International Journal of Contemporary Research in Computer Science and Technology (IJCRCST) e-ISSN: 2395-5325
Volume3, Issue 1 (January'2017)

INCREMENTAL HIGH UTILITY PATTERN TREES USING TRANSACTIONAL DATABASES

S.Dhivya,
Assistant Professor,
Department of Computer Science & Engineering,
K.L.N. College of Information Technology,
Pottapalayam,Sivagangai Dist.

T.T.Mathangi,
Assistant Professor,
Department of Computer Science & Engineering,
K.L.N. College of Information Technology,
Pottapalayam,Sivagangai Dist.

S.Jeniba,
Assistant Professor,
Department of Computer Science & Engineering,
K.L.N. College of Information Technology,
Pottapalayam, Sivagangai Dist.

Abstract: Frequent pattern mining discovers patterns in transaction databases based only on the relative frequency of occurrence of items without considering their utility.High utility pattern (HUP) mining is one of the most important in data mining due to its ability to consider the non-binary frequency values of items in transactions and different profit values for every item. On the other hand, incremental data mining provide the ability to use previous data structures and mining results in order to reduce unnecessary calculations In this project, three tree structures have been implemented called $IHUP_L$,$IHUP_{TF}$,$IHUP_{TWU}$ (Incremental High Utility Pattern).All the IHUP tree structures maintains the tf and twu values in the header table and the tree nodes. The first tree structure is arranged in the lexicographic order($IHUP_L$). It can capture the incremental data without any restructuring operation. The second tree structure is arranged in descending order based on the transaction frequency($IHUP_{TF}$).The third tree structure is arranged in the descending order based on the transaction weighted utilization($IHUP_{TWU}$) to reduce the mining time. IHUP has the "build once mine property" and is suitable for incremental mining. The experimental results shows that the tree structures are efficient and scalable for incremental mining.

Keywords— Data Mining, Frequent Pattern Mining, High Utility Pattern Mining, Incremental Mining.

I. INTRODUCTION

Data mining is the process of extracting or "mining" knowledge from large amounts of data. It allows the users to analyze data from many different dimensions or angles, categorize it, and summarize the relationships identified. Data mining is concerned with analyzing large volumes of unstructured data to discover interesting regularities or relationships which in turn lead to better understanding of the underlying processes.

Association rule mining is a method of finding relationships of the form X→Y among itemsets that occur together in a database where *X* and *Y* are disjoint itemsets. Frequent pattern mining algorithms work on transaction databases that represent the occurrence of each item in a transaction as a binary value without considering its quantity or weight (based on price, cost or profit). However, quantity and weight are significant for addressing real world decision problems that require maximizing the utility in an organization. For example, selling a laser printer may occur less frequently than sale of glossy paper in an electronic superstore, but the former gives a much higher profit per unit sold.

a. Frequent Pattern Mining

Let $I = \{i_1, i_2, \ldots \ldots i_m\}$ be a set of items and D be a transaction database $T_1, T_2, \ldots \ldots \ldots T_n$, where each transaction $T_i \in D$ is a subset of I. The support/frequency of a pattern $X\{x_1, x_2, \ldots . . x_p\}$ is the number of transactions containing the pattern in the transaction database.

The problem of frequent pattern mining is to find the complete set of patterns satisfying a minimum support in the transaction database. The downward closure property is used to prune the infrequent patterns. This property states that if a pattern is infrequent, then all of its superpatterns must also be infrequent.

Apriori algorithm is the initial solution for the frequent pattern mining problem. But it suffers from the level-wise candidate generation-and-test problem and needs several database scans. The FP-growth algorithm solves this problem by using an FP-tree based solution without any candidate generation and using only two database scans.

b. Weighted Frequent Pattern Mining

Let I = {i1, i2..., in} be a unique set of items. A transaction database, TDB, is a set of transactions in which each transaction, denoted as a tuple <tid, X>, contains a unique tid and a set of items. An itemset is called a k-itemset if it contains k items. An itemset {x1, x2, ..., xn} is also represented as x1, x2,..., xn. The support of itemset A is the number of transactions containing itemset A in the database. A weight of an item is a non-negative real number that shows the importance of each item. The weighted itemset term can be used to represent a set of weighted items. A simple way to

obtain a weighted itemset is to calculate the average value of the weights of the items in the itemset.

The main challenging problem in this area is that the itemset weighted frequency does not have the anti-monotone property. Weight of a pattern *P* is the ratio of sum of all its items weight value and length of *P*. Consider item *"a"* has weight 0.6 and frequency 4, item "b" has weight 0.2 and frequency 5, itemset "ab" has frequency 3. Then the weight of itemset "ab" will be (0.6 + 0.2)/2 = 0.4 and its weighted frequency will be 0.4 × 3 = 1.2. Weighted frequency of "a" is 0.6 × 4 = 2.4 and "b" is 0.2 × 5 = 1.0. If the minimum weighted frequency threshold is 1.2, then pattern "b" is weighted infrequent but "ab" is weighted frequent. The anti-monotone property have been used for multiplying each itemset's frequency by the global maximum weight. In the above example, if "a" has the maximum weight of 0.6, then by multiplying it with the frequency of item "b", 3.0 is obtained. Hence, pattern "b" is not pruned at the early stage and pattern "ab" will not be missed, but pattern "b" is overestimated and will be pruned later by using its actual weighted frequency. Maintaining the anti-monotone property in high utility pattern mining area is more challenging, as it considers non-binary frequency values of items in each transaction.

c. *Need for Utility Mining*

In the real world, each item in the supermarket has a different importance/price and one customer can buy multiple copies of an item. Moreover, items having high and low selling frequencies may have low and high profit values, respectively. For example, some frequently sold items such as bread, milk, and pen may have lower profit values compared to that of infrequently sold higher profit value items such as gold ring and gold necklace. Therefore, finding only traditional frequent patterns in a database cannot fulfill the requirement of finding the most valuable item sets that contribute to the major part of the total profits in a retail business. This gives the motivation to develop a mining model to discover the item sets contributing to the majority of the profit.

d. *Utility Mining*

Let $I = \{i_1, i_2, \ldots . i_m\}$ be a set of items, *D* be a transaction database, and $UT < I, U >$ be a utility table, where *U* is a subset of the real numbers that reflect the utilities of the items.

Utility is a measure of how "useful" an itemset is. The goal of utility mining is to identify high utility itemsets that drive a large portion of the total utility. The utility of an item or itemset is based on local transaction utility and external utility. The local transaction utility of an item is defined according to the information stored in a transaction, like the quantity of the item sold in the transaction. The external utility of an item is based on information from resources besides transactions, like a profit table. The external utility can be a measure for describing user preferences.

A utility mining model was defined to discover more important knowledge from a database. The importance of an itemset can be measured by the concept of utility. The dataset can be handled with non-binary frequency values of each item in transactions, and also with different profit values of each item. Therefore, utility mining represents real world market data. By utility mining, several important business area decisions like maximizing revenue or minimizing marketing or inventory costs can be considered and knowledge about itemsets contributing to the majority of the profit can be discovered.

e. *Need For Incremental Databases*

The existing works are based on a fixed database and did not consider that one or more transactions could be deleted, inserted, or modified in the database.

Consider customer X has bought three pens, four pencils, and one eraser and customer Y has bought one computer mouse. After sometime, customer Z may come to buy two breads and one milk, customer X may come and return two pencils, and customer Y may come to return the mouse. So, in the real world market new transactions can be frequently added, and the old transactions can be modified or deleted. As a result, additions, deletions, and modifications of the transactions have been considered in the real world datasets.

Moreover, if the algorithms presented in the previous works want to calculate which patterns cover 20 percent of the total profit, then their internal data structures are designed in such a way that they can only calculate the asked amount. If the amount is 15 percent of the total profit, then previous algorithms have to build their data structures again. So, any advantages from the previous design cannot be taken. However, in the real world, the businessmen need to repeatedly change the minimum threshold according to their business requirements. Thus, the "build once mine many" property (by building the data structure only once, several mining operations can be done) is required to solve the incremental mining problem.

II. RELATED WORK

Association rule mining (ARM) is one of the most widely used techniques in data mining and knowledge discovery and has tremendous applications in business, science and other domains. For example, in the business, its applications include retail shelf management, inventory predictions, supply chain management, bundling products marketing. The main objective of association rule mining is to identify frequently occurring patterns. It first finds all the itemsets whose co-occurrence frequency are beyond a minimum support threshold, and then generates rules from the frequent itemsets based on a minimum confidence threshold. Traditional ARM model treat all the items in the database equally by only considering if an item is present in a transaction or not.

The Apriori algorithm is the initial solution for the frequent pattern mining problem, but it suffers from the level-wise candidate generation-and-test problem and requires several

database scans. For the first database scan, Apriori finds all one element frequent itemsets, and based on that, generates the candidates for two-element frequent itemsets. In the second database scan, Apriori finds all of the two-element frequent itemsets, and based on that, generates the candidates for three-element frequent itemsets and so on.

Traditional frequent pattern mining algorithms [12]only consider the binary (0/1) frequency values of items in transactions and same profit values for every item. However, the customer may purchase more than one of the same item, and the unit price may vary among items. For frequent pattern mining, the downward closure property is used to prune the infrequent patterns. This property says that if a pattern is infrequent, then all its super patterns must be infrequent. The Apriori algorithm requires multiple database scans and generates many candidate itemsets. FP-Growth solved this problem by introducing a prefix-tree (FP-tree)-based algorithm without candidate set generation and testing.

The frequent item sets identified by Association Rule Mining does not reflect the impact of any other factor except frequency of the presence or absence of an item. Frequent item sets may only contribute a small portion of the overall profit, whereas non-frequent item sets may contribute a large portion of the profit. In reality, a retail business may be interested in identifying its most valuable customers (customers who contribute a major fraction of the profits to the company). These are the customers, who may buy full priced items, high margin items, or gourmet items, which may be absent from a large number of transactions because most customers do not buy these items. In a traditional frequency oriented Association Rule Mining, these transactions representing highly profitable customers may be left out. For instance, {milk, bread} may be a frequent itemset with support 40%, contributing 4% of the total profit, and the corresponding consumers is Group A, whereas {birthday cake,birthday card} may be a non-frequent itemset with support 8% (assume support threshold is 10%), contributing 8% of the total profit, and the corresponding consumers is Group B. The marketing professionals must be more interested in promoting the sale of {birthday cake, birthday card} by designing promotion campaigns or coupons tailored for Group B (valuable customers), although this item set is missed by ARM. Utility mining is likely to be useful in a wide range of practical applications.

The utility mining model [11] was defined to solve the limitations of frequent pattern mining by allowing the user to measure the importance of an itemset by its utility value. With utility mining, several important decisions in business like maximizing revenue, minimizing marketing or inventory costs can be made and more important knowledge about itemsets contributing to the majority of the profit can be discovered. These techniques can be applied to many other areas which includes network traffic measurements, telecom call records etc. Traditional Association Rule Mining problem is a special case of utility mining, where the utility of each item is always 1 and the quantity is either 0 or 1. There is no efficient strategy to find all the high utility itemsets due to the non-existence of downward closure property (anti-monotone property) in the utility mining model.

In UMining, a pruning strategy based on utility upper bound property is used. UMining H has been designed with another pruning strategy based on a heuristic method. However, some high utility itemsets may be erroneously pruned by their heuristic method. Moreover, these methods do not satisfy the downward closure property of Apriori, and therefore, overestimate too many patterns. They also suffer from excessive candidate generations and poor test methodology.

The Two-Phase algorithm [8] was developed to find high utility itemsets. In Phase I, transaction-weighted utilization was defined and transaction-weighted utilization mining model was proposed. This model maintains a Transaction-weighted Downward Closure Property. Thus, only the combinations of high transaction weighted utilization itemsets are added into the candidate set at each level during the level-wise search. Phase I may overestimate some low utility itemsets, but it never underestimates any itemsets. In phase II, only one extra database scan is performed to filter the overestimated item sets.

An efficient algorithm for utility mining using the pattern growth approach has been proposed to overcome the limitations of existing algorithms based on the candidate generation-and-test approach. A compact data representation named Compressed Transaction Utility tree (CTU-tree) [5] has been introduced which extends the CT-tree for utility mining, and a new algorithm named CTU-Mine for mining complete high utility itemsets. Compared to the two phase algorithm, this algorithm not only uses pattern growth instead of candidate generation-and-test, but also eliminates the expensive second phase of scanning the database to remove the high utility itemsets.

The challenge of utility mining is to effectively reduce the number of candidates. The Isolated Items Discarding Strategy (IIDS),[3] have been proposed which can be applied to each level-wise utility mining method to further reduce the number of redundant candidates. In each pass, a utility mining method with IIDS scans a database that is smaller than the original by skipping isolated items to efficiently improve performance. It mainly focuses on the task of efficiently discovering all high utility itemsets.

The frequent pattern mining algorithms are based on the fixed database and did not consider insertions, deletions and modifications in one or more transactions. So incremental mining algorithms have been proposed because they use previous data structures and mining results in order to reduce unnecessary calculations. They also supports the build once mine many property.

Several techniques have been developed for incremental mining and it shows that incremental prefix-tree structures such as Can Tree , CP-tree , FUFP-tree , etc., are quite possible and they are efficient by using currently available memory in the gigabyte range. The efficient dynamic database updating algorithm (EDUA) is designed for mining databases when data deletion is performed frequently in any database subset. However, these solutions are not applicable for incremental high utility pattern mining.

An algorithm, called AFPIM(Adjusting FP-tree for Incremental Mining), has been proposed to efficiently find new frequent itemsets with minimum re-computation when new transactions are added to, deleted from, or modified in the 4 transaction database. In this approach, the FP-tree structure of the original database was maintained in addition to the frequent itemsets. In most cases, without needing to re-scan the whole database, the FP-tree structure of the updated database is obtained by adjusting the preserved FP-tree according to the inserted and deleted transactions. Then the frequent itemsets of the updated database are mined from the new FP-tree structure and the corresponding association rules are discovered.

A fast updated FP-tree (FUFP-tree) [2] structure has been proposed which will make the tree update easier. It is similar to the FP-tree structure except that the links between parent nodes and their child nodes are bi-directional. Besides, the counts of the sorted frequent items are also kept in the header table of the FP-tree algorithm. Bi-directional linking and storing the counts in the header table will help fasten the maintenance process. An incremental FUFP-tree maintenance algorithm is then proposed for processing newly inserted transactions. 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 existing work on high utility pattern mining, no one has proposed any solution for incremental mining, where many new transactions can be added and existing transactions can be deleted or modified. Moreover, none of the data structures have the "build once, mine many" property.

III. PROPOSED SYSTEM

In this project, three tree structures have been implemented for high utility pattern mining in incremental databases. It exploits a pattern growth approach to avoid level -wise candidate generation and test problem of the existing high utility pattern mining algorithms. It needs only two database scans.

The first tree structure, Incremental HUP Lexicographic Tree (IHUPL), is arranged according to an item's lexicographic order. It can capture the incremental data without any restructuring operation. The second tree structure is Incremental HUP Transaction Frequency Tree (IHUPTF) which obtains a compact size by arranging items in descending order according to the transaction frequency. The third tree structure Incremental HUP Transaction Weighted Utilization Tree (IHUPTWU), to reduce the mining time, is arranged in descending order based on the TWU values. As a result, the IHUPTF tree requires smallest amount of memory, the IHUPTWU tree requires the smallest amount of overall running time during the mining phase. Moreover, IHUP tree structures has the "build once mine many" property. By building the data structure only once, several mining operations can be performed.

Benefits of the proposed system

- The tree structures are very effective since build once mine many property has been used.
- The dataset may have non-binary frequency values for each item .
- The previous data structures and mining results can be used, when one or more transactions could be inserted, deleted, or modified in the database by avoiding unnecessary calculations.

IV.SYSTEM DESIGN AND METHODOLOGY

Methodology

Let $I = \{i_1, i_2, \ldots . . i_m\}$ be a set of items and D be a transaction database $\{T_1, T_2, \ldots \ldots T_n\}$,where each transaction$T_i \in D$ is a subset of I.

Items / Tid	a	b	c	d	E	Tran Utility
T1	0	0	18	0	1	23
T2	0	6	0	1	1	71
T3	2	0	1	0	1	12
T4	1	0	0	1	1	14
T5	0	0	4	0	2	14

Fig.1.(a)Transaction Database

Item	Profit
A	3
B	10
C	1
D	6
E	5

Fig.1.(b) Utility Table

Definition 1: The internal utility or local transaction utility value $l(i_p, T_q)$, represents the quantity of item i_p in transaction T_q. For example: in Fig.1(a), l(a,T3) = 2.

Definition 2: The external utility $p(i_p)$ is the unit profit value of item i_p. For example: in Fig.1(b),p(a) = 3.

Definition 3: Utility $u(i_p, T_q)$ is the quantitative measure of utility for item i_p in transaction T_q defined by

For example: in Fig.1. u(a,T3) = 2 ×3 = 6

Definition 4: The utility of an itemset X in transaction T_q, $u(X, T_q)$,is defined by

$$u(X, T_q) = \sum_{i_p \in X} u(i_p, T_q)$$

For example: in Fig.1.u(de,T4) = 1× 6 + 1× 5 = 11.

Definition 5: The utility of an itemset is defined by

$$u(X) = \sum_{T_q \in D} \sum_{i_p \in X} u(i_p, T_q)$$

For example: in Fig.1. u(de) = u(de,T2) + u(de,T4) = 11 + 11 = 22

Definition 6: The transaction utility(tu) of transaction T_q denoted as $tu(T_q)$ describes the total profit of that transaction and is defined by

$$tu(T_q) = \sum_{i_p \in T_q} u(i_p, T_q)$$

For example: in Fig.1. tu(T5) = u(c,T5) + u(e, T5) = 4 + 10 =14.

Definition 7: Transaction-weighted utilization of an itemset X ,denoted by $twu(X)$,is the sum of the transaction utilities of all transactions containing X

$$twu(X) = \sum_{T_q \in D} tu(T_q)$$

For example : twu(de) = tu(T2) + tu(T4) = 71 + 14.

The transaction frequency of an item set X is needed for the algorithm to speed up the second database scan to find high utility itemsets from high transaction-weighted utilization item sets.

By building the data structure only once, several mining operations can be performed. Based on this property, the three tree structures can be constructed.

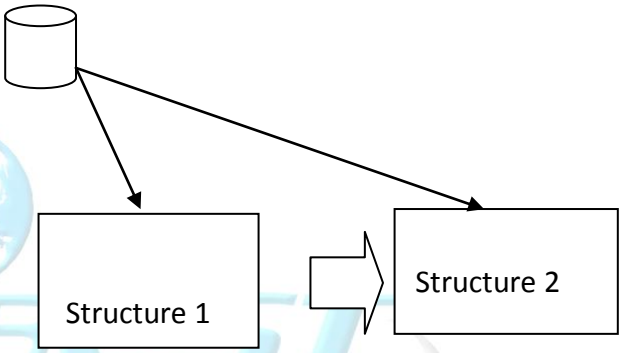

Fig. 2. Build Once Mine Many property

Tree Structures

Incremental High Utility Pattern Mining Lexicographic Tree (IHUPL):

IHUPL-Tree arranges the items in lexicographic order and inserts them as a branch inside the tree in the first database scan. All the IHUP-Trees explicitly maintain twu and tf values in both the header table and the tree nodes.

Incremental High Utility Pattern Mining Transaction Frequency Tree (IHUPTF):

In this tree, the nodes are arranged in descending order according to their transaction frequency so that items occurring in many transactions can be kept in the upper part of the tree, and therefore, higher prefix-sharing can be achieved. The IHUPTF-Tree requires the smallest amount of memory.

Incremental High Utility Pattern Mining Transaction Weighted Utilization Tree (IHUPTWU):

The IHUPTWU-Tree, which is designed according to the twu value in descending order. In the IHUPTWU-Tree, all candidate nodes are kept before the non-candidate nodes in

$u(i_p, T_q) = l(i_p, T_q) \times p(i_p)$

every branch. The IHUPTWU-Tree can be constructed any time from an IHUPL-Tree using a path adjusting method based on bubble sort technique. In the mining operation, it shows that the number of nodes participating in the mining operation in the IHUPTWU-Tree is always less than or equal to the other two tree structures, and therefore, it achieves the fastest mining time as well as the overall running time.

V. IMPLEMENTATION

Construction Of IHUP$_L$-Tree Structure:

Consider the transaction database and utility table in Fig 3(a) and 3(b). From the very beginning, the database contains only four transactions. After that, the initial database is incremented by adding two groups of transactions

	TID \ ITEM	a	b	c	d	e	Trans. Utility
Original DB	T_1	2	2	0	0	0	34
	T_2	3	0	12	4	2	88
	T_3	0	0	15	0	3	66
	T_4	4	0	0	0	0	8
db_1^+	T_5	0	10	0	8	9	277
	T_6	0	7	3	0	4	142
db_2^+	T_7	1	0	2	0	1	15
	T_8	2	0	0	1	3	33

(a)

ITEM	PROFIT($) (per unit)
a	2
b	15
c	3
d	8
e	7

(b)

Fig. 3 (a) Transaction Database (b) Utility Table

The construction process for the initial database (T1-T4) of the IHUP$_L$-Tree are described in Fig 4(a),4(b)&4(c) . The tu value of T1 is 34 in Fig. 3. Fig.4(a) shows that when T1 is added to the tree, the first node in lexicographic order is "a" with twu =34 and tf =1.The second node is "b" with the same values. Fig 4(b) shows when T2 is added to the tree. The tu of T2 is 88, so the branch in lexicographical order is "a:88, 1," "c:88, 1," "d:88, 1," and "e:88, 1." Here "a" is assigned the prefix-sharing with the existing node "a," the twu value of node "a" is 34 + 88 =122,and the tf value is 1 + 1 = 2. For other items, new nodes are created. After that, T3 and T4 are added to complete the initial database . By following the same process, db_1^+ (T5 and T6) and db_2^+ (T7 and T8) are inserted into the IHUP$_L$-Tree as shown in Fig 4(d) and 4(e)

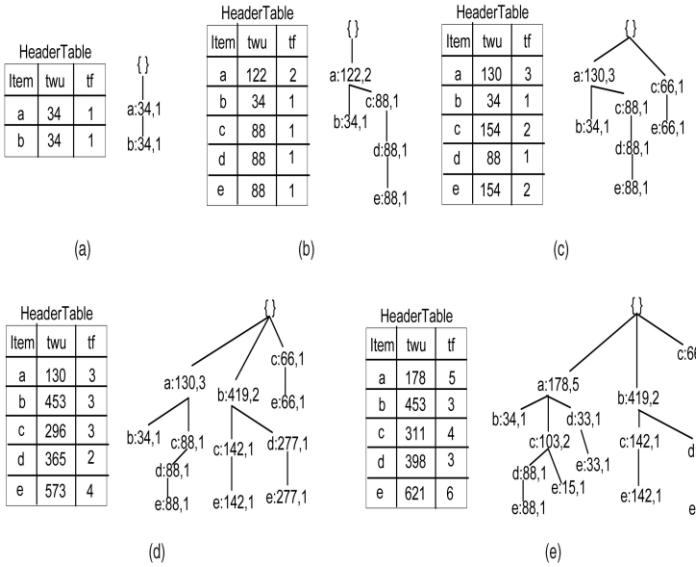

Fig.4. Construction of IHUP$_L$ Tree (a) after inserting T1 (b) after inserting T2 (c) after inserting T3 and T4 (d) after inserting db_1^+ (T5 and T6) (e) after inserting db_2^+(T7 and T8).

The deletion and modification operations can be performed in the IHUP$_L$-Tree. Suppose ,to delete T7, then the tu value of T7 and one tf value can be easily decreased from the path "a c e" in the tree shown in Fig. 4(e) Subsequently, as twu and tf values of node "e" in that path become zero, to delete node "e" from that branch. So, the modified twu and tf values of "a" will be 163 and 4, respectively, in that branch. In the same way, the modified twu and tf values of "c" will be 88 and 1,respectively, in that branch. Suppose to modify T7, simply the quantity of items "c" from two to one can be reduced. To perform this modification, the twu value of T7 will be reduced by1 × 3 = 3.From the branch "a c e" the twu value has been reduced. As IHUP$_L$-Tree always maintains the lexicographic order in both the header table and tree nodes.

Construction Of IHUP$_{TF}$-Tree Structure:

To reduce the size of the IHUPL-Tree, the prefix-sharing inside it have been increased. In order to achieve the desired compactness, the IHUPTF-Tree has been developed. In this tree, the nodes are arranged in descending order according to their transaction frequency so that items occurring in many transactions can be kept in the upper part of the tree, and therefore, higher prefix-sharing can be achieved. The IHUPTF-Tree can be constructed from an IHUPL-Tree using a path adjusting method based on a bubble sort technique at any time. Any node may be split when it needs to be swapped with any child node having a count smaller than that node. If the support counts of both nodes are equal, a simple exchange operation between them is performed. After performing each operation, swapping nodes having the same items are merged.

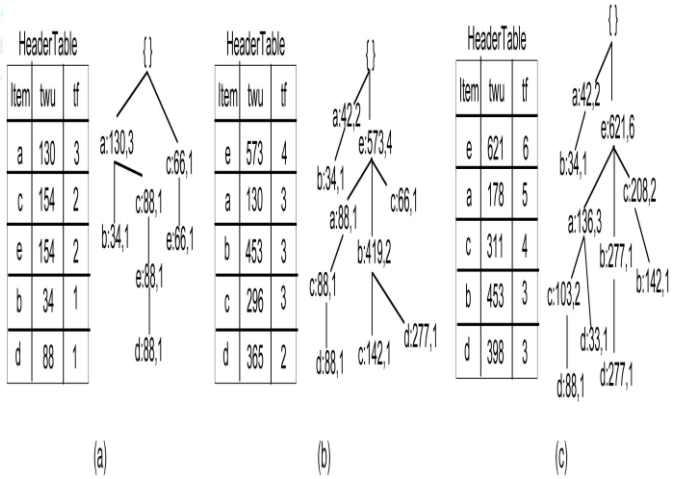

Fig.5 Construction of IHUP$_{TF}$ Tree (a) IHUP$_{TF}$ Tree upto T4 (b) IHUP$_{TF}$ Tree upto T6 (c) IHUP$_{TF}$ Tree upto T8.

Consider the transaction database and utility table in Fig. 4(a) and 4(b). In the original database (up to T4), the IHUPTF-Tree is constructed in the same way as the IHUPL-Tree shown in Fig. 4(c). Subsequently, according to the tf value, items are sorted in descending order and the new order is "a c e b d" as shown in Fig. 5(a). Items in the transactions of db_1^+ are inserted in the tree in that order. When $db_1^+$is finished, the tree is sorted according to the tf value of items, and the new order now is "e a b c d," as shown in Fig. 5(b). Likewise, db_2^+ is processed, the tree is restructured at the end of db_2^+, and the

final order is "e a c b d," as shown in Fig. 5(c). The IHUPTF-Tree in Fig. 5(c) has only 11 nodes (without root) compared to the 15 nodes of the IHUPL-Tree in Fig. 4(e). Deletion and modification operations can be done in the same way as the IHUPL-Tree.

Construction Of IHUP$_{TWU}$-Tree Structure:

Several low-twu items can appear before the high-twu items in branches of the IHUPL-Tree and IHUPTF-Tree. For example, if minutil = 198.9 in Fig. 3, then "a" is a low-twu item. However, "a" appears before high-twu items "b," "c," "d," and "e" in the IHUPL-Tree and high-twu items "b," "c," and "d" in the IHUPTF-Tree. During the mining process, these non-candidate nodes incur a huge amount of delay. For this reason the IHUPTWU-Tree, has been implemented which is designed according to the twu value in descending order. In the IHUPTWU-Tree, all candidate nodes are kept before the non-candidate nodes in every branch. The IHUPTWU-Tree can be constructed any time from an IHUPL-Tree using a path adjusting method based on bubble sort technique. The number of nodes participating in the mining operation in the IHUPTWU-Tree is always less than or equal to the other two tree structures, and therefore, it achieves the fastest mining time as well as the overall running time. Consider the transaction database and utility table in Fig. 3(a) and 3(b). In the original database (up to T4), the IHUPTWU-Tree is constructed in the same way as the IHUPL-Tree shown in Fig. 4(c). Subsequently, according to the twu value of items, the tree is restructured. The new order is "c e a d b." Items in the transactions of db_1^+ are inserted in the tree in that order. When db_1^+ is added to the tree, the tree is restructured again according to the twu value of items. The new order now is "e b d c a." Likewise, db_2^+ is processed and no restructuring is required at the end of db_2^+ because the final order "e b d c a" remains the same. The IHUPTWU-Tree in Fig. 6 has 13 nodes compared to 15 nodes in the IHUPL-Tree in Fig. 4(e) and 11 nodes in IHUPTF-Tree in Fig. 5(c). Deletion and modification operations can be done in the same way as the IHUPL-Tree.

HeaderTable

Item	twu	tf
e	621	6
b	453	3
d	398	3
c	311	4
a	178	5

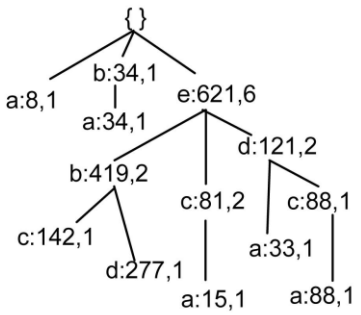

Fig 6. Construction of IHUP$_{TWU}$ Tree

VI. CONCLUSION

The key contribution of this work is to provide an efficient method for high utility pattern mining in incremental databases. Three variations of the tree structure have been implemented. Among them, the IHUP$_L$ Tree is very simple and easy to construct and handle, as it does not require any restructuring operation in spite of incremental updating of the databases. IHUP$_{TF}$-Tree that requires less memory and an IHUP$_{TWU}$ Tree requires less time to execute than previous methods. A pattern growth approach is used to avoid the level-wise candidate generation-and-test methodology. All these tree structures have the "build once mine many" property. All three tree structures require maximum of two database scans. The tree structures are scalable for handling a large number of distinct items and transactions.

VIII.REFERENCES

[1]. Chowdhury Farhan Ahmed, Syed Khairuzzaman Tanbeer, Byeong-Soo Jeong, and Young-Koo Lee, "Efficient Tree Structures for High Utility Pattern Mining in Incremental Databases" *IEEE Transactions on Knowledge and Data Engineering*,2009.

[2]. T.-P. Hong, C.-W. Lin, and Y.-L. Wu, "Incrementally Fast Updated Frequent Pattern Trees," Expert Systems with Applications, 2008.

[3]. Y.-C. Li, J.-S. Yeh, and C.-C. Chang, "Isolated Items Discarding Strategy for Discovering High Utility Itemsets," Data and Knowledge Eng., 2008.

[4]. Y.-S. Lee and S.-J. Yen, "Incremental and Interactive Mining of Web Traversal Patterns," Information Sciences, 2008.

[5]. S. Zhang, J. Zhang, and C. Zhang, "EDUA: An Efficient Algorithm for Dynamic Database Mining," Information Sciences, 2007.

[6]. A. Erwin, R.P. Gopalan, and N.R. Achuthan, "CTU-Mine: An Efficient High Utility Itemset Mining Algorithm Using the Pattern Growth Approach," *Proc. Seventh IEEE Int'l Conf. Computer and Information Technology (CIT '07)*, 2007.

[7]. J. Hu and A. Mojsilovic, "High Utility Pattern Mining: A Method for Discovery of High Utility Itemsets," *Pattern Recognition*, 2007.

[8]. C. Lucchese, S. Orlando, and R. Perego, "Fast and Memory Efficient Mining of Frequent Closed Itemsets," IEEE Trans. Knowledge and Data Eng.,Jan. 2006.

[9]. U. Yun and J.J. Leggett, "WFIM: Weighted Frequent Itemset Mining with a Weight Range and a Minimum Weight," *Proc.Fifth SIAM Int'l Conf. Data Mining (SDM '05),* 2005.

[10]. Y. Liu, W.-K. Liao, and A. Choudhary, "A Two Phase Algorithm for Fast Discovery of High Utility of Itemsets," *Proc. Ninth Pacific Asia Conf.Knowledge Discovery and Data Mining (PAKDD '05)*, 2005.

[11]. Y. Liu, W.-K. Liao, and A. Choudhary, "A Fast High Utility Itemsets Mining Algorithm," *Proc. First Int'l Conf. Utility-Based Data Mining*, 2005.

[12]. J. Wang, J. Han, Y. Lu, and P. Tzvetkov, "TFP: An Efficient Algorithm for Mining Top-K Frequent Closed Itemsets," IEEE Trans. Knowledge and Data Eng., 2005.

[13]. G. Grahne and J. Zhu, "Fast Algorithms for Frequent Itemset Mining Using FP-Trees," IEEE Trans. Knowledge and Data Eng.,2005.