An Efficient Approach for Maintaining Association Rules based on Adjusting FP-tree Structure

Jia-Ling Koh and Shui-Feng Shieh

Department of Information and Computer Education

National Taiwan Normal University

Taipei, Taiwan 106, R.O.C.

Email: jlkoh@ice.ntnu.edu.tw

Abstract

In this paper, the issue of mining and maintaining association rules in a large database of customer transactions is studied. The maintenance of association rules can be mapped into the problem of maintaining frequent itemsets in the database. Because the mining of association rules is time-consuming, we need an efficient approach to maintain the frequent itemsets when the database is updated. In this study, a general incremental updating technique is proposed for maintaining the frequent itemsets discovered in a database in the cases including insertion, deletion, and modification of transactions in the database. An efficient algorithm, called AFPIM (Adjusting FP-tree for Incremental Mining), is designed based on adjusting FP-tree structures. Our approach uses a FP-tree structure to store the compact information of transactions involving frequent and pre-frequent items in the original database. In most cases, without needing to rescan the original database, the new FP-tree structure of the updated database can be obtained by adjusting FP-tree of the original database according to the changed transactions. Experimental results show that AFPIM outperforms the existing algorithms in terms of the execution time.

1. Introduction

Data mining has attracted much attention in database research due to its wide applicability in many areas. Among the various data mining applications, mining association rules in customer transactions is an important one [1]. A customer transaction record typically consists of the customer-id, transaction time, and all the items bought in the transaction. Mining association rules from such a database is to find out all the rules like "n% of customers who purchase items X and Y also buy item Z in the same transaction", where n, X, Y, Z are initially unknown. Such rules are useful for making decisions of customized marketing.

The problem of mining association rules over customer transactions was introduced in [1]. Several efficient algorithms have been proposed for finding frequent itemsets and the association rules are derived from the frequent itemsets, such as the Apriori[1] and DHP[9] algorithms. These Apriori-like algorithms suffer from the costs to handle a huge number of candidate sets and scan the database repeatedly. [7] proposed a frequent pattern tree (FP-tree) structure for storing compressed and critical information about frequent patterns. Besides, an algorithm, named FP-growth, for mining the complete set of frequent itemsets from FP-tree is developed. This approach avoids the costly generation of a large number of candidate sets and repeated database scans, which is regarded as the most efficient strategy for mining frequent itemsets.

Updates to the transaction database could invalidate existing rules or introduce new rules. The problem of updating the association rules can be reduced to finding the new set of frequent itemsets in the updated database. A simple solution to the update problem is to re-mine the frequent itemsets of the whole updated database. However, it is clearly inefficient because all the computations done in the previous mining are wasted. The problem of maintaining discovered association rules was first studied in [4],

which proposed the FUP algorithm to incremental updating the association rules when new transaction data are added. In order to solve the problem of general cases, including insertion, deletion, and modification of transactions in the database, the FUP algorithm was modified in [6] to develop FUP2 algorithm. An incremental updating technique, similar to FUP algorithm, was proposed in [5] for mining multi-level association rules. As Apriori-like algorithms, all the FUP families have to generate large number of candidates and repeatedly scan the database.

The incremental updating techniques proposed in [12] and [13] maintained additional information in addition to the frequent itemsets. [13] proposed an incremental mining algorithm for finding frequent sequential pattern. The pre-large sequences were preserved in [13], which were sequences with supports between a lower support threshold and an upper threshold. A bound derived from the lower and upper thresholds determined when rescanning the original database was needed. In [12], the negative border was maintained along with the frequent itemsets. This algorithm required a full scan of the whole database if an itemset outside the negative border got added to the frequent itemsets or its negative border.

DUP algorithm was developed in [8] to solve the problem of maintaining association rules in the cases of insertion and deletion. In addition, DLG* algorithm was provided to constructed an association graph between frequent items for generating frequent itemsets efficiently. The bit vector representation of frequent items was designed to speed up the support computing of itemsets. However, this approach suffered from the storage cost for storing the bit vectors when the database is huge.

In this paper, we present an algorithm, called AFPIM(Adjusting FP-tree for Incremental Mining), to efficiently find new frequent itemsets with minimum re-computation when new transactions are added to, deleted from, or modified in the

transaction database. In our approach, the FP-tree structure [7] of the original database was maintained in addition to the frequent itemsets. In most cases, without needing to re-scan the whole database, the FP-tree structure of the updated database is obtained by adjusting the preserved FP-tree according to the inserted and deleted transactions. Then the frequent itemsets of the updated database are mined from the new FP-tree structure and the corresponding association rules are discovered.

The remaining of this paper is organized as follows. Section 2 gives a detailed description of the problem. The new algorithm AFPIM is described in Section 3. A performance study of AFPIM is presented in Section 4, and Section 5 concludes this paper.

2. Problem Description

2.1 Mining association rules

The following definitions refer to [1]. Let $I = \{i_1, i_2, \ldots, i_m\}$ be a set of literals, called *items*. A set of items is called an *itemset*. The number of items in an itemset is called the *length* of an itemset. Itemsets of length k are referred to as *k-itemsets*. Let *DB* be a database of transactions, where each transaction T in *DB* is an itemset such that $T \subseteq I$. For a given itemset $X \subseteq I$, we say that a transaction T *contains* itemset X if and only if $X \subseteq T$. The *support count* (or occurrence frequency) of an itemset X in *DB*, denoted as Sup_X^{DB}, is the number of transactions in *DB* containing X. Given a *minimum support threshold* s%, an itemset X is called a *frequent* itemset in *DB* if $Sup_X^{DB} \geq |DB| \times s\%$, where $|DB|$ is the number of transactions in database *DB*.

An association rule is an implication of the form: $X \Rightarrow Y$, where $X \subseteq I$, $Y \subseteq I$, and $X \cap Y = \varnothing$. Each association rule has a *support* and a *confidence*. The support of association rule $X \Rightarrow Y$ is the percentage of transactions in DB that contains both X and Y,

i.e. $Sup_{X\cup Y}^{DB}$ /|DB|. The rule $X \Rightarrow Y$ has confidence c% in DB if c% of transactions in DB that contain *X* also contain *Y*, i.e. $Sup_{X\cup Y}^{DB} / Sup_{X}^{DB}$.

The problem of mining association rules is to generate all *strong rules* whose supports and confidences are no less than the specified *minimum support* and *minimum confidence*. In [1], it was shown that the problem can be decomposed into two subproblems:

1. Find out all frequent itemsets.
2. From the set of frequent item sets found, generate all the association rules that have a confidence exceeding the minimum confidence.

The solution to the second subproblem is relatively straightforward. Therefore, the problem of mining association rules is reduced to the problem of finding all frequent itemsets[1].

2.2 Update of association rules

Let *DB* denote the original transaction database, and L^{DB} refer to the set of frequent itemsets in *DB*. After some update activities occur in database *DB*, new transactions are inserted and old transactions are removed. The modification of existing transactions can be treated as deletion followed by insertion. Let db^+ (db^-) denote the set of added (deleted) transactions, and $|db^+|$ ($|db^-|$) be the number of added (deleted) transactions. The updated database, denoted as *UD*, is obtained from $DB \cup db^+ - db^-$. Define L^{UD} to be the new set of frequent itemsets in *UD*. The number of transactions in *UD*, |*UD*|, is $|DB|+|db^+|-|db^-|$. Also, the support count of an itemset X in *UD*, denoted as Sup_X^{UD}, is equal to $Sup_X^{DB} + Sup_X^{db^+} - Sup_X^{db^-}$. These definitions are summarized in Table1.

Table 1: Definitions of symbols

database	definition	number of transactions	support count of itemset X	Frequent itemsets
DB	original database	$\|DB\|$	Sup_X^{DB}	L^{DB}
db^+	added transactions	$\|db^+\|$	$Sup_X^{db^+}$	-
db^-	deleted transactions	$\|db^-\|$	$Sup_X^{db^-}$	-
UD	updated database	$\|UD\|$	Sup_X^{UD}	L^{UD}

Therefore, the update problem of association rules is to find L^{UD} efficiently.

3. Adjusting FP-tree Structure for Incremental Mining (AFPIM) Algorithm

3.1 Basic Concept

In this paper, the concept of pre-large sequences[13] is applied. In addition to minimum support threshold, a lesser threshold, called *pre-minimum support*, is specified. For each item X in a database, if its support count is no less than minimum support, X is named a *frequent item*. If the support count of X is less than minimum support and no less than pre-minimum support, X is called a *pre-frequent item*. Otherwise, X is an *infrequent item*. In the following context, frequent items and pre-frequent items are named jointly *frequent or pre-frequent items*.

The frequent-pattern tree (FP-tree) proposed in [7] is an extended prefix-tree structure for storing compressed and crucial information in transactions about frequent itemsets. Based on the tree structure, efficient algorithms [7][14] are proposed for mining the complete set of frequent itemsets. Therefore, our strategy for maintaining association rules is designed based on adjusting FP-tree structure.

In order to solve the update problem efficiently, we maintain the following information after mining the original database *DB*.

1. All the items in *DB* along with their support count in the database.
2. The FP-tree of *DB*, which is constructed according to the frequent or pre-frequent items in *DB*.

An item X in *DB* is a frequent, pre-frequent, or in-frequent item, which will become a frequent, pre-frequent, or in-frequent item in *UD*. The possible cases of an item X are illustrated in Table 2.

Table 2

Case	**in *DB***	**in *UD***
Case 1	X is a frequent or pre-frequent item	X is a frequent item
Case 2	X is a frequent or pre-frequent item	X is a pre-frequent item
Case 3	X is a frequent or pre-frequent item	X is an infrequent item
Case 4	X is an infrequent item	X is a frequent item
Case 5	X is an infrequent item	X is a pre-frequent item
Case 6	X is an infrequent item	X is an infrequent item

If all the items belonging to cases 1, 2, or 3 illustrated in Table 2, the support counts of corresponding itemsets in *DB* can be counted from the FP-tree of *DB*. Therefore, it is needless to scan *DB* one more time. The FP-tree of *UD* can be obtained by adjusting the FP-tree of *DB* according to the transactions in db^+ and db^-. Then, the frequent itemsets in *UD* can be mined. Suppose there exist some items belonging to cases 5 or 6,

which are in-frequent in *DB*. These items can be ignored because the corresponding itemsets are not frequent in *UD*. *UD* needs to be scanned to re-construct the FP-tree of *UD* only if there exist any item belonging to case 4.

3.2 Strategies for Adjusting FP_tree

In the FP-tree of *DB*, each path follows the frequency descending order of frequent or pre-frequent items in *DB*. After insertion or deletion occurs in *DB*, there may exist some items becoming infrequent ones in *UD*. The nodes representing these items have to be removed from the FP-tree. In addition, the paths of nodes have to be adjusted when the frequency descending order of frequent or pre-frequent items in *UD* is different from the order in *DB*. Finally, the inserted and deleted transactions is added and removed from the FP-tree, respectively, to obtain the FP-tree of *UD*.

In sum, there are three steps for adjusting FP-tree：

1. Remove nodes of infrequent items,
2. adjust the path of nodes in FP-tree structure, and
3. insert or remove data from FP-tree according to the transactions in db^+ and db^-.

Table 3: Original Database DB

TID	Items Bought	Frequent or pre-frequent items (ordered in frequency descending order)
1	BDEF	FBED
2	F	F
3	ABEF	FBEA
4	CH	C
5	BF	FB
6	B	B
7	ABEF	FBEA
8	CG	C
9	BF	FB
10	CDE	CED
11	F	F
12	CD	CD
13	C	C

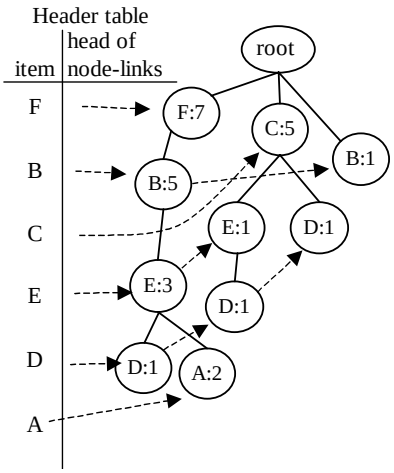


Figure 1: FP-tree of *DB*

TID	Itemset
14	BCDEF
15	BDEF
16	BCDG
17	BD
18	DH

db^+

TID	Itemset
5	BF
8	CG
12	CD

db^-

Figure 2: Inserted and deleted transactions

Table 4

Database	Items	Frequent or Pre-frequent items (in frequency descending order)
DB	A:2, B:6, C:5, D:3, E:4, F:7, G:1, H:1	**F:7, B:6, C:5, E:4, D:3, A:2**
db^+	B:4,C:2,D:5,E:2,F:2,G:1,H:1	-
db^-	B:1,C:2,D:1,F:1,G:1	-
UD	A:2, B:9, C:5, D:7, E:6, F:8, G:1, H:2	**B:9, F:8, D:7, E:6, C:5**

【Example 1】Let the original database, *DB*, be illustrated in Table3. The minimum support is 0.2 and the pre-minimum support is 0.15. Scan *DB* once to collect the set of frequent or pre-frequent items: A:2, B:6, C:5, D:3, E:4, F:7, G:1, and H:1 (:indicates the support count). The items with support counts no less than 2 (i.e. 13×0.15=1.95) are frequent or pre-frequent items in *DB*. Thus, A, B, C, D, E, and F are frequent items in *DB*. After sorting all the frequent or pre-frequent items in support descending order, the result is F:7, B:6, C:5, E:4, D:3, and A:2. Accordingly, the constructed FP-tree of DB is shown as Figure 1.

Then 5 transactions are inserted into and 3 transactions are removed from DB, where the transaction data are shown as Figure 2. The support counts of all the items in db^+ and db^- are listed in Table 4. For each item *X* in *UD*, Sup_X^{UD} can be obtained by a simple computation. The result is A:2, B:9, C:5, D:7, E:6, F:8, G:1, and H:2. In new database *UD*, a pre-frequent or frequent item must have support counts no less than 3 (i.e. (13+5-3)×0.15=2.25). Therefore, the frequent or pre-frequent 1-itemsets in *UD*, shown in frequency descending order, are B:9, F:8, D:7, E:6, and C:5.

In the following, this database example will be used to explain the three steps for adjusting FP-tree.

3.2.1 Remove nodes of infrequent items

Suppose an item *I* becoming infrequent in *UD*, the nodes representing *I* have to be removed from FP-tree. Starting from item *I*'s head in the FP-tree header and following *I*'s node-links, all the nodes representing *I* can be obtained one by one. For each node N such that N.*item-name* = *I*, set its children nodes to be the children of its parent node and remove N from *I*'s node-links. Finally, remove the entry of *I* in the header table.

【Example 2】As shown in Table 4 , A is not a frequent or pre-frequent item in *UD*. Thus, the nodes representing A are removed from the FP-tree. The resultant FP-tree is shown as Figure 3.

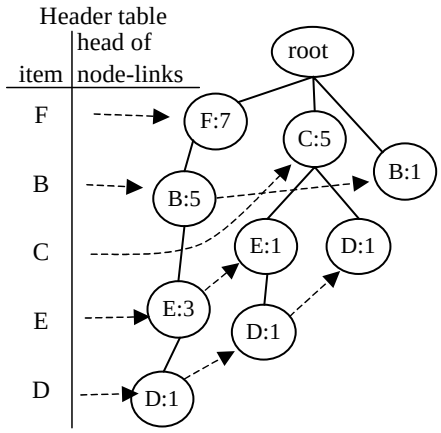


Figure 3: FP-tree after deleting nodes of infrequent items

Before adjusting：F	B	C	E	D	(in support descending order in *DB*)
	B	**F**	C	E	D	(exchange F and B)
	B	F	**E**	**C**	D	(exchange C and E)
	B	F	E	**D**	**C**	(exchange C and D)
	B	F	**D**	**E**	C	(exchange E and D)
After adjusting：	B	F	D	E	C	(in support descending order in *UD)*

Figure 4

3.2.2 Adjusting the path of nodes

Given a list of frequent or pre-frequent items in support descending order in *DB*, bubble sort algorithm is applied to decide the way for exchanging the adjacent items to meet the support descending order in *UD*. As shown in Figure 4, 4 times of item exchange is necessary to adjusting the support descending order in *DB* to fit in with the order in *UD*.

After deciding the pairs of items to be exchanged, in FP-tree of *DB*, the paths containing a pair of exchanged items have to be adjusted. The adjusting method is described as the following.

【Path adjusting method】

Suppose there exists a path in FP-tree of *DB*, where node Y is a children node of node X and the items represented in node X and Y have to be exchanged. In addition, node P is the parent node of X. Perform step 1 to 3 if X.count is greater than Y.count. Otherwise, perform step 2 to 3.

【step 1】Insertion：Insert a children node X' of node P, where X'.count is set to be X.count – Y.count. All the children nodes of X except node Y are assigned to be the children nodes of X'. In addition, X.count is reset to be equal to Y.count.

【step 2】Exchange：Exchange the parent link and children links of node X and node Y, respectively.

【step 3】Merge：Check whether there exists a children node of P, denoted as node Z, such that Z.item-name = Y.item-name. If node Z exists, Z.count is summed into Y.count and node Z is removed.

【Example 3】As shown in Figure 4, 4 iterations of item exchange are necessary to be performed on the resultant FP-tree shown in Figure 3. In the figures of this example (Figure 5 to 8), for simplifying the content in a figure, only the node links of the two exchanged items are shown.

- The first iteration of adjustment: Exchange nodes of items F and B.

  From the Header table of FP-tree, item F's node links can be obtained. Let the first node, that carries item name F, be denoted as node X, as shown in Figure 5(a). There

exists a children node Y of X, such that Y.item-name is B (as shown in Figure 5(a)). Thus, the adjusting method is performed.

(1) Insertion: For the parent node of X, root node in this case, insert a children node X' carrying item name F. The count field of X' is assigned to be X.count – Y.count, 2(i.e. 7-5). Besides, X.count is reassigned to be equal to Y.count, 5. Finally, note X' is inserted into the node-links of F, as shown in Figure 5(b).

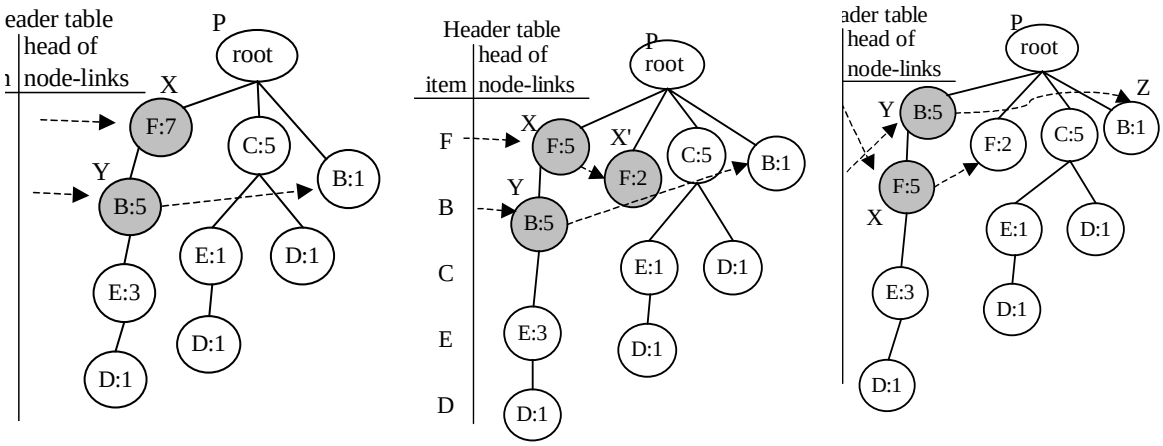
(2) Exchange: Exchange the parent link and children links of node X and node Y, respectively. The resultant tree structure is shown in Figure 5(c).

(3) Merge: There exists another children node Z of root node, such that Z carrying the same item name, B, with node Y. Thus, the count value registered in node Z is merged into node Y and Z is removed. The result is shown in Figure 5(d).

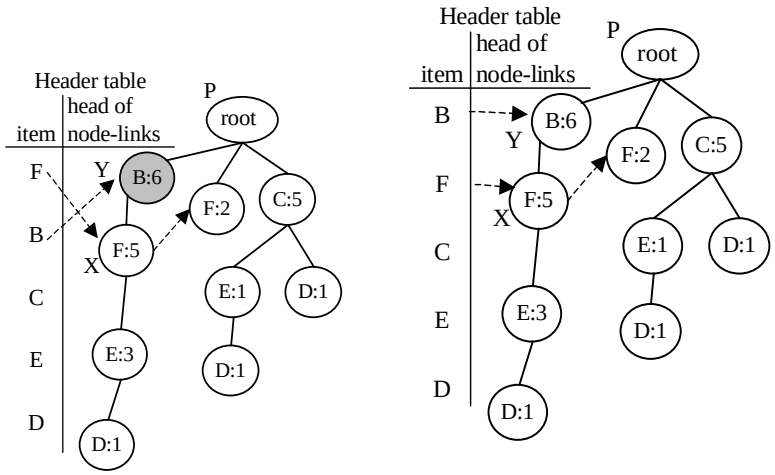
Get the next node carrying item name F. However, this node has no children node carrying item name B. Since it has reached the terminal of the node-links of item F, this iteration is complete. Finally, the entries of items F and B in Header table are exchanged as shown in Figure 5 (e).

- The second iteration of adjustment: Exchange nodes of items C and E.

1. Similar to the process of first iteration, the first node carrying item name C, be denoted as node X, and its children node Y carrying item name E are found(as shown in Figure 6(a)). Then the adjusting method is performed. In this case, node X has another children node S except node Y. Accordingly, node S is assigned to be the children node of the new added node X'. Finally, the entries of items C and E in Header table are exchanged The resultant FP-tree is shown in Figure 6(b).

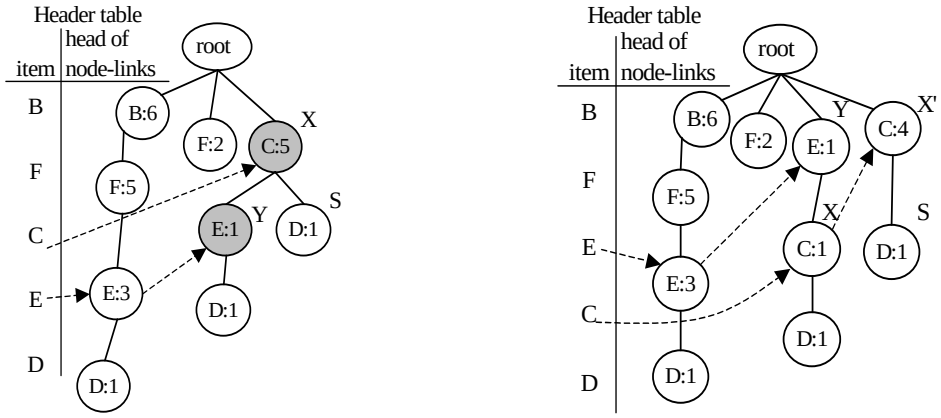


(a) (b) (c)



(d) (e)

Figure 5



(a) (b)

Figure 6

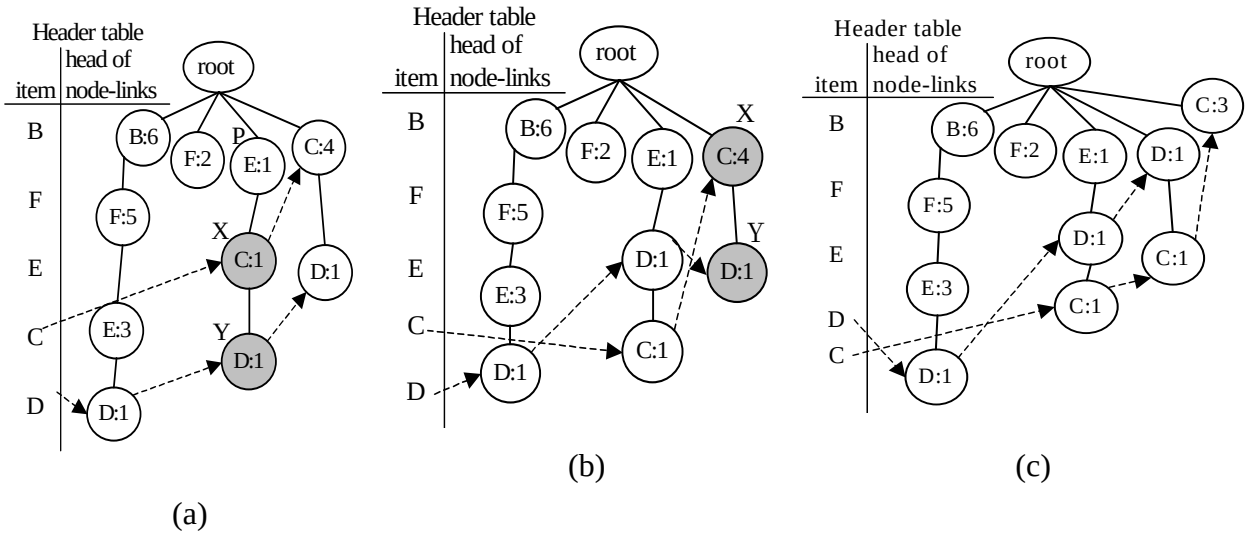


Figure 7

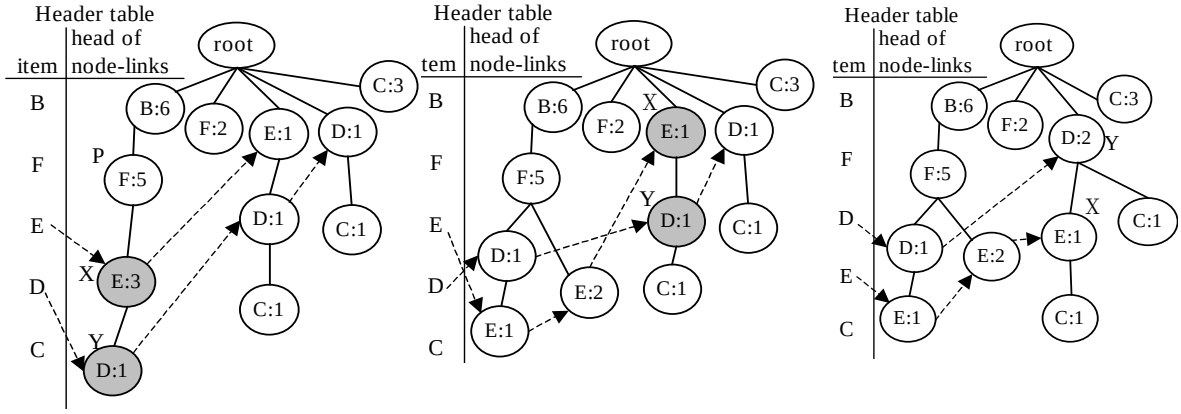
- The third iteration of adjustment: Exchange nodes of items C and D.

  The first node carrying item name C and its children node Y carrying item name D are found, which are denoted as node X and Y, respectively (as shown in Figure 7(a)). In this case, X.count is equal to Y.count. Thus, only step 2 and step 3 of adjusting method is performed. The result is shown in Figure 7(b).

  The next pair of nodes representing items C and D, denoted as node X and Y, are indicated in Figure 3.11(b). After performing the adjusting method and exchanging the entries of items F and B in Header table, the result is as shown in Figure 7(c).

- The fourth iteration of adjustment: Exchange nodes of items E and D.

  There are two pairs of nodes representing items E and D, which are indicated by node X and node Y in Figure 8(a) and Figure 8(b), respectively. After performing adjusting method for first pairs of nodes, the result is shown as Figure 8(b). The final result of adjustment in this iteration is shown in Figure 8(c).



(a) (b) (c)

Figure 8

3.2.3 Insert or remove data from FP-tree

After adjusting the path of nodes in FP-tree structure, each path in the resultant FP-tree follows the frequency descending order of frequent or pre-frequent items in *UD*. For each transaction T in db^+, the frequent or pre-frequent items of *UD*, which is contained in T, is selected and sorted in frequency descending order. Then transaction T is inserted into FP-tree as the same procedure for constructing FP-tree. Similarly, each transaction T in db^- is removed from FP-tree. Function insert_delete_tree (Program 1) is used to perform this step of task.

According to the result of example 3, function insert_delete_tree is called to inserte the transactions in db^+ and remove the transactions in db^- (shown in Figure 9). The resultant FP-tree is shown in Figure 10, which is the same with the reconstructed FP-tree of *UD*.

Function insert_delete_tree：

Input：(1)The root node R of FP-tree and Header table HT

(2) db^+, db^-

Output：FP-tree after inserting the transactions in db^+ and removing the transactions in db^-

```
For ((each transaction T in db+ ) and (each transaction T in db-))
    Select and sort the frequent or pre-frequent items in T
        according to the support descending order in UD.
    Let the sorted item list be ⟨I1, I2, ···, Ik⟩
    P:=R;
    For j:=1 to k
        Q:= the first children node of P
        Found:=False
        While (Q≠null  and  Found=False)
            If  Q.item = Ij
                Found:=True
                If  T ∈ db+
                    Q.count :=Q.count +1
                Else  /* T ∈ db-  */
                    Q.count :=Q.count -1
                    If  Q.count =0
                        Assign Q's children nodes to be children of node P
                        remove node Q
                    End If
                End else
            End If
            Else Q:= the next children node of P
        End While
        If  (T ∈ db+ and Q=null )
            Insert a node Q;
            Q.item :=  Ij
            Q.count :=1
        End If
        P:=Q
    End For
End for
```

Program 1

TID	Itemset	Frequent or Pre-frequent items in *UD* (in frequency descending order)
14	BCDEF	BFDEC
15	BDEF	BFDE
16	BCDG	BDC
17	BD	BD
18	DH	D

TID	Itemset	Frequent or Pre-frequent items in *UD* (in frequency descending order)
5	BF	BF
8	CG	C
12	CD	DC

Figure 9

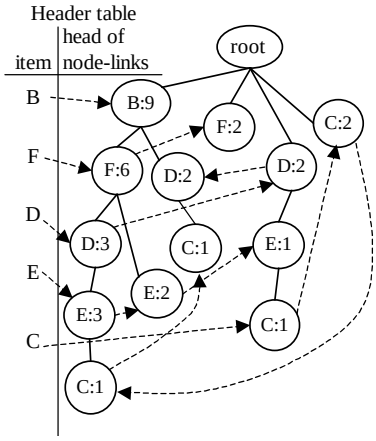


Figure 10: FP-tree of *UD*

3.3 AFPIM Algorithm

Based on the above strategies described, we have the following AFPIM algorithm to adjust FP-tree structure for incremental mining frequent item sets.

【step 1】Read in the items in *DB* and their support counts in *DB*.

【step 2】Scan db^+ and db^- once to list all items and their support counts in db^+ and db^-, individually. For each item X, compute the support count of X in *UD* according to the formula:

$$Sup_X^{UD} = \frac{Count_X^{DB} + Count_X^{db^+} - Count_X^{db^-}}{|UD|}.$$

Then collect the frequent or pre-frequent items in *UD*.

【step 3】Judge whether if all the frequent items of *UD* are covered in FP-tree of *DB*.

【step 3.1】If there exists a frequent item of *UD* not in the FP-tree, the whole *UD* needs to be scanned once to reconstruct a FP-tree of *UD* according to the frequent or pre-frequent items in *UD*.

【step 3.2】Otherwise, read in the stored FP-tree of *DB*.

【step 3.2.1】For each infrequent item in *UD*, remove the corresponding nodes from the FP-tree.

【step 3.2.2】According to the support descending orders in *DB* and *UD*, decide the pairs of items to be exchanged by applying bubble sort algorithm. Then the path adjusting method is used repeatedly to adjust the structures of paths in the FP-tree.

【step 3.2.3】Scan db^+ and db^- the second time. Call function tree_insert_delete to insert transactions of db^+ into and delete transactions of db^- from the FP-tree. Finally, the resultant FP-tree of *UD* is obtained.

【step 4】Apply FP-Growth algorithm [7]and TD-FP-Growth algorithm[14], respectively, to find out frequent item sets in *UD* from FP-tree of *UD*.

4. Performance evaluation

To evaluate the performance of AFPIM algorithm, the algorithms AFPIM, FUP[4], FUP2[6], and UWEP[2] are implemented on a personal computer running Microsoft Windows 98. In the following, AFPIM:B_U implies that the step for mining frequent itemsets from FP-tree (step 4) in algorithm AFPIM is implemented according to FP-Growth[7] algorithm. Similarly, AFPIM:T_D means that the step is implemented according to TD-FP-Growth[14] algorithm. We first show the improvement of AFPIM over FUP[4], FUP2[6], and UWEP[2], and then demonstrate the performance of AFPIM

by comparing with FP-Growth[7] and TD-FP-Growth[14], which are the most efficient non-incremental algorithms for mining frequent itemsets.

4.1 Synthetic data generation

The experiments were performed on synthetic data generated using the same technique as in [1]. The parameters used are similar to those in [1] except the size of the changed part of the database (d+ or d-), which are listed in Table 4. In the following, Tx.Iy.Dm.dn is used to denote that |T|=x, |I|=y, |D|=m×1000, and |d|=n×1000.

The updated database is created as follows: we generate |D|+|d+| thousand transactions, of which the first |D| thousand transactions are used as *DB*, and the remaining |d+| thousand transactions are used as the increment. The first |d-| thousand transactions in *DB* are considered as deleted ones. The value of pre-minimum support used in AFPIM is 0.2% less than the value of minimum support.

Table 4: The parameters

|T|	Average size of the transactions
|I|	Average size of the potentially frequent itemsets
|D|	Number of transactions
|d+|/|d-|	Number of inserted/deleted transactions
|N|	Number of items

4.2 Comparison of AFPIM and other incremental mining algorithms

We generate databases for the following three experiments by setting |N|=100.

【Experiment 1】Effects of the minimum support on algorithms AFPIM, FUP, and UWEP.

The three algorithms are tested against the setting T10.I10.D100.d+10. The value of minimum support is varied between 1% and 3%. The results are plotted in Figure 11(a).

As shown in Figure 4.1, AFPIM:T_D is 7.53 times faster than FUP and 1.47 times faster than UWEP when the value of minimum support is 3%. When the value of

minimum support is decreased to 1%, AFPIM:T_D is 20.27 times faster than FUP and 8.82 times faster than UWEP. In general, the smaller the minimum support is, the larger the speed-up ratio of AFPIM over FUP and UWEP is. The reason is that a small minimum support will induce a large number of frequent itemsets. This factor affects the computation cost of AFPIM least.

【Experiment 2】Effects of the minimum support on algorithms AFPIM and FUP2.

We perform an experiment on the dataset T10.I4.D100.d+10.d-10. The value of minimum support is varied between 1% and 6%. The experiment results are shown in Figure 11(b), in which AFPIM always has a better performance than FUP2. Moreover, similar to the results shown in the previous experiment, the speed-up ratio of AFPIM over FUP2 is more significant when the minimum support is smaller.

【Experiment 3】Effect of the size of updates on AFPIM and FUP2.

This experiment is to find out how the size of db^+ and db^- affects the performance of the two algorithms. We use the setting T10.I4.D100.d+n.d-n for the experiment, with a minimum support of 4%. That is, an initial database consisting of 100 thousand transactions is used. From this database, n thousand transactions are deleted and another n thousand are added to it. Figure 11(c) shows that AFPIM is 10 to 21 times faster than FUP2 for $10 \leq n \leq 50$. In addition, the speed-up ratio increases as the size of updates increases.

4.3 Comparison of AFPIM and non-incremental mining algorithms

We compare the execution time of AFPIM algorithm with respect to the time of running FP-Growth and TD-FP-Growth on the whole data set in *UD*.

【Experiment 4】Scale-up experiment on AFPIM and FP-tree non-incremental mining algorithms.

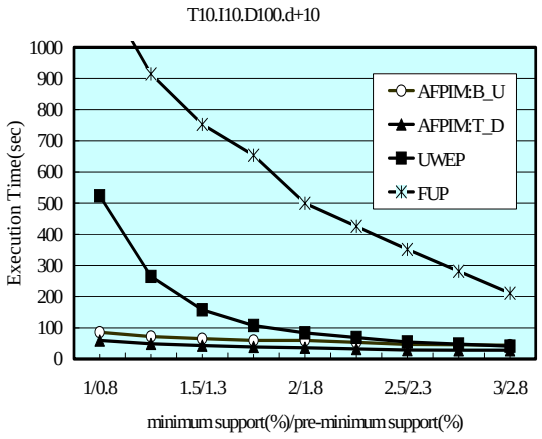


Figure 11(a)

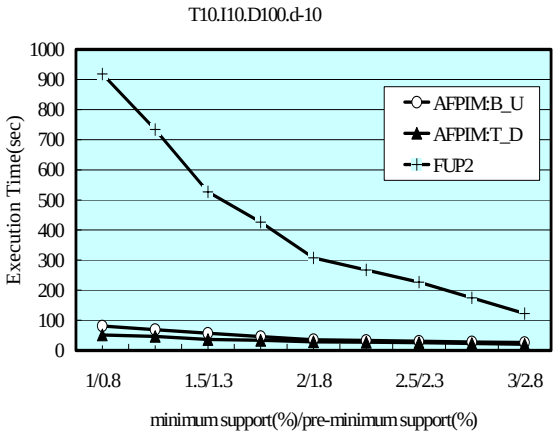


Figure 11(b)

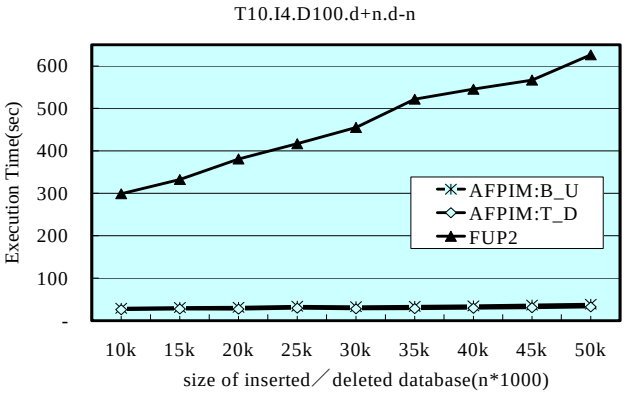


Figure 11(c)

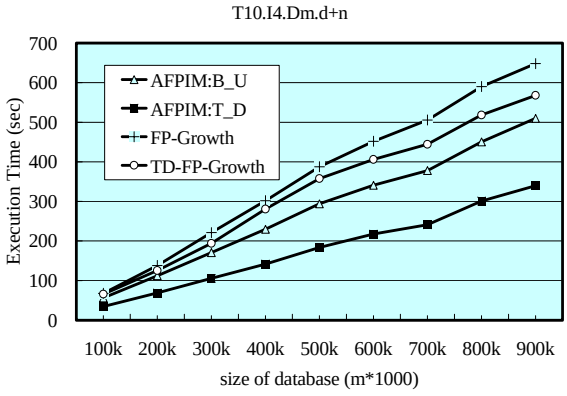


Figure 11(d)

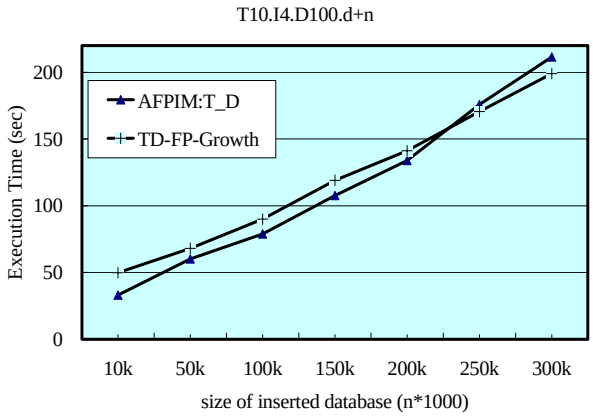


Figure 11(e)

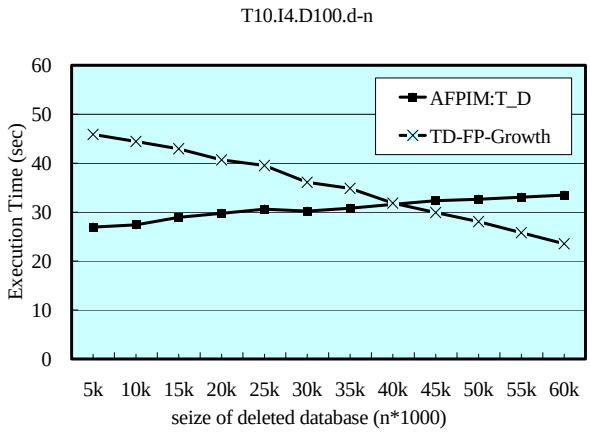


Figure 11(f)

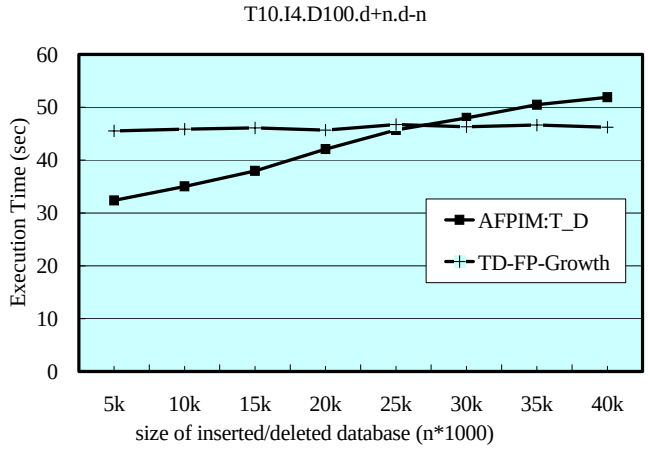


Figure 11(g)

To find out if AFPIM work also for large databases, experiments with scale-up databases are conducted. We use the setting T10.I4.Dm.d+$\frac{m}{10}$. The minimum support is set to 1%. The results are shown in Figure 11(d). The execution times of all the four algorithms increase linearly as m increases. In general, AFPIM is 2 times faster than re-running FP-tree mining algorithms.

【Experiment 5】Effect of the size of updates on AFPIM:T_D and TD-FP-Growth algorithm.

Next, we examine how much the size of the changed part in the database is the boundary that

FPIM:T_D outperforms the non-incremental mining algorithm TD-FP-Growth. Three series of experiments are conducted. We set T10.I4.D100.d+n for the insertion case, T10.I4.D100.d-n for the deletion case, and T10.I4.D100.d+n.d-n for the update case. Besides, |N| is set to 1000 and the minimum support is set to 1%.

In the insertion case, we increase the number of inserted transactions from 10 thousands to 300 thousands. Figure 11(e) shows that algorithm AFPIM: T_D has a better performance than re-running TD-FP-Growth algorithm in database ($DB+db^+$) when the number of the inserted transactions is less than 2.25 times of the size of the original database.

In the deletion case shown in Figure 11(f), the number of deleted transactions is increased from 5 thousands to 60 thousands. The algorithm AFPIM: T_D outperforms re-running TD-FP-Growth algorithm in database (DB-db^-) when the number of the deleted transactions is less than 40% of the size of the original database.

In the update case, the number of inserted transactions equals to the number of deleted transactions, which is increased from 5 thousands to 40 thousands. Figure 11(g) shows that, when the sum of the numbers of the inserted and deleted transactions is limited to 50% of $|DB|$, algorithm AFPIM: T_D is more efficient than re-running TD-FP-Growth algorithm in database (DB+ db^+-db^-).

5. Conclusion

In this paper, an efficient and general incremental updating technique is proposed for maintaining association rules discovered in transaction databases. This technique updates the association rules when old transactions are removed from the database or new transactions are added. It uses the FP-tree structure constructed from a previous mining to reduce the possibility of re-scanning the updated database. The FP-tree structure of the updated database is obtained, by adjusting the previous FP-tree according to the inserted and deleted transactions, to discover the corresponding association rules. Performance studies show that the proposed AFPIM algorithm is significantly faster than the related incremental mining algorithms. This technique works well over wide ranges of system parameter values. In particular, it works well for small minimum support setting.

Reference

[1] R. Agrawal and R. Srikant, “Fast Algorithm for Mining Association Rule in Large Databases,” in Proc. of The 20th International Conference on Very Large DataBases, 1994.

[2] N.F. Ayan, A.U. Tansel, and E. Arkun, “An Efficient Algorithm to Update Large Itemsets with Early Pruning,” in Proc. of the 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, August, pages 287-291, 1999.

[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, 1996

[4] D.W. Cheung, J. Han, V.T. Ng, and C.Y. Wong, “Maintenance of Discovered Association Rules in Large Databases: An Incremental Update Technique,” in Proc. of the 12th International Conference on Data Engineering (ICDE'96), February, pages106-114, 1996.

[5] D.W. Cheung, V.T. Ng, and B.W. Tam, “Maintenance of Discovered Knowledge: A Case in Multi-level Association Rules,” in Proc. of 2nd International Conference on Knowledge Discovery and Data Mining, pages

[6] D.W. Cheung, S.D. Lee, and Benjamin Kao, “A General Incremental Technique for Maintaining Discovered Association Rules,” in Proc. of the 5th International Conference on Database Systems for Advanced Applications, April 1-4, pages 185-194, 1997.

[7] J. Han, J. Pei, and Y. Yin, “Mining Frequent Patterns without Candidate Generation,” in Proc. of the ACM SIGMOD Int. Conf. on Management of Data, pages 1-12, Dallas, Texas, USA, 2000.

[8] G. Lee, K.L. Lee and A.L.P. Chen, “Efficient Graph-Based Algorithms for Discovering and Maintaining Association Rules in Large Databases, ” Knowledge and Information Systems, Springer-Verlag, Vol. 3, pages 338-355,

[9] J.S. Park, M.S. Chen, and P.S. Yu, “An Effective Hash-based Algorithm for Mining Association Rules,” in Proc. of the ACM SIGMOD International Conference on Management of Data (SIGMOD'95), May, pages 175-186,

[10] S. Parthasarathy, M.J. Zaki, M. Ogihara, S. Dwarkadas, “Incremental and Interactive Sequence Mining,” In Proc. of the 8th International Conference on Information and Knowledge Management (CIKM99), pages 251-258, Kansas City, MO, November 1999.

[11] P.S.M. Tsai, C.C. Lee, and A.L.P. Chen, “An Efficient Approach for Incremental Association Rule Mining,” in Third Pacific-Asia Conference, PAKDD-99 Proceedings, pages 74-83, 1999.

[12] S. Thomas, S. Bodagala, K. Alsabti, and S. Ranka, “An Efficient Algorithm for the Incremental Updation of Association Rules in Large Databases,” in Proc. of 3rd International conference on Knowledge Discovery and Data Mining (KDD 97), New Port Beach, California, August, pages 263-266, 1997.

[13] 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, Orlando, Florida,

[14] K Wang, L. Tang, J. Han, and J. Liu, “Top down FP-Growth for Association Rule Mining, ” to appear in the 6th Pacific Area Conference on Knowledge Discovery and Data Mining, May 6-8, Taipei, Taiwan, PAKDD-2002.