Effective and Efficient Utility Mining Technique for Incremental Dataset

Kavitha JeyaKumar[1], Manjula Dhanabalachandran[1], and Kasthuri JeyaKumar[2]
[1]Department of Computer Science and Engineering, Anna University, India
[2]Department of Electronics and Communication Engineering, Sri Ramaswami Memorial University, India

Abstract: *Traditional association rule mining, which is based on frequency values of items, cannot meet the demands of different factors in real world applications. Thus utility mining is presented to consider additional measures, such as profit or price according to user preference. Although several algorithms were proposed for mining high utility itemsets, they incur the problem of producing large number of candidate itemsets, results in performance degradation in terms of execution time and space requirement. On the other hand when the data come intermittently, the incremental and interactive data mining approach needs to be processed to reduce unnecessary calculations by using previous data structures and mining results. In this paper, an incremental mining algorithm for efficiently mining high utility itemsets is proposed to handle the above situation. It is based on the concept of Utility Pattern Growth (UP-Growth) for mining high utility itemsets with a set of effective strategies for pruning candidate itemsets and Fast Update (FUP) approach, which first partitions itemsets into four parts according to whether they are high-transaction weighted utilization items in the original and newly inserted transactions. Experimental results show that the proposed Fast Update Utility Pattern Tree (FUUP) approach can thus achieve a good trade between execution time and tree complexity.*

Keywords: *Data mining, utility mining, incremental mining.*

Received January 30, 2014; accepted October 14, 2014

1. Introduction

In the traditional data mining techniques, finding association rules [1, 2, 3, 11, 12, 13, 14, 21, 22, 23, 24] in transactional databases is most commonly seen. In the past, most of the algorithms for mining association rules based on Apriori algorithm [1]. The mining process first finds frequent itemsets based on user defined support threshold and then generates association rules from the frequent itemsets based on user defined confidence threshold. This level-by-level approach cause iterative database scans and high computational costs.

On the contrary, the pattern-growth approaches construct tree structures to recursively find association rules without generating candidate itemsets. One of the most important is Frequent-Pattern Tree (FP-Tree) mining algorithm [14]. The construction process was executed tuple by tuple from the first transaction to the last one. After that, a recursive mining procedure called Frequent-Pattern Growth (FP-Growth) was executed to derive frequent patterns from the FP-Tree. They showed the approach could have better performance than Apriori.

The above mentioned approaches are based on frequency values of items, which cannot meet the demands of different factors like profit or price in real world applications. Utility mining [5, 29, 30] was proposed to solve the above mentioned problem by considering the factors like cost, profit or other scales of user preference. Thus the issue of high utility itemsets mining is raised and many studies [4, 9, 17, 19, 20, 25,26, 27] have addressed this problem. Liu *et al.* [19, 20] proposed the two phase utility mining algorithm for efficiently extracting all high utility itemsets based on the downward closure property. Although, two phase algorithm reduces search space, it still generates too many candidates and requires multiple database scans. To overcome this problem, Li *et al.* [17] proposed an Isolated Items Discarding Strategy (IIDS) to reduce the number of candidates. To efficiently generate High Transaction Weighted Itemsets (HTWI's) and to avoid multiple scans, Ahmed *et al.* [4] proposed a tree based algorithm named Incremental High Utility Pattern (IHUP). Although, IHUP achieves better performance than IIDS and two-phase, it still produces too many HTWI's in phase 1. As advancement, Tseng *et al.* [27] proposed Utility Pattern Growth (UP-Growth) for mining high utility itemsets with a set of effective strategies for pruning candidate itemsets.

In real world applications, the problem of discovering the frequent itemsets becomes more time consuming if the dataset is incremental in nature. This may introduce new frequent itemsets and some existing itemsets would become invalid. Several approaches [6, 7, 8, 10, 11, 15, 18, 28] are proposed to address this issue. Thus, designing an efficient algorithm that can maintain association rules as a database grows is thus critically important. One

noticeable incremental mining algorithm was the Fast-Updated Algorithm (FUP) which was proposed by Cheung *et al.* [6] for avoiding shortcomings mentioned above. It primary calculates frequent itemsets from new transactions and compares them with the previous found frequent itemsets from the original database. Different procedures are then done according to the comparison results. For some cases, FUP can avoid or reduce the number of re-scanning the original database, thus saving computation time in incremental mining.

In this paper we focus on incremental utility mining. An incremental mining Fast Update Utility Pattern (FUUP) tree algorithm for efficiently mining high utility itemsets is proposed to handle the above mention situation. It is based on the concept of UP-Growth for mining high utility itemsets with a set of effective strategies for pruning candidate itemsets and FUP approach, which first partitions itemsets into four parts according to whether they are high-transaction weighted utilization items in the original and newly inserted transactions. Experimental results show that the proposed FUUP tree approach can thus achieve a good trade between execution time and tree complexity. The remainder of this paper is organized as follows. Related works are reviewed in section 2. High utility pattern mining problem definition described in section 3. The proposed incremental FUUP mining algorithm is described in section 4. An example to illustrate the proposed algorithm is described in section 5. Experimental results for showing the performance of the proposed approach are provided in section 6. Conclusion is finally given in section 7.

2. Review of Related Works

In this section, some related researches are briefly reviewed. They are the FUP algorithm and the concept of mining high utility itemsets.

2.1. The FUP Algorithm

In real-world application, data mining is used to extract the useful patterns or rules from a large number of database for making the important decisions or strategies. One common type of data mining is to discover association rules from transaction data, such that the presence of certain items in a transaction implies the presence of some other items. In the past, [1,2] then proposed several mining algorithms to find association rules in a level-wise mining procedure. In addition, [14] the FP-Tree algorithm for efficiently mining frequent itemsets without generation of candidate itemsets.

In real world applications, transactions in a database do not usually remain a stable condition. Some new association rules may be generated and some old ones may become invalid. Conventional batch-mining algorithms solve this problem by re-processing the entire updated database when new transactions are inserted into the original database. They, however, require lots of computational time and waste existing discovered knowledge. Cheung *et al.* [6] proposed the FUP algorithm to effectively update the discovered association rules in incremental mining. Considering an original database and some new transactions, the following four cases (illustrated in Figure 1) may occur:

- *Case* 1. An itemset is frequent (large) both in an original database and in newly inserted transactions.
- *Case* 2. An itemset is frequent (large) in an original database but not frequent (small) in newly inserted transactions.
- *Case* 3. An itemset is not frequent (small) in an original database and but frequent (large) in newly inserted transactions.

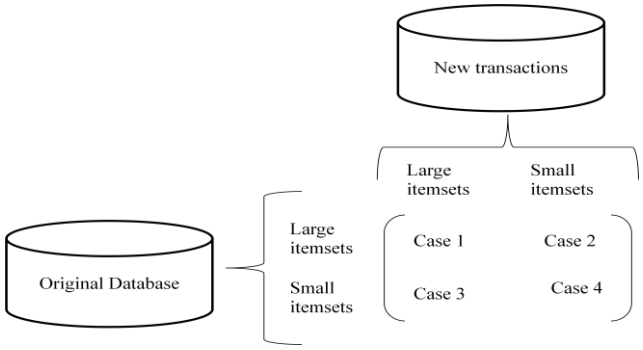

Figure 1. Four cases when new transactions are inserted into existing database.

- *Case* 4. An itemset is not frequent (small) both in an original database and in newly inserted transactions.

Since itemsets in case 1 are frequent (large) in both the original database and the new transactions, they will still be frequent (large) after the weighted average of the counts .Similarly, itemsets in case 4 will still be non-frequent (small) after the new transactions are inserted. Thus cases 1 and 4 will not affect the final frequent (large) itemsets. Case 2 may remove existing frequent (large) itemsets, and case 3 may add new frequent (large) itemsets. Based on the FUP approach, the cases 1, 2, and 4 are more efficiently handled than conventional batch mining algorithms.

2.2. Mining High Utility Itemsets

In the Past, several mining algorithms were proposed for efficiently discovering high utility itemsets. Yao *et al.* [29] proposed an algorithm for efficiently mining high utility itemsets based on mathematical properties of utility constraints. Two pruning strategies based on utility upper bounds and expected utility upper bounds respectively were adopted to reduce the search space. These pruning strategies were then incorporated into

the mining approach Umining and its heuristic successor, Umining_H [30]. Liu *et al.* [19, 20] designed a two phase algorithm for efficiently discovering all high utility itemsets. It consisted of two phases to generate and test high utility was first used as effective upper bound of each candidate itemset in the transaction such that the "transaction-weighted downward closure property" could be kept in the search space to decrease the number of candidate itemsets. An additional database scan was then performed to find out the real utility values of the remaining candidates and identifies the high utility itemsets.

Although, two phase algorithm reduces search space, it still generates too many candidates and requires multiple database scans. To overcome this problem, Li *et al.* [17] proposed an IIDS to reduce the number of candidates. To efficiently generate HTWI's and to avoid multiple scans, Ahmed *et al.* [4] proposed a tree based algorithm named IHUP. Although IHUP achieves better performance than IIDS and Two-phase, it still produces too many HTWI's in phase I. Such a large number of HTWUI's will degrade the mining performance in phase I substantially in terms of execution time and memory consumption.

As stated above, the number of generated HTWUI's is a critical issue for the performance of algorithms. As advancement, Tseng *et al.* [27] proposed UP-Growth for mining high utility itemsets with a set of effective strategies for pruning candidate itemsets. Correspondingly, a compact tree structure, called UP-Tree, was designed to maintain the important information of the transaction database related to the utility patterns. High utility itemsets are then generated from the UP-Tree efficiently with only two scans of the database. Four strategies mentioned in Table 1, are used for efficient construction of UP-Tree and the processing in UP-Growth. By these strategies, the estimated utilities of candidates can be well reduced by discarding the utilities of the items which are impossible to be high utility or not involved in the search space. The strategies can not only efficiently decrease the estimated utilities of the potential high utility itemsets but also effectively reduce the number of candidates.

Table 1. Four strategies of UP-Growth.

Strategies	Explanation
DGU	Discarding global unpromising items and their actual utilities from transactions and transaction utilities of the database.
DGN	Decreasing global node utilities for the nodes of global up-tree by actual utilities of descendant nodes during the construction of global UP-Tree.
DLU	Discarding local unpromising items and their estimated utilities from the paths and path utilities of conditional pattern bases.
DLN	Decreasing local node utilities for the nodes of local up-tree by estimated utilities of descendant nodes during the construction of global UP-Tree

3. Problem Definition

In this section, we first give some definition similar to those presented in the previous works [19, 29, 30] and define the problem of utility mining. Let $I=\{i_1, i_2, \ldots, i_j, \ldots, i_m\}$ be a set of items and D be a transaction database $\{T_1, T_2, \ldots, T_k, \ldots, T_n\}$ where each transaction $T_i \in D$ is a subset of I.The calculations are done according to the according to the original database and utility given in Tables 2 and 3.

- *Definition* 1: The quantity of item i_j in transaction T_k called local transaction utility or internal utility value, represented as $l(i_j, T_k)$. For eg., $l(A, T_3)$.
- *Definition* 2: The unit profit value of each item i_j called external utility $P(i_j)$. For eg., $P(B)=150$.
- *Definition* 3: The utility for an item i_j in transaction T_k is the quantitative measure of Equation 1.

$$u(i_j, T_k)=l(i_j, T_k)\times P(i_j) \quad (1)$$

For eg., $a(A, T_3)=3\times3=9$.

- *Definition 4*: The utility of an itemset X in transaction T_k, is defined by Equation 2.

$$u(X, T_k) = \sum_{i_j \in X} u(i_j, T_k) \quad (2)$$

Where, X is the set of k distinct items $\{i_1, i_2, \ldots, i_k\}$, $X \subseteq T_k$, $i_j \in I$, $1 \le j \le k$, k is the length of X. An itemset with length k is called k-itemset.

For eg., $u(AB, T_1)=3\times3+2\times150=9+300=309$.

- *Definition* 5*:* The utility of an itemset X is defined by Equation 3.

$$u(X) = \sum_{T_k \in D} \sum_{i_j \in X} u(i_j, T_k) \quad (3)$$

For eg., $u(AB)=u(AB, T_1)+u(AB, T_6)+u(AB, T_7)=309+303+456=1068$.

Table 2. An original database in the example.

TID	A	B	C	D	E	F	tu
1	3	2	0	3	0	0	459
2	2	0	0	4	2	0	406
3	3	0	5	0	0	3	74
4	1	0	3	0	1	2	146
5	1	0	0	3	2	0	353
6	1	2	0	4	0	0	503
7	2	3	2	0	1	1	578
8	0	0	0	0	0	2	40
9	0	0	3	3	0	0	153
10	3	0	0	4	0	0	209

Table 3. The utility table.

Item	Profit($)
A	3
B	150
C	1
D	50
E	100
F	20

- *Definition* 6: The accumulated utility value of the items in each transaction T_k, is the transaction utility (tu) is defined by Equation 4.

$$tu(T_k) = \sum_{i_j \in T_k} u(i_j, T_k) \quad (4)$$

For eg., $tu(T_1)= u(A, T_1)+ u(B, T_1)+ u(D, T_1)=3\times3+ 2\times150+ 3\times50= 9+ 200+ 150= 459$.

- *Definition* 7: The minimum utility threshold λ given by the percentage of total transaction utility values of the database. In Table 3, the summation of all the transaction utility values is 2921. If λ is 35%, then the minimum utility count is defined as Equation 5.

$$muc = \lambda \times \sum_{T_k \in D} tu(T_k) = 0.35 \times 2921 = 1022.35 \quad (5)$$

- *Definition* 8: An itemset X is a high utility itemset if the following condition in Equation 6 satisfied.

$$u(X)>=muc \quad (6)$$

For eg., $u(AB)$=1068(≥1022.35).

AB is a high utility itemset.

- *Definition* 9*:* The Transaction-Weighted Utility (TWU) of an itemset X is the summation of all Transaction Utility (tu) containing X in Equation 7.

$$twu(X) = \sum_{X \subseteq T_k \in D} tu(T_k) \quad (7)$$

For eg., $twu(AB)= tu(T_1)+ tu(T_6)+ tu(T_7)= 459+ 503+ 578= 1540$.

- *Definition* 10: An itemset X is a high transaction weighted utilization itemset if Equation 8 satisfied.

$$twu(X)>=muc \quad (8)$$

For eg., $twu(AB)$=1540(≥1022.35).

AB is a High Transaction Weighted Utilization itemset (HTWU). The downward closure property can be maintained using transaction weighted utilization. For any itemset X, if X is not a HTWU, any superset of X is a low utility itemset.

4. The Proposed Incremental FUUP Mining Algorithm

An FUUP tree must be built in advance from the initially original database before new transactions come. Its initial construction is similar to that of an UP tree according to the strategy of Discarding Global Unpromising (DGU) and DGN. The database is first scanned to find the items with their TWU larger than a minimum utility threshold, which called promising items. Other items are called unpromising. Next, the promising items are sorted in descending order and reorganized transaction utility is evaluated. At last, the reorganized transaction is scanned again to construct the tree according to the sorted order of promising items. The construction process is executed tuple by tuple, from the first transaction to the last one. After all transactions are processed, the final UP tree for the original database is completely constructed.

When new transactions are added, the proposed incremental maintenance algorithm will process them to construct the FUUP tree. The new transactions are first scanned to find the promising and unpromising items according to the TWU of newly inserted transactions. Then, it 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 FUUP tree are correspondingly updated whenever necessary.

In the process for updating the FUUP tree, item deletion is done before item insertion. When an originally large item becomes small, it is directly removed from the FUUP tree and its parent and child nodes are then linked together. On the contrary, when an originally small items becomes large, it is added to the end of the header-table and then inserted into the leaf nodes of FUUP tree. It is reasonable to insert the item at the end of the header-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 FUUP tree can thus be least updated in this way, and the performance of the proposed incremental algorithm can be greatly improved. The entire FUUP tree can be re-constructed in a batch way when a sufficiently large number of transactions are inserted. The details of the proposed algorithm are described below.

Algorithm 1: Incremental FUUP Mining

Input: D, Utility table, λ, , d
Output: FUUP tree
for each i_j in T_k find $u(i_j, T_k)$,$tu(T_k)$
find muc^d, $twu^d(X)$
if($twu^d(X)> muc^d$)
  put X in $HTWUI^d$
for each X in both $HTWUI^D$ and $HTWUI^d$
  update $twu^u(X)$ and put in $HTWUI^U$
for each X in $HTWUI^D$ and not in $HTWUI^d$
  update $twu^u(X)$
  If $twu^u(X)>muc^U$ put in $HTWUI^U$
  Else remove X from FUUP tree
for each X not in $HTWUI^D$ and in $HTWUI^d$
  Rescan X in D and update $twu^u(X)$
  If $twu^u(X)>muc^U$ put in $HTWUI^U$and Rescan items
Sort X in Rescan items
Reorganize and insert at the end of FUUP tree

- *Explanation*:
- *Input*: An original database D, its corresponding Header-table storing the High Transaction Weighted Utility Items and UP-Tree ($HTWUI^D$), an utility table (each of all items with profit value), a minimum utility threshold (λ), minimum utility count of original database muc^D, and a set of d new transactions.
- *Output*: A new FUUP tree for the updated database by using the algorithm.
- *Step* 1. Calculate the utility value $u(i_j, T_k)$ of each item i_j in each transaction T_k and find out the corresponding transaction utility $tu(T_k)$.

- *Step* 2. Calculate the minimum utility count of new transactions ($muc^d = \lambda \times \sum_{T_k \in d} tu(T_k)$).
- *Step* 3. Calculate the *twu* from the new transactions as the summation of the utilities of the new transactions $twu^d(X) = \sum_{X \subseteq T_k \in d} tu(T_k)$.
- *Step* 4. Check whether the $twu^d(X)$ of each items from the new transactions is larger than are equal to the Minimum Utility Count (muc^d) in the new transactions. If *X* satisfied the condition, put it in the set of $HTWUI^d$ from the new transactions.
- *Step 5*. For each item *X* in the set of $HTWUI^D$ from the original database, if it also appears in the set of $HTWUI^d$ in the new transactions, do the following substeps (Case 1):
  1. Set the updated transaction–weighted utility of item *X* in the header-table as $(X)=twu^D(X)+ twu^d(X)$. Where, $twu^d(X)$ is the TWU items *X* in the original database $HTWUI^D$ and $twu^d(X)$ is the TWU items *X* in the new transactions $HTWUI^d$ respectively.
  2. Put X in the set of updated $HTWUI^U$, which will be processed in Step 8.
- *Step* 6. For each item *X* in the set of $HTWUI^D$ from the original database, if it does not appear in the set of $HTWUI^d$ in the new transactions, do the following substeps (Case 2):
  1. Set the updated TWU of item *X* as $twu^u(X)= twu^D(X) + twu^d(X)$.
  2. Check the condition that whether the updated $twu^u(X)$ is larger than or equal to the updated minimum utility count(muc^D+muc^d).
  3. If X satisfied the above condition (ie., item *X* still large after the database is updated), put it in the set of updated $HTWUI^U$ and update the *twu* of *X* in the headear-table.
  4. Otherwise if *X* does not satisfy the condition (ie., item *X* will become small after the database is updated), remove *X* from the header-table. Connect each parent node of *X* directly to the corresponding child node of *X* and remove *X* from the FUUP tree.
- *Step 7*. For each item *X*, if it does not appear in the set of $HTWUI^D$ from the original database, but appears in the set of $HTWUI^d$ in the new transactions, do the following substeps (Case 3):
  1. Rescan the original database to determine the Transaction-Weighted Utility (TWU^D) of item *X* .
  2. Set the updated transaction–weighted utility of item *X* as $twu^u(X)= twu^D(X) + twu^d(X)$.
  3. Check the condition that whether the updated $twu^u(X)$ is larger than or equal to the updated Minimum Utility Count (muc^D+muc^d). If *X* satisfied the above condition, item *X* will be large after the database is updated. Add item *X* both in the set of updated High Transaction Weighted Utility Itemset in Updated database ($HTWUI^{U)}$ and in the set of Rescan-item and put the transaction ID's with item *X* in the set of rescan-transactions.
- *Step* 8. Sort the items in the Rescan-items in a descending order of their updated twu.
- *Step* 9. Insert the items in the Rescan-items to the end of the header-table according to the descending order of their *twu*.
- *Step* 10. Find the re-organized transaction according to the $HTWUI^U$ in the Rescan transactions and find corresponding *tu*.
- *Step* 11. For each original transaction in the re-organized rescan-transactions with an item *X* existing in the rescan-items, if *X* has not been at the corresponding branch of the FUUP tree for the transactions, insert *X* at the end of the branch and set its node utility as *tu*, node count as 1; otherwise, if node *X* already existing, add 1 to the count of the node *X* and add *tu* with the previous value.
- *Step* 12. Find the re-organized transaction according to the $HTWUI^U$ in the new transactions and find the corresponding *tu*.
- *Step* 13. For each re-organized transaction new transactions with an item *X* existing in the $HTWUI^U$, if *X* has not been at the corresponding branch of the FUUP tree for the new transaction; insert *X* at the end of the branch and set its node utility as corresponding *tu*, node count as 1; otherwise, update the existing node utility of *X* and corresponding branch nodes according to the DGN strategy and also increase the node count by 1.

In step 6, a parent node may have the same child nodes after deletion. In this case child nodes are merged and their node utility and counts are summed up. In step 11, a corresponding branch is the branch generated from the large items in a transaction and corresponding to the order of items appearing in the Header-table. After step 13, the final updated FUUP tree is constructed; the new transaction can then be integrated into the original database. Based on the FUUP tree, the desired association rules can then be found by the UP-Growth mining approach (Tseng *et al.* [27]).

5. An Example

Considering there is an example to show the original database in Table 2. It consists of ten transactions and six items, denoted {A} to {F}. Assume the minimum high utility threshold is set at 35%, and also assume the user-defined profit values for the items are given in a utility table shown in Table 3. The minimum high utility count is first calculated as total utility multiply a user-specified minimum utility threshold, which is

2921*0.35 (=1022.35). The UP-Tree is constructed for the original database, shown in Figure 2 and its corresponding high transaction weighted utilization and actual utilities given in Table 4. Now, the incremental FUUP algorithm is used efficiently to update the high utility itemsets for record insertion. The new transactions to be inserted are shown in Table 5. It consists of 5 transactions and also 6 items, denoted {A} to {F}.

- *Step 1*. The utility value of each item occurring in each new transaction in Table 5 is calculated. Take the first transaction as an example to illustrate the process. The items with quantities in the first transaction are (A:3, C:4, D:8). The profits for items {A}, {C}, and {D} are 3, 1, and 50, respectively, from Table 3. The transaction utility for the first transaction in Table 4 is thus calculated as *tu*(*T*11)=(3*3)+(4*1)+(8*50), which is 413. The transaction utilities for the other four transactions in Table 5 can be calculated in the same way. After Step 1, the results are shown in Table 5 itself.

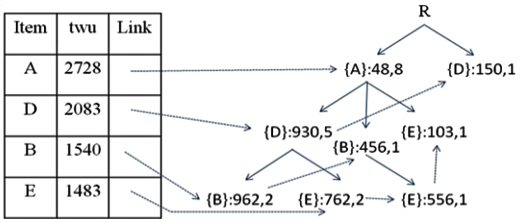

Figure 2. Final UP tree of original database.

Table 4. The high transaction-weighted utility items and their actual utility values for the original database.

Items	High Transaction-Weighted Utility	Actual Utility
{A}	2728	48
{B}	1540	1050
{D}	2083	1050
{E}	1483	600

Table 5. Five newly inserted transactions and its transaction utilities in the example.

TID	A	B	C	D	E	F	tu
11	3	0	4	8	0	0	413
12	3	0	3	7	0	0	362
13	3	0	2	6	0	0	311
14	1	0	0	0	1	1	123
15	4	3	0	0	0	0	462

Table 6. The transaction-weighted utility values of the items in the new transactions.

Items	twu
{A}	1971
{B}	462
{C}	1086
{D}	1086
{E}	123
{F}	123

Table 7. The high transaction-weighted utilization items for the updated database.

Items	twu
{A}	4399
{D}	3169

- *Step 2*. The minimum utility count for new transactions is then calculated. In Table 5, the total utility in the new transactions is calculated as (413+362+311+123+462), which is 1671. The Minimum Utility Count (muc^d) in the new transactions is then calculated as (1671*0.35), which is 584.85.
- *Step 3*. The TWU of items in the new transactions are first calculated. Take item {A} as an example to illustrate the process. The item {A} appears in the transactions from 11 to 15, the $twu^d(A)$ is calculated as (413+362+311+123+ 462), which is 1671. The other items for {B}, {C}, {D}, {E}, and {F} are calculated in the same way. After that, the results are then shown in Table 6.
- *Step 4*. The transaction-weighted utilization items in Table 6 are then checked against the MUC^d in the new transactions, which is 584.85. In this example, only three items {A}, {C}, and {D} satisfied the condition, and put them in the set of High Transaction-Weighted Utility Items ($HTWUI^d$) in the new transactions.
- *Step 5*. For each item in the set of high transaction weighted utilization 1-itemsets $HTWUI^D$ in the original database, and also appears in the set of $HTWUI^d$ in the new transactions are then processed. In this example, the two items {A} and {D} are then processed. Take item {A} as an example to illustrate the process. The TWU of {A} in the original database is $twu^D(\{A\})$ (=2728), which was shown in Table 4. The TWU of {A} in the new transactions is $twu^d(\{A\})$ (=1671). The updated transaction-weighted utility $twu^u(\{A\})$ for the updated database of {A} is then calculated as (2728+1671), which is 4399. The item {D} is also processed in the same way. The items {A} and {D} are then put in the set of high transaction-weighted utilization items for the updated database $HTWUI^U$, Which will be processed in step 8. The updated *twu* of items {A} and {D} are also updated in the header-table. The results are then shown in Table 7.
- *Step 6*. For each item in the set of high transaction weighted utilization items $HTWUI^D$ in the original database, but does not appear in the set of high transaction-weighted utilization items $HTWUI^d$ in the new transactions are then processed. In this example, the two items {B} and {E} are then processed. Take item {B} as an example to illustrate the process. The TWU of item {B} in the original database is $twu^D(\{B\})$ (=1540). The transaction- weighted utility of {B} in the new transactions is $twu^d(\{B\})$ (=462). The updated TWU of item {B} is calculated as (1540+462), which is 2002. The item {E} is also processed in the same way. After that, the results for transaction-weighted utilization 1-itemsets for the updated database are then shown in Table 8.

Table 8. The transaction-weighted utilization items for the updated database.

Items	twu
{B}	2002
{E}	1606

Table 9. The transactions with their id in the rescan_transactions.

Transaction No	Items
3	(A, 3), (C, 5), (F, 3)
4	(A,1), (C, 3), (E, 1), (F, 2)
7	(A, 2), (B, 3), (C, 2), (E, 1), (F, 1)
9	(C, 3), (D, 3)

The updated minimum utility count for the updated database is calculated as (1022.35 + 584.85), which is 1607.2. In this example, however, only the item {B} satisfied the condition. The item {B} is thus considered as the high transaction- weighted utilization item for the updated database and put in the set of high transaction-weighted utilization items for the updated database. And also update the updated twu of item {B} in the Header-Table. But item {E} does not satisfy the condition. Therefore, item {E} is thus removed from Header-Table. In this case FUUP tree need to be processed as well. The updated FUUP tree is shown in Figure 3.

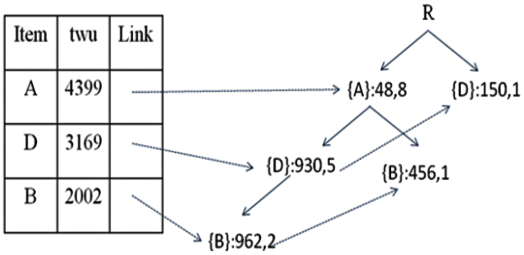

Figure 3. The header-table and FUUP tree after step 6.

- *Step 7*. For each item does not appear in the set of $HTWUI^D$ in the original database, but appear in the set of $HTWUI^d$ in the new transactions are then processed. In this example, item {C} is then processed. The original database is thus rescanned to calculate the TWU of item {C}, which is $twu^D(\{C\})$ (=951). The TWU of item {C} in the new transactions was calculated in Table 6, which is $twu^d(\{C\})$ (=1086). The updated TWU of item {C} is thus $twu^u(\{C\})$ (=951+1086) (=2037), which is larger than the updated minimum utility count muc^U (=1607.2). The item {C} is thus considered as the high transaction-weighted utilization item for the updated database and put in the set of high transaction-weighted utilization items for the updated database $HTWUI^U$ and in the set of rescan_items. And also put the transaction ID's with item {C} in the set of rescan_transactions. After step 7, $HTWUI^U$=({A}, {D}, {B}, {C}), rescan-items =({C}) and rescan_transactions={3, 4, 7, 9}. The corresponding transactions with their ID's in the rescan_transactions are shown in Table 9.
- *Step 8*. The items in the set of rescan _items are sorted in descending order of their updated TWU. In this example, rescan_items contains only {C}, and no sorting is needed.
- *Step 9*. The items in the rescan_items are inserted into the end of the header-table with its corresponding *twu* according to descending order. In this example, {C} is thus inserted.
- *Step 10*. The re-organized transactions for the items in the rescan transactions according to the $HTWUI^U$ and its corresponding transaction utility is given in Table 10.

Table 10. Re-organized transaction and their transaction utilities.

TID	Reorganized Transaction	Transaction Utility
11	(A,3), (D,8), (C,4)	413
12	(A,3), (D,7), (C,3)	362
13	(A,3), (D,6), (C,2)	311
14	(A,1)	3
15	(A,4),(B,3)	462

Table 11. The corresponding branches for the original transactions with item {C}.

TID	Items	Corresponding Branches
11	A, C, D	A, D, C
12	A, C, D	A, D, C
13	A, C, D	A, D, C
14	A, E, F	A
15	A, B	A, B

- *Step 11*. The FUUP tree is updated according to the re-organized Rescan transactions with an item existing in the rescan_items. In this example, rescan_items ={C}. The corresponding branches for the original transactions with c are shown in Table 11.

The first branch is then processed. This branch shares the same prefix {A} as the current FUFP-tree. A new node ({C}:14, 1) is thus created and linked to ({A}:48, 8) as its child. Note that the transaction utility and count for item {A} is not increased since they have already been considered in the construction of the previous FUUP tree. The same process is then executed for the other three corresponding branches. The final results are shown in Figure 4.

- *Step 12*. The re-organized transactions for the items in the new transactions according to the $HTWUI^U$ and its corresponding transaction utility is given in Table 12.

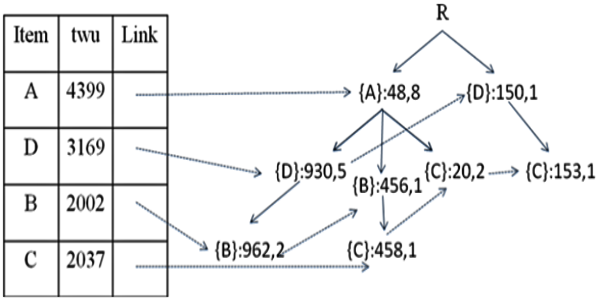

Figure 4. The header-table and FUUP tree after step 11.

Table 12. Re-organized transaction and their transaction utilities

TID	Reorganized Transaction	Transaction Utility
3	(A, 3), (C, 5)	14
4	(A, 1), (C, 3)	6
7	(A, 2), (B, 3), (C, 2)	458
9	(D, 3), (C, 3)	153

Table 13. The corresponding branches for the new transactions.

TID	Items	Corresponding Branches
3	A, C, F	A,C
4	A, C, E, F	A, C
7	A, B, C, E, F	A, B, C
9	C, D	D, C

- *Step 13*. The FUFP-tree is updated according to the new transactions with items existing in the $HTWUI^U$. In this example, $HTWUI^U$=({A}, {D}, {B}, {C}). The corresponding branches for the new transactions with any of these items are shown in Table 13.The first branch share the same prefix ({A}, {D}) as the current FUUP tree. A new node {C}:413, 1) is thus created and linked to the node {D}. The counts for items ({A}, {D}) and then increased by 1 since they have not yet counted in the construction of the previous FUUP Tree and its transaction utility is updated according to the DGN strategy. The same process is then executed for the other four branches. The final results are shown in Figure 5. Based on the final FUUP tree, the desired association rules can then be found by the UP-Growth mining approach.

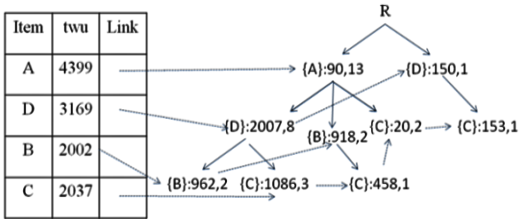

Figure 5. The final FUUP-Tree after all the new transactions are processed

6. Experimental Evaluation

A series of experiments was conducted to evaluate the performance of the batch UP-Tree construction algorithm and the incremental FUUP Tree mining algorithm for processing new transactions. When new transactions came, the batch UP-Tree construction algorithm integrated new transactions into the original database and constructed a new UP-Tree from the updated database. The process was executed whenever new transactions came. The incremental FUUP Tree mining algorithm processed new transactions incrementally in the way mentioned in section 4.

The experiments were implemented in java and executed on a PC with 2.8 GHz CPU and 3.5 GB memory. The public IBM data generator was used in our experiments (IBM Quest Data Mining Project, 1996) [16] to evaluate the performance. In the experimental evaluation, the average length of maximal potentially frequent itemsets was represented as I, the total number of different items was represented as N, and the total number of transactions was represented as D, respectively. The batch UP-Tree construction algorithm and the incremental FUUP Tree mining algorithm were then made compare the performance.

The first 200,000 transactions were then generated (IBM Quest Data Mining Project) [16] to initially mine the high transaction weighted utilization itemsets with their actual utility values. The next 2000 transactions were then generated sequentially used each time as new inserted transactions. The Minimum Utility threshold (min_util) was set at 0.2% to evaluate the performance of two compared algorithms. Figure 3 then showed the execution time required by the batch UP-Tree construction algorithm and the proposed incremental algorithm FUUP for processing each 2000 new transactions on the T10I4N4KD200K dataset.

It can be observed from Figure 6 that the proposed incremental algorithm FUUP Tree for record insertion ran faster than the batch UP-Tree algorithm. The reason was that the proposed algorithm did not re-mine the desired high utility itemsets from the last updated database. The UP-Tree algorithm, however, had to process the updated database in a batch way whenever transactions are inserted. Experiments were then also made to evaluate the efficiency of the proposed incremental algorithm FUUP Tree in different minimum utility threshold values and shown in Figure 7 for the execution time on T10I4N4KD200K dataset.

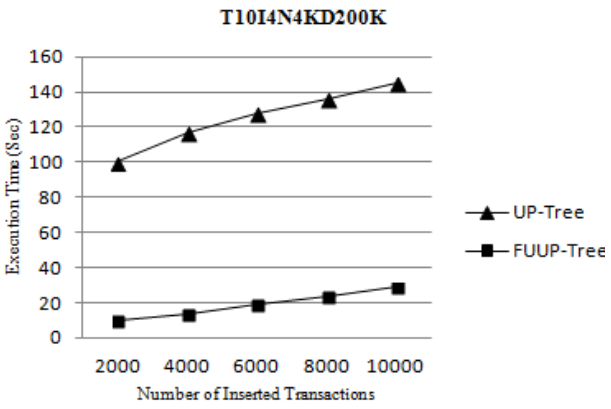

Figure 6. The comparison of the execution time for different numbers of inserted transactions.

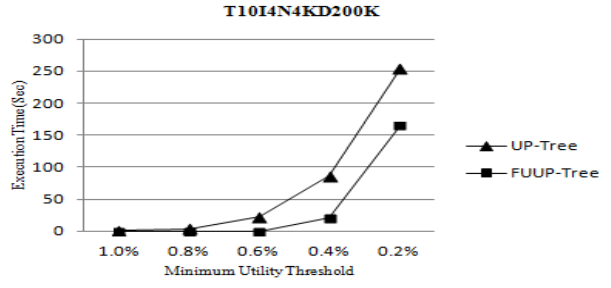

Figure 7. The comparison of the execution time at different minimum utility thresholds.

The minimum utility thresholds were then set from 1.0% to 0.2% (decrease 0.2% each time). It can easily be observed from Figure 7 that the execution time by the proposed FUUP Tree was much less than that by the UP-Tree in a batch way for handling record insertion at different minimum utility thresholds. The reason was the same as the content mentioned in previous experiment. In summary, the proposed incremental algorithm for mining high utility itemsets was efficiently and effectively than the UP-Tree algorithm in a batch way to maintain the updated database for record insertion.

7. Conclusions

In this paper, we have proposed the FUUP Tree mining algorithm to efficiently and effectively handle new transaction insertion in data mining. When new transactions are added, the proposed incremental mining algorithm processes them to maintain the FUUP 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 FUUP tree are correspondingly updated whenever necessary. It is reasonable to insert a new large item at the end of the header-table since when an originally small item becomes large due to new transactions, its updated support is usually only a little larger than the minimum support. Experimental results also show that the proposed FUUP tree mining algorithm runs faster than the batch UP-tree construction algorithm for handling new transactions and generates nearly the same tree structure as the UP-Tree algorithm. 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 Proceedings of ACM International Conference on Management of Data*, Washington, pp. 207-216, 1993.

[2] Agrawal R. and Srikant R., "Fast Algorithm for Mining Association Rules," *in Proceedings of the 20th International Conference on Very Large Data Bases*, San Francisco, pp. 487-499, 1994.

[3] Agrawal R., Srikant R., and Vu Q., "Mining Association Rules with Item Constraints," *in Proceedings of The Third International Conference on Knowledge Discovery in Databases and Data Mining*, New Port Beach, pp. 67-73, 1997

[4] Ahmed C., Tanbeer S., Jeong B., and Lee Y., "Efficient tree Structures for High Utility Pattern Mining in Incremental Databases," *IEEE Transactions on Knowledge and Data Engineering*, vol. 21, no. 12, pp. 1708-1721, 2009.

[5] Chan R., Yang Q., and Shen Y., "Mining High Utility Itemsets," *in Proceedings of The 3rd IEEE International Conference on Data Mining*, Melbourne, pp. 19-26,2003.

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

[7] Cheung D., Lee S., and Kao B., "A General Incremental Technique for Maintaining Discovered Association Rules," *in Proceedings of the 5th International Conference on Database System for Advanced Applications*, Melbourn, pp. 185-194, 1997.

[8] Das A. and Bhattacharyya D., "Rule Mining for Dynamic Databases," *Australasian Journal of Information Systems*, vol. 13, no. 1, Kolkata, pp. 19-39, 2004.

[9] Erwin A., Gopalan R., and Achuthan N., "Efficient mining of High Utility Itemises from Large Datasets," *in Proceedings of Advances in Knowledge Discovery and Data Mining*, Osaka, pp. 554-561, 2008

[10] Ezeife C. and Su Y., "Mining Incremental Association Rules with Generalized FP Tree," *in Proceedings of 15th Canadian Conference on Artificial Intelligence*, Calgary, pp. 147-160, 2002.

[11] Feldman R., Aumann Y., Manilla H., and Lipshtat O., "Borders: An Efficient Algorithm for Association Generation in Dynamic Databases," *Journal of Intelligent Information System*, vol. 12, no. 1, pp. 61-73, 1999.

[12] Fukuda T., Morimoto Y., Morishita S., and Tokuyama T., "Mining Optimized Association Rules for Numeric Attributes," *in Proceedings of the 5th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, Montreal, pp. 182-191, 1996.

[13] Han J. and Fu Y., "Discovery of Multiple-level Association Rules from Large Database," *in Proceedings of the 21th International Conference on Very Large Data Bases*, San Francisco, pp. 420-431, 1995.

[14] Han J., Pei J., Yin Y., and Mao R., "Mining Frequent Patterns without Candidate Generation: a Frequent-Pattern Tree Approach," *Data Mining and Knowledge Discovery*, vol. 8, no. 1, pp 53-87, 2004.

[15] 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.

[16] IBM Quest Data Mining Project," Quest Synthetic Data Generation Code," http://www.almaden.ibm.com/cs/quest/syndata.html, Last visited 2012

[17] Li Y., Yeh J., and Chang C., "Isolated Items Discarding Strategy for Discovering High Utility Itemsets," *Data and Knowledge Engineering*, vol. 64, no 1, pp. 198-217, 2008.

[18] Lin C., Hsieh Y., Yin K., Hung M., and Yang D., "ADMiner: An Incremental Data Mining Approach Using a Compressed FP-Tree," *Journal Of Software*, vol. 8, no. 8, pp. 2095-2103, 2013.

[19] Liu Y., Liao W., and Choudhary A., "A Fast High Utility Itemsets Mining Algorithm," *in Proceedings of the 1st international Workshop on Utility-Based Data Mining*, Chicago, pp. 90-99, 2005.

[20] Liu Y., Liao W., and Choudhary A., "A Two Phase Algorithm for Fast Discovery of High Utility of Itemsets," *in Proceedings of the 9th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining*, Hanoi pp. 689-695, 2005.

[21] Mannila H., Toivonen H., and Verkamo A., "Efficient Algorithm for Discovering Association Rules," *Technical Report AAAI*, 1994.

[22] Park J., Chen M., and Yu P., "Using a Hash-Based Method with Transaction Trimming for Mining Association Rules," *IEEE Transactions on Knowledge and Data Engineering*, vol. 9, no. 5, pp. 813-825, 1997.

[23] Srikant R. and Agrawal R., "Mining Generalized Association Rules," *in Proceedings of The Twenty-first International Conference on Very Large Data Bases*, San Francisco, pp. 407-419, 1995.

[24] Srikant R. and Agrawal R., "Mining Quantitative Association Rules in large Relational Tables, " *in Proceedings of ACM SIGMOD International Conference on Management of Data*, Montreal, pp. 1-12, 1996.

[25] Shie B., Tseng V., and Yu P., "Online Mining of Temporal Maximal Utility Itemsets from Data Streams," *in Proceedings of the 2010 ACM Symposium on Applied Computing*, Switzerland, pp. 1622-1626, 2010.

[26] Tseng V., Chu C., and Liang T., "Efficient Mining of Temporal High Utility Itemsets from Data Streams," *in Proceedings of Workshop on Utility-Based Data Mining Workshop*, pp.18-27, Philadelphia, 2006.

[27] Tseng V., Wu C., Shie B., and Yu P., "UP-Growth: An Efficient Algorithm for High Utility Itemsets Mining," *in Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Washington, pp. 253-262, 2010.

[28] Yafi E., Al-Hegami A., Alam A., and Biswas R., "YAMI: Incremental Mining of Interesting Association Patterns," *International Arab Journal of Information technology*, vol. 9, no. 6, pp.504-510, 2012.

[29] Yao H., Hamilton H., and Butz C., "A Foundational Approach to Mining Itemset Utilities from Databases," *in Proceedings of 4th SIAM International Conference on data Mining*, florida, pp. 211-225, 2004.

[30] Yao H. and Hamilton H., "Mining Itemset Utilities from Transaction Databases," *Data and Knowledge Engineering*, vol. 59, no. 3, pp.603-626, 2006.

Kavitha JeyaKumar, Assistant Professor. She received the M.Tech. Degree from the Department of Computer Science and Engineering at Anna University, Chennai, in 2009. She is now a Ph.D. candidate in the Department of Computer Science and Engineering at Anna University, Chennai. Her research interests include Data mining, Image Processing, Cloud Computing.

Manjula Dhanabalachandran, Associate Professor. She received her B.E Electronics and Communications Engineering degree in 1983 from Thiagarajar College of Engineering, Madurai and M.E Computer Science and Engineering in 1987 and Ph.D (Computer Science and Engineering) in 2004 from Anna University Chennai. Her research interests include Data mining, Image Processing, Cloud Computing, Network security.

Kasthuri JeyaKumar, Assistant Professor. She received the M.E. degree from the Department of Electronics and Communication Engineering at SRM University, Chennai, in 2008. She is now a Ph. D. candidate in the Department of Electronics and Communication Engineering at SRM University, Chennai. Her research interests include Data mining, Image Processing, VLSI.