New Generation Computing, 28(2010)5-20
Ohmsha, Ltd. and Springer

Using the Structure of Prelarge Trees to Incrementally Mine Frequent Itemsets

Chun-Wei LIN[1], Tzung-Pei HONG[2,3], and Wen-Hsiang LU[1]
[1]*Department of Computer Science and Information Engineering*
National Cheng Kung University, Tainan, 701, TAIWAN, R.O.C.
[2]*Department of Computer Science and Information Engineering*
National University of Kaohsiung, Kaohsiung, 811, TAIWAN, R.O.C.
[3]*Department of Computer Science and Engineering*
National Sun Yat-sen University, Kaohsiung, 804, TAIWAN, R.O.C.
{p7895122;whlu}@mail.ncku.edu.tw,tphong@nuk.edu.tw

Received 1 December 2008
Revised manuscript received 5 May 2009

Abstract The frequent pattern tree (FP-tree) is an efficient data structure for association-rule mining without generation of candidate itemsets. It was used to compress a database into a tree structure which stored only large items. It, however, needed to process all transactions in a batch way. In the past, we proposed a Fast Updated FP-tree (FUFP-tree) structure to efficiently handle new transactions and to make the tree update process become easier. In this paper, we propose the structure of prelarge trees to incrementally mine association rules based on the concept of pre-large itemsets. Due to the properties of pre-large concepts, the proposed approach does not need to rescan the original database until a number of new transactions have been inserted. The proposed approach can thus achieve a good execution time for tree construction especially when a small number of transactions are inserted each time. Experimental results also show that the proposed approach has a good performance for incrementally handling new transactions.

Keywords: Data Mining, FP-tree, Prelarge-tree Algorithm, Incremental Mining, Maintenance.

§1 Introduction

Years of effort in data mining have produced a variety of efficient techniques. Among them, finding association rules in transaction databases is most

commonly seen in data mining.[1–5,7,9,10,15–17,19–21,23] In the past, many algorithms for mining association rules from transactions were proposed, most of which were based on the Apriori algorithm,[1] which generated and tested candidate itemsets level-by-level. This may cause iterative database scans and high computational costs. Han *et al.* thus proposed the Frequent-Pattern-tree (FP-tree) structure for efficiently mining association rules without generation of candidate itemsets.[11] They showed the approach could have a better performance than the Apriori approach.

Both the Apriori and the FP-tree mining approaches belong to batch mining. One noticeable incremental mining algorithm was the Fast-Updated Algorithm (called FUP), which was proposed by Cheung *et al.*[6] for avoiding the shortcomings mentioned above. The FUP algorithm modified the Apriori mining algorithm[3] and adopted the pruning techniques used in the DHP (Direct Hashing and Pruning) algorithm.[17] Although the FUP algorithm could indeed improve mining performance for incrementally growing databases, original databases still needed to be scanned when necessary. Hong *et al.* thus proposed the pre-large concept to further reduce the need for rescanning original database.[12] The algorithm did not need to rescan the original database until a number of new transactions had been inserted. Since rescanning the database spent much computation time, the maintenance cost could thus be reduced in the pre-large-itemset algorithm.

Hong *et al.* modified the FP-tree structure and designed the fast updated frequent pattern trees (FUFP-trees) to efficiently handle newly inserted transactions based on the FUP concept.[13] The FUFP-tree structure was similar to the FP-tree structure except that the links between parent nodes and their child nodes were bi-directional. In this paper, we attempt to further modify the FUFP-tree algorithm for incremental mining based on the concept of pre-large itemsets.[12] A structure of prelarge tree is proposed and a mining algorithm based on the tree is provided to get the association rules. The proposed algorithm does not require rescanning the original databases to construct the prelarge tree until a number of new transactions have been processed. Experimental results also show that the proposed algorithm has a good performance for incrementally handling new transactions.

The remainder of this paper is organized as follows. Related works are reviewed in Section 2. The proposed Prelarge-tree maintenance algorithm is described in Section 3. An example to illustrate the proposed algorithm is given in Section 4. Experimental results for showing the performance of the proposed algorithm are provided in Section 5. Conclusions are finally given in Section 6.

§2 Review of Related Works

2.1 The FUFP-tree Algorithm

The FUFP-tree construction algorithm is based on the FP-tree algorithm.[11] The links between parent nodes and their child nodes are, however, bi-directional. Bi-directional linking will help fasten the process of item deletion

in the maintenance process. Besides, the counts of the sorted frequent items are also kept in the Header_Table.

An FUFP-tree must be built in advance from the original database before new transactions come. When new transactions are added, the FUFP-tree maintenance algorithm will process them to maintain the FUFP-tree. It first partitions items into four parts according to whether they are large or small in the original database and in the new transactions. Each part is then processed in its own way. The Header_Table and the FUFP-tree are correspondingly updated whenever necessary.

In the process for updating the FUFP-tree, item deletion is done before item insertion. When an originally large item becomes small, it is directly removed from the FUFP-tree and its parent and child nodes are then linked together. On the contrary, when an originally small item becomes large, it is added to the end of the Header_Table and then inserted into the leaf nodes of the FUFP-tree. It is reasonable to insert the item at the end of the Headrer_Table since when an originally small item becomes large due to the new transactions, its updated support is usually only a little larger than the minimum support. The FUFP-tree can thus be least updated in this way, and the performance of the FUFP-tree maintenance algorithm can be greatly improved. The entire FUFP-tree can then be re-constructed in a batch way when a sufficiently large number of transactions have been inserted.

Several other algorithms based on the FP-tree structure have been proposed. For example, Qiu *et al.* proposed the QFP-growth mining approach to mine association rules.[18] Mohammad proposed the COFI-tree structure to replace the conditional FP-tree.[22] Ezeife constructed a generalized FP-tree, which stored all the large and non-large items, for incremental mining without rescanning databases.[8] Koh *et al.* adjusted FP trees also based on two support thresholds,[14] but with a more complex adjusting procedure and spending more computation time than the one proposed in this paper. Some related researches are still in progress.

2.2 The Prelarge-itemset Algorithm

Hong *et al.* proposed the pre-large concept to reduce the need of rescanning original database[12]for maintaining association rules. A pre-large itemset is not truly large, but may be large with a high probability in the future. A pre-large itemset was defined based on two support thresholds, a lower support threshold and an upper support threshold. The upper support threshold is the same as that used in the conventional mining algorithms. The support ratio of an itemset must be larger than the upper support threshold in order to be considered large. On the other hand, the lower support threshold defines the lowest support ratio for an itemset to be treated as pre-large. An itemset with its support ratio below the lower threshold is thought of as a small itemset. Prelarge itemsets act like buffers in the incremental mining process and are used to reduce the movements of itemsets directly from large to small and vice-versa.

Considering an original database and transactions which are newly in-

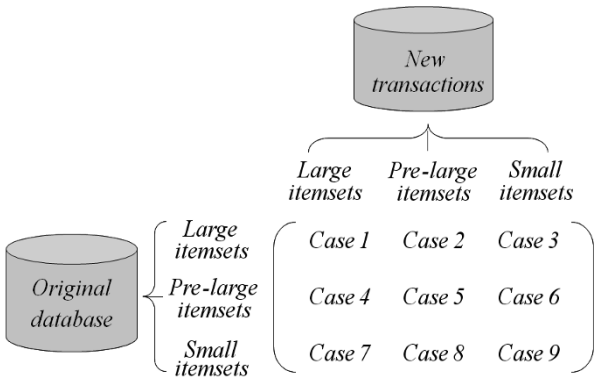


Fig. 1 Nine Cases Arising from Adding New Transactions to Existing Databases

serted by the two support thresholds, itemsets may fall into one of the following nine cases illustrated in Fig. 1.

Cases 1, 5, 6, 8 and 9 will not affect the final association rules according to the weighted average of the counts. Cases 2 and 3 may remove existing association rules, and cases 4 and 7 may add new association rules. If we retain all large and pre-large itemsets with their counts after each pass, then cases 2, 3 and case 4 can be handled easily. Also, in the maintenance phase, the ratio of new transactions to old transactions is usually very small. This is more apparent when the database is growing larger. It has been formally shown that an itemset in Case 7 cannot possibly be large for the entire updated database as long as the number of transactions is smaller than the number f shown below:[12)]

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

where f is the safety number of the new transactions, S_u is the upper threshold, S_l is the lower threshold, and d is the number of original transactions.

§3 The Proposed Prelarge-tree Structure and Maintenance Approach

Before the proposed structure and algorithm are stated, the notation used in the approach is first described below.

3.1 Notation

D: *the original database*;
T: *the set of new transactions*;
U: *the entire updated database*, i.e., $D \cup T$;
d: *the number of transactions in* D;
t: *the number of transactions in* T;
S_l: *the lower support threshold for pre-large itemsets*;
S_u: *the upper support threshold for large itemsets*, $S_u > S_l$;

I: *an itemset*;
$S^D(I)$: *the number of occurrences of I in D*;
$S^T(I)$: *the number of occurrences of I in T*;
$S^U(I)$: *the number of occurrences of I in U*;
P_ItemsD: *the set of pre-large items from D*;
P_ItemsT: *the set of pre-large items from T*;
L_ItemsT: *the set of large items from T*;
Insert_Items: *the set of items for which the new transactions have to be re-processed for updating the prelarge trees*;
Branch_Items: *the set of items for which the original database has to be re-processed for updating the prelarge trees*;
Rescan_Items: *the set of items for which the original database has to be re-scanned to determine whether the items are large.*

3.2 The Proposed Structure of Prelarge Trees

A prelarge tree must be built in advance from the initially original database before new transactions come. Its initial construction is stated as follows. The database is first scanned to find the large items which have their supports larger than the upper support threshold and the pre-large items which have their minimum supports lie between the upper and lower support thresholds. Next, the large and the pre-large items are sorted in descending frequencies. The database is then scanned again to construct the prelarge tree according to the sorted order of large and pre-large items. The construction process is executed tuple by tuple, from the first transaction to the last one. After all transactions are processed, the prelarge tree is completely constructed. The frequency values of large items and pre-large items are kept in the Header_Table and Pre_Header_Table, respectively. Besides, a variable c is used to record the number of new transactions since the last re-scan of the original database with d transactions.

3.3 The Proposed Pre-Large Tree Maintenance Algorithm

Based on the proposed structure of prelarge trees, the details of the corresponding algorithm to maintain the structure for incremental mining is described below.

The prelarge-tree maintenance algorithm:

INPUT: An old database consisting of $(d+c)$ transactions, its corresponding Header_Table and Pre_Header_Table, its corresponding prelarge tree, a lower support threshold S_l, an upper support threshold S_u, and a set of t new transactions.

OUTPUT: A new prelarge tree for the updated database.

STEP 1: Calculate the safety number f of new transactions according to the following formula:[12)]

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

STEP 2: Scan the new transactions to get all the items and their counts.

STEP 3: Divide the items in the new transactions into three parts according to whether they are large (appearing in the Header_Table), pre-large (appearing in the Pre_Header_Table) or small (not in the Header_Table or in the Pre_Header_Table) in the original database.

STEP 4: For each item I which is large in the original database, do the following substeps (***Cases 1***, ***2*** and ***3***):

Substep 4-1: Set the new count $S^U(I)$ of I in the entire updated database as:

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

where $S^D(I)$ is the count of I in the Header_Table (original database) and $S^T(I)$ is the count of I in the new transactions.

Substep 4-2: If $S^U(I)/(d+c+t) \geq S_u$, update the count of I in the Header_Table as $S^U(I)$, and put I in the set of *Insert_Items*, which will be further processed in STEP 8; Otherwise, if $S_u > S^U(I)/(d+c+t) \geq S_l$, remove I from the Header_Table, put I in the head of Pre_Header_Table with its updated frequency $S^D(I)$, and keep I in the set of *Insert_Items*; Otherwise, item I is still small after the database is updated; remove I from the Header_Table and connect each parent node of I directly to its child node in the prelarge tree.

STEP 5: For each item I which is pre-large in the original database, do the following substeps (***Cases 4***, ***5*** and ***6***):

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

$$S^U(I) = S^D(I) + S^T(I).$$

Substep 5-2: If $S^U(I)/(d+c+t) \geq S_u$, item I will be large after the database is updated; remove I from the Hedaer_Table, put I with its new frequency $S^D(I)$ in the end of Header_Table, and put I in the set of *Insert_Items*; Otherwise, if $S_u > S^U(I)/(d+c+t) \geq S_l$, item I is still pre-large after the database is updated; update I with its new frequency $S^D(I)$ in the Pre_Header_Table and put I in the set of *Insert_Items*; Otherwise, remove item I from the Pre_Header_Table.

STEP 6: For each item I which is neither large nor pre-large in the original database but large or pre-large in the new transactions (***Cases 7*** and ***8***), put I in the set of *Rescan_Items*, which is used when rescanning the database in STEP 7 is necessary.

STEP 7: If $t+c \leq f$ or the set of *Rescan_Items* is *null*, then do nothing; Otherwise, do the following substeps for each item I in the set of *Rescan_Items*:

Substep 7-1: Rescan the original database to decide the original count $S^D(I)$ of I.

Substep 7-2: Set the new count $S^U(I)$ of I in the entire updated database as:

$$S^U(I) = S^D(I) + S^T(I).$$

Substep 7-3: If $S^U(I)/(d+c+t) \geq S_u$, item I will become large after the database is updated; put I in both the sets of *Insert_Items* and *Branch_Items* and insert the items in the *Branch_Items* to the end of the Header_Table according to the descending order of their updated counts; Otherwise, if $S_u > S^U(I)/(d+c+t) \geq S_l$, item I will become pre-large after the database is update; put I in both the sets of *Insert_Items* and *Branch_Items*, and insert the items in the *Branch_Items* to the end of the Pre_Header_Table according to the descending order of their updated counts. Otherwise, do nothing.

Substep 7-4: For each original transaction with an item I existing in the *Branch_Items*, if I has not been at the corresponding branch of the prelarge tree for the transaction, insert I at the end of the branch and set its count as 1; Otherwise, add 1 to the count of the node I.

Substep 7-5: Otherwise, neglect I.

STEP 8: For each new transaction with an item I existing in the *Insert_Items*, if I has not been at the corresponding branch of the prelarge tree for the new transaction, insert I at the end of the branch and set its count as 1; otherwise, add 1 to the count of the node I.

STEP 9: If $t+c > f$, then set $d=d+t+c$ and set $c=0$; otherwise, set $c=t+c$.

In STEP 9, a corresponding branch is the branch generated from the large and pre-large items in a transaction and corresponding to the order of items appearing in the Header_Table and the Pre_Header_Table. After STEP 9, the final updated prelarge tree is maintained by the proposed algorithm. The new transactions can then be integrated into the original database. Based on the prelarge tree, the desired association rules can then be found by the FP-growth mining approach as proposed in[11] on only the large items.

§4 An Example

In this session, an example is given to illustrate the proposed algorithm for maintaining a prelarge tree when new transactions are inserted. Table 1 shows a database to be used in the example. It contains 10 transactions and 9 items, denoted a to i.

Assume the lower support threshold S_l is set at 30% and the upper one S_u at 50%. Here, not only the frequent items are kept in the prelarge tree but also the pre-large items. For the given database, the large items are b, c, f and d, and the pre-large items are a, e, h and g, from which the Header_Table and the Pre_Header_Table can be constructed. The prelarge tree is then formed from the database, the Header_Table and the Pre_Header_Table. The results are shown in Fig. 2.

Table 1 The Original Database in the Example

TID	Items
1	$a,\ b,\ c,\ f,\ g$
2	$a,\ b,\ c,\ f,\ g$
3	$a,\ d,\ e,\ f,\ h$
4	$e,\ h,\ i$
5	$e,\ d,\ h,\ f$
6	$b,\ c,\ d$
7	$b,\ d,\ i$
8	$b,\ c,\ d$
9	$b,\ c,\ e,\ f,\ h$
10	$a,\ b,\ c,\ f,\ g$

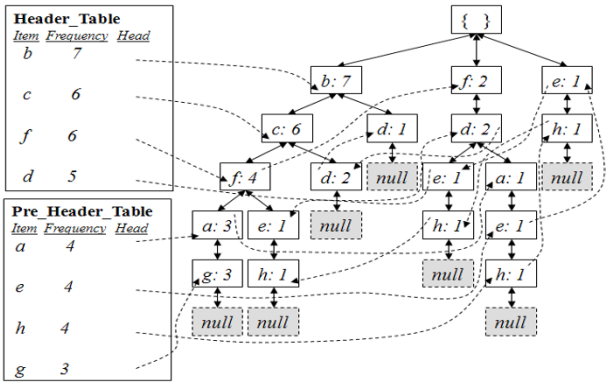


Fig. 2 The Header_Table, Pre_Header_Table and the Prelarge Tree Constructed

Assume the three new transactions shown in Table 2 appear. The proposed prelarge-tree maintenance algorithm proceeds as follows. The variable c is initially set at 0.

Table 2 The Three New Transactions

TID	Items
11	$a,\ b,\ c,\ e,\ f$
12	$e,\ h,\ i$
13	$d,\ e,\ f,\ h$

STEP 1: The safety number f for new transactions is calculated as:

$$f = \left\lfloor \frac{(S_u - S_l)d}{1 - S_u} \right\rfloor = \left\lfloor \frac{(0.5 - 0.3)10}{1 - 0.5} \right\rfloor.$$

STEP 2: The three new transactions are first scanned to get the items and their counts. The results are shown in Table 3.

STEP 3: All the items a to i in Table 3 are divided into three parts, $\{b\}\{c\}\{f\}\{d\}$, $\{a\}\{e\}\{h\}\{g\}$, and $\{i\}$ according to whether they are large (appearing in the Header_Table), pre-large (appearing in the Pre_Header_Table) or small in the original database. The results are shown in Table 4, where the

Table 3 The Counts of All Items in the New Transactions

New transactions					
Item	Count	Item	Count	Item	Count
a	1	d	1	g	0
b	1	e	3	h	2
c	1	f	2	i	1

Table 4 Three Partitions of the Items from the New Transactions

Pre-large items in the original database		*Pre-large items in the original database*		*Small items in the original database*	
Item	Count	Item	Count	Item	Count
b	1	a	1	i	1
c	1	e	3		
f	2	h	2		
d	1	g	0		

counts are only from the new transactions.

STEP 4: The items in the new transactions which are large in the original database are first processed. In this example, items b, c, f and d (the first partition) satisfy the condition and are processed. Take item b as an example to illustrate the substeps. The count of item b in the Header_Table is 7, and its count in the new transactions is 1. The new count of item b is thus 7+1 (= 8). The new support ratio of item b is 8/(10+0+3), which is larger than the minimum support threshold, 0.5. Item b is thus still a large item after the database is updated. The frequency value of item b in the Header_Table is thus changed as 8, and item b is then put into the set of *Insert_Items*. Items c and f are similarly processed. Item d will, however, become pre-large after the database is updated. The item d is thus removed from the Header_Table and put into the head of the Pre_Header_Table with its updated frequency value and into the set of *Insert_Items*.

STEP 5: The items in the new transactions which are pre-large in the original database are processed. They include items a, e, h and g. Take item a as an example to illustrate the substeps. The count of item a in the Pre_Header_Table is 4, and its count in the new transactions is 1. The new count of item a is thus 4+1(= 5). The new support ratio of item a is 5/(10+0+3), which lies between 0.3 and 0.5. Item a is thus still a pre-large item after the database is updated. The frequency value of item a in the Pre_Header_Table is thus changed as 5, and item a is then put into the set of *Insert_Items*. Item h is similarly processed. The count of item e in the Pre_Header_Table is 4, and its count in the new transactions is 3. The new count of item e is thus 4 + 3 (=7). The new support ratio of item e is then 7/(10+0+3), which is larger than 0.5. Item e will thus become large after the database is updated. It is then removed from the Pre_Header_Table and put in the end of the Header_Table and in the set of *Insert_Items*. The frequency value of item e in the Hedaer_Table is thus changed as 7. At last, item g will become small after the database is updated. Item g is thus removed from the Pre_Header_Table and from the prelarge tree. After STEP 5, *Insert_Items*=$\{a, b, c, d, e, f, h\}$.

STEP 6: Since the item i is neither large nor pre-large in the original database (not appearing in the Header_Table and in the Pre_Header_Table), but large in the new transactions, it is put into the set of *Rescan_Items*, which is used when rescanning in STEP 7 is required. After STEP 6, *Rescan_Items*=$\{i\}$.

STEP 7: Since $t+c=3+0 < f(=4)$, rescanning the original database is unnecessary. Nothing is done in this step.

STEP 8: The prelarge tree is updated according to the new transactions with items existing in the *Insert_Items*. In this example, *Insert_Items* = $\{a, b, c, d, e, f, h\}$. The corresponding branches for the new transactions with any of these items are shown in Table 5.

Table 5 Three Partitions of the Items from the New Transactions

TID	Items	Corresponding branches
1	a, b, c, e, f	b, c, f, a, e
2	e, h, i	e, h
3	d, e, f, h	f, d, e, h

The first branch shares the same prefix (b, c, f, a) as the current prelarge tree. The counts for items b, c, f and a are then increased by 1 since they have not yet counted in the construction of the previous prelarge tree. A new node (e:1) is thus created and linked to (a: 4) as its child. The same process is then executed for the other two branches. The final results are shown in Fig. 3.

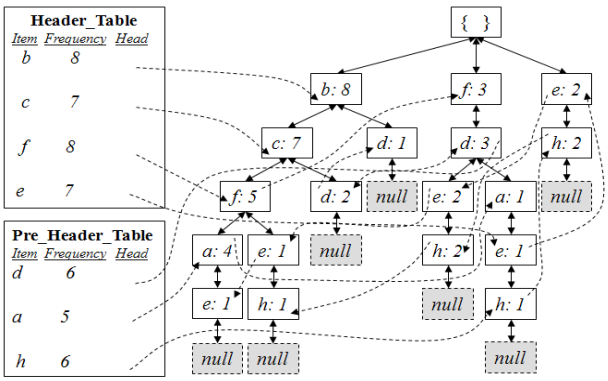


Fig. 3 The Final Results of the Prelarge Tree

STEP 9: Since $t\ (=3) + c\ (=0) < f(=4)$, set $c = t + c = 3 + 0 = 3$. After STEP 9, the prelarge tree is updated. Note that the final value of c is 3 in this example and f - c = 1. This means that one more new transaction can be added without rescanning the original database for Case 7. Based on the prelarge tree shown in Fig. 3, the desired large itemsets can then be found by the FP-growth mining approach as proposed in the reference[11] on only the large items.

§5 Experimental Results

Experiments were made to compare the performance of the batch FP-tree

construction algorithm, the FUFP-tree maintenance algorithm and the prelarge-tree maintenance algorithm. When new transactions came, the batch FP-tree construction algorithm integrated new transactions into the original database and constructed a new FP-tree from the updated database. The process was executed whenever new transactions came. The incremental FUFP-tree maintenance algorithm and the prelarge-tree maintenance algorithm processed new transactions incrementally in the way mentioned before.

A real dataset called *BMS-POS*[24] were used in the experiments. This dataset was also used in the KDDCUP 2000 competition. The BMS-POS dataset contained several years of point-of-sale data from a large electronics retailer. Each transaction in this dataset consisted of all the product categories purchased by a customer at one time. There were 515,597 transactions with 1657 items in the dataset. The maximal length of a transaction was 164 and the average length of the transactions was 6.5. The first 500,000 transactions were extracted from the *BMS-POS* database to construct an initial tree structure. The value of the minimum support thresholds were set at 7% to 15% for the three maintenance algorithms, with 2% increment each time. The next 3,000 transactions were then used in incremental mining. For the prelarge-tree maintenance algorithm, the lower minimum support thresholds were set as the values of the minimum support threshold minus 0.8%, which are 6.2%, 8.2%, 10.2%, 12.2% and 14.2%, respectively. Figure 4 shows the execution times of the three algorithms for different threshold values.

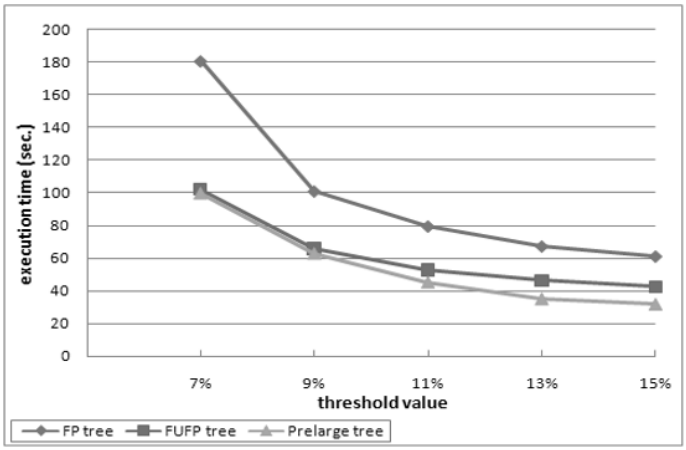


Fig. 4 The Comparisons of the Execution Times

It can be observed from Fig. 4 that the proposed prelarge-tree maintenance algorithm ran faster than the other two. When the minimum support threshold was larger, the effects became more obvious. Note that the FUFP-tree maintenance algorithm and the prelarge-tree maintenance algorithm may generate a less concise tree than the FP-tree construction algorithm since the latter completely follows the sorted frequent items to build the tree. As mentioned above, when an originally small item becomes large due to new transactions, its

updated support is usually only a little larger than the minimum support. It is thus reasonable to put a new large item at the end of the Headrer_Table. The difference between the FP-tree, the FUFP-tree and the prelarge-tree structures (considering large items) will thus not be significant. For showing this effect, the comparison of the numbers of nodes for the three algorithms is given in Fig. 5. It can be seen that the three algorithms generated nearly the same sizes of trees. The effectiveness of the prelarge-tree maintenance algorithm is thus acceptable.

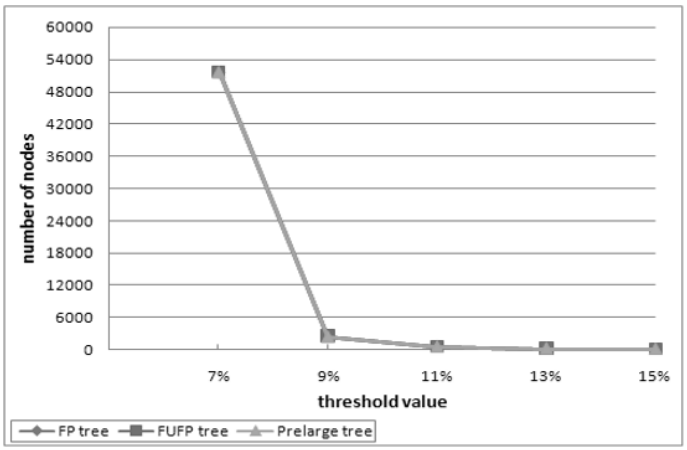


Fig. 5 The Comparisons of the Numbers of Nodes

Experiments were then made to show the execution times and the numbers of nodes of the three algorithms for different numbers of transactions inserted. The first 500,000 transactions were extracted from the *BMS-POS* database to construct an initial FP-tree. The next 3,000 transactions were then sequentially used each time as new transactions for the experiments. The minimum support threshold values set at 15% to demostrate the execution times and the number of nodes. The results are shown in Figs. 6 and 7, respectively. Note the execution times and the number of nodes were measured for each sequential 3,000 transactions.

As mentioned before, when the number of inserted transactions reached the safety number, the original database would be processed again. In the experiments, when the lower threshold value was set at 14.2% and the upper threshold value was set at 15%, the safety number was calculated as f = 500,000*(0.15-0.142)/(1-0.15) = 4,705. The processing times for 6,000 and 12,000 new transactions were thus more than those for the others in Fig. 6. Besides, it can be seen that the prelarge-tree maintenance algorithm ran faster than the other two in Fig. 6. It can also be seen that the prelarge-tree maintenance algorithm had nearly the same (frequent) node numbers as the other two in Fig. 7.

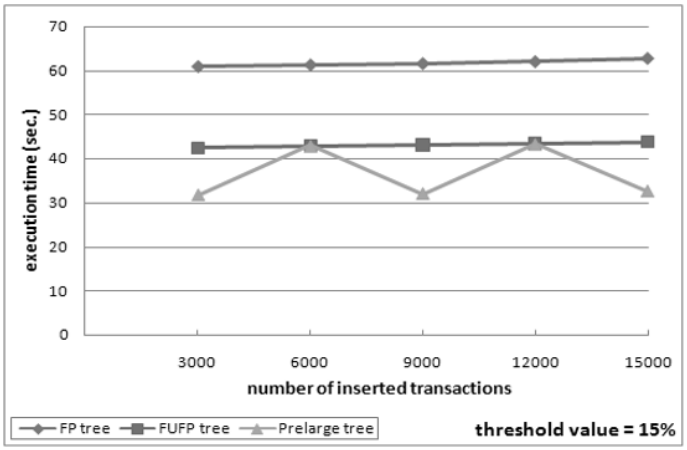


Fig. 6 The Comparisons of the Execution Times at the 15% Threshold

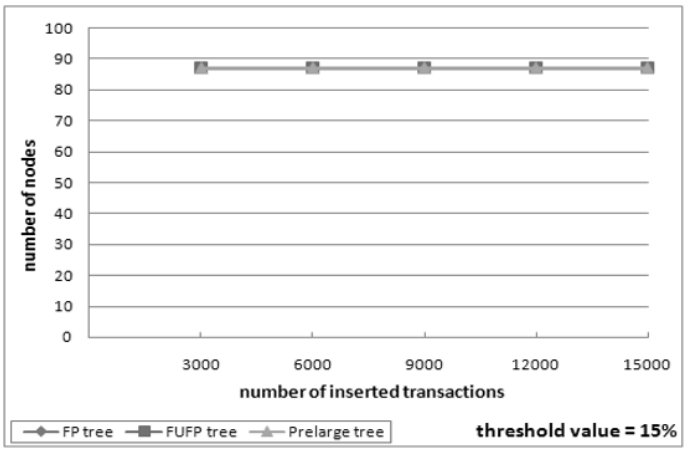


Fig. 7 The Comparisons of the Numbers of Nodes at the 15% Threshold

§6 Conclusions

In this paper, we have proposed the prelarg-tree maintenance algorithm for incremental mining based on the concept of pre-large itemsets. The prelarge-tree structure is used to efficiently and effectively handle new transactions. Using two user-specified upper and lower support thresholds, the pre-large items act as a gap to avoid small items becoming large in the updated database when transactions are inserted. When new transactions are added, the proposed prelarge-tree maintenance algorithm processes them to maintain the prelarge tree. It first partitions items of new transactions into three parts according to whether they are large, pre-large or small in the original database. Each part is then processed in its own way. The Header_Table, the Pre_Header_Table, and the prelarge-tree are correspondingly updated whenever necessary.

Experimental results also show that the proposed prelarge-tree mainte-

nance algorithm runs faster than the batch FP-tree and the FUFP-tree algorithm for handling new transactions and generates nearly the same number of frequent nodes as them. The proposed approach can thus achieve a good trade-off between execution time and tree complexity.

References

1) Agrawal, R., Imielinksi T. and Swami, A., "Mining association rules between sets of items in large database," in *The ACM SIGMOD Conference*, pp. 207-216, 1993.
2) Agrawal, R., Imielinksi T. and Swami, A., "Database mining: a performance perspective," *IEEE Transactions on Knowledge and Data Engineering, 5, 6*, pp. 914-925, 1993.
3) Agrawal R. and Srikant, R., "Fast algorithm for mining association rules," in *The International Conference on Very Large Data Bases*, pp. 487-499, 1994.
4) Agrawal, R., Srikant R. and Vu, Q., "Mining association rules with item constraints," in *The Third International Conference on Knowledge Discovery in Databases and Data Mining*, pp. 67-73, 1997.
5) Chen, M. S., Han J. and Yu, P. S., "Data mining: An overview from a database perspective," *IEEE Transactions on Knowledge and Data Engineering, 8, 6*, pp. 966-883, 1996.
6) Cheung, D. W., Han, J., Ng, V. T. and Wong, C. Y., "Maintenance of discovered association rules in large databases: An incremental updating approach," in *The Twelfth IEEE International Conference on Data Engineering*, pp. 106-114, 1996.
7) Cheung, D.W., Lee, S. D. and Kao, B., "A general incremental technique for maintaining discovered association rules," in *Database Systems for Advanced Applications*, pp. 185-194, 1997.
8) Ezeife, C. I., "Mining incremental association rules with generalized FP-tree," in *The 15th Conference of the Canadian Society for Computational Studies of Intelligence on Advances in Artificial Intelligence*, pp. 147-160, 2002.
9) Fukuda, T., Morimoto, Y., Morishita S. and Tokuyama, T., "Mining optimized association rules for numeric attributes," *Journal of Computer and System Sciences, 58, 1*, pp. 182-191, 1996.
10) Han J. and Fu, Y., "Discovery of multiple-level association rules from large database," in *The Twenty-first International Conference on Very Large Data Bases*, pp. 420-431, 1995.
11) Han, J., Pei, J. and Yin, Y., "Mining frequent patterns without candidate generation," in *The 2000 ACM SIGMOD International Conference on Management of Data*, pp. 1-12, 2000.
12) Hong, T. P., Wang, C. Y. and Tao, Y. H., "A new incremental data mining algorithm using pre-large itemsets," *Intelligent Data Analysis, 5, 2*, pp. 111-129, 2001.
13) Hong, T. P., Lin, C. W. and Wu, Y. L., "Incrementally fast updated frequent pattern trees," *Expert Systems with Applications 34, 4*, pp. 2424-2435, 2008.
14) Koh, J. L. and Shieh, S. F., "An effcient approach for maintaining association rules based on adjusting FP-tree structure," in *The Ninth International Conference on Database Systems for Advanced Applications*, pp. 417-424, 2004.

15) Lin, M. Y. and Lee, S. Y., "Incremental update on sequential patterns in large databases," in *The Tenth IEEE International Conference on Tools with Artificial Intelligence*, pp. 24-31, 1998.

16) Mannila, H., Toivonen, H. and Verkamo, A. I., "Efficient algorithm for discovering association rules," in *The AAAI Workshop on Knowledge Discovery in Databases*, pp. 181-192, 1994.

17) Park, J. S., Chen, M. S. and Yu, P. S., "Using a hash-based method with transaction trimming for mining association rules," *IEEE Transactions on Knowledge and Data Engineering, 9, 5*, pp. 812-825, 1997.

18) Qiu, Y., Lan, Y. J. and Xie, Q. S., "An improved algorithm of mining from FP-tree," in *The Third International Conference on Machine Learning and Cybernetics*, pp. 26-29, 2004.

19) Sarda, N. L. and Srinivas, N. V., "An adaptive algorithm for incremental mining of association rules," in *The Ninth International Workshop on Database and Expert Systems*, pp. 240-245, 1998.

20) Srikant, R. and Agrawal, R., "Mining generalized association rules," in *The Twenty-first International Conference on Very Large Data Bases*, pp. 407-419, 1995.

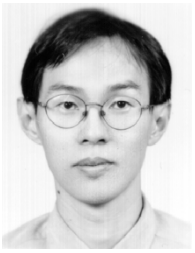
21) Srikant, R. and Agrawal, R., "Mining quantitative association rules in large relational tables," in *ACM SIGMOD International Conference on Management of Data*, pp. 1-12, 1996.

22) Zaiane, O. R. and Mohammed, E. H., "COFI-tree mining: A new approach to pattern growth with reduced candidacy generation," in *IEEE International Conference on Data Mining*, 2008.

23) Zhang, S., "Aggregation and maintenance for database mining," *Intelligent Data Analysis*, pp. 475-490, 1999.

24) Zheng, Z., Kohavi, R. and Mason, L., "Real world performance of association rule algorithms," in *The International Conference on Knowledge Discovery and Data Mining*, pp. 401-406, 2001.

Chun-Wei Lin: He received his B.S. and M.S. degrees from the Department of Information Management in I-Shou University, Taiwan, in 2002 and 2006, respectively. He is currently a Ph.D. student in computer science and information engineering in National Cheng Kung University. His research interests include data mining, web mining, fuzzy theory and ontology.



Tzung-Pei Hong, Ph.D.: He received his B.S. degree in chemical engineering from National Taiwan University in 1985, and his Ph.D. degree in computer science and information engineering from National Chiao-Tung University in 1992. He was in charge of the whole computerization and library planning for National University of Kaohsiung in Preparation from 1997 to 2000 and served as the first director of the library and computer center in National University of Kaohsiung from 2000 to 2001, as the Dean of Academic Affairs from 2003 to 2006 and as the Vice President from 2007 to 2008. He is currently a professor at the Department of Computer Science and Information Engineering and at the Department of Electrical Engineering. His current research interests include parallel processing, machine learning, data mining, soft computing, management information systems, and www applications.

Wen-Hsiang Lu, Ph.D.: He received his B.S., M.S., and Ph.D. degrees in computer science and information engineering from National Chiao Tung University, Hsinchu, Taiwan. He is an Assistant Professor in the Department of Computer Science and Information Engineering (CSIE) at National Cheng Kung University, Tainan, Taiwan. His current research focuses on web mining, information retrieval, natural language processing, and medical informatics.