Maintaining and Mining Sequential Patterns in Incremental Sequence database

WEI-HUA HAO
Department of Electrical Engineering
Mucha Vocational High School
edhao@mcvs.tp.edu.tw

NANCY P. LIN
Department of Computer Science and Information Engineering
Tamkang University
nancylin@mail.tku.edu.tw

HUNG-JEN CHEN
Department of Industrial Engineering and Management
St. John's University
chenhj@mail.sju.edu.tw

CHUNG-I CHANG
Department of Information Management,
St. Mary's Medicine Nursing and Management College
taftdc@mail.tku.edu.tw

HAO-EN CHUEH
Department of Information Management,
Yuanpei University
hechueh@mail.ypu.edu.tw

***Abstract**-Real world databases are dynamic, new data are stored into database over time. The most naïve solution of mining sequential patterns over an incremental database is to rerun the database from scratch, which didn't take the advantage of previous work. The other way is that merge the new sequential patterns set with previous discovered sequential patterns. However, Algorithms that pruned off infrequent sequences are essentially not suitable for merging due to the information loss. In this paper, we proposed a novel algorithm IMSP that transform original sequence database into a frequency data model. In the model constructing process, no candidates were generated and with only one database scan. The advantages of IMSP are proven by example.*

Keywords: *Data mining, Sequential patterns, Candidates, Closed sequential patterns, Lattice structure.*

1. Introduction

Most real world records are time related, such as super market transaction records, scientific records, environment monitoring records and e-Learning data.

The major problem of previous works which base on apriori-like approaches is generating too many candidate sequences and multiple scan of database during the mining process, which urges larger working space, consequence is more unlikely to fit all data into main memory.

1.1 related works

In order to minimize both the working space and searching space it is necessary to introduce closed sequential patterns to our algorithm. Closed itemset and maximal itemset, sequence, concept had been introduced in [2][3][4][5][6]. In GSP, the candidates were generate-and-test repeatedly until no more new frequent sequences are discovered.

To alleviate the drawback of generating huge amount of candidates in the mining process, many researchers had proposed their counter approach; Garofalakis had proposed SPIRIT[7], a Apriori-like algorithm, that generate less candidates via constrains. Han had proposed Prefixspan[8] and Freespan[9] algorithms based on projected databases to improve efficiency. These two algorithms applied a divide-and-conquer approach that generate many smaller projected databases of the original sequence database, only mining the frequent sequences that confined in each projected databases by discovering participated frequent patterns. None the less, these are all apriori-like algorithms still inherit the drawback of candidate gen-and-test.

Many researches of incremental mining of sequential patterns were developed in recent years. An incremental method SPADE[14] of mining sequential patterns was proposed by Zaki. In this paper, equivalent class was introduced to construct sequence lattice in incremental manner. Newly read sequence data from sequence database are updated into the lattice. Other research has presented diverse methods solving the incremental sequence database problem in [23][24]. Many proposed methods on incremental sequence mining have to tackle the problems of dealing with the newly append sequences to the original sequential database to form into previous constructed frequent sequential patterns, and to adjust the sequential patterns with change of minimum support, which is usually change after, or during, the mining process. In the practice fields, e-commerce and eLearning applications, facing an incremental sequence database is inevitable. How to save mining time with less memory is essential to evaluate sequential patterns algorithm. In this aspect, not to rebuild previous construct sequential patterns is almost an essential part to solve this problem.

1.2 Contributions

In our previous studies [4][5], that mining sequential patterns without candidates, and scan sequence database only once. To our best knowledge, this is the optimal condition of scan times. Note that, the FP-tree needs to scan database twice.

The main contributions of this paper are: constructing closed sequence data model without generating candidates, require less memory space and no need to rerun mining algorithm from scratch.

The rest of this paper is organized as follows. In section 2, we define the basic definitions and properties of sequential patterns. In section 3, we analyzed incremental sequence database. In section 4, we describe the adjustable minimum support. The algorithm of IMSP and its example are given in section 5 and 6, respectively. Section 7 gives conclusion and future work.

2. Preliminary

Lemma- Decrement Monotonic:

According to the downward closure property, if a sequence is frequent then all its subsequences are also frequent. The size of frequent sequence $|L_{K+1}|$ is no large than $|L_K|$. In apriori-like algorithm, the size of each length of sequences is decrement monotonic. All apriori-like mining algorithms are compliant to this lemma.

Let the size of searching space, |ss|, is the summary of all frequent sequences.

$$|\mathbf{ss}| = \sum_{i=1}^{k} |L_i|$$

Here, k is the length of longest frequent sequences in frequent sequences set.

Most previous work of apriori-like algorithm didn't discuss about working space, *ws*, that is the minimum space requirement of memory to run the mining algorithm. It is defined as the sum of maximum candidate space and frequent sequences space.

$$ws = \mathbf{max}\,(|C_{k+1}| + |\mathrm{ss_k}|)$$

Usually the size of candidate set is much larger than frequent sequences set, which makes the candidate became the most crucial part when evaluating efficiency.

The problem can be described as follows: Assume $I=\{i_1, i_2, \cdots, i_n\}$ be a set of all items (or events). An itemset is a non-empty set of finite items. A sequence is an ordered list of itemsets. A sequence *s* is denoted as $s=\langle i_1,\ldots,i|s|\rangle$, where ii is an itemset, that is , i_i I for $1\le i\le k$. s_i is also defined as an element of sequence, and denoted as $(x_1\ x_2\cdots x_l)$, where $x_j\in I$ for $1\le j\le l$. In fact, the brackets are usually omitted if $|i_i|=1$. An item can appear at most once in an element of a sequence, but can appear more than one time in different elements of a sequence. The length of a sequence is defined as the number of instances of items in a sequence. A sequence with length l is called an l-sequence. A sequence $x=\langle x_1,x_2,\cdots,x_n\rangle$ is called a subsequence of $y=\langle y_1,y_2,\cdots,y_m\rangle$ and *y* a super sequence of *x*, denoted as $x\ \ y$, *y* contains *x*, if there exist integers $1\le j_1<j_2<\cdots<j_n\le m$ such that $\mathrm{x_1} \prec \mathrm{y_{j1}}, \mathrm{x_1} \prec \mathrm{y_{j1}}, \ldots, \mathrm{x_n} \prec \mathrm{y_{jm}},$. A sequence database D is a set of tuples denoted as $\langle$ SID, s $\rangle$, where SID is a sequence identification number and s is a sequence. Given a k-items sequence s, its support is supp(*s*) which is defined as the number of transactions in D that including s.

Cardinality of sequence s denotes the number of distinct SID values in the id-list of sequence database for a sequence s. A sequence X is closed sequence if there exist no super-sequence that contain X in the equivalent class [6].

Given a sequence database (SDB), a minimum support (*minsup*), and an appended sequence database (ASDB). Let $\mathrm{SDB_{k+1}}$ be an updated sequence database that $\mathrm{SDB}_{k+1} = \mathrm{SDB_k}+\mathrm{ASDB}_k$. When original sequence database has changed, the algorithm must make full use of the previously discovered information to adapt with the change. This frequent sequences set (FSS) is transform from original sequence database without distortion, $\mathrm{FSS}_k=f(\mathrm{SDB}_k)$. When new sequence, ASDB, has been appended to original sequence database the data model also change dependently. $\mathrm{FSS}_{k+1} = f(\mathrm{SDB}_k)+ f(\mathrm{ASDB}_k)$.

Definition 1 Closed Sequential Patterns (CSP) are frequent sequence has no Supersequence in same equivalent class (EC).

$$CSP = \{s \mid s \in EC \ and\ \neg\exists s' \in EC \ such\ that\ \ s \prec s'\}$$

Definition 2 Frequent Closed Sequential Patterns are closed sequential patterns that satisfy the threshold of minimum support.

3. Incremental Sequence Database

In incremental sequence database the absolute support of some sequences will increase also. Hence, the frequent sequence set (FSS) will be altered. We can regard frequent sequence set as a function, mining algorithm, of SDB and predefined fixed minimum support (minsupp).

$$FSS_k = f(SDB_k,\ minsupp_k)$$

So, when sequence database SDB has been changed from SDB_k to SDB_{k+1} the frequent sequence set changed accordingly. Let $ASDB_k$ be a sequence database to be added to original sequence database. The most up-to-date sequence database SDB_{k+1} is derived from SDB_k and $ASDB_k$.

$$SDB_{k+1} = SDB_k + ASDB_k$$

To discover frequent sequence set from the most up-to-date sequence database, the most intuition way is to run the mining algorithm on it from scratch.

$$FSS_{k+1} = f(SDB_{k+1},\ minisupp_{k+1})$$

But this approach is very time consuming, didn't take the advantage of previous discovered frequent sequence set, $FSS_k = f(SDB_k,\ minisupp_k)$. Normally, the size of ASDB is much smaller than SDB.

$$|SDB| > |ASDB|$$

It is very inefficient to rerun the mining algorithm on original sequence database from scratch.

3.1 Linear Algorithm

For a linear algorithm, the updated frequent sequence set can be obtain from pervious discovered frequent sequence set and apply mining algorithm on appended sequence database separately.

$$FSS_{k+1} = f(SDB_k + ASDB)$$

For a linear mining algorithm the frequent sequence set can be obtained with merging of $f(SDB_k)$ and $f(ASDB_k)$.

$$FSS_{k+1} = merge(f(SDB_k), f(ASDB_k))$$

Hence, the advantage of previously work is taken. The only overhead is the effort of merging two frequent sequence sets. Merge is to deal with the mutual part of two frequent sequence sets.

3.2 Nonlinear Algorithm

A nonlinear mining algorithm can't process original sequence database and incremented sequence database separately via merging. The most up-to-date frequent sequence set is not equals to previous minded frequent sequence set merge with updated frequent sequence set.

$$FSS_{k+1} \neq merge(f(SDB_k), f(ASDB_k))$$

That is

$$FSS_{k+1} \neq merge(FSS_k + f(ASDB_k))$$

The transfer function is the algorithm of finding frequent sequences from sequence database SDB, such as apriori-like algorithms, FP-Tree, FMMSP and etc.

4. Adjustable Minimum Support

The minimum support measure plays a major role in sequential patterns mining algorithm. Lacking the knowledge of the sequences and their frequency, knowledge worker defined minimum support measures are often inappropriate. Especially when applying an algorithm that must predefine the minimum support in advance.

Most previous work has focus on single minimum support threshold. Few had discussed about multilevel minimum support threshold.

$$FSS_k = f(minsupp_k,\ SDB_k)$$

$FSS_{k+1} = f(minsupp_{k+1}, SDB_k)$, where $minsupp_{k+1} \neq minsupp_k$.

In this paper we propose a new algorithm, IMSP, to alleviate this problem via mining frequent sequences in a form of maximal sequential pattern, rather than mining the full set of frequent sequences. The reason why we mine maximal sequential patterns is that they are compact representations of frequent sequential patterns.

5. IMSP and Merge Algorithm

IMSP provides a categorized, in frequency, data model represent original sequence database without distortion. With a approach of incremental strategy, sequences of D are read one by one, transformed and load into the data model, Frequent Sequences Set (FSS). Sequence S read from database is compared to the existed sequences in the FSS. Comparison is in descending order in each array, but in ascending order from F1 to higher frequent sequence array. The relationship between S and SFSS are $S \cap S_{FSS} = \phi$, $S \cap S_{FSS} = S$, $S \cap S_{FSS} = S_{FSS}$ or S and SFSS are partially mutual to each other. That is mutual sequence $S_m = S \cap S_{FSS} \neq \phi$ and $S \neq S_{FSS}$. The new sequence S is processed according to the type of relationship. The first case is simple; we just append the new sequence S to the array of F1. In case 2 and 3, the mutual part is upgraded to higher frequent array. In case 4, S is upgraded to higher frequent array. Each frequent array contains maximal sequences only. For example, sequence 〈ABC〉 and 〈C〉 will not coexist in the same array because 〈ABC〉 is the maximal sequence of 〈C〉.

```
//Input: sequence database, SDB
//Output: FSS
Initial 2-dimension array FSS={ F1, F2, …}
Function Upgrade(S){
   move S from allocated array to higher frequent array }
For each sequence in SDB{
   Read sequence S from sequence database D
   For n= 1 to Top{
         Case S$\cap$ each sequence in $F_n = \phi$ :{
            append S to Fn ;
            break;
            }
         Case   S$\subset$ Fn.sequence   :{
            Upgrade(S);
            mark S;
            }
         Case   S$\supset$ Fn.sequence:{
            Upgrade(Fn.sequence);
            Mark Fn.sequence
            }
         Case   S$\cap$ Fn.sequence=Smutual :{
            Upgrade(Smutual);
            S= Smutual;
            }
   }
}
```

Fig. 1 IMSP algorithm

```
Input: 2 frequent Sequence Sets FSS1 and FSS2
Output: Merged Frequent Sequent Set
If   |FSS1|>|FSS2|
      For each frequent array FSS.Fn{
            Append sequences S from FSS2.Fn to FSS1.Fn
            IMSP(S)}
Return FSS1 as most up-to-date frequent Sequence Set
If   |FSS1|<|FSS2|
      For each frequent array FSS.Fn{
            Append sequences S from FSS1.Fn to FSS2.Fn
            IMSP(S)}
Return FSS1 as most up-to-date frequent Sequence Set
```

Fig.2 MergeFSS algorithm

6. Example and Performance Analysis

In figure 2 is a simple sequence database, SDB. SID represents Student Identifier. The itemset I={A,B,C,D,E}. The incremental sequence database is represented by ASDB which contains sequences of items.

Let SDB has kept 5 tuples of sequences depicted in Fig 3.

SID	Sequence
1	〈ACD〉
2	〈ABCE〉
3	〈BCE〉

(SDB)

SID	Sequence
4	〈BE〉
5	〈ABCDE〉

(ASDB)

Fig.3 original sequence database SDB and new appended sequence database ASDB

The first sequence 〈ACD〉 from database SDB is reside at Frequent-1 set (F1) depict in Figure.4. For all sequence from SDB are compared with sequences in F1. Since there is no other sequence in F1 the comparison process stops after allocate 〈ACD〉 to F1.

F1	F2	F3	…	Fn
〈ACD〉				

Fig. 4 FSS(SDB) containing 〈ACD〉

Next sequence 〈ABCE〉 is compared with all sequences in F1. The mutual sequence of 〈ACD〉 and 〈ABCE〉 is 〈AC〉 which will be upgraded to higher frequent set F2. As depicted in Fig 5.

F1	F2	…	Fn
〈ACD〉	〈AC〉		
〈ABCE〉			

Fig. 5 FSS with 〈ACD〉 and 〈ABCE〉 and their mutual sequence 〈AC〉

Since sequence 〈BCE〉 is contained by 〈ABCE〉 of F1, so 〈BCE〉 is upgraded to F2. Sequence 〈BCE〉 is a new sequence to F2. In F2, The mutual sequence of new comer 〈BCE〉 and 〈AC〉 is 〈C〉 that will be upgraded to F3. In F3, the empty set has no sequences to compare to. As depicted in Figure.6.

F1	F2	F3	…
〈ACD〉	〈AC〉	〈C〉	
〈ABCE〉	〈BCE〉		

Fig. 6 example of upgrading mutual sequence

Apply IMSP to appended sequence database ASDB. The frequent sequence set, f(ASDB), is depicted in Fig. 7.

F1	F2	…	Fn
〈ABCDE〉	〈BE〉		

Fig.7 frequent sequent set f(ASDB)

Next is applying Merge Algorithm to these two frequent sequence sets. Sequences of f(ASDB) are appended to f(SDB) respectively. Compare sequences in descending order to merge each sequence into most-up-date frequent sequence set. The process is demonstrated in Fig.8.

F1	F2	F3	…
〈ACD〉	〈AC〉	〈C〉	
〈ABCE〉	〈BCE〉		
〈ABCDE〉	〈BE〉		

Merge FSS(SDB) with FSS(ASDB)

F1	F2	F3	…
〈ACD〉	〈AC〉	〈C〉	
〈ABCE〉	〈BCE〉	〈BE〉	
〈ABCDE〉			

upgrade <BE> to F3

F1	F2	F3	…
〈ABCE〉	〈AC〉	〈C〉	
〈ABCDE〉	〈BCE〉	〈BE〉	
	〈ACD〉		

Upgrade <ACD> to F2

F1	F2	F3	…
〈ABCE〉	〈BCE〉	〈C〉	
〈ABCDE〉	〈ACD〉	〈BE〉	
		〈AC〉	

Upgrade <AC> to F3

F1	F2	F3	F4	…
〈ABCE〉	〈BCE〉	〈BE〉	〈C〉	
〈ABCDE〉	〈ACD〉	〈AC〉		

Upgrade <C> to F4

F1	F2	F3	F4	…
〈ABCDE〉	〈BCE〉	〈BE〉	〈C〉	
	〈ACD〉	〈AC〉		
	〈ABCE〉			

upgrade <ABCE> to F2

F1	F2	F3	F4	…
〈ABCDE〉	〈ACD〉	〈BE〉	〈C〉	
	〈ABCE〉	〈AC〉		
		〈BCE〉		

upgrade <BCE> to F3

F1	F2	F3	F4	…
〈ABCDE〉	〈ACD〉	〈AC〉	〈C〉	
	〈ABCE〉	〈BCE〉	〈BE〉	

upgrade <BE> to F4

Fig.9 Formation of updated frequent sequent set

The updated frequent sequence set is also known as searching space which is represented by closed sequences. The threshold of minimum support is adjustable in our algorithm. Let minsupp=3 then frequent sequences are easy be classified from data model. Furthermore, the threshold could be two values to set up a range of frequent sequences. Such as setting upper bound minsuppup=4, lower bound minsupplow=2. Setting threshold range is to discover sequential patterns that are not frequent enough to regard as strong rules.

7. Conclusions and future works

With the result of performance analysis with examples, the accomplishment include: 1)full set of frequent sequences set is discovered, 2) our algorithm is linear which means it is maintainable for incremental sequence database, 3)adjustable minimum support, 4) compact searching space and 5) no candidates are generated.

In this paper, we proposed IMSP, a novel algorithm for mining frequent maximal sequential sequences. It has alleviate the drawbacks of the candidate maintenance-and-test paradigm, construct more compact searching space compare to the previously developed maximal pattern mining algorithms and an adjustable minimum support.

The future challenges are to deal with dynamic sequence database, not only adding new sequences but also deleting obsolete or impropriate sequences.

References

[1] R. Agrawal and R. Srikant. "Mining sequential patterns", In Proc. 1995 Int. Conf. Data Engineering (ICDE'95), pages 3–14, Taipei, Taiwan, Mar. 1995.

[2] C. Lucchese, S. Orlando and R. Perego, "Fast and Memory Efficient Mining of Frequent Closed Itemsets", IEEE Transactions on Knowledge and Data Engineering, Vol. 18, No. 1, January 2006.

[3] P. Songram, V. Boonijin and S. Intakosum, "Closed Multidimensional Sequential Pattern Mining", Proceeding of the Third Conference on Information Technology: New Generations (ITNG'06).

[4] Nancy P. Lin, Wei-Hua Hao and Hung-Jen Chen, "Fast Accumulation Lattice Algorithm for Mining Sequential Patterns", Proceedings of the 6th WSEAS International Conference on Applied Computer Science (ACOS'07), pp. 230-234, Hangzhou, China, April 15-17, 2007.

[5] Nancy P. Lin, Wei-Hua Hao, Hung-Jen Chen, Hao-En Chueh and Chung-I Chang, "Fast Mining Sequential Patterns", Proceedings of the 7th WSEAS International Conference on Simulation, Modeling and Optimization Applied Computer Science (SMO '07), pp. 405-408,Beijing ,China ,September 15-17, 2007.

[6] R. Srikant and R. Agrawal, "Mining Sequential Patterns: Generalizations and Performance Improvements," Proc. Fifth Int'l Conf. Extending Database Technology (EDBT '96), pp. 3-17, Mar. 1996.

[7] M. Garofalakis, R. Rastogi, K. Shim, "SPIRIT: Sequential pattern mining with regular expression constraints," Proceedings of the 25th International Conference on Very Large Databases (VLDB'99), pp. 223-234, 1999.

[8] Jian Pei, Jiawei Han, Behzad Mortazavi-Asl, Janyong Wang, Helen Pinto, Qiming Chen, Umeshwar Dayal, Mei-Chun Hsu, "Mining Sequential Patterns by Pattern-Growth: The PrefixSpan Approach", IEEE Transactions on Knowledge and Data Engineering, vol. 16, No. 11, November 2004.

[9] J. Han, J. Pri, B. Mortazavi-Asl, Q. Chen, U. Dayal, and M.-C. Hsu, "FreeSpan: Frequent Pattern-Projected Sequential Pattern Mining", Proc. 2000 ACM SIGKDD Int'l Conf. Knowledge Discovery in Database (KDD '00), pp. 355-359, Aug. 2000.

[10] J. Han, J. Pei and Y. Yin, "Mining Frequent Patterns without Candidate Generation", Proc. 2000 ACM-SIGMOD Int'l Conf. Management of Data (SIGMOD '00), pp.1-12, May 2000.

[11] Wang, J., Han, J. a, "BIDE: efficient mining of frequent closed sequences", Data Engineering, 2004. Proceedings. 20th International Conference on 30 March-2 April 2004 Page(s):79 – 90.

[12] J. Wang, J. Han, and J. Pei, "CLOSET+: Searching for the Best Strategies for Mining Frequent Closed Itemsets", KDD'03, Washington, DC, Aug. 2003.

[13] X. Yan, J. Han, and R. Afshar, " CloSpan: Mining Closed Sequential Patterns in Large Databases". SDM'03, San Francisco, CA, May 2003.

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

[15] F. Masseglia, P. Poncelet, M. Teisseire, "Incremental mining of sequential patterns in large databases," Actes des Jouenes Bases de Donnes Avances (BDA'00), Blois, France, 1999.

[16] Weimin Ouyang, Qingsheng Cai, "An incremental updating techniques for discovering generalized sequential patterns," Journal of Software, Vol. 9, No. 10, pp. 778-780, 1998.