Data & Knowledge Engineering 96–97 (2015) 78–89

Contents lists available at ScienceDirect

Data & Knowledge Engineering

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

Editorial

Fast updated frequent-itemset lattice for transaction deletion

Bay Vo [a], Tuong Le [b,c,*], Tzung-Pei Hong [d,e], Bac Le [f]

[a] Faculty of Information Technology, Ho Chi Minh City University of Technology, Viet Nam
[b] Division of Data Science, Ton Duc Thang University, Ho Chi Minh City, Viet Nam
[c] Faculty of Information Technology, Ton Duc Thang University, Ho Chi Minh City, Viet Nam
[d] Department of Computer Science and Information Engineering, National University of Kaohsiung, Kaohsiung, Taiwan, ROC
[e] Department of Computer Science and Engineering, National Sun Yat-sen University, Kaohsiung, Taiwan, ROC
[f] Department of Computer Science, University of Science, VNU-Ho Chi Minh, Viet Nam

A R T I C L E I N F O

Available online 11 April 2015

Keywords:
Data mining
Frequent-itemset lattice
Transaction deletion

A B S T R A C T

The frequent-itemset lattice (FIL) is an effective structure for mining association rules. However, building an FIL for a modified database requires a lot of time and memory. Currently, there is no approach for updating an FIL with deleted transactions. Therefore, this paper proposes an approach for maintaining FILs for transaction deletion without rescanning the original database if the number of eliminated transactions is smaller than the threshold determined based on the pre-large and diffset concepts. A diffset-based approach is first used for fast building an FIL. Then, two proposed approaches (tidset-based and diffset-based) are used for updating the FIL with transaction deletion. The experiment was conducted to show that the diffset-based approach outperforms the tidset-based and the batch-mode approaches.

1. Introduction

Association rule (AR) mining [1,15,24] is an important problem, which attracts so much attention of scientists, in data mining and knowledge discovery. They have wide applications, such as basket data analysis, semantic web mining, text mining and so on. The traditional methods for mining ARs are divided into two phrases: (i) Mining frequent itemsets (FIs) from databases [5,7,8,18] and streaming databases [4,19] and (ii) mining ARs from FIs. According to the experiments, phase (ii) is easily implemented but it requires a lot of processing time. Recently, frequent-itemset lattices (FILs) and frequent-closed-itemset lattices (FCILs) have been proposed for effectively mining ARs [13,20,21,25]. Building FILs/FCILs takes longer than getting frequent (closed) itemsets, but generating ARs from FILs/FCILs is more efficient than doing so from frequent (closed) itemsets [17,20]. Therefore, mining ARs based on FILs/FCILs outperforms the traditional approach when both phases of mining are considered.

In practical applications, databases are typically modified, meaning that transactions are often inserted or deleted. For instance, inserted transactions will be added to database of the system when customers buy something. Besides, when customers return their orders or there are a number of errors in orders, those will be removed from the transaction databases. Therefore, mining ARs, frequent itemsets, class association rules and high utility patterns from modified databases [6,9,16,23] have attracted much research interest. Fast-UPdate (FUP) [3] is the first algorithm for mining ARs from incremental databases. FUP is an Apriori-based algorithm that generates candidates and repeatedly scans databases. Since then, methods based on FP-tree [9,10,12] and IT-tree [14] have been developed for databases with transaction insertion. Incremental mining from sequence databases has also been developed [2]. Some studies have considered transaction deletion [11]. However, there are no methods proposed for maintaining FILs with transaction deletion.

* Corresponding author at: Division of Data Science, Ton Duc Thang University, Ho Chi Minh City, Viet Nam.
E-mail addresses: bayvodinh@gmail.com (B. Vo), lecungtuong@tdt.edu.vn, tuonglecung@gmail.com (T. Le), tphong@nuk.edu.tw (T.-P. Hong), lhbac@fit.hcmus.edu.vn (B. Le).

http://dx.doi.org/10.1016/j.datak.2015.04.006
0169-023X/© 2015 Elsevier B.V. All rights reserved.

To deal with the problem of maintaining frequent itemsets for transaction modification, the pre-large concept is proposed to reduce the need for rescanning an original database. With this concept, the original database does not need to be rescanned if the number of deleted transactions or inserted transactions is equal to or less than a safety threshold, thus reducing the maintenance cost. The pre-large concept was later used by La et al. [13] and Vo et al. [21] for fast updating FCILs with transaction insertion.

This paper proposes an approach for maintaining FILs with transaction deletion. First, the proposed approach uses the DFIL algorithm based on the diffset concept to build FILs. Then, two methods for updating FILs (tidset-based and diffset-based methods) with transaction deletion are used.

The rest of this paper is organized as follows. Section 2 presents the basic concepts and an effective algorithm based on the diffset concept for building FILs. Two algorithms for maintaining FILs based on the tidset and the diffset concepts, respectively, with transaction deletion are proposed in Section 3. Section 4 presents the results of experiments that compare the run time of the proposed algorithms with that of the batch-mode approaches to show the effectiveness of the proposed algorithms. Finally, Section 5 summarizes the results and offers some future topics.

2. Basic concepts

2.1. Frequent-itemset lattice building algorithm

Given a database D with n transactions, with each transaction including a set of items belonging to I, where I is the set of all items in D. An example of a transaction database D_1 is presented in Table 1. The support of an itemset X, denoted by $\sigma(X)$, where $X \subseteq I$, is the number of transactions in D which contains all the items in X. An itemset X is called a frequent itemset if $\sigma(X) \geq minSup \times n$, where $minSup$ is a given threshold.

Vo et al. [21] proposed the DFIL algorithm for building FILs. It is summarized as follows.

Definition 1. Let $n(X)$ be a node of a k-itemset X. The child-nodes of $n(X)$ based on the equivalence class property associated with $n(X)$ are:

$$y_{EC}(n(X)) = \{n(XA) | \forall A \in I, A \notin X\} \tag{1}$$

Definition 2. Let X be a k-itemset. The child-nodes of $n(X)$ based on the lattice property associated with $n(X)$ are:

$$y_L(n(X)) = \{n(Y) | Y \text{ is a } (k+1)-\text{item set}, n(Y) \notin y_{EC}(n(X)) \text{ and } X \subset Y\} \tag{2}$$

Definition 3. Each node, $n(X)$, in the FIL is a tuple:

$$\langle \mathrm{X}, t(\mathrm{X}), y_{EC}(n(\mathrm{X})), y_L(n(\mathrm{X}))\rangle, \tag{3}$$

where:

- X is an itemset;
- $t(X)$ is the set of IDs associated with the transactions containing X; and
- $\gamma_{EC}(n(X))$ contains the child-nodes based on the equivalence class property associated with X.
- $\gamma_L(n(X))$ contains the child-nodes based on the lattice property associated with X.

Theorem 1. Let $n(XA)$ be a node of a k-itemset XA. $\forall n(XB) \in \gamma_{EC}(n(X))$, if A is before B in the order of frequent 1-itemsets (sorted in ascending order of frequency), then $\nexists n(Y) \in \gamma_{EC}(n(XB)) \cup \gamma_L(n(XB))$ so that $n(Y) \in \gamma_L(n(XA))$.

Theorem 2. Let $n(XA)$ be a node of a k-itemset XA. $\forall n(Z) \in \gamma_L(n(X))$, $\nexists\, n(Y) \in \gamma_L(n(Z))$ so that $n(Y) \in \gamma_L(n(XA))$.

To understand the application of Theorems 1 and 2, the process of updating a lattice when $n(XA)$ (a k-itemset) is created, is described below. The existing algorithms [22] have to consider all nodes of Y in the four cases shown in Table 2.

Table 1
Example of a transaction database D_1.

Transaction	Items
1	*A, C, T, W, V*
2	*C, D, W*
3	*A, C, T, W*
4	*A, C, D, W*
5	*A, C, D, T, W*
6	*C, D, T*

Table 2
Four cases considered when updating the lattice.

Case	Nodes of k-itemsets	Nodes of $(k+1)$-itemsets
1	$Z \in \gamma_{EC}(n(X))$	$Y \in \gamma_{EC}(n(Z))$
2	$Z \in \gamma_{EC}(n(X))$	$Y \in \gamma_{L}(n(Z))$
3	$Z \in \gamma_{L}(n(X))$	$Y \in \gamma_{EC}(n(Z))$
4	$Z \in \gamma_{L}(n(X))$	$Y \in \gamma_{L}(n(Z))$

However, from Theorems 1 and 2, for cases 1, 2 and 4 (Table 2), there are no node of k-itemsets, $n(Y)$, in the current lattice as the child nodes of $n(XA)$ based on the lattice property. Therefore, the building FIL algorithm can easily find the nodes which belong to $\gamma_L(n(XA))$. The process first visits all child nodes based on the lattice property of $n(X)$ ($n(Y) \in \gamma_L(n(X))$). With each $n(Y)$, the process then visits all children based on the equivalence class property of $n(Y)$ ($n(YB) \in \gamma_{EC}(n(Y))$). With each $n(YB)$, if $XA \subset YB$, the process only updates $n(YB)$ which belongs to children based on the lattice property of $n(XA)$ ($n(YB) \in \gamma_L(n(XA))$). Separating *Children* of a node X on FIL into $\gamma_{EC}(n(X))$ and $\gamma_L(n(X))$ makes the proposed algorithm better than the algorithms by Vo and Le [22,23] because it eliminates a large number of candidates in cases 1, 2, and 4 in Table 2.

Vo et al. [21] also used the diffset concept for fast building FILs. The diffset definition was first presented by Zaki and Gouda [24]. They proposed a method to fast determine the support associated with a k-itemset based on the support of the (k-1)-itemset. Let the tidsets associated with XA, XB and XAB be $t(XA)$, $t(XB)$ and $t(XAB)$, respectively. Then

$$t(XAB) = t(XA) \cap t(XB). \tag{4}$$

Let $d(XAB) = t(XA) \setminus t(XB)$ [24] be the TIDs that exist in $t(XA)$ but not in $t(XB)$. $d(XAB)$ is called the diffset associated with XAB. Let $d(XA)$ be the diffset of XA and $d(XB)$ be the diffset of XB, $d(XAB) = d(XB) \setminus d(XA)$. The support associated with XAB is determined as:

$$\sigma(XAB) = \sigma(XA) - |d(XAB)|. \tag{5}$$

Diffset associated with an itemsets XAB is the transaction IDs in the tidset associated with XA which do not exist in the tidset associated with XB. Therefore, diffset-based approaches required less memory than tidset-based approaches. Besides, computing the intersection between two tidsets consumes more time than computing the difference between two diffsets. In conclusion, diffset-based approaches are always better than tidset-based approaches.

Fig. 1 presents the diffset-based algorithm for building FILs.

The DFIL algorithm is applied to the example database in Table 1 with $minSup = 50\%$ to illustrate its use. First, DFIL finds all the frequent 1-itemsets and sorts them in ascending order of frequency. The result of this step is $I_1 = \{A, D, T, W, C\}$. The algorithm then uses the depth-first-search strategy to generate the candidates associated with each equivalence class. The frequent k-itemsets are then combined with the remaining k-itemsets in this equivalence class to create the $(k+1)$-itemset candidates. The frequent itemsets from these candidates are used to create the frequent $(k+2)$-itemsets. When each node X in the lattice is created, the algorithm calls the procedure *Update_Lattice* to update the child nodes based on the lattice property associated with X, which is created in the previous steps. For instance, the first frequent 1-itemset in I_1, A, is combined with the remaining frequent 1-itemsets $\{D, T, W, C\}$ to create the candidates $\{AD, AT, AW, AC\}$. However, AD is excluded because $\sigma(AD) = 2 < minSup \times n = 50\% \times 6 = 3$. The frequent 2-itemsets associated with the equivalence class A are thus $\{AT, AW, AC\}$. Then, the algorithm combines the first frequent 2-itemset in this list, AT, with the remaining frequent 2-itemsets in this list $\{AW, AC\}$ to create the frequent 3-itemsets $\{ATW, AWC\}$. At last, ATW is combined with AWC to create the frequent 4-itemset $\{ATWC\}$, finishing the processing of the equivalence class A. The algorithm then similarly processes the remaining equivalence class $\{D, T, W, C\}$. The results for the example database are shown in Fig. 2. Note that the dashed and solid lines represent child-nodes based on the lattice property and based on the equivalence class property, respectively.

2.2. Pre-large concept

The pre-large concept was proposed that based on a safety threshold f to reduce the need of rescanning the original database for efficiently maintaining association rules with transaction insertion. For transaction deletion, Hong et al. [10] proposed the following formula for determining the safety number f:

$$f = \left\lfloor \frac{(S_U - S_L) \times |D|}{S_U} \right\rfloor \tag{6}$$

where S_U is the upper threshold, S_L is the lower threshold, and $|D|$ is the number of original database D's transactions. When the number of deleted transactions is equal to or less than f, the algorithm does not need to rescan the original database.

When two thresholds are used, each itemset has three cases: frequent, pre-large, and infrequent. This divides itemsets in the original and new databases into nine cases [10], as presented in Table 3.

```
Input: transaction database $D$ with $n$ transactions and frequent threshold $minSup$
Output: lattice containing all frequent itemsets of $D$
1. Generate null node, $L_{root}$, as the root of the lattice
2. Derive the frequent 1-itemsets $FI_1$ from $D$
3. for each $A_i \in FI_1$, the algorithm adds $n(A_i) = \langle A_i, d(A_i)$, null, null$\rangle$ to $\gamma_{EC}(L_{root})$
4.   Enumerate_Lattice($FI_1$)
5.  return $L_{root}$

Procedure Enumerate_Lattice(the nodes at level k $FI_k$)
1. for each $I_i \in FI_k$ do
2.     let $FI_{k+1} \leftarrow \{\}$
3.     for each $I_j \in FI_k$ with j > i do
4.         $X = I_i \cup I_j$
5.         $d(X) = d(I_j) \setminus d(I_i)$
6.         $\sigma(X) = \sigma(I_i)$ - $|d(X)|$
7.         if $\sigma(X) \geq minSup \times n$ then
8.             add $n(X)$ to $\gamma_{EC}(I_i)$ and $\gamma_L(I_j)$
9.             add $X$ to $FI_{k+1}$
10.            Update_Lattice($I_i$, $X$)
11.    end for
12.    Enumerate_Lattice($FI_{k+1}$)
13. end for

Procedure Update_Lattice($n(P)$, $n(X)$)
1. for each Child $I_i$ in $\gamma_L(n(P))$ do
2.     for each Child $I_j$ in $\gamma_{EC}(I_i)$ do
3.         if $X \subset I_j$ then
4.             add $n(I_j)$ to $\gamma_L(X)$
```

Fig. 1. DFIL algorithm.

Cases 2, 3, 4, 7 and 8 do not affect the final large itemsets according to the weighted average of the counts. Case 1 may remove existing large itemsets, and cases 5, 6 and 9 may add new large itemsets. If all large and pre-large itemsets with their counts are retained after each pass, then cases 1, 5 and 6 can be easily handled. It has been theoretically shown that an itemset in case 9 cannot possibly be large enough to become a frequent itemset in the final updated database as long as the number of deleted transactions is smaller than *f*.

For example, consider the database in Table 1 with $|D_1| = 6$. Assume that $S_U = 50\%$ and $S_L = 40\%$. Then, if the number of deleted transactions is equal to or less than $f = \left\lfloor \frac{(0.5-0.4)\times 6}{0.5} \right\rfloor = 1$, the algorithm does not need to rescan the original database to determine the support of infrequent itemsets, which are mined from the original database.

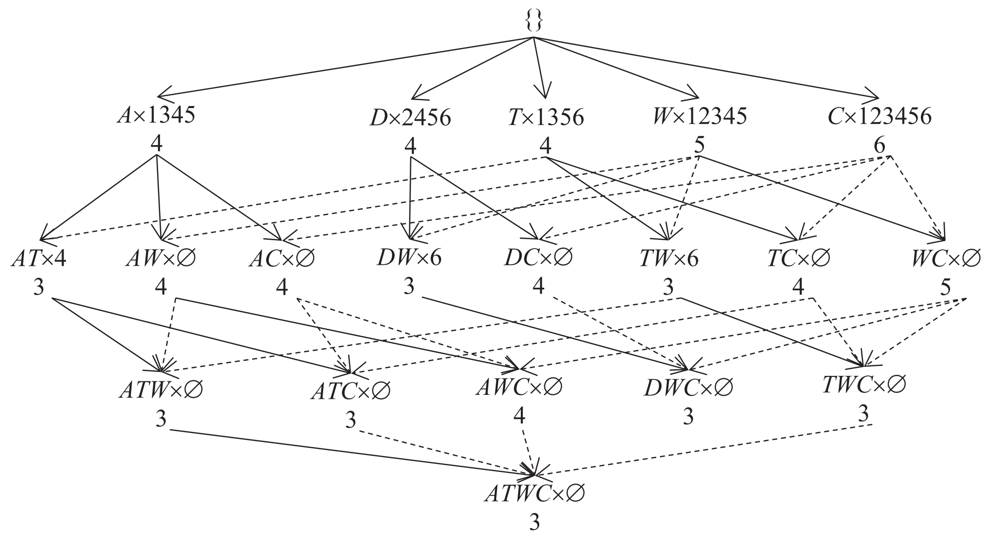

Fig. 2. Frequent-itemset lattice for database D_1 with $minSup = 50\%$ obtained by DFIL algorithm.

Table 3
Nine cases of itemsets for transaction deletion.

Case	Original – deleted	Results
1	Large – large	Large or pre-large or small, determined from existing information
2	Large – pre-large	Always large
3	Large – small	Always large
4	Pre-large – large	Pre-large or small, determined from existing information
5	Pre-large – pre-large	Large or pre-large or small, determined from existing information
6	Pre-large – small	Large or pre-large, determined from existing information
7	Small – large	Always small
8	small – pre-large	Always small
9	small – small	Pre-large or small, determined from existing information

3. Fast updated FIL algorithm for transaction deletion

Using the FIL building algorithm presented in Section 2, this paper proposes the tidset-based and the diffset-based approaches for updating FILs (TiFU-FIL and DiFU-FIL) with transaction deletion.

Remark 1. Let $n(XA)$ and $n(XB)$ be a node of a k-itemsets XA and XB in the lattice. If $d(XA)$ and $d(XB)$ are unchanged then $d(XAB)$ is unchanged, therefore, there is no need to update $n(XAB)$.

Based on Remark 1, the algorithm only considers the changed nodes which have deleted transactions.

3.1. Tidset-based approach (TiFU-FIL)

This section first introduced the proposed TiFU-FIL algorithm for updating FILs. It is presented in Fig. 3.

```
Algorithm: TiFU-FIL
Input:
- original database $D$
- safety threshold $f$ determined from $D$
- deleted transactions $D'$
- upper threshold $S_U$ and lower threshold $S_L$
- FIL
Output: Updated FIL
Method:
1. If the FIL is empty, the algorithm builds the FIL for $D'$ using $S_L$ and computes the safety threshold $f = \left\lfloor \frac{(S_U - S_L)\times|D'|}{S_U} \right\rfloor$;
2. If the number of transactions in $D'$ is larger than $f$, the algorithm calls the function FIL to build the FIL for $D - D'$ using $S_L$ and computes $f = \left\lfloor \frac{(S_U - S_L)\times(|D| - |D'|)}{S_U} \right\rfloor$;
3. The algorithm then
- clears the tidset information in each node in FIL;
- updates the node information in the first level of $L_1$;
- mark the nodes in $L_1$ whose information changes and whose supports satisfy $S_L$;
- calls the procedure UPDATE-FIL to update all the nodes in the FIL with $L_1$ as parameter;
- updates $f = f - |D'|$.
4. If $D$ is empty, then $D = D'$; otherwise, $D = D - D'$.

Procedure UPDATE-FIL($L_r$)
1.for each $l_i \in \gamma_{EC}(L_r)$ do
2.    if $l_i$ is marked then
3.        for each $l_j \in \gamma_{EC}(L_r)$, with $j > i$ do
4.            if $l_j$ is marked then
5.                let $O$ be the direct child node of $l_i$ and $l_j$
6.                if $O$ exists then
7.                    $t(O) = t(l_i) \cap t(l_j)$
8.                    if $|t(O)| > 0$ then
9.                        $\sigma(O) = \sigma(O) - |t(O)|$
10.                           if $\sigma(O) \geq S_L \times (|D| - |D'|)$ then
11.                               mark $O$
12.   UPDATE-FIL($l_i$)
```

Fig. 3. TiFU-FIL algorithm.

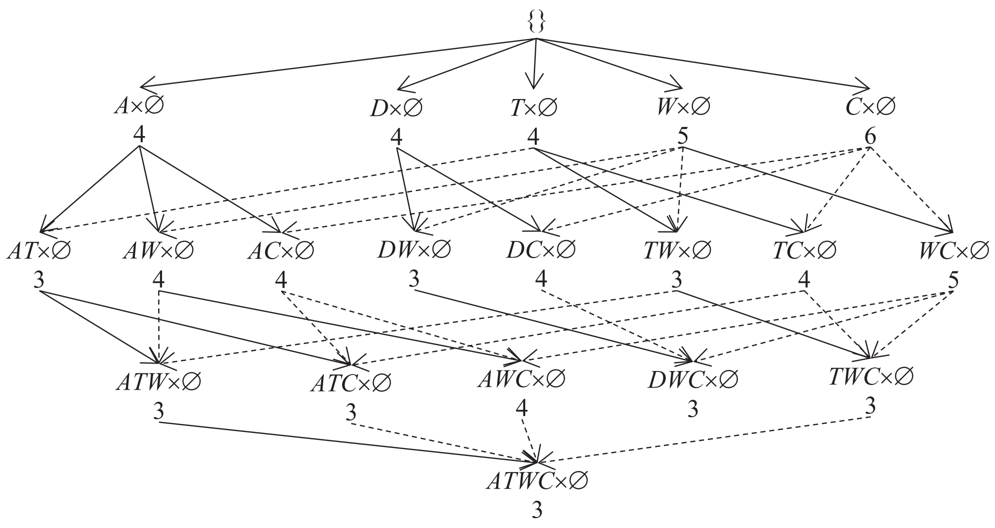

Fig. 4. The FIL after the first substep of step 3 for the second round of transaction deletion.

The complexity of TiFU-FIL algorithm depends on ΔD, the number of deletion transactions: (i) If $\Delta D \leq f$ then the complexity (in the worst case) of this algorithm is $O(n)$ where n is the number of nodes in the lattice. (ii) If $\Delta D > f$ then the complexity of this algorithm is equal the complexity of DFIL algorithm, with the worst case is $O(2^{|I|})$ where I is the number of items in database.

An example is given below to illustrate the process of TiFU-FIL in three rounds of transaction deletions with the initial database $D = \emptyset$, $S_L = 50\%$, and $S_U = 60\%$.

3.1.1. First round (D_1 with the six transactions in Table 1)

The algorithm builds the FIL for $S_L = 50\%$, as shown in Fig. 2, which computes $f = \left\lfloor \frac{(S_U - S_L) \times |D_1|}{S_U} \right\rfloor = \left\lfloor \frac{(0.6-0.5)\times 6}{0.5} \right\rfloor = 1$, and set $D = D_1$.

3.1.2. Second round (D_2 with transaction number 6 deleted)

Because the number of deleted transactions is equal to f, the algorithm skips step 2 and directly performs step 3 as follows.

1. The algorithm clears all the tidset information associated with all the nodes in the FIL (see Fig. 4).
2. The algorithm inserts the tidset information (only for deleted transactions) associated with the frequent 1-itemsets in the FIL and then marks the updated nodes (Fig. 5).
3. The algorithm recursively calls the procedure *UPDATE-FIL* in depth-first search to update the tidset information of all the nodes in the FIL. The result of this step is shown in Fig. 6.

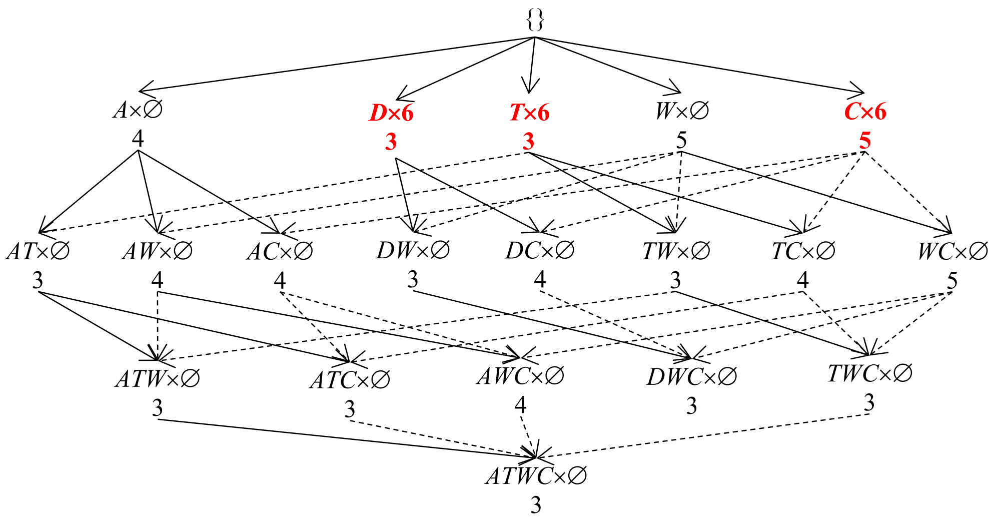

Fig. 5. The FIL after the second substep of step 3 for the second round of transaction deletion.

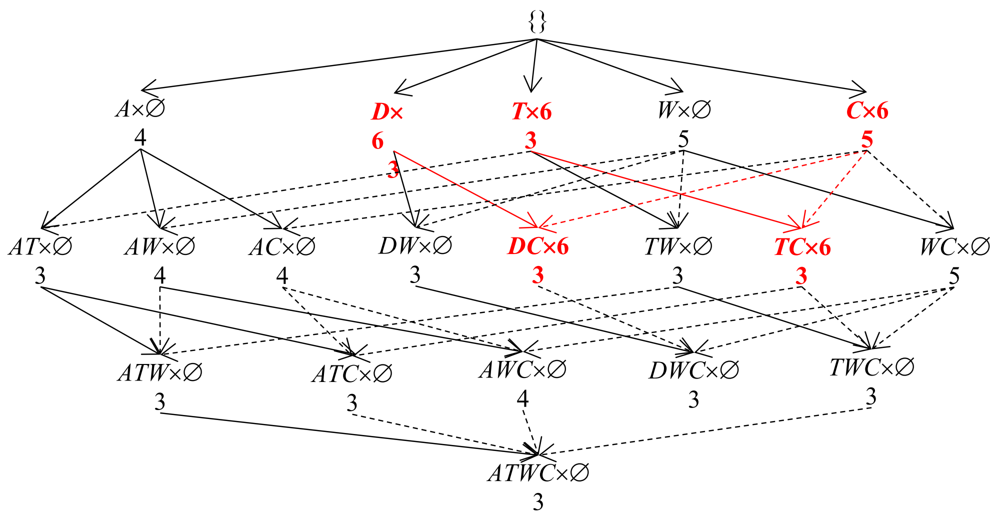

Fig. 6. The FIL after the third substep of step 3 for the second round of transaction deletion.

4. The algorithm updates the safety threshold $f = f - |D_2| = 1 - 1 = 0$.
5. The algorithm updates the database $D = D_1 - D_2$.

Fig. 6 shows that only a small number of nodes whose tidset information was updated in the FIL are used to update the lattice.

3.1.3. Third round (D_3 with transaction number 5 deleted)

Because $|D_3| = 1 > f = 0$, the algorithm performs step 2.

1. The algorithm calls the DFIL algorithm to create the FIL for D with only four transactions (1, 2, 3 and 4). The result of this step is shown in Fig. 7.
2. The algorithm calculates the safety threshold $f = \left\lfloor \frac{(S_U - S_L)\times|D|}{S_U} \right\rfloor = \left\lfloor \frac{(0.6-0.5)\times 4}{0.6} \right\rfloor = 0$. Therefore, for the fourth round of deletion, the algorithm always rebuilds the FIL.

Note that in actual applications, the number of transactions deleted is usually much less than that in a database. Thus, f is not zero in the case.

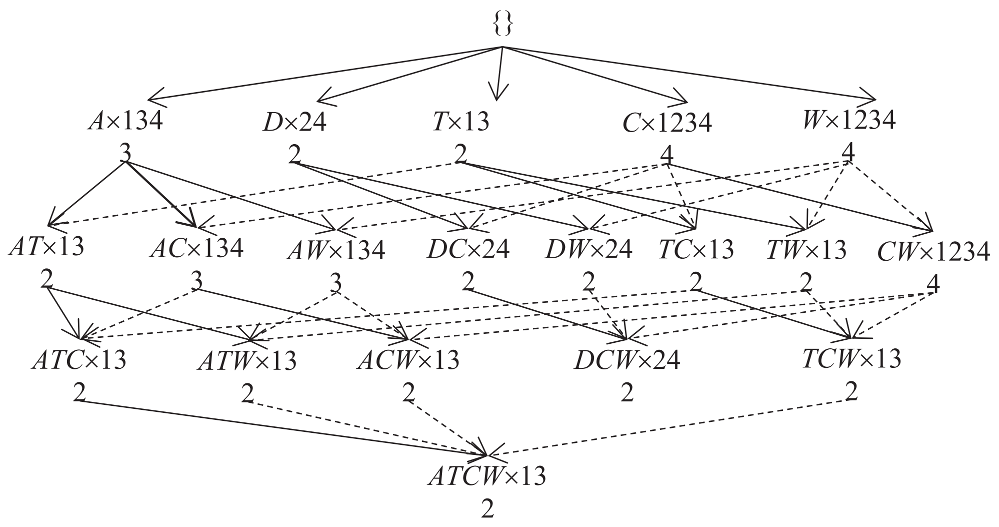

Fig. 7. The FIL after the first substep of step 2 for the third round of transaction deletion.

```
UDATE-PFIL ($L_r$)
1.for all $l_i \in \gamma_{EC}(L_r)$ do
2.  if $l_i$ is marked then
3.     for each $l_j \in \gamma_{EC}(L_r)$, with $j > i$ do
4.        if $l_j$ is marked then
5.           let $O$ be the directly child node of $l_i$ and $l_j$
6.           if $O$ exists then
7.              if level of $L_r$ is equal to 1 then
8.                 $d(O) = d(l_i) \setminus d(l_j)$
9.              else
10.                $d(O) = d(l_j) \setminus d(l_i)$
11.             $\sigma^T(O) = \sigma^T(l_i) - |d(O)|$
12.             if $\sigma^T(O) > 0$ then
13.                $\sigma(O) = \sigma(O) + \sigma^T(O)$
14.                if $\sigma(O) \geq S_L \times (|D| - |D'|)$ then
15.                   mark $O$
16.    UDATE-PFIL($l_i$)
```

Fig. 8. DiFU-FIL algorithm.

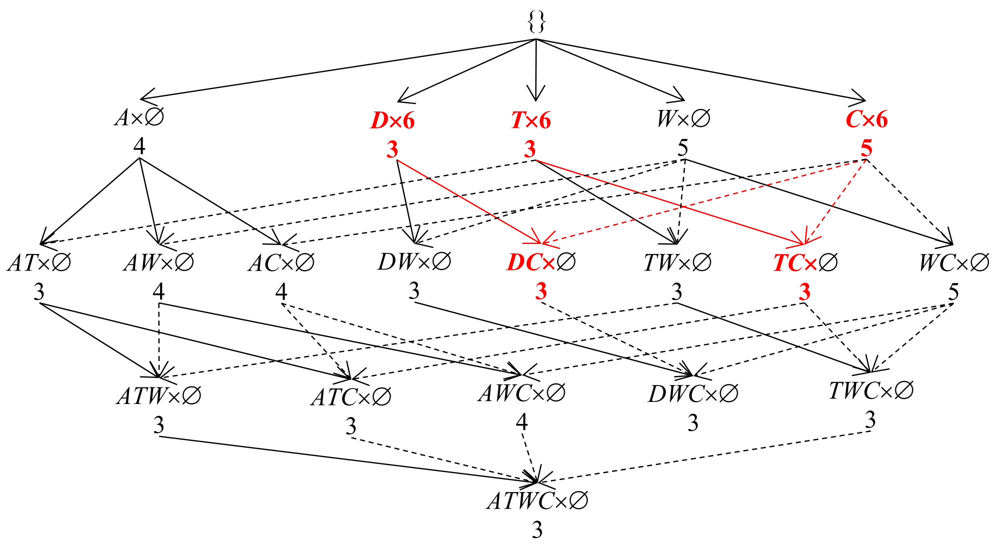

Fig. 9. FIL in step 3 for the second round of transaction deletion obtained using the DiFU-FIL algorithm.

3.2. Diffset-based approach (DiFU-FIL)

In this section, the diffset-based maintenance approach (DiFU-FIL) of an FIL is proposed (see Fig. 8). Note that only the procedure *UDATE-FIL* is given since the other procedures are the same as those in Fig. 3.

To use the diffset concept, each node in the lattice has a temporal support (σ^T) field. The σ^T field is associated with a node of a frequent 1-itemset and denotes the number of items in its tidset (only for deleted transactions). The σ^T value associated with a node of a frequent *k*-itemset *XAB* ($k > 1$) is determined using Eq. (6). It can be rewritten as follows:

$$\sigma^T(XAB) = \sigma^T(XA) - |t(XAB)|, \tag{7}$$

Table 4
Statistical summary of experimental databases.

Database	# of trans	# of items
Accidents	340,183	468
Mushroom	8,124	120
Pumsb_star	49,046	7,117
Retail	88,162	16,470

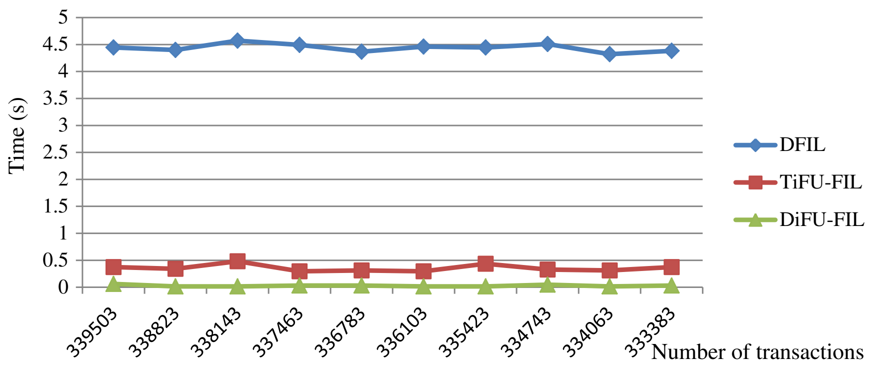

Fig. 10. Execution time of TiFU-FIL, DiFU-FIL ($S_U = 55\%$ and $S_L = 52\%$) and DFIL ($S_U = 55\%$) for Accidents.

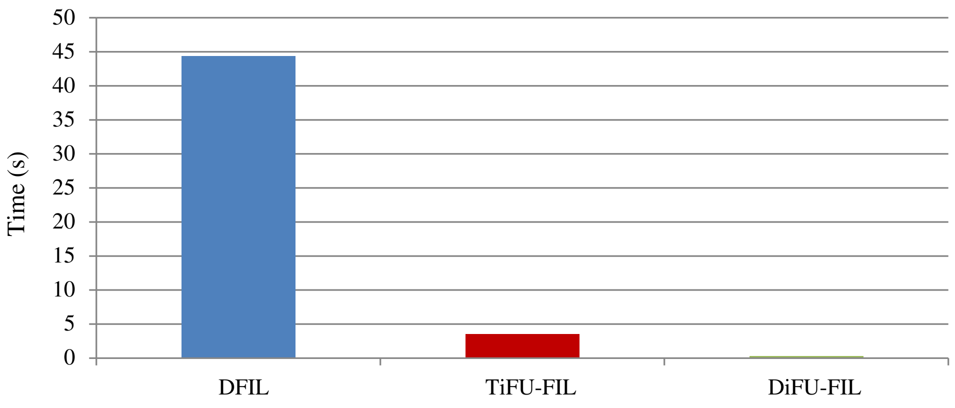

Fig. 11. Total time for ten runs of TiFU-FIL, DiFU-FIL ($S_U = 55\%$ and $S_L = 52\%$) and DFIL ($S_U = 55\%$) for Accidents.

where A is before B in the order of the counts of frequent 1-itemsets and $|t(XAB)|$ is the number of elements in the diffset of XAB.

To illustrate the DiFU-FIL algorithm, the FIL in step 3 for the second round of transaction deletions from Section 3.2 is shown in Fig. 9.

The resulting FIL in Fig. 9 is clearly better than that in Fig. 6 in terms of memory usage, which also speeds up the execution time. Hence, DiFU-FIL is more effective than TiFU-FIL.

4. Experimental results

All experiments presented in this section were performed on a laptop with an Intel i3 M380 2.53-GHz CPU and 2 GB of RAM running Windows 7. All the programs were coded in C#. The experiments were conducted using the following UCI databases[1]: Accidents, Mushroom, Pumsb_star, and Retail. A statistical summary of these databases is shown in Table 4.

Hong et al. [10] proposed the method for maintaining pre-large tree structure in incremental mining. Then, frequent itemsets were generated from the tree. Finally, association rules were mined from the above frequent itemsets. This is a traditional method to maintain association rules. On the other hand, this paper proposes an approach for maintaining the frequent-itemset lattice. This is a lattice-based approach for maintaining association rules. In this paper, we focus on the comparison of lattice-based approaches.

In this section, the total execution time of the TiFU-FIL, DiFU-FIL and DFIL algorithms are compared. Note that TiFU-FIL and DiFU-FIL use an upper threshold S_U and a lower threshold S_L. DFIL uses only an upper threshold S_U to build the FIL.

Fig. 10 compares the total run time of DFIL, TiFU-FIL and DiFU-FIL for the Accidents database. The results show that the time required for FIL maintenance is smaller than that for building an FIL using a batch mode. For 10 transaction deletions, the total runtime for DFIL, TiFU-FIL and DiFU-FIL are 44.389, 3.557 and 0.281 s, respectively (Fig. 11).

Fig. 12 compares the total run time of DFIL, TiFU-FIL and DiFU-FIL for the Mushroom database. Then, Fig. 13 shows the total runtime for 10 transaction deletion. The total runtime of DFIL, TiFU-FIL and DiFU-FIL are 4.282, 0.594 and 0.468 s, respectively.

Similar results were obtained for the Pumsb_star and Retail databases (Figs. 14 to 17). Updating the lattice with transaction deletion is always faster than using batch mode, and the diffset-based approach is more efficient than the tidset-based approach.

[1] http://fimi.cs.helsinki.fi/data/.

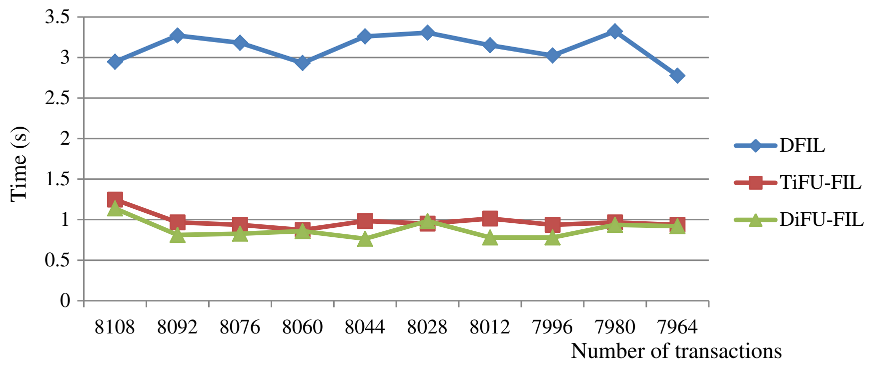

Fig. 12. Execution time of TiFU-FIL, DiFU-FIL ($S_U = 10\%$ and $S_L = 9\%$) and DFIL ($S_U = 10\%$) for the Mushroom database.

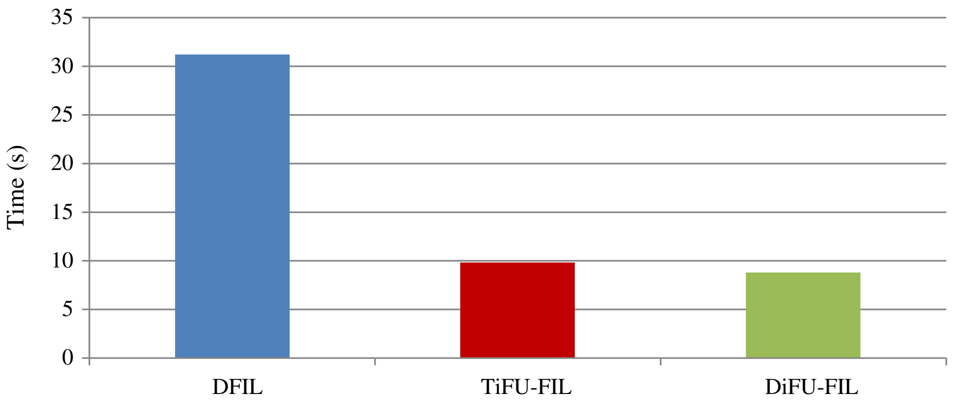

Fig. 13. Total time for ten runs of TiFU-FIL, DiFU-FIL ($S_U = 10\%$ and $S_L = 9\%$) and DFIL ($S_U = 10\%$) for the Mushroom database.

5. Conclusions and future work

This study proposes two approaches (tidset-based and diffset-based) for maintaining FILs with transaction deletion. Firstly, DFIL is used for fast building an FIL from a database. After that, when some transactions are deleted, TiFU-FIL or DiFU-FIL is used for updating the FIL. The pre-large concept is used in TiFU-FIL/DiFU-FIL to update the FIL without rescanning the original database if the number of deleted transactions is smaller than or equal to the calculated safety threshold *f*. The experimental results show that the two approaches outperform the batch-mode algorithm in building FILs, with the diffset-based approach (DiFU-FIL) being more efficient than the tidset-based approach (TiFU-FIL).

In future work, we will study how to store more information to avoid rescanning the original database. Besides, using a frequent-closed-itemset lattice (FCIL) for generating association rules is very effective. Therefore, a method for updating FCILs with transaction deletion will be developed.

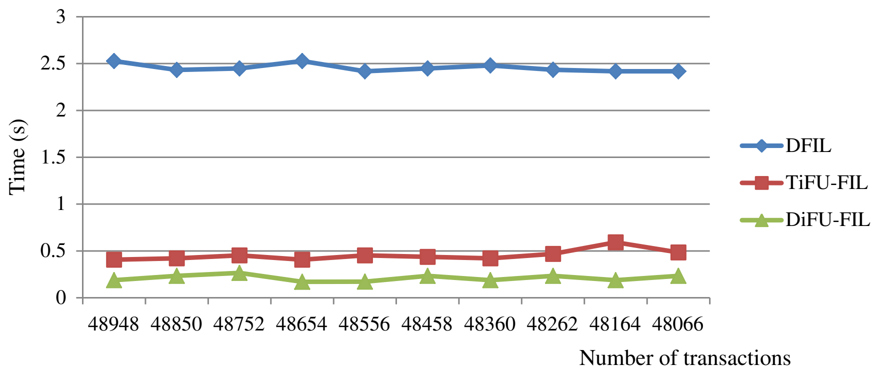

Fig. 14. Execution time of TiFU-FIL, DiFU-FIL ($S_U = 40\%$ and $S_L = 38\%$) and DFIL ($S_U = 40\%$) for the Pumsb_star database.

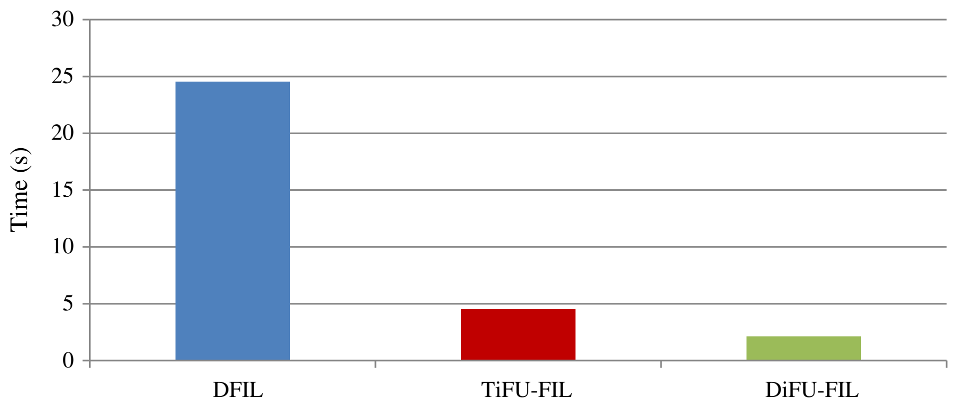

Fig. 15. Total time for ten runs of TiFU-FIL, DiFU-FIL ($S_U = 40\%$ and $S_L = 38\%$) and DFIL ($S_U = 40\%$) for the Pumsb_star database.

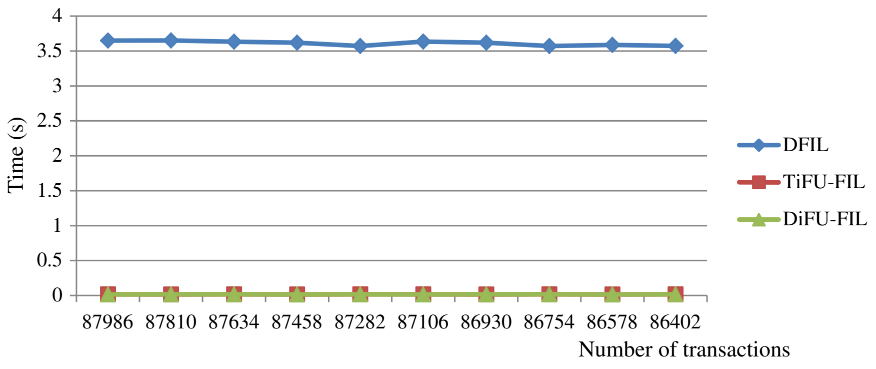

Fig. 16. Execution time of TiFU-FIL, DiFU-FIL ($S_U = 80\%$ and $S_L = 78\%$) and DFIL ($S_U = 80\%$) for the Retail database.

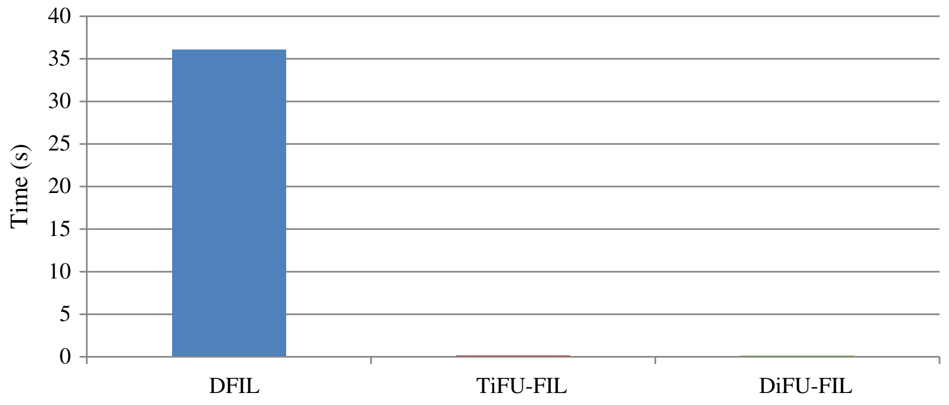

Fig. 17. Execution time of TiFU-FIL, DiFU-FIL ($S_U = 80\%$ and $S_L = 78\%$) and DFIL ($S_U = 80\%$) for the Retail database.

Acknowledgments

This research was funded by Vietnam National Foundation for Science and Technology Development (NAFOSTED).

References

[1] R. Agrawal, R. Srikant, Fast algorithms for mining association rules, VLDB (1994) 487–499.
[2] H. Chen, X. Yan, J. Han, IncSpan: incremental mining of sequential patterns in large databases, KDD (2004) 527–532.
[3] D.W. Cheung, J. Han, V.T. Ng, C.Y. Wong, Maintenance of discovered association rules in large databases: an incremental updating approach, ICDE (1996) 106–114.

[4] M. Dallachiesa, T. Palpanas, Identifying streaming frequent items in ad hoc time windows, Data Knowl. Eng. 87 (2013) 66–90.
[5] J. Dong, M. Han, BitTableFI: an efficient mining frequent itemsets algorithm, Knowl.-Based Syst. 20 (4) (2007) 329–335.
[6] T.G. Gharib, H. Nassar, M. Taha, A. Abraham, An efficient algorithm for incremental mining of temporal association rules, Data Knowl. Eng. 69 (8) (2010) 800–815.
[7] G. Grahne, J. Zhu, Fast algorithms for frequent itemset mining using FP-trees, IEEE Trans. Knowl. Data Eng. 17 (10) (2005) 1347–1362.
[8] J. Han, J. Pei, Y. Yin, Mining frequent patterns without candidate generation, SIGMOD (2000) 1–12.
[9] T.P. Hong, C.W. Lin, Y.L. Wu, Incrementally fast updated frequent pattern trees, Expert Syst. Appl. 34 (4) (2008) 2424–2435.
[10] T.P. Hong, C.W. Lin, Y.L. Wu, Maintenance of fast updated frequent pattern trees for record deletion, Comput. Stat. Data Anal. 53 (7) (2009) 2485–2499.
[11] T.P. Hong, C.Y. Wang, An efficient and effective association-rule maintenance algorithm for record modification, Expert Syst. Appl. 37 (1) (2010) 618–626.
[12] J.L. Koh, S.F. Shied, An efficient approach for maintaining association rules based on adjusting FP-tree structures, DASFAA (2004) 417–424.
[13] P.T. La, B. Le, B. Vo, Incrementally building frequent closed itemset lattice, Expert Syst. Appl. 41 (6) (2014) 2703–2712.
[14] T.P. Le, T.P. Hong, B. Vo, B. Le, An efficient incremental mining approach based on IT-tree, RIVF (2012) 57–61.
[15] J.M. Luna, J.R. Romero, S. Ventura, Grammar-based multi-objective algorithms for mining association rules, Data Knowl. Eng. 86 (2013) 19–37.
[16] T.T.L. Nguyen, N.T. Nguyen, Updating mined class association rules for record insertion, Appl. Intell. (2015), http://dx.doi.org/10.1007/s10489-014-0614-1 (in press).
[17] N. Pasquier, Y. Bastide, R. Taouil, L. Lakhal, Efficient mining of association rules using closed itemset lattices, Inf. Syst. 24 (1) (1999) 25–46.
[18] W. Song, B. Yang, Z. Xu, Index-BitTableFI: an improved algorithm for mining frequent itemsets, Knowl.-Based Syst. 21 (6) (2008) 507–513.
[19] L. Troiano, G. Scibelli, Mining frequent itemsets in data streams within a time horizon, Data Knowl. Eng. 89 (2014) 21–37.
[20] B. Vo, T.P. Hong, B. Le, A lattice-based approach for mining most generalization association rules, Knowl.-Based Syst. 45 (2013) 20–30.
[21] B. Vo, T. Le, T.P. Hong, B. Le, An effective approach for maintenance of pre-large-based frequent-itemset lattice in incremental mining, Appl. Intell. 41 (3) (2014) 759–775.
[22] B. Vo, B. Le, Interestingness measures for association rules: combination between lattice and hash tables, Expert Syst. Appl. 38 (9) (2011) 11630–11640.
[23] U. Yun, H. Ryang, Incremental high utility pattern mining with static and dynamic databases, Appl. Intell. 42 (2) (2015) 323–352.
[24] M.J. Zaki, K. Gouda, Fast vertical mining using diffsets, KDD (2003) 326–335.
[25] M.J. Zaki, C.J. Hsiao, Efficient algorithms for mining closed itemsets and their lattice structure, IEEE Trans. Knowl. Data Eng. 17 (4) (2005) 462–478.

Bay Vo received his BSc, MSc and PhD degrees in Computer Science from the University of Science, Vietnam National University, Ho Chi Minh City, Vietnam in 2002, 2005 and 2011 respectively. His research interests include association rules, classification, mining in incremental databases, distributed databases and privacy preserving in data mining.

Tuong Le received his BSc degree in Information Technology from University of Science, Vietnam National University, Ho Chi Minh City, Vietnam and MSc degree in Computer Science from University of Information Technology, Vietnam National University, Ho Chi Minh City, Vietnam in 2009 and 2014 respectively. His research interests include data mining and social network analysis.

Tzung-Pei Hong received his PhD degree in computer science and information engineering from National Chiao-Tung University in 1992. He served as the first director of the library and computer center, the Dean of Academic Affairs and the Vice President in National University of Kaohsiung. He is currently a distinguished professor at the Department of Computer Science and Information Engineering in NUK. He has published more than 400 research papers in international/national journals and conferences and has planned more than fifty information systems. He is also the board member of more than thirty journals and the program committee member of more than two hundred conferences. His current research interests include parallel processing, machine learning, data mining, soft computing, management information systems, and www applications.

Bac Le received the BSc degree, in 1984, the MSc degree, in 1990, and the PhD degree in Computer Science, in 1999. He is an Associate Professor, Vice Dean of Faculty of Information Technology, Head of Department of Computer Science, University of Science, Vietnam National University, Ho Chi Minh City, Vietnam. His research interests are in artificial intelligent, soft computing, and knowledge discovery and data mining.