Maintenance of Discovered Sequential Patterns for Record Modification

Ching-Yao Wang[1], Tzung-Pei Hong[2] and Shian-Shyong Tseng[1]
[1]Institute of Computer and Information Science, National Chiao-Tung University
[2]Department of Electrical Engineering, National University of Kaohsiung
cywang@cis.nctu.edu.tw, tphong@nuk.edu.tw, sstseng@cis.nctu.edu.tw

Abstract

In the past, some researchers proposed efficient incremental mining algorithms for maintenance of sequential patterns as records were inserted or deleted. In addition to record insertion and deletion, record modification is also very commonly seen. Although maintenance of sequential patterns for record modification can be performed by usage of the deletion procedure and then by the insertion procedure, twice the computation time of a single procedure is needed. In this paper, we thus attempt to design an effective maintenance algorithm for sequential patterns as records are modified. Our proposed algorithm utilizes previously discovered large sequences in the maintenance process, thus greatly reducing numbers of rescanning databases and improving the performance. Experimental results also show the performance of the proposed approach. It is thus useful for dynamic database mining.

1. Introduction

Mining useful information and helpful knowledge from large databases has evolved into an important research area [1][3]. Among them, finding sequential patterns in temporal transaction databases is important since it allows modeling of customer behavior [2][7][9]. Although customer behavior models can be efficiently extracted by Agrawal and Srikant's mining algorithm [2] to assist managers in making correct and effective decisions, the sequential patterns discovered may become invalid or inappropriate when databases are updated. Conventional approaches may re-mine entire databases to get up-to-date sequential patterns. However, when a database is massive in size, this will require considerable amounts of computation time.

In the past, some approaches were proposed to improve maintenance performance. Examples for associations rules are the FUP algorithm proposed by Cheung et al. [4][5], the adaptive algorithm proposed by Sarda and Srinivas [8], the incremental mining algorithm based on the pre-large concept proposed by Hong *et al.* [6], the incremental updating technique based on negative borders proposed by Thomas *et al.* [10]. As to maintenance of sequential patterns, Lin and Lee proposed the FASTUP algorithm for inserted records [7]. Wang *et al.* then proposed a maintenance algorithm for deleted records [11]. The common idea of all the above approaches is that previously information mined should be utilized as much as possible to reduce maintenance costs.

In addition to record insertion and deletion, record modification in databases is also commonly seen in applications. Developing an efficient mining algorithm to maintain discovered information as records are modified is thus important in the field of data mining. Although maintenance of sequential patterns for record modification can be performed by usage of the deletion procedure and then by the insertion procedure, twice the computation time of a single procedure is needed. In this paper, we thus attempt to design an effective maintenance algorithm for sequential patterns as records are modified.

2. Maintenance of Sequential Patterns for Record Modification

When records are modified in databases, the original sequential patterns may become invalid, or new implicitly valid patterns may appear in the resulting updated databases. For example, assume a database has 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 occurred, and *Trans_content* stores what items were purchased in a transaction. Also assume that the database consists of twenty records, sorted first by *Cust_id* and then by *Trans_time*, as shown in Table 1.

Let a *sequence* be an ordered list of itemsets and a *customer sequence* be a sequence of all transactions for a customer in order of transaction times. Note

Table 1: The twenty records sorted first by *Cust_id* and then by *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
9	1998/01/02	A
9	1998/01/03	B, C
9	1998/01/07	G
10	1998/01/05	E, F, G

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. These records in Table 1 are transformed into customer sequences as shown in Table 2.

Table 2: The customer sequences transformed from the records 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)>
9	<(A)(B, C)(G)>
10	<(E, F, G)>

Assume the minimum support is set at 50% (i.e., four customer sequences for this example). All the large sequences mined from the customer sequences in Table 2 by Agrawal and Srikant's AprioriAll approach [2] are presented in Table 3.

If the two records, *Cust_id* = 2 with *Trans_time* = 1998/01/11 and *Cust_id* = 3 with *Trans_time* = 1998/01/07, are modified as shown in Table 4, the large sequences in Table 3 will be modified as those shown in Table 5.

Table 3: All large sequences mined from the customer sequences in Table 2

Large sequences			
1-sequence	Count	2-sequence	Count
<(A)>	7	<(A)(B)>	5
<(B)>	6		
<(C)>	5		
<(G)>	6		

Table 4: The two modified records

Cust_id	Trans_time	Trans_content
2	1998/01/11	D
3	1998/01/07	A, E, G

Table 5: The large sequences after record modification

Large sequences			
1-sequence	Count	2-sequence	Count
<(A)>	7	<(A)(B)>	5
<(B)>	6		
<(E)>	5		
<(G)>	6		
<(E, G)>	5		

The original large sequence <(C)> will become small and the original small sequences <(E)> and <(E, G)> will become large after the two records are modified. Conventional batch-mining algorithms must re-process the entire updated database to find the updated sequential patterns for managing these situations.

3. Review of the FUP and the FASTUP Approaches

In 1996, Cheung et al. first proposed and applied the concept of intermediate information to design an incremental mining algorithm, called FUP [4][5], for association rules as new records are inserted. Using FUP, large itemsets with their counts in preceding runs are recorded for later use in maintenance. As new transactions are added, FUP first scans them to generate candidate 1-itemsets (only for these new transactions), and then compares these itemsets with the previous ones retained in the intermediate information. FUP partitions candidate 1-itemsets into two parts according to whether they are large for the original database. If a candidate 1-itemset from the newly inserted transactions is also 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 all previous large itemsets with their counts are kept by FUP. Whether an original large itemset is still large after new transactions are inserted is determined from its support ratio as its total count over the total number of transactions. By contrast, if a candidate 1-itemset from the newly inserted transactions does not exist among the large 1-itemsets in the original database, one of two possibilities arises. If this candidate 1-itemset is not large for the new transactions, then it cannot be large for the entire updated database, which means no action is necessary. If this candidate 1-itemset is large for the new transactions but not among the original large 1-itemsets, 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.

Lin and Lee then proposed the FASTUP algorithm [7] to maintain sequential patterns as new records are inserted by extending the FUP algorithm. Maintaining sequential patterns is much harder than maintaining association rules since the former must consider both itemsets and sequences.

4. Four Cases Caused from Record Modification

When records are modified in a database, they are first merged with the other records from the same customers to form the modified customer sequences. For example, assume the original records and customer sequences are the same as those shown in Tables 1 and 2. When the two original records are modified as shown in Table 4, the two corresponding original customer sequences shown in Table 6 will be transformed into the modified customer sequences shown in Table 7.

Table 6: The two original customer sequences

Cust_id	Customer sequence
2	<(C, D)(A)(E, F, G)>
3	<(A, H, G)>

Table 7: The two modified customer sequences

Cust_id	Customer sequence
2	<(D)(A)(E, F, G)>
3	<(A, E, G)>

The subsequences from the modified customer sequences and from the corresponding old customer sequences are then compared to obtain the subsequence differences resulting from record modification. Let modified candidate sequences for record modification be defined as the sequences mentioned above, with their count differences not being zero. For the above example, the candidate 1-sequences are shown in Table 8.

Table 8: The candidate 1-sequences with their counts

Candidate 1-sequences	
Candidate 1-sequence	Count difference
<(C)>	-1
<(E)>	1
<(H)>	-1
<(C, D)>	-1
<(E, G)>	1
<(A, E)>	1
<(A, H)>	-1
<(H, G)>	-1
<(A, H, G)>	-1
<(A, E, G)>	1

Considering original customer sequences in terms of minimum support threshold and modified sequences in terms of positive or negative count differences, there exist four cases illustrated in Figure 1.

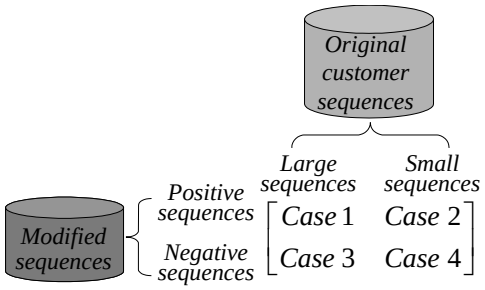

Figure 1: Four cases arising from modifying records in existing databases

In Figure 1, Cases 1 and 4 will not affect the final sequential patterns. The final sequences in Case 1 will always be large and those in Case 4 will always be small. Case 2 may add new large sequences, and Case 3 may remove existing large sequences. If we retain all large sequences with their counts in the original database, then Cases 1 and 3 can be easily

handled. Thus, only sequences in Case 2 need to be rescanned. The maintenance cost can thus be reduced.

5. The Proposed Maintenance Algorithm for Record Modification

In the proposed algorithm, the originally large sequences with their counts from preceding runs are retained for later use in maintenance. As records are modified, they are first merged with the other original records from the same customers to form modified customer sequences. The candidate 1-sequences are then found from the customer sequences before and after record modification. The candidate 1-sequences are then compared to the large 1-sequences which were previously retained. They are thus divided into two parts according to whether they are large or small in the original database. If a candidate 1-sequence is also among the previously retained large 1-sequences, its new total count for the entire updated database can easily be calculated from its current count and previous count. On the contrary, if a candidate 1-sequence does not exist among the previously retained large 1-sequences and its current count is positive, this sequence may be large for the entire updated database, and rescanning the original database is necessary; otherwise, it must be a small sequence and can be ignored. The proposed algorithm can thus find all large 1-sequences for the entire updated database. After that, candidate 2-sequences from modified 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.

The proposed maintenance algorithm for record modification:

INPUT: A predefined minimum support *s*, a set of large sequences for the original database *D* consisting of *d* customer sequences, and a set of *t* modified customer sequences from the modified transactions.

OUTPUT: A set of sequential patterns for the updated database.

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

STEP 2: Find all the candidate k-sequences C_k^T with their count differences from the modified customer sequences.

STEP 3: Divide the candidate k-sequences in C_k^T into two parts according to whether they are large or small for the original database.

STEP 4: Do the following substeps for each k-sequence I existing in both C_k^T and L_k^D:

Substep 4-1: Set the new count $S^U(I) = S^D(I) + S^T(I)$.

Substep 4-2: If $S^U(I)/d \geq s$, then assign I as a large sequence, set $S^D(I) = S^U(I)$, and keep I with $S^D(I)$; otherwise, remove I.

STEP 5: For each sequence I that is in C_k^T but not in L_k^D, if $S^T(I)$ is positive, then put it into the rescan-set R.

STEP 6: If R is null, then do nothing; otherwise, rescanning the original database to determine whether the sequences in the rescan set R are large.

STEP 7: Form candidate $(k+1)$-sequences C_{k+1}^T, each of which must be a $(k+1)$-sequence difference from the record modification and must have all their k-subsequences existing in the large k-sequences.

STEP 8: Set $k = k+1$.

STEP 9: Repeat STEPs 2 to 8 until no candidate k-sequences are found.

After Step 9, the large sequences for the updated database can thus be found.

6. An Example

Assume that the initial customer sequences are the same as those shown in Table 2 and *s* is set at 50%. The large sequences for the given data set are thus the same as those shown in Table 3.

If the two records, *Cust_id* = 2 & *Trans_time* = 1998/01/11 and *Cust_id* = 3 & *Trans_time* = 1998/01/07 from Table 1 have been modified to those shown in Table 4, C_1^T with their count differences are thus the same as those shown in Table 8.

According to whether they are large or small for the original database, C_1^T are divided into two parts as shown in Table 9. The new count of the originally large 1-sequence <(C)> thus becomes 4, since its new support ratio is less than *s*, <(C)> thus becomes a small sequence. The other candidate 1-sequences with positive count differences are then put in the rescan set *R*. Results are shown in Table 10.

Table 9: Two parts of C_1^T in Table 8

Originally large 1-sequence		Originally small 1-sequence	
1-sequence	Count Difference	1-sequence	Count difference
<(C)>	-1	<(E)>	1
		<(H)>	-1
		<(C, D)>	-1
		<(E, G)>	1
		<(A, E)>	1
		<(A, H)>	-1
		<(H, G)>	-1
		<(A, H, G)>	-1
		<(A, E, G)>	1

Table 10: The 1-sequences put in the rescan set *R*

1-sequence	Count difference
<(E)>	1
<(E, G)>	1
<(A, E)>	1
<(A, E, G)>	1

Since *R* is not null, rescanning the original transactions in Table 2. Table 11 shows the resulting large 1-sequences in *R*.

Table 11: The resulting large 1-sequences in *R*

1-sequence	New count
<(E)>	5
<(E, G)>	5

The modified 2-sequences with count differences not equal to zero are those with the 1-subsequences <(C)> or <(C, D)> at the front. Since the 1-subsequences <(C)> or <(C, D)> are not large from the previous results, no candidate 2-sequences are formed. The final large sequences are shown in Table 12.

Table 12: The final large sequences after record modification

Large sequences			
1-sequence	Count	2-sequence	Count
<(A)>	7	<(A)(B)>	5
<(B)>	6		
<(E)>	5		
<(G)>	6		
<(E, G)>	5		

7. Experiments

The section reports on experiments made to show the performance of the proposed approach. It was implemented in C++ on a Pentium-III 800 dual-processor workstation with 512M RAM. The experimental datasets were generated by the program developed by Agrawal and his co-workers in [2]. 100000 customer sequences were generated at random, with each sequence having an average of 5 transactions of single items for a customer. The mean size of the maximal potentially sequential patterns was 4. Figure 2 shows the relationships between computational times and numbers of modified customer sequences for our proposed approach and for the Apriorial approach, with the minimum support = 0.6%.

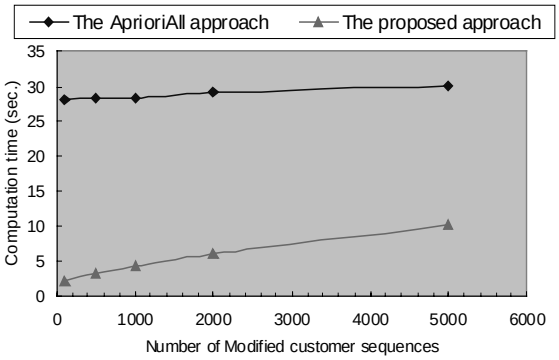

Figure 2. The relationships between computational times and numbers of modified customer sequences (the minimum support = 0.6%)

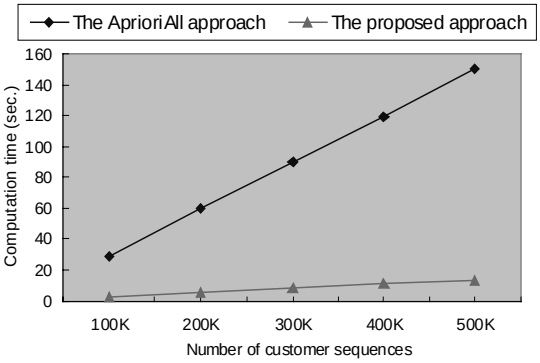

Figure 3. The relationships between computational times and numbers of customer sequences (the minimum support = 0.6%)

From Figure 2, it is easily seen that the computational times by the proposed approach are much less than those by the Apriorial approach for sequence modification. Experiments were then made

for a comparison of different datasets. The relationships between computational times and numbers of customer sequences, when the number of modified customer sequences is 500 and the minimum support is 0.6%, are shown in Figure 3.

From Figure 3, it is easily seen that the execution time increased by the proposed algorithm for datasets with larger numbers of customer sequences is little when compared to those by the Aprioriall algorithm.

8. Conclusions

In this paper, we have proposed an efficient maintenance algorithm to sequential patterns for record modification. Experimental results have also showed the performance of the proposed approach. In summary, the proposed algorithm has the following attractive features.

1. It utilizes previously discovered information in maintenance.
2. It focuses on modified customer sequences, thus greatly reducing the number of candidate sequences.
3. It uses a simple check to further filter the candidate sequences in modified customer sequences.
4. The performance is even better for a larger database.

These characteristics are especially useful for real-world applications.

Acknowledgement

This research was supported by the Ministry of Education and the National Science Council of the Republic of China under Grand No. 89-E-FA04-1-4, “High Confidence Information Systems”.

References

[1] R. Agrawal, T. Imielinksi and A. Swami, “Database mining: a performance perspective,” IEEE Transactions on Knowledge and Data Engineering, Vol. 5, No. 6, pp. 914-925, 1993.

[2] R. Agrawal and R. Srikant, ”Mining sequential patterns,” The Eleventh IEEE International Conference on Data Engineering, pp. 3-14, 1995.

[3] M. S. Chen, J. Han and P. S. Yu, “Data mining: an overview from a database perspective,” IEEE Transactions on Knowledge and Data Engineering, Vol. 8, No. 6, pp. 866-883, 1996.

[4] D. W. Cheung, J. Han, V. T. Ng and C. Y. Wong, “Maintenance of discovered association rules in large databases: an incremental updating approach,” The Twelfth IEEE International Conference on Data Engineering, pp. 106-114, 1996.

[5] D. W. Cheung, S. D. Lee and B. Kao, “A general incremental technique for maintaining discovered association rules,” The International Conference on Database Systems for Advanced Applications, pp. 185-194, Melbourne, Australia, 1997.

[6] T. P. Hong, C. Y. Wang and Y. H. Tao, “A new incremental data mining algorithm using pre-large itemsets,” An International Journal: Intelligent Data Analysis, Vol. 5, No. 2, pp. 111-129, 2001.

[7] M. Y. Lin and S. Y. Lee, “Incremental update on sequential patterns in large databases,” The Tenth IEEE International Conference on Tools with Artificial Intelligence, pp. 24-31, 1998.

[8] N. L. Sarda and N. V. Srinivas, “An adaptive algorithm for incremental mining of association rules,” The Ninth International Workshop on Database and Expert Systems Applications, pp. 240-245, 1998.

[9] R. Srikant and R. Agrawal, “Mining sequential patterns: generalizations and performance improvements,” The Fifth International Conference on Knowledge Discovery and Data Mining, pp. 269-274, 1995.

[10] S. Thomas, S. Bodagala, K. Alsabti and S. Ranka “An efficient algorithm for the incremental update of association rules in large databases,” The International Conference on Knowledge Discovery and Data Mining, pp. 263-266, 1997.

[11] C. Y. Wang, T. P. Hong and S. S. Tseng "A Pattern-maintenance Algorithm for Record Deletion in Temporal Transaction Databases," The Sixth Conference on Artificial Intelligence and Applications, Kaohsiung, Taiwan, 2001.