An Efficient Approach for Mining Frequent Item sets with Transaction Deletion Operation

Bay Vo[1, 2], Thien-Phuong Le[3], Tzung-Pei Hong[4], Bac Le[5], Jason Jung[6]
[1]Division of Data Science, Ton Duc Thang University, Vietnam
[2]Faculty of Information Technology, Ton Duc Thang University, Vietnam
[3]Faculty of Technology, Pacific Ocean University, Vietnam
[4]Department of Computer Science and Information Engineering, National University of Kaohsiung, Taiwan
[5]Department of Computer Science, University of Science, Vietnam
[6]Department of Computer Engineering, Chung-Ang University, Republic of Korea

Abstract: *Deletion of transactions in databases is common in real-world applications. Developing an efficient and effective mining algorithm to maintain discovered information is thus quite important in data mining fields. A lot of algorithms have been proposed in recent years, and the best of them is the pre-large-tree-based algorithm. However, this algorithm only rebuilds the final pre-large tree every deleted transactions. After that, the FP-growth algorithm is applied for mining all frequent item sets. The pre-large-tree-based approach requires twice the computation time needed for a single procedure. In this paper, we present an incremental mining algorithm to solve above issues. An itemset tidset-tree structure will be used to maintain large and pre-large item sets. The proposed algorithm only processes deleted transactions for updating some nodes in this tree, and all frequent item sets are directly derived from the tree traversal process. Experimental results show that the proposed algorithm has good performance.*

Keywords: *Data mining, frequent item sets, incremental mining, pre-large item sets, item set-tidset tree.*

Received November 1, 2012; accepted June 19, 2013; published online October 29, 2015

1. Introduction

Mining Frequent Item sets (FIs) is the most important task in mining association rules [1, 2, 16, 18, 19, 20]. A lot of algorithms for mining FIs have been proposed. Some of famous ones are Apriori [1, 17], FP-growth [4], Eclat [22] and so on. All Apriori, FP-growth and Eclat algorithms use batch mining. In real-world applications, transactions are commonly inserted or deleted or updated [3, 5, 6, 7, 8, 9, 10, 11, 13]. Therefore, designing an efficient algorithm for the maintenance of association rules as databases change is critically important. The first incremental mining algorithm was the Fast-Update (FUP) algorithm [3]. Like Apriori-based algorithms, FUP generates candidates and repeatedly scans the database, although it avoids a lot of unnecessary checking. After that, Hong *et al.* [6] proposed the pre-large concept to further reduce the need for rescanning the original database. This algorithm does not require the original database to be rescanned until a number of new transactions have been inserted. The maintenance cost is thus reduced with the pre-large concept.

Many pre-large-based algorithms for handling of inserted transactions have been proposed. Some of ones are Pre-FUFP [15], Pre-FUIT [12]. In addition to, transaction insertion, transaction deletion from databases is also commonly seen in real-applications. A lot of algorithms have been proposed in recent years, and the best of them is pre-large tree maintenance algorithm [14]. However, it only rebuilds the final pre-large tree every deleted transactions. After that, the FP-growth must be used to mine all FIs [4]. In fact, original large item sets are derived from the pre-large tree of the original database. So, the pre-large tree maintenance algorithm does not utilize the item sets which have been mined from the original database.

The aim of this paper is to present an algorithm for handling of deleted transactions based on the concept of pre-large item sets. Pre-large item sets are defined by using lower and upper support thresholds. It does not require the original database to be rescanned until a number of transactions have been deleted. The proposed algorithm uses the Item set Tidset-tree (IT-tree) for storing the pre-large and large items ets which are mined from the original database. When some transactions are deleted from the database, the proposed algorithm only processes them for maintaining large and pre-large item sets. Experimental results show that the proposed algorithm outperforms the pre-large tree maintenance algorithm.

The rest of this paper is organized as follows. Related works are reviewed in section 2. The Inc-Eclat algorithm is described in section 3. An example to illustrate the proposed algorithm is given in section 4. Experimental results that show the performance of the

proposed algorithm are provided in section 5. Finally, conclusions and future work are presented in section 6.

2. Related Works

2.1. Pre-large Concepts

The concept of pre-large item sets was proposed by Hong *et al.* [5]. It uses two thresholds, namely the upper threshold and the lower threshold, to set the pre-large item sets. The upper threshold is similar to min sup. The lower threshold defines the lowest support ratio for an itemset that is to be treated as pre-large. An itemset with a support ratio below the lower threshold is seen as small. Hong *et al.* [5] also proposed the pre-large-itemset algorithm. It is based on a safety number f of inserted transactions to reduce the need for rescanning the original database for the efficient maintenance of the large item sets.

Considering an original database and some transactions that are to be deleted by the two support thresholds, Lin *et al.* [14] proposed the following formula for computing f in the case of deletion of transactions:

$$f = \left\lfloor \frac{(S_u - S_l)d}{S_u} \right\rfloor \quad (1)$$

Where S_u is the upper threshold, S_l is the lower threshold and d is the number of original transactions.

Cases 2, 3, 4, 7, and 8 do not affect the final large item sets according to the weighted average of the counts. Case 1 may remove the existing large item sets, and cases 5, 6 and 9 may add new large item sets. If all large and pre-large item sets 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 in the final updated database as long as the number of deleted transactions is smaller than the number f [14]. A summary of the nine cases and their results are given in Table 1.

Table 1. Nine cases and their results.

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

2.2. Maintenance of Fast Updated Frequent Pattern Trees for Transaction Deletion

Transaction deletion from databases is commonly seen in real-world applications. Some algorithms have been proposed such as Fast Updated Frequent Pattern tree (FUFP-tree) [7], pre-large tree [14]. Hong *et al.* [7] modified the FP-tree structure and designed a FUFP-tree for handling deleted transactions based on the FUP algorithm [3]. The FUFP-tree structure is similar to the FP-tree structure, with the difference being that the links between the parent nodes and their child nodes are bi-directional in the former. Bi-directional linking will help fasten the process of item deletion in the maintenance process. When transactions are deleted from the database, the FUFP-tree based approach will process them to maintain the FUFP-tree. It partitions items into four parts according to whether they are frequent or infrequent in the original database and in deleted transactions. Considering an original database and some transactions to be deleted, the following four cases may arise:

- ***Case*** 1: An item set is frequent both in an original database and in deleted transactions.
- ***Case*** 2: An item set is frequent in an original database but not frequent in deleted transactions.
- ***Case*** 3: An item set is not frequent in an original database but frequent in deleted transactions.
- ***Case*** 4: An item set is not frequent both in an original database and in deleted transactions.

Cases 2 and 3 will not affect the final frequent item sets. Item sets in case 1 are frequent in both the original database and deleted transactions. Thus, some existing frequent item sets may be removed after the database is updated. At last, item sets in case 4 are infrequent in both the original database and deleted transactions. Some frequent item sets may thus be added it however, requires the original database to be rescanned for rebuilding the FUFP-tree of the final updated database. After that, the FP-growth algorithm must be used to mine all FIs [4].

In order to reduce the need for rescanning the original database, Lin *et al.* [14] proposed a pre-large tree structure and designed an algorithm to rebuild the pre-large tree based on the concept of pre-large item sets. The pre-large tree is similar to the FUFP-tree. When some transactions are deleted from the database, the pre-large-tree-based approach will process them to maintain the pre-large tree. Unlike the FUFP-tree-based approach, it partitions items into nine cases according to whether they are large or pre-large or small in the original database and in deleted transactions. The summary of the nine cases and their results is given in Table 1. The algorithm does not require the original database to be rescanned until a number of deleted transactions have been processed. When some transactions are deleted from the database, some nodes are removed from or inserted into the pre-large tree. After that, the FP-growth algorithm is applied for the pre-large tree of the entire database to mine all frequent item sets. So, the pre-large-tree-based approach does not utilize item sets which have been mined from the original database.

As we known, IT-tree-based approach [22] is one of the famous approaches for mining FIs in static transaction databases. It is based on equivalence

classes, scans the database only once, uses the depth-first traversal technique to generate item sets and to compute the supports of the item sets fast by tidset intersections. This paper proposes an incremental algorithm for handling of deleted transactions based on the IT-tree structure and pre-large item sets. Like the pre-large-tree-based approach, when transactions are deleted from the database, the proposed approach will partition items into nine cases according to whether they are large or pre-large or small in the original database and in deleted transactions. The summary of the nine cases and their results is given in Table 1. Its main idea is to use the depth-first traversal technique to update the final supports of the item sets from their tidsets in deleted transactions. The supports of the item sets in deleted transactions are computed by tidset intersections. All FIs are mined using depth-first order traversal. The advantage of the IT-tree-based approach is to utilize item sets which have been mined from the original database. The proposed algorithm only processes deleted transactions for rebuilding the final IT-tree. Besides, the concept of pre-large item sets is used to reduce the need for rescanning the original database to save maintenance cost. The algorithm does not require the original database to be rescanned until many deleted transactions have been processed.

3. Proposed Algorithm

Notations used in the proposed algorithm are listed in Table 2.

Table 2. The notation used in the proposed algorithm.

Symbol	Description
D	The Original Database
T	The Set of Deleted Transactions
U	The Final Database, i.e., $D - T$
d	The Number of Transactions in D
S_l	The Lower Support Threshold for Pre-Large Item Sets
S_u	The Upper Support Threshold for Large Item Sets, $S_u > S_l$
X	An Item Set
Tr	An IT-Tree Storing the Set of Pre-Large and Large Item Sets from D
R	a Set of Item Sets for which the Original Database must be Rescanned to Update their Final Support.
$T^{-}(X)$	Tidset of Item X in T
$S^U(X)$	The Support Count of X in U
$S^D(X)$	The Support Count of X in D, $S^D(X)$
$S^T(X)$	The Support Count of X in T, $S^T(X)=\|T^{-}(X)\|$
$\sigma_D(X)$	The Support of X in D, $\sigma_D(X)=S^D(X) / d$
$\sigma_T(X)$	The Support of X in T, $\sigma_T(X)=S^T(X) / t$
$\sigma_u(X)$	The Support of X in U, $\sigma_U(X)=(S^D(X) - S^T(X)) / (d - t - c)$

3.1. Building an IT-tree from an Original Database

Given an original database and two support thresholds, the upper support threshold is similar to min sup, the lower threshold defines the lowest support ratio for an itemset that is to be treated as pre-large. An IT-tree, which stores all pre-large and large item sets, must be built in advance from the initially original database before some transactions are deleted. Its initial construction is stated as follows. The database is firstly transformed into the vertical data format in which each itemset has a corresponding tidset. Next, a set of all large and pre-large item sets with their tidsets is created at first level. After that, these nodes are combined to create nodes at higher level using depth-first order traversal. The downward-closure property [1] is also used to prune unpromising item sets. The tree is completely constructed when no new nodes are created.

3.2. Algorithm for Maintaining IT-tree

The proposed algorithm only processes deleted transactions for updating the final supports of item sets using the depth-first search technique. The decreasing supports of item sets are rapidly computed using tidset intersections. The algorithm uses the downward-closure property [1] to prune unpromising item sets while traversing the IT-tree. FIs can be directly derived from the tree traversal process. According to whether the number of deleted transactions exceeds the safety threshold f, the proposed algorithm requires the original database to be rescanned.

If the number of deleted transactions is less than f, the UP-TID procedure for updating the IT-tree is performed. The main idea of this procedure is to maintain the large and pre-large item sets stored in the IT-tree using depth-first order traversal. Firstly, a set of item sets at level 1 in Tr with their tidsets is determined from deleted transactions. Next, the final support counts of them are updated. Item sets at level 1 in Tr that does not satisfy S_l are removed from Tr. After L and R are determined, the procedure UP-TID-EXTEND extends the nodes in L to one more level by combining the nodes following them. With each pair (X, Y), this procedure will compute the intersection of $T^{-}(X)$ and $T^{-}(Y)$. If itemset XY exists in Tr then the final support count is updated. Otherwise, if XY is small in T then itemset XY is inserted into R. After L_i are created, the procedure UP-TID-EXTEND will be called recursively to update the support counts of all child nodes of the nodes in L_i. When the deleted transactions are processed, other transactions can be deleted continuously. If the number total of deleted transactions does not exceed f, the UP-TID procedure is called and all processes are the same as above.

Assume after that some transactions need to be deleted, if the number total of deleted transactions (including newest deleted transactions) exceeds f, the algorithm must rescan the original database to determine whether the item sets in R are large or pre-large in the final updated database. The large or pre-large item sets in R are inserted into Tr. A new safety threshold f is then computed. Algorithm 1 will compare the number of newest deleted transactions and f. All processes are then repeated as above.

Algorithm 1: Maintaining IT-tree.

 # Procedure Inc-IT(*Tr, D, T, S*$_l$*, S*$_u$*, R*)
 # S$_l$ *is a lower support threshold,*
 # S$_u$ *is an upper threshold,*
 # R is a set of item sets
 # Tr is an IT-tree that stores large and pre-large item sets #derived from the original database D consists of (*d*+*c*) *#transactions with c*

is a variable which is used to record # #the number of deleted transactions since the last #rescan of the original database with d transactions.

#A set of t deleted transactions $f = \left\lfloor \frac{(S_u - S_l)d}{S_u} \right\rfloor$

```
If (t + c < f) {
        #processes deleted transactions for updating Tr
        UP-TID(Tr, D, T, Sl, Su, R)
        set c=c+t
    }
Else    {
        #rescans d transactions to determine whether
        #item sets in R are large or pre-large in U
        L_item sets=rescan (d)
        Tr=Tr∪L_item sets
        set R=⌀; d=d+c; c=0
        Inc-IT(Tr, D, T, Sl, Su, R)
    }
```

#Procedure UP-TID(Tr, D, T, Sl, Su, R)

Update the final support count of each itemset at level 1 in Tr based on their tidset in T, after that remove nodes at level 1 that does not satisfy Sl threshold.

L={$i_{T^-(i)}$*|itemset i belongs to a set of item sets at level 1 in Tr and* $\sigma_U(i) \geq S_l$ *}*

R={ $i_{|T^-(i)|}$ *| itemset i belongs to 1-item sets in T, but not exists at level 1 in Tr, and* $\sigma_T(i) < S_l$*}*

UP-TID-EXTEND(Tr, L, R, Sl)

#Procedure UP-TID-EXTEND(Tr, L, R, Sl)

$\forall X_{T^-(X)} \in L$:

Create the new set L_i *by joining* $X_{T^-(X)}$ *with* $Y_{T^-(Y)}$ *following X in L:*

If XY exists in Tr then

If $\sigma_U(XY) \geq S_l$ *then add XY and* $T^-(XY)$ *into* L_i *and update the support count of XY in Tr*

Otherwise remove XY from Tr

Otherwise, if $\sigma_T(XY) < S_l$ *then add XY with* $|T^-(XY)|$ *into R*

If $|L_i| \geq 2$ *then UP-TID-EXTEND(Tr,* L_i*, R,* S_l*)*

As the number of deleted transactions is not limited, a variable *c* is used to store it since the last rescan of the original database with *d* transactions. The details of the algorithm are shown in Figure 1.

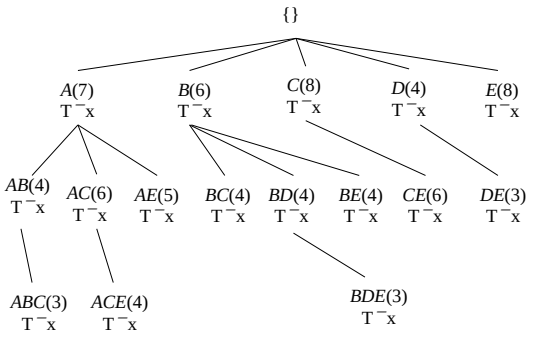

Figure 1. IT-tree that are constructed from the original database.

4. Example

In this section, an example is given to illustrate the proposed incremental algorithm. Assume that the initial *D* database includes 10 transactions shown in Table 3. For S_l=30% and S_u=50%, an IT-tree that are constructed from *D* are shown in Figure 2. In fact, the tidsets of the item sets, which are mined from the original database, are not retained as the algorithm does not consider them in the maintenance process.

Table 3. An example of an original database.

Original Database	
TID	**Items**
1	ACE
2	ABDE
3	BCDE
4	ACE
5	ACE
6	ABC
7	BDE
8	ABCE
9	ABCD
10	CEF

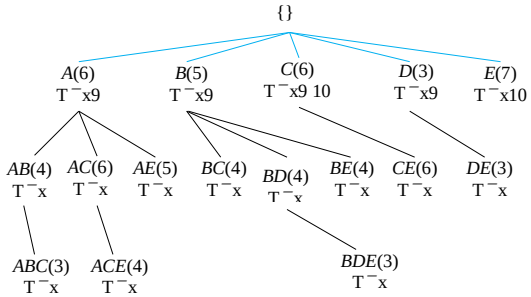

Figure 2. IT-tree *Tr* after nodes at level 1 have been processed.

Assume that the last two transactions (with TIDs 9 and 10, respectively) are deleted from the original database. Table 4 shows the vertical layout of two deleted transactions.

Table 4. Vertical format of the two deleted transactions.

Item	Tidset
A	9
B	9
C	9 10
D	9
E	10
F	10

For the above data, the proposed maintenance algorithm proceeds as follows. The variable *c* is initially set at 0 and the safety *f* threshold is computed:

$$f = \left\lfloor \frac{(S_u - S_l)d}{S_u} \right\rfloor = \left\lfloor \frac{(0.5 - 0.3)10}{0.5} \right\rfloor = 4 \quad (2)$$

As $t+c = 2+0 = 2 \leq f$, rescanning the original database is unnecessary, the UP-TID procedure is called. The final support counts of item sets at level 1 in *Tr* will be updated based on their tidsets in *T*.

In this example, *L*={*A*, *B*, *C*, *D*, *E*} and R=⌀. Take item set {*A*} as an example, $|T^-(A)|$=1, the support count of {*A*} in *T* is thus 1. As $\sigma_U(A)$=(7-1)/(10-2-0) ≥ 0.3, {*A*} is inserted into *L*. Item sets {*B*}, {*C*}, {*D*}, {*E*} are similarly processed. For item set {*F*}, $|T^-(F)|$= 1, the support counts of {*F*} in *T* is thus 1. As {*F*} is small in *D* and $\sigma_T(F)$=1/2≥0.3, {*F*} is removed from *Tr*. The results are shown in Figure 3.

After *L* and *R* are determined, the UP-TID-EXTEND is called. With *L*= {*A*, *B*, *C*, *D*, *E*}, this procedure creates nodes at higher level, it joins each child node with all its following nodes. For example, consider node *A* at level 1 of the IT-tree in Figure 3. It will be combined with the other nodes as follow:

- {*A*} joins {*B*} to create a new node *AB*, $T^{-}(A)$={9} and $T^{-}(B)$={9}, so $T^{-}(AB)=T^{-}(A)\cap T^{-}(B)$={9}. As {*AB*} exists in *Tr* and $\sigma_U(AB)$=(4-1)/(10-2-0)≥0.3, $L_i=L_i \cup \{AB\}$.
- {*A*} joins {*C*} to create a new node *AC*, $T^{-}(A)$={9} and $T^{-}(C)$={9, 10}, so $T^{-}(AC)=T^{-}(A)\cap T^{-}(C)$={9}. As {*AC*} exists in *Tr* and $\sigma_U(AC)$=(6-1)/(10-2-0)≥ 0.3, $L_i= L_i \cup \{AC\}$.
- {*A*} joins {*D*} to create a new node *AD*, $T^{-}(A)$= {9} and $T^{-}(D)$={9}, so $T^{-}(AD)=T^{-}(A)\cap T^{-}(D)$={9}. As {*AD*} does not exist in *Tr* and $\sigma_T(AD)$=1/2≥0.3, {*AD*} does not belong to *R*.
- {*A*} joins {*E*} to create a new node *AE*, $T^{-}(A)$={9} and $T^{-}(E)$={10}, so $T^{-}(AE)=T^{-}(A)\cap T^{-}(E)$= {⌀}. As {*AE*} exists in *Tr* and $\sigma_U(AE)$=(5-0)/(10-2-0) ≥ 0.3, $L_i= L_i \cup \{AE\}$.

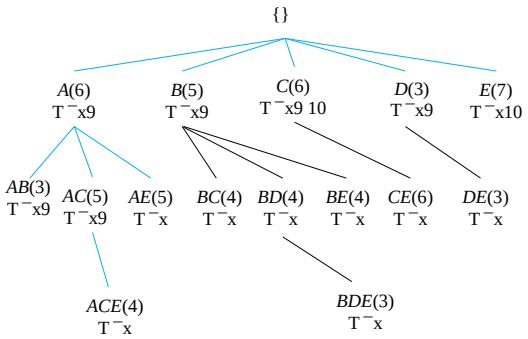

Figure 3. IT-tree *Tr* after the branch corresponding to {*A*} has been processed.

After that, the procedure UP-TID-EXTEND is repeated recursively with L_i={*AB*, *AC*, *AE*} and the final support counts of corresponding nodes will be updated. This process will be repeated recursively to update the final support counts of all nodes in *Tr*. The results are shown in Figure 4 and the final *Tr* are shown in Figure 5.

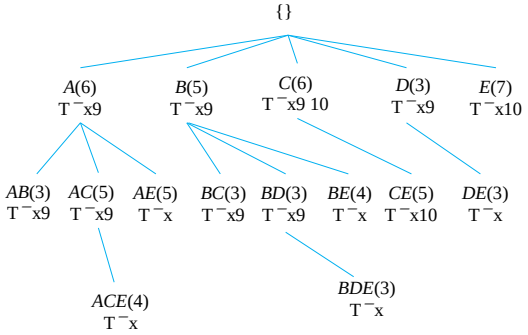

Figure 4. Final IT-tree *Tr*.

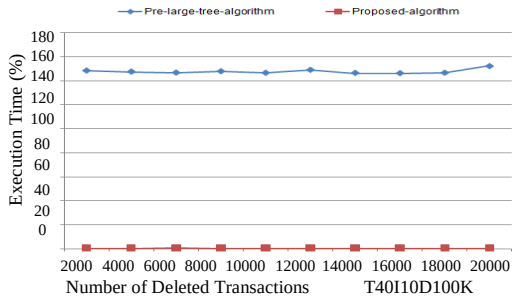

Figure 5. Comparison of execution times for sequentially deleted transactions (T40I10D100K database).

When the Inc-IT algorithm finishes, *R*= ∅. Besides, the final value of *c* is 2 in this example and *f*–*c*=2. This implies that two more transactions can be deleted without rescanning the original database.

5. Experimental Results

Experiments were conducted to show the performance of the proposed algorithm. All the algorithms were implemented on a PC with a Core 2 Duo (2×2GHz) CPU and 2GBs of RAM running Windows 7. All the algorithms were coded in C++. Two databases from http://fimi.cs.helsinki.fi/data/ were used for the experiments. The T40I10D100K (with 100000 transactions) and kosarak (with 990002 transactions) databases were used.

Experiments were made to compare execution times for mining all FIs using the pre-large-tree-based algorithm [14] and the proposed algorithm when a number of transactions were deleted from the database. The T40I10D100K database was used for the first experiment. A total of 100000 transactions were used to construct an initial IT-tree and pre-large tree. Sets of 2000 transactions were then sequentially used as the deleted transactions for the experiment. The upper and the lower support thresholds were set at 2% and 5%, respectively. Figure 6 shows execution times of the two algorithms for processing each set of 2000 deleted transactions. For the experiment with incremental threshold values, the lower support threshold was set from 2% to 6% (in 1% increments) and the upper support threshold was set from 5% to 9% (in 1% increments). Sets of 2000 transactions were used as the deleted transactions for the experiment. Figure 7 shows a comparison of execution times for various threshold values. It can be seen that the proposed maintenance algorithm runs faster than the pre-large-tree-based algorithm.

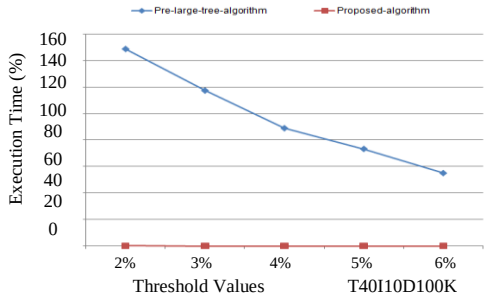

Figure 6. Comparison of execution times for various threshold values (T40I10D100K database).

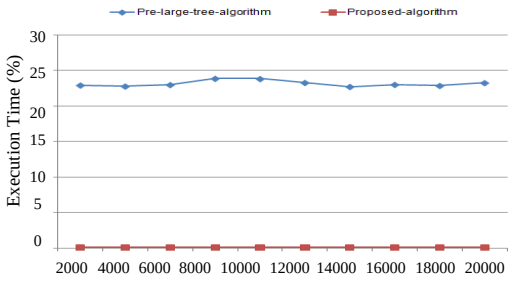

Figure 7. Comparison of execution times for sequentially deleted transactions (kosarak database).

Next, the kosarak database was used for the experiments. A total of 990002 transactions were used to build an initial IT-tree and pre-large tree. Sets of 2000 transactions were sequentially used as deleted transactions for the experiment. The upper and the lower support thresholds were set to 0.5% and 1%, respectively. Figure 8 shows execution times of the two algorithms for processing each set of 2000 deleted transactions. For the experiment with incremental threshold values, the lower support threshold was set from 0.5% to 2.5% (in 1% increments) and the upper support threshold was set from 1% to 3% (in 1% increments). Sets of 2000 transactions were used as the deleted transactions for the experiment. Figure 9 shows a comparison of execution times for various threshold values. It can be observed that the proposed maintenance algorithm also runs faster than the pre-large-tree-based algorithm.

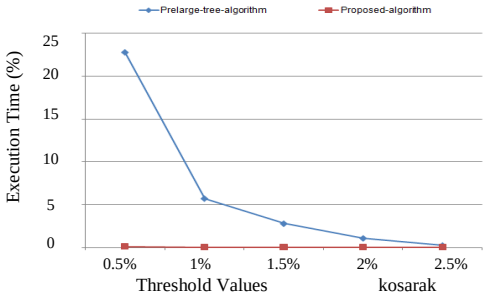

Figure 8. Comparison of execution times for various threshold values (kosarak database).

6. Conclusions and Future Work

This paper proposed an approach based on the concept of pre-large item sets for the maintenance of an IT-tree during transaction deletion. The proposed algorithm uses the IT-tree structure to facilitate tree traversal and the updating of itemset supports. Instead of batch mining, the proposed algorithm only concentrates on processing deleted transactions. The incremental supports of candidate item sets can be rapidly computed using tidset intersections. Besides, pre-large item sets are used to reduce the number of database scans. User-specified upper and lower support thresholds are used to avoid the small items directly becoming large in the updated database when transactions are deleted. All the tasks are processed using the IT-tree structure. With these strategies, the execution time of the proposed approach is lower than the pre-large-tree-based algorithm.

The proposed algorithm belongs to calculations of tidset intersections. Dynamic Bit Vectors (DBVs) [21] is an efficient data structure for mining FIs. DBVs can be used to compress a database in one scan and shorten the length of the tidset, speeding up the tidset intersections process. In the future, we will apply DBVs structure for handling of deleted transactions.

References

[1] Agrawal R. and Srikant R., "Fast Algorithm for Mining Association Rules," *In VLDB'94 Proceedings of the 20th International Conference on Very Large Data Bases*, Santiago de Chile, Chile, pp. 487-499, 1994.

[2] Bodon F. and Ronyai L., "Trie: An Alternative Data Structure for Data Mining Algorithms," *Mathematical and Computer Modelling*, vol. 38, no. 7, pp. 739-751, 2003.

[3] Cheung D., Han J., Ng V., and Wong C., "Maintenance of Discovered Association Rules in Large Databases: An Incremental Updating Approach," *in Proceedings of the 12th IEEE International Conference on Data Engineering*, New Orleans, USA, pp. 106-114, 1996.

[4] Han J., Pei J., and Yin Y., "Mining Frequent Patterns without Candidate Generation," *in Proceedings of ACM SIGMOD International Conference on Management of Data*, Dallas, Taxas, USA, pp. 1-12, 2000.

[5] Hong T., Wang C., and Tao Y., "A New Incremental Data Mining Algorithm using Pre-Large Itemsets," *Intelligent Data Analysis*, vol. 5, no. 2, pp. 111-129, 2001.

[6] Hong T., Lin C., and Wu Y., "Incrementally Fast Updated Frequent Pattern Trees," *Expert Systems with Applications*, vol. 34, no. 4, pp. 2424-2435, 2008.

[7] Hong T., Lin C., and Wu Y., "Maintenance of Fast Updated Frequent Pattern Trees for Record Deletion," *Computational Statistics and Data Analysis*, vol. 53, no. 7, pp. 2485-2499, 2009.

[8] Hong T. and Wang C., "An Efficient and Effective Association-Rule Maintenance Algorithm for Record Modification," *Expert Systems with Applications*, vol. 37, no. 1, pp. 618-626, 2010.

[9] Hong T., Wang C., and Tseng S., "An Incremental Mining Algorithm for Maintaining Sequential Patterns using Pre-Large Sequences," *Expert Systems with Applications*, vol. 53, no. 6, pp. 7051-7058, 2011.

[10] Koh J. and Shied S., "An Efficient Approach for Maintaining Association Rules based on Adjusting FP-Tree Structures," available at: http://www.csie.ntnu.edu.tw/~jlkoh/publications/dasfaa04.pdf, last visited 2004.

[11] Le T., Hong T., Vo B., Le B., and Hwang D., "Improving Efficiency of Incremental Mining by Trie Structure and Pre-Large Itemsets," *Computing and Informatics*, vol. 33, no. 3, pp. 609-632, 2014.

[12] Le T., Vo B., Hong T., and Le B., "An Efficient Incremental Mining Approach based on IT-

Tree," *in Proceedings of tIEEE International Conference on Computing and Communication Technologies, Research, Innovation, and Vision for the Future*, Ho Chi Minh, VietNam, pp. 57-61, 2012.

[13] Lin X., Deng Z., and Tang S., "A Fast Algorithm for Maintenance of Associations Rules in Incremental Databases," *in Proceedings of the 2nd International Conference on Advanced Data Mining and Applications*, XiAn, China, pp. 56-63, 2006.

[14] Lin C., Hong T., and Lu W., "Maintenance of the Prelarge Trees for Record Deletion," *in Proceedings of the 12th WSEAS International Conference on Applied Mathematics*, Stevens Point, Wisconsin, USA, pp. 105-110, 2007

[15] Lin C., Hong T., and Lu W., "The Pre-FUFP Algorithm for Incremental Mining," *Expert Systems with Applications*, vol. 36, no. 5, pp. 9498-9505, 2009.

[16] Srikant R. and Agrawal R., "Mining Generalized Association Rules," available at: http://www.vldb.org/conf/1995/P407.PDF, last visited 1995.

[17] Senhadji S., Khiat S., and Belbachir H., "Association Rule Mining and Load Balancing Strategy in Grid Systems," *The International Arab Journal of Information Technology*, vol. 11, no. 4, pp. 338-344, 2014.

[18] Srikant R. and Agrawal R., "Mining Quantitative Association Rules in Large Relational Tables," *in Proceedings of ACM SIGMOD International Conference on Management of Data*, New York, USA, pp. 1-12, 1996.

[19] Thomas S., Bodagala S., Alsabti K., and Ranka S., "An Efficient Algorithm for the Incremental Updation of Association Rules in Large Databases," available at: https://www.aaai.org/ Papers/KDD/1997/KDD97-055.pdf, last visited 1997.

[20] Toivonen H., "Sampling Large Databases for Association Rules," available at: http://www.vldb.org/conf/1996/P134.PDF, last visited 1996.

[21] Vo B., Hong T., and Le B., "DBV-Miner: A Dynamic Bit-Vector Approach for Fast Mining Frequent Closed Item Sets," *Expert Systems with Applications*, vol. 39, no. 8, pp. 7196-7206, 2012

[22] Zaki M., "Scalable Algorithms for Association Mining," *IEEE Transactions on Knowledge and Data Engineering*, vol. 12, no. 3, pp. 372-390, 2000.

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

Thien-Phuong Le received his MSc degrees in Information System from the University of Science, Vietnam National University of Ho Chi Minh, Vietnam in 2011. His research interests include association rules, clustering, classification, mining in incremental databases.

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 hundreds 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, Ho Chi Minh City. His research interests are in artificial intelligent, soft computing, and knowledge discovery and data mining.

Jason Jung is an associate professor of Computer Engineering Department at Chung-Ang University, Korea. He was a postdoctoral researcher in INRIA Rhone-Alpes, France in 2006, and a visiting scientist in Fraunhofer Institute (FIRST) in Berlin, Germany in 2004. He received the B.Eng. in Computer Science and Mechanical Engineering from Inha University in 1999. He received M.S. and Ph.D. degrees in Computer and Information Engineering from Inha University in 2002 and 2005, respectively. His research topics are knowledge engineering on social networks by using machine learning, semantic Web mining, and ambient intelligence. He has about 25 international journal articles published in Knowledge-Based Systems, Information Retrieval, Information Processing & Management, Knowledge and Information Systems, and Expert Systems with Applications. Also, he is an editorial member of Journal of Universal Computer Science and International Journal of Intelligent Information and Database Systems. Moreover, he has been editing 10 special issues in Information Sciences, Journal of Network and Computer Applications, Computing and Informatics and so on.