An Efficient FUSP-Tree Update Algorithm for Deleted Data in Customer Sequences

Chun-Wei Lin[1], Tzung-Pei Hong[2] and Wen-Hsiang Lu[1]
[1]Department of Computer Science and Information Engineering
National Cheng Kung University, Taiwan, R.O.C.
[2]Department of Computer Science and Information Engineering
National University of Kaohsiung, Taiwan, R.O.C.
p7895122@mail.ncku.edu.tw, tphong@nuk.edu.tw, whlu@mail.ncku.edu.tw

Abstract

In the past, the fast-updated sequential-pattern tree (call FUSP-tree) structure was proposed for mining sequential patterns from a set of customer sequences. An incremental mining algorithm was also designed for handling newly added transactions. Since data may also be deleted in real applications, an FUSP-tree maintenance algorithm for deletion of customer sequences is thus proposed in this paper for reducing the execution time in reconstructing the tree. Experimental results also show that the proposed tree-update algorithm has a good performance than the batch FUSP-tree algorithm for handling the deletion of customer sequences. The proposed tree-update algorithm thus makes the tree update process become easy and efficient.

1. Introduction

Mining useful information and helpful knowledge from large databases has evolved into an important research area [1][2][3]. Among the classes of knowledge sought, finding sequential patterns in temporal transaction databases is very useful to business since it allows modeling of customer behavior [4][10].

Mining sequential patterns was first proposed by Agrawal *et al.* in 1995 [4], and is a non-trivial task. Although the models of customer behavior can be extracted by mining algorithms 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.

Han *et al.* proposed the Frequent-Pattern-tree (FP-tree) structure for efficiently mining association rules without generation of candidate itemsets [6]. Hong *et al.* thus modified the FP-tree structure and designed the fast updated frequent pattern trees (FUFP-trees) [8] to efficiently handle newly inserted transactions based on the FUP concept [3]. Lin *et al.* then extended the FUFP-trees to maintain sequential patterns and proposed the FASTUP algorithm [10]. Their approach worked well except when newly coming candidate sequences were not frequent (or called large) in the original database. If this occurred frequently, the performance of the FASTUP algorithm would decrease.

Hong *et al.* also used the pre-large concept to reduce the need of rescanning original databases for maintaining sequential patterns [7]. A pre-large sequence is not truly large, but may be large with a high probability in the future. Cheng *et al.* then further proposed the IncSpan (Incremental mining of sequential patterns) algorithm for efficiently maintaining sequential patterns in a tree structure [5]. IncSpan used the frequent and semi-frequent sequences, which were a little like large and pre-large sequences in Hong *et al.*'s approach [7], to speed up the maintenance for items appended to old sequences. An IncSpan tree was designed to keep the frequent and semi-frequent sequences. The approach did not generate the frequent sequences from the tree as in the FP-tree, but used the tree to maintain the frequent sequences. Besides, Lin *et al.* proposed the FUSP-tree structure and the FUSP-tree maintenance algorithm for efficiently handling new transactions and finding sequential patterns [9].

In addition to data insertion, data deletion is also common seen in the real-world applications. In this paper, the previous FUSP-tree structure is used in this paper and the FUSP-tree maintenance algorithm is proposed for reducing the execution time in re-constructing the tree when data in customer sequences are deleted from a database. This approach may be especially efficient for mining the traversal paths from the web log since only one item (web page) exists in a frequent itemset. The proposed approach is especially useful for this situation.

2. Review of Related Works

The sequential patterns discovered from mining algorithms may become invalid when data in customer sequences are deleted. If all the data in a customer sequence are deleted, then the customer sequence is deleted. Considering an original database and some customer sequences to be deleted, itemsets may fall into one of the following four cases illustrated in Figure 1.

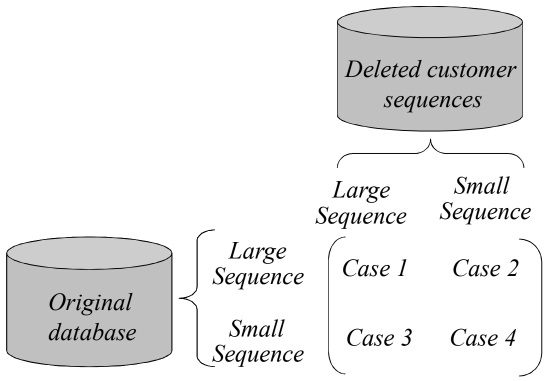

Figure 1: Four cases arising from the deleted customer sequences and existing databases

Lin *et al.* proposed an FUSP-tree structure and the FUSP-tree maintenance algorithm for efficiently handling new transactions and finding sequential patterns [9]. The FUSP-tree structure is similar to the FP-tree [6] and FUFP-tree [8] structure except each node consists of the connection relation with the prefix item.

An FUSP tree must be built in advance from the original customer sequences before the customer sequences are maintained. The concept of the FUSP tree is extended from both the FUFP tree [8] and the IncSpan tree [5]. The FUSP tree is compact because it takes the characteristic of the FUFP tree that only frequent items are kept. Besides, the complete customer sequences are kept in the FUSP tree to avoid and reduce the cost of rescanning the original database. A FUSP tree is shown in Figure 2.

In Figure 2, only the frequent items (1-itemsets) are kept. Like the IncSpan tree [5], the link between two connected nodes is marked by the symbol *s* (representing the sequence relation) if the sequence is within the sequence relation in a sequence; otherwise, the link is marked by the symbol *i*, which indicates the sequence is within the itemset relation in a sequence. The tree thus contains the complete customer sequences from the database.

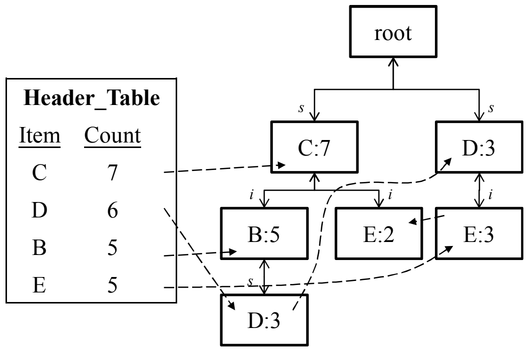

Figure 2: The initial constructed FUSP tree with its Header_Table

3. The Proposed Maintenance Algorithm

The proposed maintenance algorithm for handling deleted data in customer sequences is stated below.

INPUT: An old database which contains the customer sequences, its corresponding Header_Table which sorts the frequent 1-itemsets initially in descending order, its corresponding FUSP tree, a support threshold *s*, and a set of *t* deleted transactions.

OUTPUT: An updated FUSP tree by using the FUSP-tree maintenance algorithm.

STEP 1: Form the deleted customer sequences from the *t* deleted transactions.

STEP 2: Scan the deleted customer sequences to get all the 1-itemsets and their counts.

STEP 3: Find the number *q* of deleted customers (with all their records are deleted) from the deleted customer sequences.

STEP 4: Divide the 1-itemsets in the deleted customer sequences into two parts according to whether they are large or small in the original database.

STEP 5: For each 1-itemset *I* which are large both in the deleted customer sequences and in the original database (appearing in the Header_Table), do the following substeps (**Case 1**):

Substep 5-1: Set the new count $S^U(I)$ of I in the entire updated database as:

$$S^U(I) = S^D(I) - S^T(I),$$

where $S^D(I)$ is the count of I in the Header_Table (original database) and $S^T(I)$ is the count of I in the deleted customer sequences.

Substep 5-2: If $S^U(I) \geq (d-q)*s$, I will still be large after the database is updated; update the count of *I* in the Header_table as $S^U(I)$ and add *I* to the set of *Reduced_Items*, which will be processed in STEP 7;
Otherwise, the 1-itemset *I* will become small after the database is updated; Remove *I* from the Header_Table, connect each parent node of *I* directly to the corresponding child node of *I*, and remove *I* from the FUSP tree.

STEP 6: For each 1-itemset *I* which are small in the deleted customer sequences but large in the original database (appearing in the Header_Table), do the following substeps (**Case 2**):

Substep 6-1: Set the new count $S^U(I)$ of *I* in the entire updated database as:
$$S^U(I) = S^D(I) - S^T(I).$$

Substep 6-2: Update the count of *I* in the Header_Table as $S^U(I)$.

Substep 6-3: Put *I* in the set of *Reduced_Items*, which will be further processed in STEP 7.

STEP 7: For each deleted customer sequence with an item *J* existing in the set of *Reduced_Items*, if *J* has been at the corresponding branch of the FUSP tree for the deleted customer sequences, subtracts 1 from the count of node *J*. If the count of node *J* becomes 0, remove the node *J* from the FUSP tree.

STEP 8: For each 1-itemset *I* which are both small in the deleted customer sequences and in the original database (not appearing in the Header_Table), do the following substeps (**Case 4**):

Substep 8-1: Rescan the original database to calculate the count $S^D(I)$ of 1-itemset *I* in the original database. Set the new count $S^U(I)$ of *I* in the entire updated database as:
$$S^U(I) = S^D(I) - S^T(I).$$

Substep 8-2: If $S^U(I) \geq (d-q)*s$, I will become large after the database is updated; Add the 1-itemset *I* in the set of *Rescan_Items* for further processing.

STEP 9: If *Rescan_Items* is ***null***, do nothing;
Otherwise, sort the 1-itemsets in the *Rescan_Items* in a descending order of their updated counts and do the following substeps:

Substep 9-1: Insert the 1-itemsets in the *Rescan_Items* to the end of the Header_Table according to the sorted list.

Substep 9-2: For each original database with a 1-itemset *J* existing in the *Rescan_Items*, if *J* has not been at the corresponding branch of the FUSP tree for the customer sequences, insert *J* at its sequential position of the branch in the original databases and set its count as 1; Otherwise, add 1 to the count of the node *J*.

In STEP 7, a corresponding branch is the branch generated from the customer sequences according to the sequence order of the large 1-itemsets in the transaction. After STEP 9, the final updated FUSP tree by the proposed maintenance algorithm for deleted transactions is constructed. Based on the FUSP tree, the desired large sequences for the updated database can be determined by a recursive procedure like the FP-growth approach [6] but more complex than it.

4. An Example

In this section, an example is given to illustrate the proposed FUSP-tree maintenance algorithm for deleted customer sequences. Table 1 shows a database to be used in the example. It contains 10 customer sequences and 9 items, denoted A to I.

Table 1: The customer sequences

Cust_id	Customer sequence
1	<(B, C)(D)(I)>
2	<(D, E)(F, H)>
3	<(A)(B, C)>
4	<(A, B, C)(G, H, I)>
5	<(D, E)(F)>
6	<(A)(B, C)(D, F)>
7	<(B, C)(D, F)(G, H)>
8	<(C, E)(I)>
9	<(C, E)>
10	<(A)(D, E, G)(I)>

Assume the minimum support threshold is set at 50%. The FUSP tree and the Header_Table found from the database are the same as that shown in Figure 2. Assume eight transactions are deleted, which form the following five deleted customer sequences in Table 2.

Table 2: The deleted customer sequences

Cust_id	Customer sequence
4	<(G, H, I)>
7	<(G, H)>
8	<(C, E)(I)>
9	<(C, E)>
10	<(A)(D, E, G)(I)>

The final results by the proposed algorithm are shown in Figure 3.

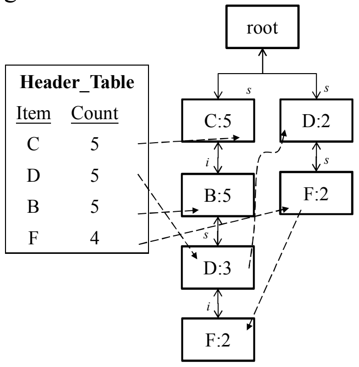

Figure 3: The final updated FUSP tree

5. Experimental Results

Experiments were made to compare the performance of the batch FUSP-tree construction algorithm and the FUSP-tree maintenance algorithm for handling deleted customer sequences. A simulation database is thus generated from IBM data generator. There were 4,000,000 transactions with 1,000 items in the dataset. In the experiments, the last 4,000 transactions are deleted from the database.

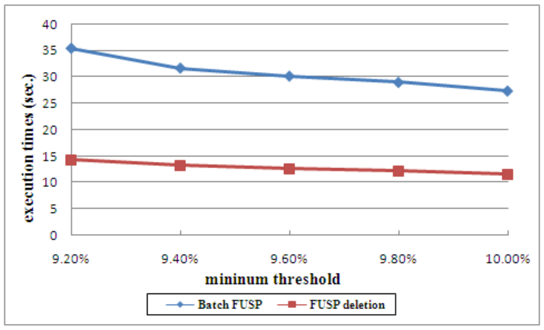

Figure 4: The execution time of the two algorithms in five different thresholds

It can be observed from Figure 4 that the proposed FUSP-tree maintenance algorithm for deleted customer sequences ran faster than the batch one in all the five different minimum thresholds.

6. Conclusions

In this paper, we have used the FUSP-tree structure to efficiently and effectively handle the deleted customer sequences. An FUSP-tree maintenance algorithm for deleted transactions has also been proposed. From the experimental results, it is obvious to observe that our proposed maintenance algorithm has a better performance than the batch FUSP-tree construction algorithm.

7. References

[1] R. Agrawal, T. Imielinksi and A. Swami, "Mining association rules between sets of items in large database," *The ACM SIGMOD Conference*, pp. 207-216, 1993.

[2] 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.

[3] R. Agrawal and R. Srikant, "Fast algorithm for mining association rules," *The International Conference on Very Large Data Bases*, pp. 487-499, 1994.

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

[5] H. Cheng, X. Yan and J. Han, "IncSpan: incremental mining of sequential patterns in large database," *The ACM SIGKDD international conference on Knowledge discovery and data mining*, pp.527-532, 2004.

[6] J. Han, J. Pei and Y. Yin, "Mining frequent patterns without candidate generation," *The 2000 ACM SIGMOD International Conference on Management of Data*, pp. 1-12, 2000.

[7] T. P. Hong, C. Y. Wang and S. S. Tseng, "Incremental data mining for sequential patterns using pre-large sequences," *The fifth world multi-conference on Systemics, Cybernetics and Informatics*, 2001.

[8] T. P. Hong, C. W. Lin and Y. L. Wu, "Incrementally fast updated frequent pattern trees," *Expert Systems with Applications* , Vol. 34, Issue 5, pp. 2424-2435, 2008.

[9] C. W. Lin, T. P. Hong, W. H. Lu and W. Y. Lin, "An Incremental FUSP-Tree Maintenance Algorithm," *International Conference on Intelligent System Design and Applications*, Vol. 1, pp. 445 - 449, 2008.

[10] 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.