the different algorithms suit different kind of problem, thus a classification of them can be useful. In this paper the different aspects of classifying the sequence pattern mining algorithms is described. The main features of the best-known algorithms are outlined. A basic level-wise algorithm, the GSP is introduced in detail. It is an important method, because it can handle not only the simple sequence mining problem, but also the problem of defining time constraints, sliding time windows and taxonomies in sequential patterns as well.

The organization of the paper is as follows. Section 2 introduces the problem of sequential pattern mining including the problem of handling time constraints. In Section 3 the aspects of classification are discussed and a classification of the best known algorithm based on the aspects is shown. Section 4 explains the GSP algorithm, a basic level-wise, "candidate generate and test" algorithm in detail. In Section 5 two novel level-wise methods are suggested, which solve the frequent sequence mining problem efficiently. Conclusion can be found in Section 6.

2 Problem Statement

The problem of sequential pattern mining was first introduced by Agrawal and Srikant in [2]. Let D denote the set of transactions. Each transaction consist a customer identifier (CustID), a transaction time (time) and the set of items, called itemset. The ordering of the items in an itemset is irrelevant, but in order to handle the transactions easier, and without loss of generality, it is assumed that they are in lexicographical order. A sequence is defined as an ordered list of itemsets.

An *itemset* is denoted with $i=(i_1, i_2, ..., i_m)$ where i_j is an item. A *sequence* is denoted with $s=<s_1, s_2, ...,s_n>$, where s_i is an itemset. An item can occur in an itemset only once while in a sequence multiple times. A sequence $<a_1, a_2, ...,a_n>$ is contained by another sequence $<b_1, b_2, ..., b_m>$ if there exists integers $i_1< i_2< ...< i_n$ such that $a_1 \subseteq b_{i1}, a_2 \subseteq b_{i2}, ..., a_n \subseteq b_{in}$.

The length of a sequence is defined in two different ways in the literature. In [2] the *length* of a sequence is defined as the number of itemsets contained by the sequence. [3] and the later results of the authors interpret the definition of the *length* of a sequence as the number of items in the whole sequence. Because the latter definition is more current, in this paper the second definition is used. A *customer sequence* is a list of itemsets bought by the given customer. In this case the itemsets are ordered regarding their transaction times, and the identifier of the sequence transaction (TID) is the customer identifier. The support of a sequence s (denoted as *sup(s)*) is the number of customer transactions which contain the sequence s.

Beyond the basic problem of sequential pattern mining several constraints can be added to make the results more adequate for the user. This can be for example a sliding window. When using sliding window the transactions are grouped together such that those transactions belong to the same group whose transaction time is between a user-given intervals. This is practically a window which slides through the transactions. When defining time constraints the user can give a maximum gap and a minimum gap between the transactions belonging to the same group. The purpose of the minimum gap is to not to distinguish those transactions which are too close to each other, and defining the maximum gap means that the transactions which are far from each other are irrelevant. For example for a book store it is irrelevant, whether a customer buys two books in three years. It has more relevance when the time interval between the two events is less than a year.

When given a user-defined minimum support threshold (*minsup*) the task of sequential pattern mining is to discover those sequences which are contained by at least *minsup* number of sequences in the database. These sequences are called frequent sequences. If the size of the sequence is k, it is denoted as *k-sequence*.

3 Classification of the algorithms

Like in frequent itemset mining the algorithms for the problem of frequent sequence mining differs in many ways. When comparing the performance of two algorithms it is important that they should achieve about the same task.

In this paper the following aspects are considered to be relevant when classifying frequent sequence discovering algorithms:

- The type of the discovered frequent sequences.
- The method of the search space traversal.
- The number of disk access, i.e. which basic frequent itemset mining algorithm serves as a basis for the sequence mining method.
- The representation of the transactions, i.e. how the support of the transaction can be counted.

3.1 Type of the frequent sequences

It is an important aspect that frequent patterns are searched for by the algorithm. Both the execution time and the memory requirement depend strongly on the type of the discovered frequent sequences. The reason for that is that the number of the different types of frequent sequences differs significantly.

The most general approach is to discover all the sequences whose support is over the minimum support threshold. The following algorithms discover all the frequent sequences: AprioriAll [2], GSP [3], SPIRIT [4], SPADE [5], FreeSpan [6], PrefixSapn [7] and SPAM [8].

Another approach is to discover only the maximal frequent sequences. A sequence s is maximal if s is not contained in any other sequence. The maximal frequent sequences are discovered by the following algorithms: AprioriSome [2], DynamicSome[2] and DFS-MINE [9]. The benefit of mining only the maximal sequences is that in this case the search space of the mining algorithm is reduced significantly, however the drawback of it, that the support values of the non-maximal sequences are not present. This approach is useful when the application needs only the maximal sequences.

The definition of the closed sequence is similar to that of the closed itemset. The sequence s is a frequent closed sequence, if s is frequent, and there exists no supersequence q of s, such that the support of q is the same as the support of s. The benefit of discovering only the frequent closed sequences is that all the frequent sequences and their support values can be derived from them, and it reduces the search space significantly, however not as much as the maximal sequence discovering methods. CloSpan [10] and BIDE [11] are algorithms which discover the frequent closed sequences.

3.2 Search space traversal

When traversing the search space the algorithms use two types of approaches. The first is the breadth-first search traversal, BFS for short, and the second is the depth-first search traversal, DFS for short. The DFS has an elegant recursive implementation, while BFS needs more iteration. BFS is used by the AprioriAll, AprioriSome, DynamicSome and GSP algorithms. The DFS traversal is used by FreeSpan, PrefixSpan, SPAM and DFS-MINE. The SPADE algorithm offers a possibility to the user to choose between the two methods.

3.3 Number of disk access

Because of the high cost of the I/O process, it is an important issue how many times the algorithm has to access the database. In the frequent itemset mining area there exist basically two types of algorithms regarding the number of disk access. The first type incorporates the methods like the Apriori [10] algorithm, i.e. Apriori-like algorithms, which access the database at least as many times as the size of the longest itemset is. In sequence mining there exist several algorithms whose approach is based on the principles of the Apriori algorithm, like AprioriAll, AprioriSome, DynamicSome, GSP, SPADE and SPIRIT. The other approach of the itemset mining is the two-phase mining like the FP-growth algorithm [12]. The FP-growth algorithm scans the database only twice, while a so-called FP-tree is created in the main memory. The process is then executed recursively by creating further conditional FP-trees in the memory. The idea of the FreeSpan and PrefixSpan algorithms is based on the principle of the FP-growth algorithm, thus they access the database much less than the Apriori-like algorithms do. However, the memory requirements of the former algorithms are much huger than that of the latter.

3.4 Transaction representation

Two approaches are wide-spread in counting the support of the transactions, the horizontal and the vertical representation. Both of the two representations can be implemented both as lists and as bitmaps. Thus four types of representation exist. The existing sequential pattern mining algorithms use lists in case of horizontal representation and in case of vertical representation both list and bitmaps.

The horizontal transaction representation means that for each transaction the itemsets are listed which are contained by the transaction. By vertical representation for each sequence there is a list of transaction identifiers which holds the given sequence. If N denotes the number of transactions in the database, then using a bitmap vertical representation means that for each sequence there exists a bit vector of length N. The i^{th} item of the vector is set *1* if the transaction with the identifier i contains the given sequence, else it is set to *0*.

The benefit of the vertical representation is that the support counting of a candidate sequence can be achieved easily by intersecting the TID lists or vectors of two sequences. The drawback is, however, that the length of the list or the vector depends on the number of transactions, which can be very large.

All of the algorithms mentioned so far use a horizontal representation except SPADE, which uses a linked list for the transaction identifiers, and SPAM, which uses a vertical bitmap representation. Table 1 shows the different features of the most commonly known frequent sequence mining algorithms regarding the aspects described so far.

Table 1. Classification of the algorithms

Algorithm	BFS/ DFS	All/ Max/ Closed	Fundamental algorithm	Transaction representation
AprioriAll [2]	BFS	All	Apriori	Horizontal
AprioriSomew [2]	BFS	Max	Apriori	horizontal
DynamicSome [2]	BFS	Max	Apriori	horizontal
GSP[3]	BFS	All	Apriori	horizontal
SPIRIT [4]	BFS	All	Apriori	Horizontal
SPADE [5]	both	All	Apriori	Vertical list
FreeSpan [6]	DFS	All	FP-growth	Horizontal
PrefixSpan [7]	DFS	All	FP-growth	Horizontal
SPAM [8]	DFS	All	Apriori	Vertical bitmap
DFS-MINE [9]	DFS	Max		Horizontal
CloSpan [10]	DFS	Closed	FP-growth	Horizontal
BIDE [11]	DFS	Closed	—	Horizontal

Regarding the features of the methods depicted in Table 1. The algorithms which discover all the frequent sequences can be categorized mainly into three classes which is depicted in Fig. 1.

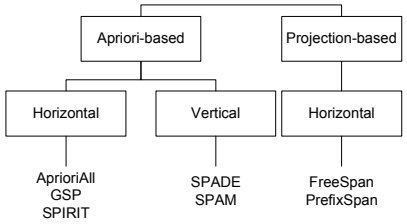

Fig. 1. Classification of the algorithms

4 GSP algorithm

The GSP (Generalized Sequential Patterns) is one of the first sequential pattern mining algorithms. Its idea is based on the Apriori frequent itemset mining algorithm, and uses the a-priori hypothesis, namely, a sequence can be only frequent, if all its subsequences are frequent as well. The GSP is a level-wise, "candidate generate and test" method. The GSP algorithm uses the length definition as the number of items contained by the transaction.

The algorithm works as follows. During the first database scan the 1-sequences (practically the frequent

items) are discovered. It is done by counting the support of each item in the database such that when processing a transaction a counter of a certain item is incremented only once, no matter how many times the transaction contains the given item. From the frequent one sequences 2-candidates are generated. There are two types of 2-candidates, namely, the candidates which are in the form $<(x,y)>$, where x and y are different frequent items, and $x < y$, and the candidates of the form $<(x)(y)>$, where x and y are arbitrary frequent items. After generating the candidates a further database scan follows, and the support of the candidates is counted.

In general the rule for generating a candidate of size k is as follows. Let s_1 and s_2 be two sequences of size $(k-1)$. Let s_1' be a sequence derived from s_1 such that the first item of the s_1 is omitted. Let s_2' be a sequence derived from s_2 such that the last item of it is omitted. If $s_1' = s_2'$, then s_1 and s_2 are joined to cerate a sequence s_3 of size k. The resulting sequence s_3 is generated as follows. The prefix of s_3 is the sequence s_1. The only question is how the last item of s_2 is inserted to the end of the sequence. If the last item of s_2 is a single item in the last itemset, then it is inserted as a new itemset at the end of s_3. If there were other items in the same itemset before the last item of s_2, then it is inserted at the end of the last itemset of s_3. s_3 becomes only a candidate if all its $(k-1)$-subsequences are also frequent, thus this has to be checked as well. For this reason all the (k-1) subsequences are generated from s_3 by omitting each single item from the sequence. If all the resulting sequences are frequent, then s_3 becomes a candidate.

After generating the k-candidates, a database scan is executed and the support of each candidate is counted. To reduce the number of candidates to be checked when processing a transaction, a hash-tree is created whose leafs index the candidates. Using this hash-tree the process of counting the support of the candidate is enhanced. The GSP algorithm solves not only the basic frequent sequence mining problem, but also can handle time constraints, sliding windows and item taxonomies. For this reason, it is worth to find algorithms which have these benefits, but are more efficient than the GSP algorithm.

5 BGSP and SM-Tree algorithms

The BGSP (Bitmap-based GSP) and the SM-Tree (State Machine-Tree) algorithms are based on the "candidate generate and test" approach of the GSP algorithm. However they differ in several important aspects.

In the next subsections the novel methods are introduced, and some implementation details are discussed as well, because the performance (execution time and the memory requirement) of an algorithm depends strongly on its implementation. The main difference in the two proposed methods is the way they count the support of the candidate sequences longer than two.

5.1 Discovering the frequent 2-sequences

Both of the algorithms use the same method to count the 1 and 2-sequences. The 1-sequences i.e. the items are counted using an array with size of n, where n denotes the number of items that can appear in the database. For detecting that an item has already been counted in a transaction another array is used. In order not to clear the array each time when a new transaction is read, not a binary array is used, but the i^{th} column is set to the number of the transaction when i was present in the sequence. When an item is processed in the sequence, the value of the i^{th} element of the "shadow" array is checked and if it is the same as the number of the transaction just processed, then it shows that the item already appeared in the transaction and its counter have not being incremented, otherwise the counter in the other array is incremented. After scanning the whole database the frequent items are found and an index table is created in order to indexing the frequent items quickly.

For counting the 2-candidates two matrices are used which is more efficient than using a hash-tree as the GSP algorithm does, because the matrices can be indexed in a direct way. For counting the candidates having the form of $<(x,y)>$ an upper triangular matrix, denoted with M_1, is needed because in this case $x < y$. For counting the support of sequences in the form of $<(x)(y)>$ a matrix, denoted with M_2, is needed because in this case both of the items can have any value of the whole set of frequent items. The dimensions of both matrices are the same as the number of the frequent items. Using these matrices the counting of the 2-candidates can be achieved in the most efficient way because of the direct indexing possibility provided by the index table and the matrices. To avoid the counting of one item more than one times in a transaction, two "shadow" matrices are used as in the first phase.

The 3-candidates are generated by traversing the two matrices. The rules for creating the candidates are the following.

Rule1: if $sup(M_1[s_1,s_2]) > minsup$ and $sup(M_1[s_2,s_3]) > minsup$ then they are joined, if $sup(M_1[s_1,s_3])> minsup$. Then the resulting sequence is $<(s_1, s_2, s_3)>$.

Rule 2: if $sup(M_1[s_1,s_2]) > minsup$ and $sup(M_2[s_2,s_3]) > minsup$ then they are joined, if $sup(M_2[s_1,s_3])> minsup$. Then the resulting sequence is $<(s_1, s_2)\ (s_3)>$.

Rule 3: if $sup(M_2[s_1,s_2]) > minsup$ and $sup(M_1[s_2,s_3]) > minsup$ then they are joined, if $sup(M_2[s_1,s_3])> minsup$. Then the resulting sequence is $<(s_1)\ (s_2, s_3)>$.

Rule 4: if $sup(M_2[s_1,s_2]) > minsup$ and $sup(M_2[s_2,s_3]) > minsup$ then they are joined, if $sup(M_2[s_1,s_3])> minsup$. Then the resulting sequence is $<(s_1)\ (s_2)(s_3)>$.

One of the most time consuming step of the whole frequent sequence discovering process is to discover the support of the candidates. For this reason this step has to be enhanced as well. The following two subsections suggest two ways to achieve this process.

5.2 BGSP algorithm

The main contribution of the BGSP algorithm is that it uses a bitmap representation for the candidate sequences. The aim of such a representation is to enhance the process of detecting whether a sequence contains a candidate sequence. For this reason each sequence is represented as a matrix. The matrix has as many columns, as many itemsets the sequence has. The number of the rows equals the number of the frequent items. In each column those cells are set to 1 whose item is contained by the given itemset. Because such a representation is not efficient regarding the memory requirement (it has to store values not only for items which are present in the sequence, but also for all the other items as well), it is worth using only when the number of possible frequent items is not large.

In order to check whether the sequence *s* contains the sequence *q* they bitmaps have to be compared. In a loop the columns of *q* have to be compared to the columns of *s* by using a binary OR operation. If a column of *q* matches the column of *s*, the next column of *q* should be compared to the further columns of *s*. The number of candidates to be checked can be reduced by using hash-trees to store the bitmaps of the candidates.

The bitmap representation of the two sequences is a benefit as well when creating a candidate. Let s_1 and s_2 be two k-frequent sequences. Let their bitmap representations be B_1 and B_2. The task is to find whether the two sequences share the same *(k-1)*-subsequence. For this reason the first items of s_1 and the last item of s_2 have to be pruned. If the resulting two sequences are the same, then s_1 and s_2 are joined. Checking of the matching can be done simply by setting the first item of the first level in B_1 to zero, and the last item in the last level of B_2 as well. Afterwards the two bitmaps have to be compared using a binary AND operation. If the bitmaps are the same, then the join of the two sequences can lead to a candidate.

5.3 SM-Tree algorithm

The main contribution of the SM-Tree (State Machine-Tree) approach is to use finite state machines to discover whether a candidate sequence is contained by a transaction. In order to handle efficiently these machines they are joined to form an SM-Tree.

A sequence is represented as the list of items and the separating items, which can be for instance the minus sign. For example the sequence *<(ab)(c)(de)>* is represented as *ab-c-de*. A finite state machine with the alphabet $\sum$ for a given candidate sequence can be created as follows. Starting from the start state for each new item and for each minus sign as well, a new state is created and the transition between the states contains the item. These are in the state diagram of the finite state machines all forward edges. There are several backward edges as well. A backward edge is created between the state having no transition with the minus sign and the state just after the last minus sign so far. From each state there exist transitions to all the items such that the state will be the same (self loops). The accept state of the machine is the state for the last item of the sequence. It is susceptible of proof that the finite state machine gets to the accept state iff the input string of the sequence contains the string of the candidate sequence for which the machine was created. Fig. 2 shows the state diagram of the sequence *<(ab)(c)(de)>*.

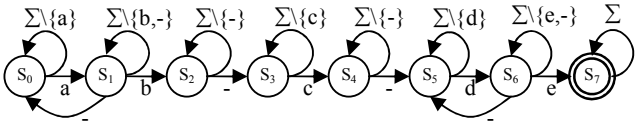

Fig.2. State diagram of the sequence *<(ab)(c)(de)>*

When dealing a large number of candidates sequences an efficient way has to be found to handle the several state machines. For this reason a JOIN operation is introduced between the state machines. Two state machines of sequences can be joined if the sequences share a prefix in common. Because all state machines have a start state, all of them can be joined to form a so-called State Machine-Tree. Fig. 3 shows the joining step of two state machines (the self loops are not shown).

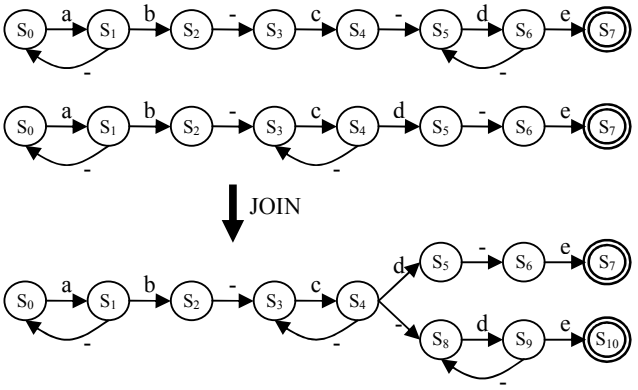

Fig.3. Joining of two state machines of the sequences *<(ab)(c)(de)> and <(ab)(cd)(e)>*

Because multiple state machines are joined, the tree can have more than one current state at the same time. In order to maintain the several current states of the tree, tokens are used. On each current state of the tree a token is placed. When a new item is read from the input string all the new current states are found using the transition

functions. If a token reaches a leaf of the tree, the counter of the corresponding sequence is incremented. When the last item of the input sequence has been processed the support counting phase is finished. Thus each item of the input string is processed exactly once.

Fig. 4 shows the execution time of the SM-Tree and that of the SPAM algorithms.

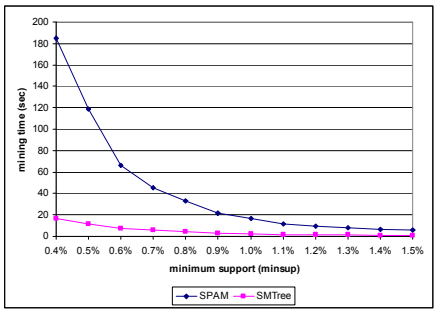

Fig.4. Execution time of the SPAM and the SM-Tree algorithms on the dataset D25C10T5S4I1.25

6 Conclusion

In this paper we have covered some of the efficiency issues of the algorithmic sequential pattern mining. The aim of our examination was to classify the most commonly known frequent sequence discovering algorithms. We have established a system of aspects for classifying these algorithms. After classifying the well-known algorithms, the basic level-wise algorithm, the GSP algorithm was described in detail. Due to its level-wise approach the GSP algorithm uses less memory than the database projection-based methods, thus it is worth enhancing the GSP algorithm.

In the second part of the paper two novel methods called Bitmap-based GSP and SM-Tree are contributed. The main contribution of both algorithms is to enhance the process of the subsequence testing step in the whole process. This is an important part of the algorithm hence it is executed very often. The new BGSP method uses a bitmap representation for the sequences, thus comparing two sequences can be done with a simple binary operator. However it is efficient only in case of low number of items because of the sparse property of the bitmaps. The SM-Tree algorithm uses joined state machines to count the support of the candidates. It has the advantage over the hash-tree of the GSP algorithm that using the SM-Tree the items of the transactions are processed exactly once while using the hash-tree they are processed several times. The SM-Tree algorithm is efficient independently of the number of items.

Acknowledgments

This work has been supported by the fund of the Hungarian Academy of Sciences for control research and the Hungarian National Research Fund (grant number: T042741).

References:

[1] R. Agrawal and R. Srikant, Fast algorithms for mining association rules, *Proc. of the 20th Int'l Conference on Very Large Databases, Santiago, Chile, Sept. 1994*

[2] R. Agrawal and R. Srikant, Mining Sequential Patterns, *In Proc. of the 11th Int'l Conference on Data Engineering,* Taipei, Taiwan, March 1995.

[3] R. Srikant and R. Agrawal, Mining Sequential Patterns: Generalizations and Performance Improvements, *In Proc. of the 5th Int. Conference extending Database Technology (EDBT'96),* Avignong, France, 1996, pp. 3-17.

[4] M. N. Garofalakis, R. Rastogi and K. Shim, SPIRIT:Sequential Pattern Mining with Regular Expression Constraints, *In Proc. of of the 25th International Conference on Very Large Data Bases,* Edinburgh, Scotland, UK, 7-10. Sept, 1999, pp. 223-234.

[5] M. Zaki. SPADE: An efficient algorithm for mining frequent sequences. *Machine Learning*, 40:31–60, 2001.

[6] J. Han, J. Pei, B. Mortazavi-Asl, Q. Chen, U. Dayal, MC Hsu, FreeSpan: Frequent Pattern-Projected Sequential Pattern Mining, *In Proc. of the 6th Int. Conf. on Knowledge Discovery and Data Mining (KDD2000)*, Boston, USA, pp. 20-23. 2000.

[7] J. Pei et al. PrefixSpan: Mining sequential patterns efficiently by prefix-projected pattern growth. *In Proc. of International Conference on Data Engineering*, *ICDE'01,* Heidelberg, 2001.

[8] J. Ayres, J. E. Gehrke, T. Yiu and J. Flannick, Sequential Pattern Mining Using Bitmaps. *In Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. Edmonton, Alberta, Canada, July 2002.

[9] I. Tsoukatos and D.Gunopulos, Efficient Mining of Spatiotemporal Patterns, *In Proc. of 7th International. Symposium on Spatial and Temporal Databases*, Los Angeles, USA, 2001, pp. 425-442.

[10] X. Yan, J. Han and R. Afshar, CloSpan: Mining Closed Sequential Patterns in Large Databases, *In Proc. of the 2003 SIAM International Conference on Data Mining*, San Fransisco, USA, pp. 166-177.

[11] J. Wang and J. Han, BIDE: Efficient Mining of Frequent Closed Sequences, *In Proc. of the 20th International Conference on Data Engineering*, Boston, Massachusets, 2004.

[12] J.Han, J. Pei and Y. Yin, Mining frequent patterns without candidate generation, *In Proc. of the 2000 ACMSIGMOD Int'l Conf. On Management of Data, Dallas, Texas, USA, May* 2000.