Expert Systems with Applications 38 (2011) 7051–7058

Contents lists available at ScienceDirect

Expert Systems with Applications

journal homepage: www.elsevier.com/locate/eswa

An incremental mining algorithm for maintaining sequential patterns using pre-large sequences

Tzung-Pei Hong [a,b,*], Ching-Yao Wang [c], Shian-Shyong Tseng [d]

[a] Department of Computer Science and Information Engineering, National University of Kaohsiung, Kaohsiung 811, Taiwan
[b] Department of Computer Science and Engineering, National Sun Yat-sen University, Kaohsiung 804, Taiwan
[c] Information and Communications Research Laboratories, Industrial Technology Research Institute, Hsinchu, Taiwan
[d] Institute of Computer and Information Science, National Chiao-Tung University, Hsinchu 300, Taiwan

ARTICLE INFO

Keywords:
Data mining
Sequential pattern
Large sequence
Pre-large sequence
Incremental mining

ABSTRACT

Mining useful information and helpful knowledge from large databases has evolved into an important research area in recent years. Among the classes of knowledge derived, finding sequential patterns in temporal transaction databases is very important since it can help model customer behavior. In the past, researchers usually assumed databases were static to simplify data-mining problems. In real-world applications, new transactions may be added into databases frequently. Designing an efficient and effective mining algorithm that can maintain sequential patterns as a database grows is thus important. In this paper, we propose a novel incremental mining algorithm for maintaining sequential patterns based on the concept of pre-large sequences to reduce the need for rescanning original databases. Pre-large sequences are defined by a lower support threshold and an upper support threshold that act as gaps to avoid the movements of sequences directly from large to small and vice versa. The proposed algorithm does not require rescanning original databases until the accumulative amount of newly added customer sequences exceeds a safety bound, which depends on database size. Thus, as databases grow larger, the numbers of new transactions allowed before database rescanning is required also grow. The proposed approach thus becomes increasingly efficient as databases grow.

1. Introduction

The rapid development of computer technology, especially increased capacities and decreased costs of storage media, has led businesses to store huge amounts of external and internal information in large databases at low cost. Mining useful information and helpful knowledge from these large databases has thus evolved into an important research area (Agrawal, Imielinksi, & Swami, 1993; Chen, Han, & Yu, 1996). Years of effort in data mining has produced a variety of efficient techniques. Depending on the types of databases to be processed, mining approaches may be classified as working on transactional databases, temporal databases, relational databases, and multimedia databases, among others. Depending on the classes of knowledge sought, mining approaches may be classified as finding association rules, classification rules, clustering rules, and sequential patterns, among others (Chen et al., 1996). Among them, finding sequential patterns in temporal transaction databases is important since it allows modeling of customer behavior (Agrawal & Srikant, 1995; Lin & Lee, 1998; Srikant & Agrawal, 1995).

Mining sequential patterns was first proposed by Agrawal and Srikant (1995), and is a non-trivial task. It attempts to find customer purchase sequences and to predict whether there is a high probability that when customers buy some products, they will buy some other products in later transactions. For example, a sequential pattern for a video shop may be formed when a customer buys a television in one transaction, he then buys a video recorder in a later transaction. Note that the transaction sequences need not be consecutive.

Although customer behavior models can be efficiently extracted by the mining algorithm in Agrawal and Srikant (1995) to assist managers in making correct and effective decisions, the sequential patterns discovered may become invalid when new customer sequences occur. Sequential patterns that did not originally exist may emerge due to these new customer sequences. Conventional approaches may re-mine the entire database to get correct sequential patterns for maintenance. However, when the database is massive in size, this will require considerable computation time. Developing efficient approaches to maintain sequential patterns is thus very important to real-world applications. Recently, some researchers have strived to develop incremental mining algorithms

* Corresponding author at: Department of Computer Science and Information Engineering, National University of Kaohsiung, Kaohsiung 811, Taiwan.
E-mail addresses: tphong@nuk.edu.tw (T.-P. Hong), cywang@cis.nctu.edu.tw (C.-Y. Wang), sstseng@cis.nctu.edu.tw (S.-S. Tseng).

0957-4174/$ - see front matter © 2010 Elsevier Ltd. All rights reserved.
doi:10.1016/j.eswa.2010.12.008

for maintaining association rules. Examples are the FUP algorithm proposed by Cheung, Han, Ng, and Wong (1996, 1997), the adaptive algorithm proposed by Sarda and Srinivas (1998), and the incremental mining algorithm with pre-large itemsets proposed by Hong, Wang, and Tao (2000, 2001). The common idea in these approaches is that previously mined information should be utilized as much as possible to reduce maintenance costs. Intermediate results, such as large itemsets, are kept and checked against newly added transactions, thus saving much computation time for maintenance, although original databases may still need to be rescanned.

Studies on maintaining sequential patterns are relatively rare compared to those on maintaining association rules. Lin and Lee proposed the FASTUP algorithm to maintain sequential patterns by extending the FUP algorithm (Lin & Lee, 1998). Their approach works well except when newly coming candidate sequences are not large in the original database. If this occurs frequently, the performance of the FASTUP algorithm will correspondingly decrease. In this paper, we thus attempt to develop a novel and efficient incremental mining algorithm capable of updating sequential patterns based on the concept of pre-large sequences. A pre-large sequence is not truly large, but nearly large. A lower support threshold and an upper support threshold are used to realize this concept. Pre-large sequences act like buffers and are used to reduce the movement of sequences directly from large to small and vice versa during the incremental mining process. A safety bound for newly added customer sequences is derived within which rescanning the original database can be efficiently reduced and maintenance costs can also be greatly reduced. The safety bound also increases monotonically along with increases in database size. Thus, our proposed algorithm becomes increasingly efficient as the database grows. This characteristic is especially useful for real-world applications.

The remainder of this paper is organized as follows. Mining sequential patterns is first reviewed in Section 2. The maintenance of association rules is reviewed in Section 3. The concept of pre-large sequences is described in Section 4. The notation used in this paper is defined in Section 5. The theoretical foundation for our approach is given in Section 6. A novel maintenance algorithm for sequential patterns is proposed in Section 7. An example to illustrate the proposed algorithm is provided in Section 8. Conclusions are given in Section 9.

2. Mining sequential patterns

In a database D of customer transactions, each transaction consists of at least three attributes, *Cust_id*, *Trans_time* and *Trans_content*. *Cust_id* records the unique identification of a customer, *Trans_time* stores the time a transaction occurs, and *Trans_content* stores what items were purchased in a transaction. A *sequence* is an ordered list of itemsets. A *customer sequence* is a sequence of all transactions for a customer in order of transaction times. Note that each transaction in a customer sequence corresponds to an itemset. A sequence A is contained in a sequence B if the former is a sub-sequence of the latter. Take the data in Table 1 as an example.

There are sixteen transactions sorted first by *Cust_id* and then by *Trans_time*. These transactions can be transformed into customer sequences as shown in Table 2.

Tuples 1 and 2 in Table 1 both belong to customer 1 and are thus combined into a customer sequence in Table 2. Similarly, tuples 3, 4 and 5 belonging to customer 2 are combined into another customer sequence. Note that the sequence $\langle(C,D)\rangle$ in customer sequence 2 indicates that a customer bought items C and D in one transaction. This differs from the sequence $\langle(C)(D)\rangle$, which means that a customer bought item C in a transaction and then bought item D in a later transaction. Customer sequence 1, $\langle(A)(B)\rangle$, is contained in customer sequence 4, which is $\langle(A)(E,G)(B)\rangle$, since the former is a sub-sequence of the latter. Note that, if customer sequence 1 is $\langle(A)(E,B)\rangle$, then it is not contained in customer sequence 4.

Table 1
Sixteen transactions sorted according to Cust_id and Trans_time.

Cust_id	Trans_time	Trans_content
1	1998/01/01	A
1	1998/01/20	B
2	1998/01/11	C, D
2	1998/02/02	A
2	1998/02/11	E, F, G
3	1998/01/07	A, H, G
4	1998/02/09	A
4	1998/02/19	E, G
4	1998/02/23	B
5	1998/01/05	B
5	1998/01/12	C
6	1998/01/05	A
6	1998/01/13	B, C
7	1998/01/01	A
7	1998/01/17	B, C, D
8	1998/01/23	E, G

Table 2
The customer sequences transformed from the transactions in Table 1.

Cust_id	Customer sequence
1	⟨(A)(B)⟩
2	⟨(C,D)(A)(E,F,G)⟩
3	⟨(A,H,G)⟩
4	⟨(A)(E,G)(B)⟩
5	⟨(B)(C)⟩
6	⟨(A)(B,C)
7	⟨(A)(B,C,D)⟩
8	⟨(E,G)⟩

Agrawal and Srikant proposed the AprioriAll approach to mining sequential patterns from sets of transactions (Agrawal & Srikant, 1995). Five phases are included in this approach. In the first phase, transactions are sorted first using customer ID as the major key and then using transaction time as the minor key. This phase thus converts the original transactions into customer sequences. In the second phase, large itemsets are found in customer sequences by comparing their counts with the predefined support parameter α. This phase is similar to the process of mining association rules. Note that when an itemset occurs more than once in a customer sequence, it is counted once for this customer sequence. In the third phase, large itemsets are mapped to contiguous integers and the original customer sequences are transferred to the mapped integer sequences. In the fourth phase, the integer sequences are examined for finding large sequences. In the fifth phase, maximally large sequences are then derived and output to users.

Mining sequential patterns repeatedly and level-wisely executes series of operations on customer sequences similar to the mechanism for mining association rules (Agrawal, Imielinksi, & Swami, 1993; Agrawal & Srikant, 1994; Agrawal, Srikant, & Vu, 1997; Fukuda, Morimoto, Morishita, & Tokuyama, 1996; Han & Fu, 1995; Mannila, Toivonen, & Verkamo, 1994, 1997; Savasere, Omiecinski, & Navathe, 1995). However, association rules concern relationships among items in transactions, while sequential patterns concern relationships among itemsets in customer sequences. Consider, for example, the customer sequences shown in Table 2. Assume the minimum support is set at 50% (i.e., three

Table 3
All large sequences generated for the customer sequences in Table 2.

Large sequences			
1-sequence	Count	2-sequence	Count
⟨(A)⟩	6	⟨(A)(B)⟩	4
⟨(B)⟩	5		
⟨(C)⟩	4		
⟨(G)⟩	4		

customer sequences for this example). All the large sequences mined from the customer sequences in Table 2 are presented in Table 3.

3. Maintenance of association rules

In this section, we review some methods for maintaining association rules, which provide some hints to the maintenance of sequential patterns introduced in the next section. When new transactions are added to databases, existing association rules may become invalid, or new implicitly valid rules may appear in the resulting updated databases (Cheung et al., 1996, 1997; Hong et al., 2000, 2001; Sarda & Srinivas, 1998; Zhang, 1999). In these situations, conventional batch-mining algorithms must re-mine the entire updated databases to find all up-to-date association rules. Two drawbacks are associated with maintaining database knowledge in this manner:

(a) Nearly the same computation time is needed to cope with each new transaction as was spent in mining from the original database. If the original database is large, much computation time is wasted in maintenance whenever new transactions are generated.
(b) Information previously mined from the original database, such as large itemsets, provides no help in the maintenance process.

Some approaches have been proposed to use previously mined information to improve rule-maintenance performance. Cheung et al. first proposed the concept of intermediate information and designed an incremental mining algorithm, called *FUP* (Cheung et al., 1996, 1997), to solve the problems stated above. The FUP algorithm retains previously discovered large itemsets as intermediate information during each run. It then scans the newly added transactions to generate candidate 1-itemsets (only for these transactions), and compares these itemsets with ones previously retained in the intermediate information. FUP partitions candidate 1-itemsets into two parts according to whether they are large in the original database. If a candidate 1-itemset is among the large 1-itemsets from the original database, its new total count for the entire updated database can easily be calculated from its current count and previous count since FUP retains all previous large itemsets with their counts. By contrast, if a candidate 1-itemset is not among the large 1-itemsets in the original database, it is treated in one of two ways. If a candidate 1-itemset is not large for the new transactions, it cannot be large for the entire updated database, which means no action is necessary. Additionally, if a candidate 1-itemset is large for the new transactions, the original database must be re-scanned to determine whether the itemset is actually large for the entire updated database. Using the processing tactics mentioned above, FUP is thus able to find all large 1-itemsets for the entire updated database. After that, candidate 2-itemsets from the newly inserted transactions are formed and the same procedure is used to find all large 2-itemsets. This procedure is repeated until all large itemsets have been found.

The FUP algorithm thus focuses on newly added transactions and utilizes intermediate information to save computation time in maintaining association rules. It must, however, rescan the original database to handle cases in which candidate itemsets are large in newly added transactions and not retained in the intermediate information. This situation may occur frequently, especially when the number of new transactions is small.

We proposed an incremental mining algorithm based on the concept of *pre-large itemsets* to reduce the amount of rescanning of original databases required whenever new transactions are added (Hong et al., 2000, 2001). A pre-large itemset is not truly large, but promises to be large in the future. A lower support threshold and an upper support threshold are used to realize this concept. The upper support threshold is the same as the minimum support used in conventional mining algorithms. The support ratio of an itemset must be larger than the upper support threshold in order to be considered large. On the other hand, the lower support threshold defines the lowest support ratio for an itemset to be treated as pre-large. An itemset with a support ratio below the lower threshold is thought of as a small itemset. Pre-large itemsets act like buffers and are used to reduce the movement of itemsets directly from large to small and vice versa in the incremental mining process. Therefore, when few new transactions are added, the original small itemsets will at most become pre-large and cannot become large, thus reducing the amount of rescanning necessary. A safety bound for new transactions is derived from the upper and lower thresholds and from the size of the database. This algorithm is described as follows:

Step 1: Retain all previously discovered large and pre-large itemsets with their counts.
Step 2: Scan newly inserted transactions to generate candidate 1-itemsets with counts.
Step 3: Set $k = 1$, where k is used to record the number of items currently being processed.
Step 4: Partition all candidate k-itemsets as follows.
Case 1: A candidate k-itemset is among the previous large 1-itemsets.
Case 2: A candidate k-itemset is among the previous pre-large itemsets.
Case 3: A candidate k-itemset is among the original small itemsets.
Step 5: Calculate a new count for each itemset in cases 1 and 2 by adding its current count and previous count together; prune the itemsets with new support ratios smaller than the lower support threshold.
Step 6: Rescan the original database if the accumulative amount of new transactions exceeds the safety threshold.
Step 7: Generate candidate $k + 1$-itemsets from updated large and pre-large k-itemsets, and then go to step 3 until they are null.

The above algorithm, like the FUP algorithm, retains previously mined information, focuses on newly added transactions, and further reduces the computation time required to maintain large itemsets in the entire database. The algorithm can further reduce the number of rescans of the original database as long as the accumulative amount of new transactions does not exceed the safety bound.

4. Extending the concept of pre-large itemsets to sequential patterns

Maintaining sequential patterns is much harder than maintaining association rules since the former must consider both itemsets

and sequences. In this paper, we attempt to extend the concept of pre-large itemsets to maintenance of sequential patterns. The pre-large concept is used here to postpone original small sequences directly becoming large and vice versa when new transactions are added. A safety bound derived from the lower and upper thresholds determines when rescanning the original database is needed.

When new transactions are added to a database, they can be divided into two classes:

Class 1: new transactions by old customers already in the original database;
Class 2: new transactions by new customers not already in the original database.

Newly added transactions are first transformed into customer sequences, and those belonging to class 1 mentioned above are merged with corresponding customer sequences in the original database. For example, assume that the original database includes the six customer sequences shown in Table 2 and the large sequences found from them are presented in Table 3 with the minimum support set at 50%. When the two new transactions shown in Table 4 are added to the original database, they are first transformed into the new customer sequences shown in Table 5, and then merged with the original customer sequences. Results are shown in Table 6.

The candidate sequences from the newly merged customer sequences are then listed and counted. Comparing the merged customer sequences with the corresponding old customer sequences, it is easily seen that the itemsets in new customer sequences will always be appended to the old customer sequences. Therefore, the counts of the sequences existing in the old customer sequences are not changed and only the counts of the sequences derived from the new customer sequences increase. The counts of the candidate sequences from the merged customer sequences are then defined by their count increments for insertion of new transactions. For example, the candidate 1-sequences for the merged customer sequences in Table 6 are shown in Table 7. Their counts as calculated in the manner described above are also shown there.

Considering the old customer sequences in terms of the two support thresholds, the newly merged customer sequences may fall into the following three cases illustrated in Fig. 1.

Case 1 may remove existing large sequences, and cases 2 and 3 may add new large sequences. If we retain all large and pre-large sequences with their counts in the original database, then cases 1 and 2 can be easily handled. Also, in the maintenance phase, the ratio of newly added customer sequences to original customer sequences is usually very small. This is more apparent when the database is growing larger. A sequence in case 3 cannot possibly be large for the entire updated database as long as the number of newly added customer sequences is small compared to the number of customer sequences in the original database. This point is proven below.

Table 4
Two new transactions sorted according to *Cust_id* and *Trans_time*.

Cust_id	Trans_time	Trans_content
5	1998/02/01	E, G
9	1998/02/05	E, F, G

Table 5
Two newly added customer sequences.

Cust_id	Customer sequence
5	⟨(E,G)⟩
9	⟨(E,F,G)⟩

Table 6
The two merged customer sequences.

Cust_id	Customer sequence
5	⟨(B)(C)(E,G)⟩
9	⟨(E,F,G)⟩

Table 7
The candidate 1-sequences with their counts.

Candidate 1-sequences	
1-sequence	Count
⟨(B)⟩	0
⟨(C)⟩	0
⟨(E)⟩	2
⟨(F)⟩	1
⟨(G)⟩	2
⟨(E,F)⟩	1
⟨(E,G)⟩	2
⟨(F,G)⟩	1
⟨(E,F,G)⟩	1

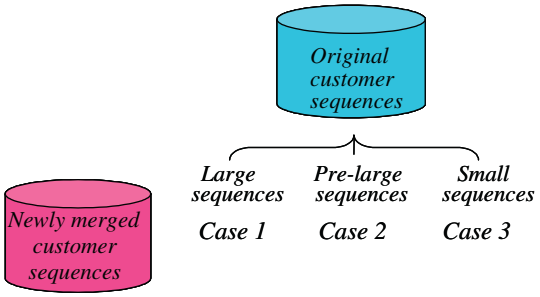

Fig. 1. Three cases arising from adding new transactions to existing databases.

5. Notation

The notation used in this paper is defined below.

D: the original customer sequences;
T: the set of newly merged customer sequences from the newly inserted customer sequences;
U: the entire updated customer sequences;
d: the number of customer sequences in D;
t: the number of customer sequences in T;
q: the number of newly added customer sequences belonging to old customers in the original database;
S_u: *the upper support threshold for large sequences;*
S_l: *the lower support threshold for pre-large sequences,* $S_l < S_u$;
L_k^D: *the set of large k-sequences from D;*
L_k^T: *the set of large k-sequences from T;*
L_k^U: *the set of large k-sequences from U;*
P_k^D: *the set of pre-large k-sequences from D;*
P_k^T: *the set of pre-large k-sequences from T;*
P_k^U: *the set of pre-large k-sequences from U;*
C_k: *the set of all candidate k-sequences from T;*
I: a sequence;
$S^D(I)$: *the number of occurrences of I in D;*
$S^T(I)$: *the number of occurrence increments of I in T;*
$S^U(I)$: *the number of occurrences of I in U.*

6. Theoretical foundation

As mentioned above, if the number of newly added customer sequences is small when compared with the number of original customer sequences, a sequence that is small (neither large nor pre-large) in the original database cannot possibly be large for the entire updated database. This is demonstrated by the following theorem.

Theorem 1. *Let S_l and S_u be, respectively, the lower and the upper support thresholds, d be the number of customer sequences in the original database, t be the number of newly added customer sequences, and q be the number of newly added customer sequences belonging to old customers. If $t \leqslant \frac{(S_u-S_l)d}{1-S_u} - \frac{qS_u}{1-S_u}$, then a sequence that is small (neither large nor pre-large) in the original database is not large for the entire updated database.*

Proof. The following derivation can be obtained from $t \leqslant \frac{(S_u-S_l)d}{1-S_u} - \frac{qS_u}{1-S_u}$:

$$t \leqslant \frac{(S_u - S_l)d}{1 - S_u} - \frac{qS_u}{1 - S_u} \quad (1)$$

$$\Rightarrow \quad t(1 - S_u) \leqslant (S_u - S_l)d - qS_u$$

$$\Rightarrow \quad t - tS_u \leqslant dS_u - dS_l - qS_u$$

$$\Rightarrow \quad t + dS_l \leqslant S_u(d + t - q)$$

$$\Rightarrow \quad \frac{t + dS_l}{d + t - q} \leqslant S_u.$$

If a sequence I is small (neither large nor pre-large) in D, then its count $S^D(I)$ must be less than $S_l * d$. Therefore,

$$S^D(I) < dS_l.$$

Since the number of newly added customer sequences is t, the count of I in T is at most t. Thus:

$$t \geqslant S^T(I).$$

The entire support ratio of I in U is $\frac{S^U(I)}{d+t-q}$, which can be further expanded to:

$$\frac{S^U(I)}{d + t - q} = \frac{S^T(I) + S^D(I)}{d + t - q} < \frac{t + dS_l}{d + t - q} \leqslant S_u.$$

I is thus not large for U. This completes the proof. □

Example 1. Assume d = 100, S_l = 50% and S_u = 60%. The number of newly added customer sequences within which the original database need not be re-scanned for rule maintenance is:

(a) If q = 10,

$$\frac{(S_u - S_l)d}{1 - S_u} - \frac{qS_u}{1 - S_u} = \frac{(0.6 - 0.5)100}{1 - 0.6} - \frac{0.6 * 10}{1 - 0.6} = 10.$$

(b) If q = 5,

$$\frac{(S_u - S_l)d}{1 - S_u} - \frac{qS_u}{1 - S_u} = \frac{(0.6 - 0.5)100}{1 - 0.6} - \frac{0.6 * 5}{1 - 0.6} = 17.5.$$

(c) If q = 0,

$$\frac{(S_u - S_l)d}{1 - S_u} - \frac{qS_u}{1 - S_u} = \frac{(0.6 - 0.5)100}{1 - 0.6} - \frac{0.6 * 0}{1 - 0.6} = 25.$$

If t is equal to or less than the number calculated, then I cannot be large for the entire updated database.

From Theorem 1, the bound of the number of new customer sequences is determined by S_l, S_u, d and q. It is easily seen from the first term, $\frac{(S_u-S_l)d}{1-S_u}$ in Formula 1, that if d grows larger, then t will grow larger too. As the database grows, our proposed approach thus becomes increasingly efficient. Also, from the second term, $\frac{qS_u}{1-S_u}$ in Formula 1, when the number of newly added customer sequences belonging to old customers in the original database is large (i.e., q is large), the allowable number of newly added customer sequences t will be reduced.

7. The proposed algorithm

In the proposed algorithm, the original large and pre-large sequences with their counts from preceding runs are retained for later use in maintenance. As new transactions are added, the proposed algorithm first transforms them into new customer sequences and merges them with the corresponding old sequences existing in the original database. The newly merged customer sequences are then scanned to generate candidate 1-sequences with occurrence increments. These candidate sequences are compared to the large and pre-large 1-sequences which were previously retained. These candidate sequences are divided into three parts according to whether they are large, pre-large or small in the original database. If a candidate 1-sequence is also among the previously retained large or pre-large 1-sequences, its new total count for the entire updated database can easily be calculated from its current count increment and previous count, since all previous large and pre-large sequences with their counts have been retained. Whether an original large or pre-large sequence is still large or pre-large after new transactions are added is then determined from its new support ratio, which is derived from its total count over the total number of customer sequences. On the contrary, if a candidate 1-sequence does not exist among the previously retained large or pre-large 1-sequences, then the sequence is absolutely not large for the entire updated database when the number of newly merged customer sequences is within the safety bound derived from Theorem 1. In this situation, no action is needed. When new transaction data are incrementally added and the total number of newly added customer sequences exceeds the safety bound, the original database must be re-scanned to find new large and pre-large sequences. The proposed algorithm can thus find all large 1-sequences for the entire updated database. After that, candidate 2-sequences from the newly merged customer sequences are formed, and the same procedure is used to find all large 2-sequences. This procedure is repeated until all large sequences have been found. The details of the proposed maintenance algorithm are described below. Two global variables, c and b, are used to accumulate, respectively, the number of newly added customer sequences and the number of newly added customer sequences belonging to old customers since the last re-scan of the original database.

7.1. The proposed maintenance algorithm for sequential patterns

Input: A lower support threshold S_l, an upper support threshold S_u, a set of large and pre-large sequences in the original database D consisting of $(d + c)$ customer sequences, the accumulative amount b of new customer sequences belonging to old customers, and a set of t newly added customer sequences transformed from new transactions.

Output: A set of final large sequential patterns for the updated database.

Step 1: Calculate the value of the first term in Formula 1 as:

$$f = \frac{(S_u - S_l)d}{1 - S_u}.$$

Step 2: Merge the newly added customer sequences with the old sequences in the original database and count the

value q, which is the number of the newly added customer sequences belonging to old customers.

Step 3: Set $b = b + q$ and calculate the second term $\frac{bS_u}{1-S_u}$ in Formula 1 as:

$$h = \frac{bS_u}{1 - S_u},$$

where b is the accumulative amount of q since the last re-scan.

Step 4: Set $k = 1$, where k is used to record the number of itemsets in the sequences currently being processed.

Step 5: Find all candidate k-sequences C_k and their count increments from the newly merged customer sequences T.

Step 6: Divide the candidate k-sequences into three parts according to whether they are large, pre-large or small in the original database.

Step 7: Do the following substeps for each k-sequence I in the original large k-sequences L_k^D:
Substep 7-1: Set the new count $S^U(I) = S^T(I) + S^D(I)$.
Substep 7-2: If $S^U(I)/(d + c + t - b) \geqslant S_u$, then assign I as a large sequence, set $S^D(I) = S^U(I)$ and keep I with $S^D(I)$;
otherwise, if $S^U(I)/(d + c + t - b) \geqslant S_l$, then assign I as a pre-large sequence, set $S^D(I) = S^U(I)$ and keep I with $S^D(I)$;
otherwise, ignore I.

Step 8: Do the following substeps for each k-sequence I in the original pre-large sequences P_k^D:
Substep 8-1: Set the new count $S^U(I) = S^T(I) + S^D(I)$.
Substep 8-2: If $S^U(I)/(d + c + t - b) \geqslant S_u$, then assign I as a large sequence, set $S^D(I) = S^U(I)$ and keep I with $S^D(I)$;
otherwise, if $S^U(I)/(d + c + t - b) \geqslant S_l$, then assign I as a pre-large sequence, set $S^D(I) = S^U(I)$ and keep I with $S^D(I)$;
otherwise, ignore I.

Step 9: Put I in the rescan-set R for each k-sequence I in the candidate k-sequences C_k that is neither in the original large sequences L_k^D nor in the pre-large sequences P_k^D, for use when rescanning in Step 10 is necessary.

Step 10: If $c + t \leqslant f - h$ or R is null, then do nothing; otherwise, rescan the original database to determine whether the sequences in the rescan-set R are large or pre-large.

Step 11: Form candidate $(k+1)$-sequences C_{k+1} from finally large and pre-large k-sequences $(L_k^U \bigcup P_k^U)$ that appear in the newly merged transactions.

Step 12: Set $k = k + 1$.

Step 13: Repeat STEPs 5 to 12 until no new large or pre-large sequences are found.

Step 14: Modify the maximal large sequence patterns according to the modified large sequences.

Step 15: If $c + t > f - h$, then set $d = d + c + t$, $c = 0$ and $b = 0$; otherwise, set $c = c + t$.

After Step 15, the finally maximal large sequences for the updated database can be determined.

8. An example

In this section, an example is given to illustrate the proposed maintenance algorithm for sequential patterns. Assume the initial customer sequences are the same as those shown in Table 2. Also assume S_l is set at 30% and S_u is set at 50%. The sets of large sequences and pre-large sequences for the given data are shown in Tables 8 and 9, respectively.

Assume the two new customer sequences shown in Table 10 are added.

Table 8
The large sequences for the customer sequences in Table 2.

Large sequences			
1-sequence	Count	2-sequence	Count
⟨(A)⟩	6	⟨(A)(B)⟩	4
⟨(B)⟩	5		
⟨(C)⟩	4		
⟨(G)⟩	4		

Table 9
The pre-large sequences for the customer sequences in Table 2.

Pre-large sequences			
1-sequence	Count	2-sequence	Count
⟨(E)⟩	3		
⟨(E,G)⟩	3		

Table 10
Two newly added customer sequences.

Cust_id	Customer sequence
5	⟨(E,G)⟩
9	⟨(A)(B,C)⟩

The global variables c and b are initially set at 0. The proposed incremental mining algorithm proceeds as follows:

Step 1: The value of the first term in Formula 1 is calculated as:

$$f = \frac{(S_u - S_l)d}{1 - S_u} = \frac{(0.5 - 0.3)8}{1 - 0.5} = 3.2.$$

Step 2: The two new sequences in Table 10 are merged with their corresponding old sequences in Table 2. Results are shown in Table 11.

Step 3: Since only the customer sequence with *Cust_id* = 5 in Table 10 belongs to old customers in Table 2, q is thus 1. $b = b + q = 0 + 1 = 1$. The value of the second term in Formula 1 is calculated as:

$$h = \frac{bS_u}{1 - S_u} = \frac{1 * 0.5}{1 - 0.5} = 1.$$

Step 4: k is set to 1, where k is used to record the number of itemsets in a sequence currently being processed.

Step 5: All candidate 1-sequences C_1 with their counts from the two merged customer sequences are found (and shown in Table 12). The counts are actually the count increments due to insertion of new transactions.

Step 6: All the candidate 1-itemsets in Table 12 are divided into three parts according to whether they are large, pre-large or small in the original database. Results are shown in Table 13.

Step 7: The following substeps are done for each of the originally large 1- sequences ⟨(A)⟩, ⟨(B)⟩, ⟨(C)⟩ and ⟨(G)⟩ (Table 8):

Table 11
The merged customer sequences.

Cust_id	Customer sequence
5	⟨(B)(C)(E,G)⟩
9	⟨(A)(B,C)⟩

Table 12
All candidate 1-sequences from the two merged customer sequences.

Candidate 1-sequences	Count
⟨(A)⟩	1
⟨(B)⟩	1
⟨(C)⟩	1
⟨(E)⟩	1
⟨(G)⟩	1
⟨(B,C)⟩	1
⟨(E,G)⟩	1

Table 13
Three partitions of all the candidate 1-itemsets in Table 12.

Originally large 1-sequences		Originally pre-large 1-sequences		Originally small 1-sequences	
1-sequence	Count	1-sequence	Count	1-sequence	Count
⟨(A)⟩	1	⟨(E)⟩	1	⟨(B,C)⟩	1
⟨(B)⟩	1	⟨(E,G)⟩	1		
⟨(C)⟩	1				
⟨(G)⟩	1				

Table 14
The total counts of ⟨(A)⟩, ⟨(B)⟩, ⟨(C)⟩ and ⟨(G)⟩.

1-sequence	Count
⟨(A)⟩	7
⟨(B)⟩	6
⟨(C)⟩	5
⟨(G)⟩	5

Substep 7-1: The total counts of the candidate 1-sequences ⟨(A)⟩, ⟨(B)⟩, ⟨(C)⟩ and ⟨(G)⟩ are calculated using $S^T(I) + S^D(I)$. Table 14 shows the results.
Substep 7-2: The new support ratios of ⟨(A)⟩, ⟨(B)⟩, ⟨(C)⟩ and ⟨(G)⟩ are calculated. For example, the new support ratio of {A} is $7/(8 + 0 + 2 - 1) = 0.78 \geqslant 0.5$. {A} is thus still a large sequence. In this example, since the new counts of all the four sequences ⟨(A)⟩, ⟨(B)⟩, ⟨(C)⟩ and ⟨(G)⟩ are larger than 0.5, they are large 1-sequences for the entire updated database.

Step 8: The following substeps are done for each of the originally pre-large 1-sequences ⟨(E)⟩ and ⟨(E,G)⟩ (Table 8):
Substep 8-1: The total count of the candidate 1-sequences ⟨(E)⟩ and ⟨(E,G)⟩ are calculated using $S^T(I) + S^D(I)$. Table 15 shows the results.
Substep 8-2: The new support ratio of ⟨(E)⟩ and ⟨(E,G)⟩ is $4/(8 + 0 + 2 - 1) = 0.43$. Since $0.3 < 0.43 < 0.5$, they are still retained in the pre-large 1-sequences.

Step 9: Since the sequence ⟨(B,C)⟩ is in the candidate 1-sequences and is neither originally large nor originally pre-large, it is then put into the rescan-set R for use if rescanning in Step 10 is necessary.

Step 10: Since $c + t = 0 + 2 \leqslant f - h$ (=2.2), rescanning the database is unnecessary and nothing is done.

Table 15
The total counts of ⟨(E)⟩and ⟨(E,G)⟩.

1-sequence	Count
⟨(E)⟩	4
⟨(E,G)⟩	4

Table 16
All candidate 2-itemsets appearing in the newly merged customer sequences.

Candidate 2-itemsets				
⟨(B)⟩⟨(C)⟩	⟨(B)⟩⟨(E)⟩	⟨(B)⟩⟨(G)⟩	⟨(B)⟩⟨(E,G)⟩	⟨(C)⟩⟨(E)⟩
⟨(C)⟩⟨(G)⟩	⟨(C)⟩⟨(E,G)⟩	⟨(A)⟩⟨(B)⟩	⟨(A)⟩⟨(C)⟩	

Table 17
All large and pre-large 2-sequences for the entire updated database.

Large 2-sequences		Pre-large 2-sequences	
Sequences	Count	Sequences	Count
⟨(A)(B)⟩	5		

Step 11: From Steps 7, 8 and 9, the final large and pre-large 1-sequences for the entire updated database are ⟨(A)⟩, ⟨(B)⟩, ⟨(C)⟩, ⟨(E)⟩, ⟨(G)⟩ and ⟨(E,G)⟩. All candidate 2-itemsets that appear in the newly merged customer sequences are shown in Table 16.

Step 12: $k = k+1 = 2$.

Step 13: Steps 5 to 12 are repeated to find large or pre-large 2-sequences. Results are shown in Table 17. Large or pre-large 3-sequences are found in the same way. No large 3-sequences were found in this example.

Step 14: The maximally large sequence derived from the large 2-sequence is:

$$A \rightarrow B(\text{Confidence} = 5/7).$$

Step 15: Since If $c + t < f - h$, $c = c + t = 0 + 2 = 2$.

After Step 15, the maximally large sequence for the updated database has been found. Also, c is 2 and b is 1. The new values of b and c will be used for processing next new transactions.

9. Conclusion

In this paper, we have proposed a novel incremental mining algorithm capable of maintaining sequential patterns based on the concept of pre-large sequences. Pre-large sequences act like buffers to postpone originally small sequences directly becoming large and vice versa, when new transactions are inserted into existing databases. We have also proven that when small numbers of new transactions are inserted, an originally small sequence will at most become pre-large, and never large. The number of rescans of original databases can thus be reduced. In summary, the proposed algorithm has the following advantages:

1. It avoids re-computing large sequences that have already been discovered.
2. It focuses on newly added customer sequences, which are transformed from newly added transactions, thus greatly reducing the number of candidate sequences.
3. It uses a simple check to further filter candidate sequences in newly added customer sequences.
4. It effectively handles the case, in which sequences are small in an original database.

The proposed algorithm also requires no rescanning of original databases until certain numbers of new transactions, determined from the two support thresholds and the database size, have been processed. The bound increases monotonically along with increases in database size, which means the proposed algorithm

becomes increasingly efficient as a database grows. This characteristic is especially useful for real-world applications.

References

Agrawal, R., Imielinksi, T. & Swami, A. (1993). Mining association rules between sets of items in large database. In *The ACM SIGMOD conference. Washington DC, USA*, pp. 207–216.

Agrawal, R., Srikant, R., & Vu, Q. (1997). Mining association rules with item constraints. In *The third international conference on knowledge discovery in databases and data mining. Newport Beach, California*, pp. 67–73.

Agrawal, R., Imielinksi, T., & Swami, A. (1993). Database mining: A performance perspective. *IEEE Transactions on Knowledge and Data Engineering, 5*(6), 914–925.

Agrawal, R., & Srikant, R. (1994). Fast algorithm for mining association rules. *The International Conference on Very Large Data Bases*, 487–499.

Agrawal, R., & Srikant, R. (1995). Mining sequential patterns. *The Eleventh IEEE International Conference on Data Engineering*, 3–14.

Chen, M. S., Han, J., & Yu, P. S. (1996). Data mining: An overview from a database perspective. *IEEE Transactions on Knowledge and Data Engineering, 8*(6), 866–883.

Cheung, D. W., Han, J., Ng, V. T., & Wong, C. Y. (1996). Maintenance of discovered association rules in large databases: An incremental updating approach. In *The 12th IEEE international conference on data engineering*, pp. 106–114.

Cheung, D. W., Lee, S. D., & Kao, B. (1997). A general incremental technique for maintaining discovered association rules. In *Proceedings of database systems for advanced applications. Melbourne, Australia*, pp. 185–194.

Fukuda, T., Morimoto, Y., Morishita, S., & Tokuyama, T. (1996). Mining optimized association rules for numeric attributes. *The ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, 182–191.

Han, J., & Fu, Y. (1995). Discovery of multiple-level association rules from large database. In *The twenty-first international conference on very large data bases. Zurich, Switzerland*, pp. 420–431.

Hong, T. P., Wang, C. Y., & Tao, Y. H. (2000). Incremental data mining based on two support thresholds. In *The fourth international conference on knowledge-based intelligent engineering systems and allied technologies*.

Hong, T. P., Wang, C. Y., & Tao, Y. H. (2001). A new incremental data mining algorithm using pre-large itemsets. *Intelligent Data Analysis, 5*(2), 111–129.

Lin, M. Y., & Lee, S. Y. (1998). Incremental update on sequential patterns in large databases. In *The 10th IEEE international conference on tools with artificial intelligence*, pp. 24–31.

Mannila, H., Toivonen, H., & Verkamo, A. I. (1994). Efficient algorithm for discovering association rules. *The AAAI Workshop on Knowledge Discovery in Databases*, 181–192.

Park, J. S., Chen, M. S., & Yu, P. S. (1997). Using a hash-based method with transaction trimming for mining association rules. *IEEE Transactions on Knowledge and Data Engineering, 9*(5), 812–825.

Sarda, N. L., & Srinivas, N. V. (1998). An adaptive algorithm for incremental mining of association rules. In *The Ninth international workshop on database and expert systems*, pp. 240–245.

Savasere, A., Omiecinski, E. & Navathe, S. (1995). An efficient algorithm for mining association rules in large database. In *The 21st international conference on very large data bases. Zurich, Switzerland*, pp. 432–444.

Srikant, R., & Agrawal, R. (1995). Mining sequential patterns: Generalizations and performance improvements. In *The fifth international conference on knowledge discovery and data mining (KDD'95)*, pp. 269–274.

Zhang, S. (1999). Aggregation and maintenance for database mining. *Intelligent Data Analysis, 3*(6), 475–490.