Journal of Computer Science 2 (4): 318-321, 2006
ISSN 1549-3636

A New Incremental Updating Algorithm for Mining Sequential Patterns

Jia-Dong Ren and Xiao-Lei Zhou
College of Information Science and Engineering, Yanshan University, Qinhuangdao 066004, China

Abstract: This study discusses how to maintain discovered sequential patterns when some information is deleted from a sequence database. A new algorithm, called *MA_D* (*M*aintenance *A*lgorithm when *D*eleting some information), is presented in order to deal with the maintenance of sequential patterns mining resulted from the updating of database and the algorithm makes full use of the information obtained from previous mining results to cut down the cost of finding new sequential patterns in an updated database. Our experimental analysis shows that the new algorithm is more efficient.

Key words: Data mining, sequential pattern, incremental updating, maintenance

INTRODUCTION

Data mining is to extract previously unknown and potentially useful information or knowledge from database. Sequential pattern mining, which discovers frequent patterns in a sequence database, is an important issue among the various data mining problems. Sequential pattern was first introduced by Agrawal and Srikant[1] and since its introduction, there have been many researches on efficient mining techniques and algorithms, extensions of sequential pattern mining method and its applications.

In general, sequential pattern mining algorithms can be sorted into two classes:

1. Apriori-based[2] candidate generation and test philosophy, with GSP[3] (Generalized Sequential Pattern mining algorithm) and SPADE[4] as its representative. *GSP* firstly discovers frequent 1-sequences and then generates candidate (k+1)-sequences from the sets of frequent k-sequences. With *SPADE* algorithm, all the frequent sequences and their negative borders build up a sequence lattice. When the incremental data arrive, the incremental parts are scanned and the sequence lattice is updated and then we can determine that which parts in the original database should be scanned according to the sequence lattice and incremental data. Here, *GSP* algorithm adopts horizontal format to mine sequential patterns, while *SPADE* algorithm adopts vertical format. Subsequently, some scholars presented FAST[5] algorithm for incremental updating of sequential patterns mining. The algorithm can be used to solve the incremental updating problem when the minimum support threshold changes and the sequence database remains invariable.

2. Projection-based pattern growth method, with Freespan[6] and Prefixspan[7] algorithms as its representative. The two algorithms apply a divide-and-conquer strategy to generate many smaller projected databases of the original database and then the frequent sequences are mined in each projected databases by exploring local frequent patterns.

The existing algorithms discuss the problems that how to mine sequential patterns quickly and how to maintain the discovered sequential patterns. So far, the research on incremental updating of sequential pattern mining has been focusing on two aspects: on the one hand, when new transactions and new data-sequences are appended to the original database, how to deal with the incremental updating of sequential pattern mining; on the other hand, when the minimum support threshold changes and the original database doesn't change, how to deal with the maintenance problem of sequential pattern mining. But in the fields of Electronic Commerce and Web usage mining, we often delete some information from sequence database, in order to save storage space or because some information is not interesting any longer or has become invalid. The incremental updating of sequential pattern mining in this circumstance has been paid little attention in previous studies.

In this study, we investigate the issue and develop a new algorithm, called *MA_D*, to deal with the problem that when some information is deleted from a sequence database, how to maintain the discovered sequential patterns. Our algorithm utilizes the information obtained from prior mining processes and stores the sets of discovered frequent sequence in the original database for further mining. Meanwhile, it adopts a new method to generate the sets of candidate sequence, which cuts down the size of candidate sets in some extent.

Problem definition: Let $I=\{i_1, i_2, \ldots, i_n\}$ be a set of all items. An *itemset* is a non-empty set of items. A sequence is an ordered list of itemsets. A sequence is denoted by $<s_1,s_2,\ldots,s_l>$, where s_j is an itemset, i.e.,

Corresponding Author: Jia-Dong Ren, College of Information Science and Engineering, Yanshan University, Qinhuangdao 066004, China

$s_j \subseteq I$ for $1 \le j \le l$. s_j is also called an element of the sequence and denoted as $(x_1x_2\ldots x_m)$, where $x_k \in I$ for $1 \le k \le m$. The number of instances of items in a sequence is called the length of the sequence. A sequence with length l is called a l-sequence. A sequence $a = <a_1,a_2,\ldots,a_n>$ is called a subsequence of $b = <b_1,b_2,\ldots,b_m>$ and b a super sequence of a, denoted as $a \subseteq b$, if there exist integers $1 \le j_1 < j_2 < \ldots < j_n \le m$ such that $a_1 \subseteq b_{j1}$, $a_2 \subseteq b_{j2}$, … , $a_n \subseteq b_{jn}$.

A sequence database D is a set of tuples $\langle sid, s \rangle$, where *sid* is a sequence-id and s is a sequence. A tuple $\langle sid, s \rangle$ is said to contain a sequence a, if a is a subsequence of s, i.e., $a \subseteq s$. The number of tuples in a sequence database D containing sequence a is called the *support* of a, denoted as $sup(a)$.

Given a sequence database D and some user specified minimum support *min_sup*, a sequence a is a sequential pattern in D if $sup(a) \ge min\_sup$. The sequential pattern mining problem is to find the complete set of sequential pattern with respect to D and *min_sup*.

When some information is deleted from an original sequence database, some formerly discovered sequential patterns may become invalid and some new frequent patterns may appear in the resulting updated database. The incremental updating of sequential pattern mining is to discover all the frequent sequences in the updated database with respect to the same minimum support threshold. When the database is updated, the incremental updating method must utilize previously discovered information to avoid re-mining the whole updated database from scratch. The objective of maintaining and updating sequential patterns is to respond to each mining quickly when some information is deleted from a sequence database and to minimize the overall runtime for the whole process accordingly.

The *MA_D* algorithm: Let *DB* be an original sequence database, *dd* be the database consisting of deleted information ($dd \subset DB$), *DB-dd* be the updated sequence database, s be the user specified minimum support threshold, D be the size of *DB*, d be the number of deleted data-sequences in *DB*, $\dot{\cup}_k L_k$ be the sets of frequent k-sequence in *DB*, $\dot{\cup}_k F_k$ be the sets of frequent k-sequence in *DB-dd*, C_1 be the sets of candidate 1-sequence in *DB*.

An overview: When some information is deleted from an original database, the incremental updating problem of sequential pattern mining can be considered into the following two cases:

* Only some transactions (but not data-sequences) are deleted from the sequence database, the minimum support count remains constant.

  Because some transactions are deleted, the support count of some sequences that contain these deleted items may diminish and not satisfy the minimum support count. They might become infrequent sequences. In this case, we can deal with it easily by deleting the infrequent sequences from the old set of sequential patterns. The method can be described as follows: Scanning the updated database *DB-dd* only once, we can obtain the support count of the sequences in $\dot{\cup}_k L_k$. And then, infrequent sequences are filtered out from $\dot{\cup}_k L_k$ and the frequent ones remain. In this case, it is obvious that we can obtain the set of new sequential patterns without any mining operations.
* When some transactions and data-sequences are deleted from the original database, the size of the database will become small, which results in smaller minimum support threshold (The minimum support count will be $s*(D-d)$). Besides some formerly frequent sequences have become infrequent ones, new frequent sequences may appear.

  For the case that the formerly frequent sequences become infrequent ones in *DB-dd*, we first scan the updated database *DB-dd* once and obtain the new support count of the originally old frequent sequences in $\dot{\cup}_k L_k$. After that, the sequences that don't satisfy the new support count $s*(D-d)$ will be filtered from $\dot{\cup}_k L_k$ and the frequent sequences still will be preserved. This case is the same as above mentioned.

  It is a vital issue that how to discover all the new frequent sequences in *DB-dd*. Here, in order to solve the problem, we take GSP (Generalized Sequential Pattern) algorithm for example to analyze the general method used for finding frequent sequences. GSP algorithm is in general seen as a breadth-first traversal algorithm, it discovers all the frequent sequences by making multiple passes over the database. At the 1st pass, GSP algorithm discovers all the frequent 1-sequences; at the kth pass, it generates the set of potentially frequent k-sequences (C_k) from the set of frequent (k-1)-sequences (usually called candidates) and then scans the database to compute the support of each candidate sequence and discover the frequent k-sequences. The process iterates until no more new frequent sequences are generated.

  Based on the above discussion, we shall conclude that three kinds of k-sequences appear at the kth pass:

* Set of frequent k-sequences (L_k) obtained from prior mining process;
* α Candidate sequences obtained from candidate k-sequences above mentioned (C_k) minus L_k, i.e., $C_k - L_k$;
* β Candidate sequences obtained from all the k-sequences minus L_k and $C_k - L_k$;

To the three kinds of k-sequences above mentioned, we discuss the generation of new frequent sequences, respectively.

With the minimum support count, i.e., $s^*(D-d)$, we scan the updated database *DB-dd* once and obtain the new support count of the sequences in L_k. And then, the infrequent sequences in L_k will be filtered out. That is to say, no new frequent sequences are generated from L_k;

* α Naturally, $C_k - L_k$ should be scanned, compared and filtered as candidate sequences;
* β Because of new minimum support count ($s^*(D-d)$), the formerly old candidate (k-1)-sequences may become frequent ones, which will result in new candidate k-sequences generation;

When some information is deleted from a sequence database, the incremental updating problem of sequential pattern mining lies in how to obtain new candidate sequences from the set of sequences in the third case above mentioned.

The generation of new candidate sequences: In order to find new frequent sequences, we must find the set of candidate sequences containing these new frequent sequences. This study adopts a new candidate sets generation method depicted as follows.

First, scanning the updated sequence database *DB-dd* once, we can count the support of the new candidate 1-sequences, denoted by NC_1, which are not contained in L_1. Obviously, the new set of candidate 1-sequences can be written as $NC_1=C_1-L_1$, thus the new frequent 1-sequences, denoted by NL_1, is $NL_1=\{x|x\in NC_1 \wedge x.sup\geq s^*(D-d)\}$. The new frequent 1-sequences NL_1 and the originally old frequent 1-sequences L_1 make up all the frequent 1-sequences, denoted by F_1, in *DB-dd*, i.e., $F_1=NL_1 \cup L_1$, obviously, $NL_1 \cap L_1 = \varnothing$.

Let us now consider the problem how to seek for the set of frequent k-sequences F_k when $k\geq 2$. In order to improve the efficiency of incremental mining, we should try to reduce the number of the candidate sequences. Take k=2 for example, because of $F_1 = NL_1 \cup L_1$ and $NL_1 \cap L_1 = \varnothing$, according to a basic Apriori[2] property: "Any subsequences of a frequent sequence must be frequent sequences.", the new candidate 2-sequences NC_2 are generated by the sequences in NL_1 or generated by a sequence from NL_1 and a sequence from L_1(This part of NC_2 can be labeled as NC_k^2). Another part of NC_2 can be denoted by $C_2 - L_2$(labeled as NC_k^1). It is obvious that the two part of NC_2 (NC_k^1 and NC_k^2) is mutually exclusive. Scanning the updated database $DB-dd$ once for counting the support of the sequences in NC_2 and then choosing the frequent sequences, we can obtain the new frequent 2-sequences,denoted by NL_2. The updated sequential patterns $F_2=NL_2 \cup L_2$. When $k\geq 3$, we execute above operations iteratively, until $F_{k-1}=\varnothing$.

The incremental updating algorithm of sequential pattern mining (*MA_D* algorithm). Based on above discussion, we shall now depict the *MA_D* algorithm as follows:

Algorithm *MA_D*

Input: *DB* the original database, $\cup_k L_k$ the set of frequent k-sequences in *DB*, s the minimum support threshold, D the size of *DB*, *dd* the deleted database and d the number of deleted data-sequences, C_1 the set of all the 1-sequences in *DB*.

Output: The set $\cup_k F_k$ of all frequent sequences in *DB-dd*.

Method

Step 1: Scanning the updated sequence database *DB-dd* only once, we can count the new support of the sequences in $\cup_k L_k$. And then, infrequent sequences are filtered out from $\cup_k L_k$ and the frequent ones remain.

Step 2: Scanning the updated sequence database once again, we can find new frequent 1-sequences NL_1, thus new frequent 1-sequences and formerly old frequent 1-sequences form all the frequent 1-sequences, i.e., $F_1 = NL_1 \cup L_1$.

Step 3: Seeking for new frequent k-sequences NL_k when $k\geq 2$. NL_k and formerly old frequent k-sequences L_k form all the frequent k-sequences in *DB-dd*, i.e., $F_k = NL_k \cup L_k$. This step iterates until no more new frequent sequences generate.

Our algorithm makes full use of the information obtained from prior mining processes. In the entire mining process, we only need to scan the updated sequence database k passes (Here, k means that the length of the longest sequence pattern in NL_k). We adopt a new candidate generation method to decrease the size of candidate sequences and improve the efficiency of sequential pattern mining.

RESULTS

The two algorithms are implemented in Java language and tested on a Pentium IV-2.4G Windows-XP system with 512MB of main memory and JBuilder 8.0 as the Java execution environment. To make the time measurements more reliable, no other application is running on the machine while the experiments are running. The datasets are maintained in main memory during the algorithms processing, avoiding hard disk accesses. During the execution processes, the datasets are stored and operated in a data structure, called *Vector*, in Java language. And each element in the *Vector* data structure is a data-sequence in the synthetic dataset.

The dataset for our experiments is generated using the standard synthetic data set generator from IBM Almaden[8]. The data set generator has been used in most studies on sequential pattern mining and it

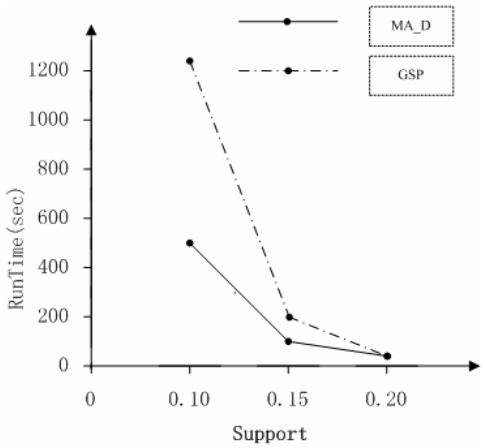

Fig. 1: Performance comparison on the updated dataset *Update_Dataset* (*MA_D* and *GSP*)

generates datasets that imitate real-world transactions, where customers tend to make a sequence of transactions involving a set of items. Sequence and transaction sizes are clustered around a mean and some of them may have larger sizes. Note that a sequence may have repeated transactions, but a transaction cannot have repeated items. Each data-sequence in the synthetic dataset is stored in the nested *Vector* data structure. With this data structure, we can judge whether a sequence is contained in a data-sequence and then the support of the sequence is counted. Based on the support of the sequence, we can further estimate whether the sequence is frequent or not.

In our synthetic dataset, the number of items is set to 1, 000 and there are 10, 000 sequences in the dataset. The average number of elements in a sequence is set to 8. The average number of items within elements is set to 8. The average length of maximal patterns is set to 8 and maximal frequent transactions set to 8. These values were chosen in order to follow closely the parameters usually chosen in other studies. After some transactions and sequences are randomly deleted from the initial synthetic dataset, we can obtain the updated dataset *Update_Dataset*.

Firstly, we use *GSP* algorithm to mine sequential patterns with different minimum support threshold on the initial synthetic dataset. The mining results, i.e., all the frequent *k*-sequences are initially saved in the *Vector* data structure in the main memory and then outputted to the hard disk memory in a random access file format to be used for further incremental mining. Then, incremental mining is performed over the updated dataset. *MA_D* algorithm utilizes the results obtained in the prior mining process to mine sequential patterns over the updated dataset *Update_Dataset* with corresponding minimum support threshold, whereas *GSP* algorithm mines sequential patterns over *Update_Dataset* from scratch.

Figure 1 shows the experiment conducted on the updated dataset *Update_Dataset* using different minimum support thresholds. The label "MA_D" corresponds to the *MA_D* algorithm while "GSP" stands for *GSP* used for mining the updated dataset from scratch.

Figure 1 clearly indicates that the performance gap between the two algorithms increases with decreasing minimum support. We can observe that *MA_D* is faster than running *GSP* from scratch. It can also be noticed that *GSP* algorithm provides the worst performance when the support is lower. The main reason is that when the support threshold is lower, *GSP* must generate numerous candidate sequence sets. The experimental result shows that *MA_D* outperforms *GSP* in significant magnitude.

CONCLUSION

We discuss the incremental updating problem of sequential pattern mining when some information is deleted from a sequence database. A new algorithm, called *MA_D*, is presented for the incremental updating of sequential pattern mining. The algorithm makes full use of the set of sequential patterns obtained from the prior mining processes, improves the efficiency of sequential pattern mining and cuts down the execution time. The performed experiments show that *MA_D* enhances *GSP* by several orders of magnitude for incremental updating of sequential pattern mining.

However, the multiple passes on the database could be a problem in the *MA_D* algorithm. Further work may include the problem that how to reduce the passes on the database.

REFERENCES

1. Agrawal, R. and R. Srikant, 1995. Mining sequential pattern. In: Proc. 11th Intl. Conf. Data Engineering (ICDE'95), Taipei, Taiwan, pp: 3-14.
2. Agrawal, R., T. Imielinski and A. Swami, 1993. Mining association rules between sets of items in large databases. In: Proc. of the 1993 ACM SIGMOD Conf., Washington DC, USA, pp: 207-216.
3. Srikant, R. and R. Agrawal, 1996. Mining sequential pattern: generalizations and performance improvements. In: Proc. of the 5th Intl. Conf. on Extending Database Technology (EDBT'96), Avignon, France, pp: 3-17.
4. Zaki, M.J., 2001. SPADE: An efficient algorithm for mining frequent sequences. Machine Learning J., 42: 31-60.
5. Weimin Ouyang and Qingsheng Cai, 1998. An incremental updating techniques for discovering generalized sequential patterns. J. Software, 9: 778-780.
6. Han, J., J. Pei and B. Mortazavi-Asl, 2000. FreeSpan: Frequent pattern-projected sequential pattern mining. In: Proc. of the 6th Intl. Conf. on Knowledge Discovery and Data Mining (KDD2000), Boston, USA, pp: 355-359.
7. Pei, J., J. Han and B. Mortazavi-Asl *et al.*, 2001. PrefixSpan: Mining sequential patterns efficiently by prefix-projected pattern growth. In: Proc. of the 17th Intl. Conf. on Data Engineering, Heidelberg, Germany, pp: 215-224.
8. http://www.almaden.ibm.com/software/quest/ Resources/datasets/ syndata.html# assocSynData