International Journal On Engineering Technology and Sciences – IJETS™
ISSN (P): 2349-3968, ISSN (O): 2349-3976
Volume 2 - Issue 5, May 2015

HUI Miner Algorithms for Transactional Databases

Ms. Arya P.
ME Student
Department of CSE
Kathir College of Engineering
Coimbatore

Mr. D. Ravi
Assistant Professor
Department of CSE
Kathir College of Engineering
Coimbatore

Mrs. R. Subathra
Professor and Head
Department of CSE
Kathir College of Engineering
Coimbatore

Abstract— Data mining is the process of revealing nontrivial, previously unknown and highly useful information from large databases. Discovering these useful patterns hidden in a database plays an essential role in several data mining techniques, such as frequent pattern mining, weighted pattern mining and high utility pattern mining. Mining high utility itemsets from a transactional database refers to the mining of itemsets with high utility like profits. Even though a number of relevant algorithms have been proposed in this regard, they produce a large number of candidate itemsets for high utility itemsets. Such a candidate itemset degrades the mining performance in terms of execution time and space requirement. The situation may become even worse when the database contains lots of long transactions. In the proposed system there are three algorithms, namely UP-Growth UP-Growth+ and Apriori, for mining high utility itemsets with a set of effective strategies for pruning candidate itemsets. The information regarding the high utility itemsets is maintained in a tree-based data structure named utility pattern tree (UP-Tree) such that candidate itemsets can be generated efficiently with only two scans of database. Experimental results prove that the proposed algorithms reduce the number of candidate items effectively and also outperform other algorithms substantially in terms of runtime, especially when the database contains a lot of long transactions.

Index Terms— Data mining, candidate itemsets, high utility itemsets, transactional database

I. INTRODUCTION

Data mining is the process of revealing nontrivial, previously unknown and potentially useful information from large databases. Discovering useful patterns hidden in a database plays an essential role in several data mining tasks, such as frequent pattern mining, weighted frequent pattern mining, and high utility pattern mining. Among them, frequent pattern mining [6] is a fundamental research topic that has been applied to different kinds of databases, such as transactional databases, streaming databases and time series databases and various application domains, such as bioinformatics, web click-stream analysis and mobile environments. Nevertheless, relative importance of each item is not considered in frequent pattern mining. To address this problem, weighted association rule [3] was proposed. In this framework, weights of items, such as unit profits of items in transactional databases, are considered. With this concept, even if some items appear infrequently, they might still be found if they have high weights. However in this frame work the quantities of item are not specified yet.

In view of this, utility mining [4] [5] emerges as an important topic in data mining field. Mining high utility itemsets from databases refers to finding the itemsets with high profit. Utility of an itemsets is defined as the product of its external utility and its internal utility. An itemsets is called a high utility itemsets [7] if its utility is no less than a user specified minimum utility threshold; otherwise it‟s called a low utility itemsets. In other words, pruning search space for high utility itemsets mining is difficult because a superset of a low-utility itemset may be a high utility Itemset.

Mining high utility itemsets from databases is an important task has a wide range of applications such as website click stream analysis, business promotion in chain hypermarkets, cross marketing in retail stores, online e-commerce management, and mobile commerce environment planning, and even finding important patterns in biomedical applications.

A. Statement about the Problem

Mining high utility itemsets from transactional database refers to the discovery of itemsets with high utility like profits. Although a number of relevant algorithms have been proposed in recent years, they incur the problem of producing a large number of candidate itemsets. Such a large number of candidate itemsets degrades the mining performance in terms of execution time and space requirements. The situation may be worse when the database contains lots of long transactions or long high utility itemsets. Here we are implementing two algorithms UP-Growth (Utility Pattern growth)[9] and UP-Growth+; for mining high utility itemsets [2] with a set of effective strategies for pruning candidate itemsets and finally applying Apriori algorithm to obtain the high utility itemsets. Mining potential high utility itemsets(PHUIs) from global UP-Tree and local UP-Trees by UP-Growth with two strategies Discarding Local Unpromising items (DLU) and Decreasing Local Node utilities (DLN) or by UP-Growth+ with two strategies Discarding Local unpromising items(DNU) and Decreasing node utilities (DNN). Then identify actual high utility itemsets from the set of PHUIs. The information of high utility itemsets is maintained in tree based data structure, named UP-tree such that candidate itemsets can be generated efficiently with only two scans of database.

B. Objective and Scope of the System

The main objective of this thesis is to implement two efficient algorithms named UP-Growth and UP-Growth+ and apply Apriori algorithm [1] for mining high utility itemsets from transactional database. A data structure named UP-Tree is used for maintaining the information of high utility itemsets.

Potential high utility itemsets can be efficiently generated from UP-tree with only two database scans. In an UP-Tree, each node N consists of N.name, N.count, N.nu, N.parent and a set of child nodes. N.name is the node‟s item name. N.count is the node‟s support count. N.nu is the node‟s node utility, i.e., overestimated utility of the node. N.parent records the parent node of N. A table named header table is employed to facilitate the traversal of UP-Tree.

II. PREVIOUS WORKS

A number of relevant algorithms have been proposed in recent years for mining high utility itemsets from transactional databases.

Extensive studies have been proposed for mining frequent patterns. Among the issues of frequent pattern mining, the most famous are association rule mining [3] and sequential pattern mining. One of the well-known algorithms for mining association rules is Apriori, which is the pioneer for efficiently mining association rules from large databases. Pattern-growth based association rule mining algorithms such as FP-Growth[10] were afterward proposed. It is widely recognized that FP-Growth achieves a better performance than Apriori-based algorithms since it finds frequent itemsets without generating any candidate itemset and scans database just twice. In the framework of frequent itemset mining, the importance of items to users is not considered. Thus, the topic called weighted association rule mining [3] was brought to attention.

Although weighted association rule mining considers the importance of items, in some applications, such as transaction databases, items‟ quantities in transactions are not taken into considerations yet. Thus, the issue of high utility itemset mining is raised and many studies have addressed this problem. Liu et al. proposed an algorithm named Two- Phase which is mainly composed of two mining phases. In phase I, it employs an Apriori-based level-wise method to enumerate HTWUIs (High Transactional Weighted Utilities). Candidate itemsets with length k are generated from length k-1 HTWUIs and their TWUs are computed by scanning the database once in each pass. After the above steps, the complete set of HTWUIs is collected in phase I. In phase II, HTWUIs that are high utility itemsets are identified with an additional database scan. Although two-phase algorithm reduces search space by using TWDC (Transactional Weighted Downward Closure) property, it still generates too many candidates to obtain HTWUIs and requires multiple database scans. To overcome this problem, Li et al. proposed an isolated items discarding strategy (IIDS) to reduce the number of candidates. By pruning isolated items during level-wise search, the number of candidate itemsets for HTWUIs in phase I can be reduced. However, this algorithm still scans database for several times and uses a candidate generation-and-test scheme to find high utility itemsets.

To efficiently generate HTWUIs in phase I and avoid scanning database too many times, Ahmed et al. proposed a tree-based algorithm, named IHUP. A tree based structure called IHUP-Tree is used to maintain the information about itemsets and their utilities. Each node of an IHUP-Tree consists of an item name, a TWU value and a support count. IHUP algorithm has three steps: 1) construction of IHUP-Tree, 2) generation of HTWUIs, and 3) identification of high utility itemsets. In step 1, items in transactions are rearranged in a fixed order such as lexicographic order, support descending order or TWU descending order. Then the rearranged transactions are inserted into an IHUP-Tree. Fig. 1 shows the global IHUP Tree for the database in Table 1, in which items are arranged in the descending order of TWU. For each node in Fig. 1, the first number beside item name is its TWU and the second one is its support count. In step 2, HTWUIs are generated from the IHUP-Tree by applying FP-Growth .Thus, HTWUIs in phase I can be found without generating any candidate for HTWUIs. In step 3, high utility itemsets and their utilities are identified from the set of HTWUIs by scanning the original database once.

Although IHUP achieves a better performance than IIDS and Two-Phase, it still produces too many HTWUIs in phase I. Note that IHUP and Two-Phase produce the same number of HTWUIs in phase I, since they both use TWU framework to overestimate itemsets‟ utilities.

However, this framework may produce too many HTWUIs in phase I since the overestimated utility calculated by TWU is too large. Such a large number of HTWUIs will degrade the mining performance in phase I substantially in terms of execution time and memory consumption. Moreover, the number of HTWUIs in phase I also affects the performance of phase II since the more HTWUIs the algorithm generates in phase I, the more execution time for identifying high utility itemsets it requires in phase II. As stated above, the number of generated HTWUIs is critical issue for the performance of algorithms.

III. EFFICIENT ALGORITHMS FOR MINING HIGH UTILITY ITEMSETS FROM TRANSACTIONAL DATABASES

The framework of the proposed methods consists of three steps. First scan the database twice to construct a global UP Tree with two strategies Discarding Global Unpromising Items (DGU) and Decreasing Global Node Utilities (DGN) during constructing a Global UP-Tree. Second recursively generate PHUIs from global UP-Tree and local UP-Trees by UP-Growth with two strategies Discarding Local Unpromising items (DLU) and Decreasing Local Node utilities (DLN) or by UP-Growth+ with two strategies Discarding Local unpromising items(DNU) and Decreasing node utilities (DNN) . Then identify actual high utility itemsets from the set of PHUIs.

A. The Proposed Data Structure: UP-Tree

To facilitate the mining performance and avoid scanning original database repeatedly, we use a compact tree structure, named UP-Tree, to maintain the information of transactions and high utility itemsets. Two strategies are applied to minimize the overestimated utilities stored in the nodes of global UP-Tree.

In an UP-Tree, each node N consists of N.name, N.count, N.nu, N.parent and a set of child nodes. N.name is the node‟s item name. N.count is the node‟s support count. N.nu is the node‟s node utility, i.e., overestimated utility of the node. N.parent records the parent node of N. A table named header table is employed to facilitate the traversal of UP-Tree. In header table, each entry records an item name; an overestimated utility.

The construction of a global UP-Tree can be performed with two scans of the original database. In the first scan, TU (Transaction Utility) of each transaction is computed. At the same time, TWU (Transaction Weighted Utility) of each single item is also accumulated. By TWDC property, an item and its supersets are unpromising to be high utility itemsets if its TWU is less than the minimum utility threshold. Such an item is called an unpromising item. UP-tree is constructed with two strategies DGU and DGN. First strategy is discarding global unpromising items and their actual utilities from transactions and transaction utilities of the database. New TU after pruning unpromising items is called reorganized transaction utility (RTU). RTU of a reorganized transaction Tr is denoted as RTU (Tr). By reorganizing the transactions, not only less information is needed to be recorded in UP-Tree, but also smaller overestimated utilities for itemsets are generated. Strategy DGU uses RTU to overestimate the utilities of itemsets instead of TWU.The second strategy is 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.

B. The Proposed Mining Method: UP-Growth

UP growth algorithm is applying after the construction of global UP-Tree for mining high utility itemsets. The common method for generating patterns in tree-based algorithms contain three steps: 1) Generate conditional pattern bases by tracing the paths in the original tree; 2) construct conditional trees (also called local trees) by the information in conditional pattern bases; and 3) mine patterns from the conditional trees. However, strategies DGU and DGN cannot be applied into conditional UP-Trees since actual utilities of items in different transactions are not maintained in a global UP-Tree. We cannot know actual utilities of unpromising items that need to be discarded in conditional pattern bases unless an additional database scan is performed. To overcome this problem, a naıve solution is to maintain items‟ actual utilities in each transaction into each node of global UP-Tree. However, this is impractical since it needs lots of memory space. For the two strategies, we maintain a minimum item utility table to keep minimum item utilities for all global promising items in the database. Minimum item utility of item Ip in database D, denoted as miu(Ip), is Ip‟s utility in transaction Td if there does not exist a transaction Td‟ in D such that u(Ip,Td‟)<u(Ip,Td).

Minimum item utilities are utilized to reduce utilities of local unpromising items in conditional pattern bases instead of exact utilities. An estimated value for each local unpromising item is subtracted from the path utility of an extracted path. Path utility of a path p in Im‟s conditional pattern base

(abbreviated as {im}-CPB) is denoted as PU(p, {Im}-CPB) and defined as NIm‟s node utility where p is retrieved by tracing NIm in the UP-Tree.

UP growth is implemented by using two strategies. First strategy is DLU (Discarding Local Unpromising items). Local unpromising items and their estimated utilities are discarding from the paths and path utilities of conditional pattern bases. DLU can be recognized as local version of DGU. It provides a simple but useful schema to reduce overestimated utilities locally without an extra scan of original database. Second strategy is DLN (Decreasing Local Node utilities) for the nodes of local UP-Tree by estimated utilities of descendant nodes.

The process of mining PHUIs by UP-Growth is described as follows: First, the node links in UP-Tree corresponding to the item im, which is the bottom entry in header table, are traced. Found nodes are traced to root of the UP-Tree to get paths related to im. All retrieved paths, their path utilities and support counts are collected into im‟s conditional pattern base. A conditional UP-Tree can be constructed by two scans of a conditional pattern base. For the first scan, local promising and unpromising items are learned by summing the path utility for each item in the conditional pattern base. Then, DLU is applied to reduce overestimated utilities during the second scan of the conditional pattern base. When a path is retrieved, unpromising items and their estimated utilities are eliminated from the path and its path utility by (1). Then the path is reorganized by the descending order of path utility of the items in the conditional pattern base. DLN is applied during inserting reorganized paths into a conditional UP-Tree.

C. The Proposed Mining Method: UP-Growth+

UP-Growth achieves better performance than FP-Growth by using DLU and DLN to decrease overestimated utilities of itemsets. However, the overestimated utilities can be closer to their actual utilities by eliminating the estimated utilities that are closer to actual utilities of unpromising items and descendant nodes. There is also an improved method, named UP-Growth+, for reducing over estimated utilities more effectively. In UP-Growth, minimum item utility table is used to reduce the overestimated utilities. In UP-Growth+, minimal node utilities in each path are used to make the estimated pruning values closer to real utility values of the pruned items in database.

Minimal node utility for each node can be acquired during the construction of a global UP-Tree. First, we add an element, namely N.mnu, into each node of UP-Tree. N.mnu is minimal node utility of N. When N is traced, N.mnu keeps track of the minimal value of N.name‟s utility

in different transactions. If N.mnu is larger than u(N. name, Tcurrent), N. mnu is set to u (N.name, Tcurrent). When a local UP Tree is being constructed, minimal node utilities can also be acquired by the same steps of global UP-Tree. In the mining process, when a path is retrieved, minimal node utility of each node in the path is also retrieved. In UP-Growth+ algorithm, minimum item utilities are replaced with minimum node utilities. UP-Growth+ algorithm uses two strategies .That are, DNU and DNN. First strategy is discarding local unpromising items and their estimated node utilities from the paths and path utilities of conditional pattern bases and Second strategy is decreasing local node utilities for the nodes of local UP-Tree by estimated utilities of descendant nodes.

D. Concept of Up-Tree and Up-Growth Algorithm With an Example

The proposed method consists of three steps

- Scan the database twice to construct a global UP-tree with two strategies DGU and DGN.
- Recursively generate potential high utility itemsets from global UP-tree and local UP-trees by UP-Growth with two strategies DLU and DLN or by UP-Growth+ with the two strategies DNU and DNN.
- Identifying actual high utility itemsets from the set of potential high utility itemsets.

Consider the transactional database given in table.

Transaction id (TID)	*Transaction*
T1	*(A,1)(C,1)(D,1)*
T2	*(A,2)(C,6)(E,2)(G,5)*
T3	*(A,1)(B,2)(C,1)(D,6)(E,1)(F,5)*
T4	*(B,4)(C,3)(D,3)(E,1)*
T5	*(B,2)(C,2)(E,1)(G,2)*

- Here each transaction contains item name and its quantity. For e.g.: (A, 1) indicates item A with quantity 1.
- The profit of an item is calculated by subtracting cost price (CP) from its selling price (SP). Suppose item A has CP=10 and SP=15, then profit (A) =5. Items and their unit profits are given in table.

Item	*A*	*B*	*C*	*D*	*E*	*F*	*G*
Unit profit	*5*	*2*	*1*	*2*	*3*	*1*	*1*

- Utility of an item Ip in transaction Td U (Ip,Td) = q(Ip,Td)* P(Ip)
  ie, U({A},T1) = 1*5=5
- Utility of an itemset X in the transaction Td Σ U (Ip,Td)
  ie, U({AD},T1) = U({A},T1)+U({D},T1)
  =5+2=7
- Utility of an itemset X in the database ΣU(X,Td)
  ie, U({AD}) = U({AD},T1)+U({AD},T3)
  = ((1*5) + (1*2)) + ((1*5) + (6*2))
  = 7+17=24

- An itemset X is called an high utility itemset if and only if U(X)>= min_util.
- Transaction utility(TU) for each transaction TU (Td) = U (Td,Td)
  ie, TU (T1) = (1*5)+(1*1)+(1*2) = 8
  Similarly, TU (T2) = 27, TU (T3) = 30, TU (T4) = 20, TU (T5) = 11
- Transaction weighted utility(TWU) of an itemset TWU(X) = ΣTU (Td)
  ie, TWU(A) = U(T1,T1)+U(T2,T2)+U(T3,T3) = 8+27+30=65
  Similarly, TWU(B)=61, TWU(C)=96, TWU(D)=58, TWU(E)=88, TWU(F)=30, TWU(G)=38.

Construction of global UP-tree

Construction of global UP-tree is performed with two database scans. In the first scan each transactions TU is computed; at the same time, each one item's TWU is also accumulated.

TUs of each transaction in table 1 and TWUs of each items is shown below.

Transaction id(TID)	*Transaction*	*TU*
T1	*(A,1)(C,1)(D,1)*	*8*
T2	*(A,2)(C,6)(E,2)(G,5)*	*27*
T3	*(A,1)(B,2)(C,1)(D,6)(E,1)(F,5)*	*30*
T4	*(B,4)(C,3)(D,3)(E,1)*	*20*
T5	*(B,2)(C,2)(E,1)(G,2)*	*11*

Items and their TWUs are shown in table

Item	*A*	*B*	*C*	*D*	*E*	*F*	*G*
TWU	*65*	*61*	*96*	*58*	*88*	*30*	*38*

An item Ip is called a *promising item* if TWU (Ip) >=

min_util, where min_util is the user specified minimum utility. Otherwise it is called an *unpromising item*. Suppose 40are the min_util, then in this example F and G are the unpromising items. After getting all promising items, DGU (Discarding Global Unpromising items) is applied. This strategydiscards the global unpromising items and their actual utilities from transactions and transaction utilities of the database. That is shown below.

Transaction id	*Transaction*	*TU*
T1	*(A,1)(C,1)(D,1)*	*8*
T2	*(A,2)(C,6)(E,2)*	*22*
T3	*(A,1)(B,2)(C,1)(D,6)(E,1)*	*25*
T4	*(B,4)(C,3)(D,3)(E,1)*	*20*
T5	*(B,2)(C,2)(E,1)*	*9*

The transactions are reorganized by pruning the unpromising items and sorting the remaining promising items in fixed order (based on the TWUs). That is shown in table below.

Transaction id(TID)	*Transaction*	*RTU*
T1′	*(C,1)(A,1)(D,1)*	*8*
T2′	*(C,6)(E,2)(A,2)*	*22*
T3′	*(C,1)(E,1)(A,1)(B,2)(D,6)*	*25*
T4′	*(C,3)(E,1)(B,4)(D,3)*	*20*
T5′	*(C,2)(E,1)(B,2)*	*9*

Each transaction after the above reorganization is called a *reorganized transaction.*

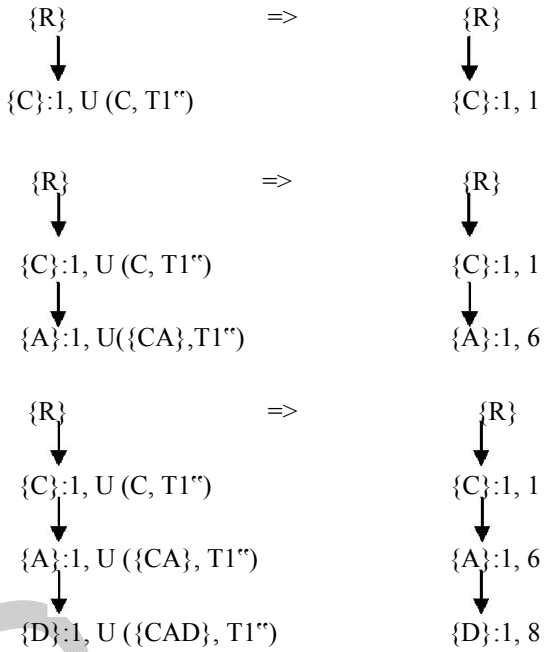
Then a function insert_reorganized_transaction is called to apply DGN (Decreasing Global Node utilities) during constructing a global UP-tree. That is, after a transaction has been reorganized, it is inserted into the global UP-tree. That process is shown below.

- When T1'= {(C, 1) (A, 1) (D, 1)} is inserted, the first node Nc is created with Nc.item ={C} and Nc.count =1. Nc.nu is increased by RTU(T1')

  minus the utilities of the rest items that are behind {C} in T1'.That is Nc.nu=RTU(T1')-((U{A},T1')+(U{D},T1'))=8-(5+2)=1.It can also be calculated as Nc.nu=U({C},T1').

- The second node NA is created with Nc.item = {A}, Nc.count =1, NA.nu=U ({C A}, T1‟) =6.

This process is shown below



- After inserting all reorganized transactions by the same way, the global UP-Tree shown in figure.1 is constructed.

- There is table named *header table* is employed to facilitate the traversal of UP-Tree. In header table, each entry records an item name, an over estimated utility, and a link. The link points to the last occurrence of the node which has the same item as the entry in the UP-tree. By following the links in header table and nodes in UP-tree, the nodes having the same name can be traversed efficiently.

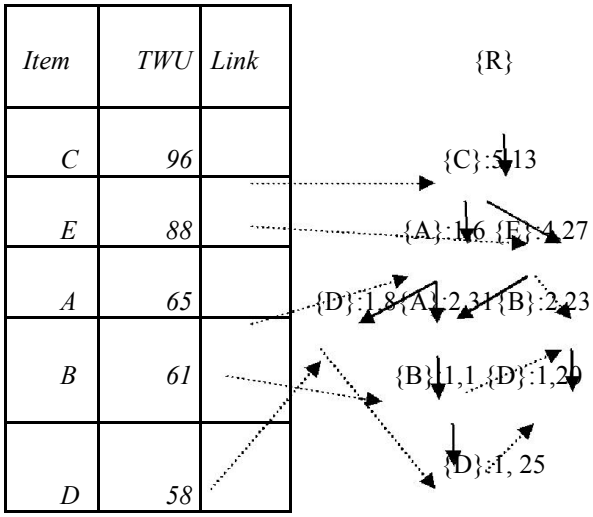


Figure: The process of mining potential high utility itemsets by UP-Growth.

- First, the node links in UP-tree corresponding to item

Im, which is the bottom entry in header table, are traced. Found nodes are traced to root of the UP-tree to get paths related to Im. All retrieved paths, their path utilities and support counts are collected in to Im's conditional pattern base.

- In this example, first, node links corresponding to D is traced. The paths obtained are {AC}, {BAEC} and {BEC}.

D's Conditional Pattern Base (CPB) is shown below.

{D}'s Conditional Pattern Base		
Path	*Support Count*	*Path utilities by DGU and DGN.*
{AC}	*1*	*8*
{BAEC}	*1*	*25*
{BEC}	*1*	*20*

order of path utility of the item in the CPB.

- In this example min_util is 40. Here item A has path utility less than the min_util.so A is the local unpromising item.

{D}'s conditional pattern base after applying DLU is shown below.

Retrieved Path:Path utility	*Reorganized path: path utility(after DLU)*	*Support Count*
<AC>:8	*<C>: 3*	*1*
<BAEC>:25	*<CBE> : 20*	*1*
<BEC>:20	*<CBE> : 20*	*1*

- DLN (Decreasing Local Node utilities) is applied during inserting reorganized paths in to a conditional UP-tree.

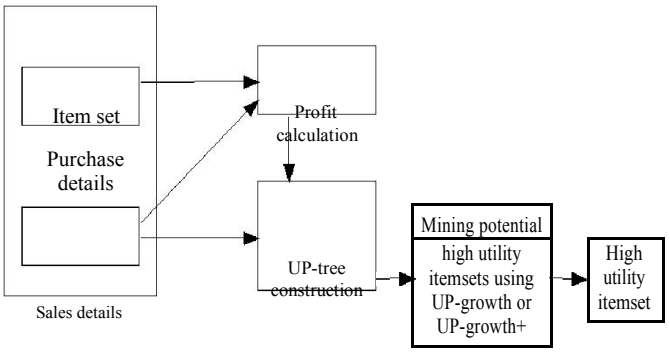
- The conditional UP-tree can be constructed by two scans of a conditional pattern base. For the first scan, local promising and unpromising items are learned by summing the path utility for each item in the conditional pattern base.

  Eg: path utility of item {A} in the {D}'s CPB is 8+25=33
  Path utility of each item is shown below.

Local item	*A*	*B*	*C*	*D*
Path Utility	*33*	*45*	*53*	*45*

- Then DLU (Discarding Local Unpromising items) is applied. That is discarding local unpromising items and their estimated utilities from the paths and path utilities of conditional pattern bases by equation, pu(p,{Im}-CPB) = NIm.nu-∑miu(i)*p.count, where *miu i*s the minimum item utility. Minimum item utility of item Ip in database D, denoted as miu(Ip), is, Ip's utility in transaction Td if there does not exist a transaction Td' in D such that U(Ip,Td')<U(Ip,Td).
- Then the path is reorganized by descending IV. ARCHITECTURAL DIAGRAM OF PROPOSED SYSTEM

Figure shows the overall process for mining high utility item sets from transactional database. The details of items are stored in a database. The purchase and sales details of these items are stored in other databases respectively. From the purchase and sales details, calculate the profit of each item. Next step is to construct the global UP-tree by using sales transaction details and profit of each item. Then recursively generate potential high utility itemsets (PHUIs) from UP-tree by UP-growth or UP-growth+ algorithm. Identify actual high utility itemsets from the set of PHUIs.



International Journal On Engineering Technology and Sciences – IJETS™
ISSN (P): 2349-3968, ISSN (O): 2349-3976
Volume 2 - Issue 5, May 2015

- According to DLN when the first reorganized path <C> is inserted in to {D}-tree, the first node Nc is created under root node with Nc.nu=3.

  This process is shown below.

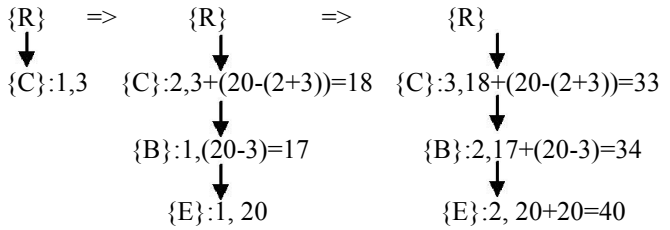


Figure: Local UP-tree for {D}

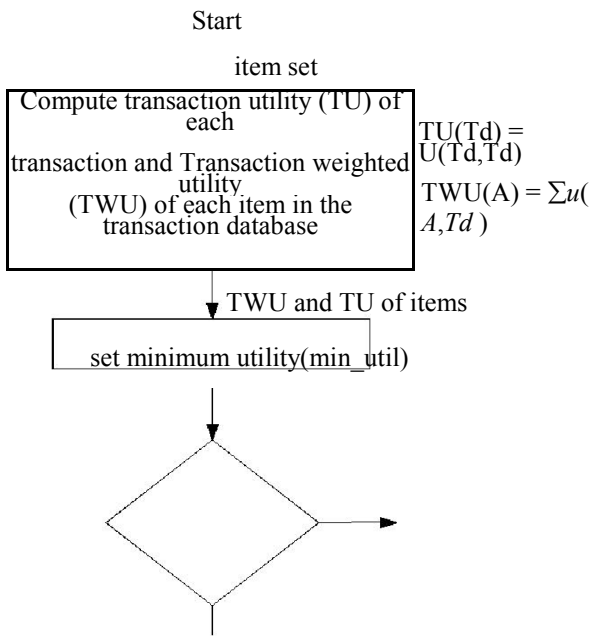
- Generating potential high utility itemsets from {D}-tree by UP-growth, the PHUIs that are involved with {D} are obtained. That are {D}, {DE}. After mining the remaining items in header table, all PHUIs in the global up-tree can be obtained. From that we can identify high utility itemsets.

A. Flow Chart of Up-Tree Construction

Input: Sales Transactions
Output: UP-Tree

Description: Figure below shows the process of up-tree construction.

First scan the database to compute Transaction utility (TU) of each transaction and Transaction Weighted utility (TWU) of each item. Then set the user specified minimum utility. If the TWU is greater than the minimum utility then consider the item as promising item and otherwise it is an unpromising item. After getting all promising items, they are sorted in the header table by descending order of TWU. Then transactions are reorganized by sorting the promising items and subtracting utilities of unpromising items from their TUs. After that reorganized transactions are inserted into UP-tree.



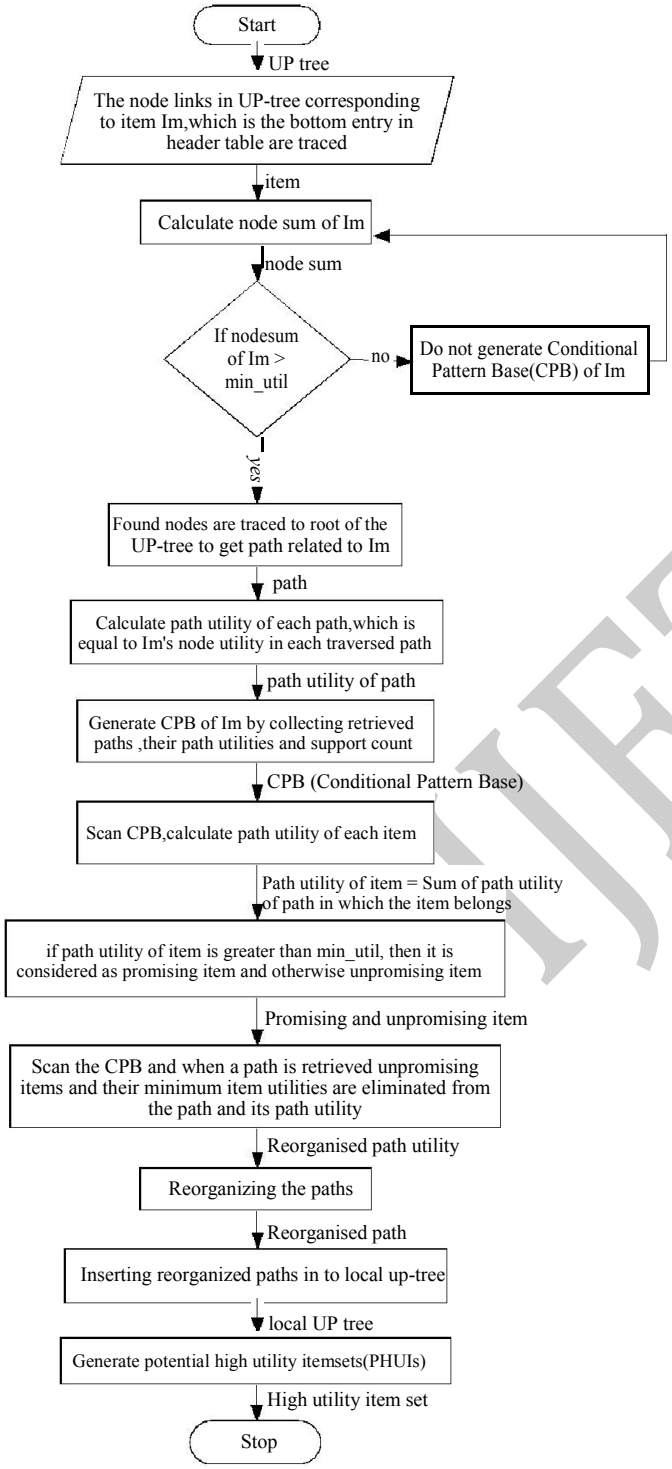
Flow chart: UP-Tree construction

B. Flow Chart of Up-Growth Algorithm

Input: UP-Tree
Output: High Utility Itemsets

Description: The process of mining PHUIs by UP-Growth is described as follows: First, the node links in UP-Tree corresponding to the item im, which is the bottom entry in header table, are traced. Found nodes are traced to root of the UP-Tree to get paths related to im. All retrieved paths, their path utilities and support counts are collected into im‟s conditional pattern base. A conditional UP-Tree can be constructed by two scans of a conditional pattern base. For the first scan, local promising and unpromising items are learned by summing the path utility for each item in the conditional pattern base. When a path is retrieved, unpromising items and their estimated utilities are eliminated from the path

and its path utility. Then the path is reorganized by the descending order of path utility of the items in the conditional pattern base. Then insert this reorganized path into local up-tree and there by generate potential high utility itemsets. From that identify the high utility itemsets. This process is shown below.

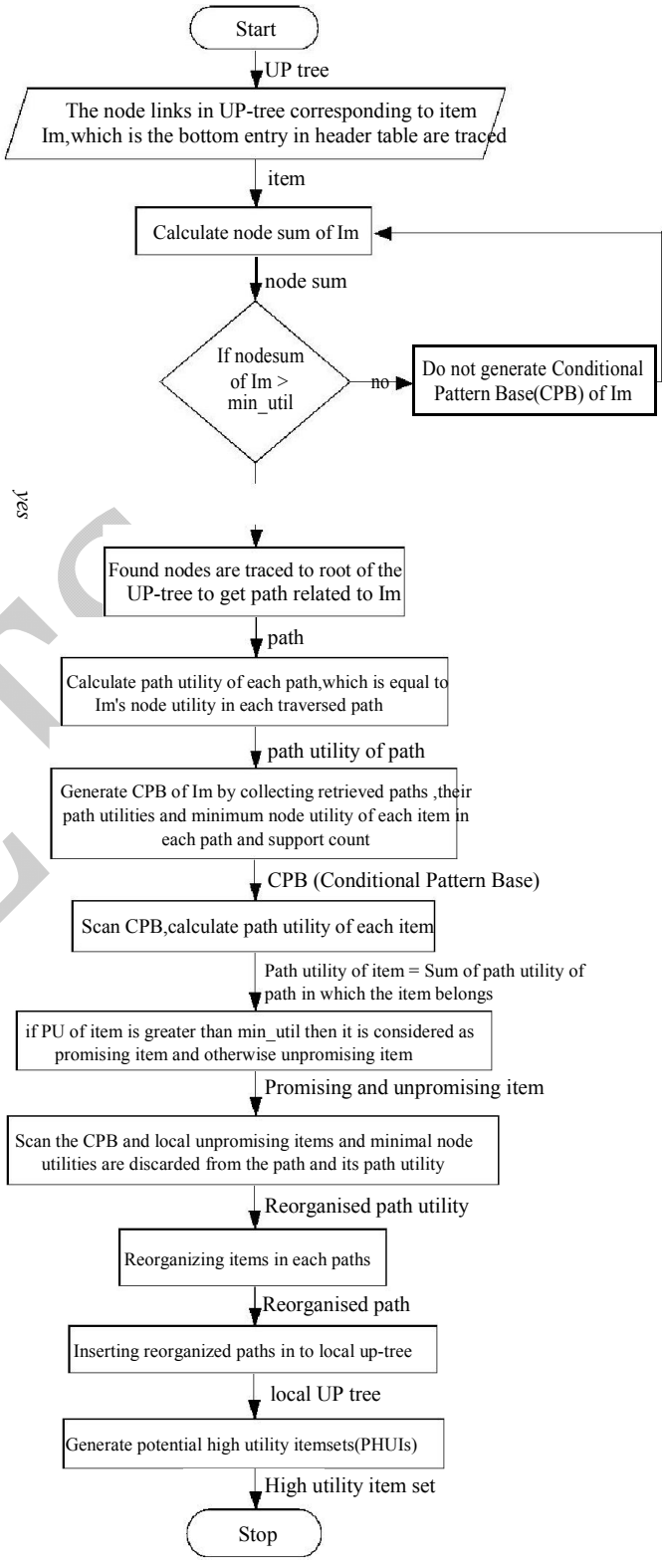


Flowchart: UP-Growth algorithm

C. Flow Chart of Up-Growth+ Algorithm

Input: UP-tree

Output: High Utility Itemsets



Flowchart: UP-Growth+ algorithm

V. CONCLUSIONS

Mining high utility itemsets from transactional database is the discovery of itemsets with high utility like profits. Although a number of relevant algorithms have been proposed in recent years, they incur the problem of producing a large number of candidate itemsets. Such a large number of candidate itemsets degrades the mining performance in terms of execution time and space requirements. The situation may become worse when the database contains lots of long transactions or long high utility itemsets [8]. Here we implemented two algorithms UP-Growth (Utility Pattern growth) and UP-Growth+ along with Apriori algorithm; for mining high utility itemsets with a set of effective strategies for pruning candidate itemsets.UP-growth and UP-Growth+ algorithms are efficient algorithms for mining high utility itemsets from transactional database. The information of high utility itemsets is maintained in a tree-based data structure named utility pattern tree (UP-Tree) such that candidate itemsets can be generated efficiently with only two scans of database. High utility itemsets can be identified by scanning reorganized transactions. Since there is no unpromising item in the reorganized transactions, I/O cost and execution time can be further reduced. This technique works well especially when the original database contains lots of unpromising items.

After finding all potential high utility itemsets, next step is to identify high utility itemsets and their utilities from the set of potential high utility itemsets by scanning original database. This step is called phase II. However, in previous studies, two problems in this phase occur: 1) number of high transactional weighted utility items is too large; and (2) scanning original database is very time consuming. In this framework, overestimated utilities of potential high utility itemsets are smaller than or equal to transactional weighted utilities of high transactional weighted utility items since they are reduced by the proposed strategies. Thus, the number of potential high utility itemset is much smaller than that of high transactional weighted utility itemsets. Therefore, in phase II, this method is much efficient than the previous methods. Moreover, although these methods generate fewer candidates in phase I, scanning original database is still time consuming since the original database is large and it contains lots of unpromising items. In view of this, in this framework, high utility itemsets can be identified by scanning reorganized transactions. Since there is no unpromising item in the reorganized transactions, I/O cost and execution time for phase II can be further reduced. This technique works well especially when the original database contains lots of unpromising items.

The entire system is tested and verified. Applying different datas and then tested. It is found that these algorithms are efficient and produces accurate results. In some cases the results produced by UP-Growth and UP-Growth+ are same. In some cases UP-Growth+ reduces the number of high utility itemsets produced than UP-Growth+. The difference between these algorithms are mainly UP-Growth algorithm using minimum item utility and this value is deducted from its local tree, whereas UP-Growth+ using minimum node utility and this value is deducted from its local trees. Many unpromising items are filtered when producing reorganized transactions and before constructing local trees. It helps to reduce the number of candidate sets generated.

Mining high utility itemsets from databases is an important task has a wide range of applications such as website click stream analysis, business promotion in chain hypermarkets, cross marketing in retail stores, online e-commerce management, and mobile commerce environment planning, and even finding important patterns in biomedical applications.

ACKNOWLEDGMENT

We would like to express our thanks to our institution, Kathir College of Engineering and all its faculties for supporting us in the work of the paper and its implementation.

REFERENCES

[1] R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules," Proc. 20th Int"l Conf. Very Large Data Bases (VLDB), pp. 487-499, 1994.

[2] J. Han, J. Pei, Y. Yin, "Mining frequent patterns without candidate generation," in Proc. of the ACM-SIGMOD Int„l Conf. on Management of Data, pp. 1-12, 2000.

[3] W. Wang, J. Yang and P. Yu, "Efficient mining of weighted association rules (WAR)," in Proc. of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD 2000), pp. 270-274, 2000.

[4] Y. Liu, W. Liao and A. Choudhary, "A fast high utility itemsets mining algorithm," in Proc. of the Utility-Based Data Mining Workshop, 2005.

[5] V. S. Tseng, C. J. Chu and T. Liang, "Efficient Mining of Temporal High Utility Itemsets from Data streams," in Proc. of ACM KDD Workshop on Utility-Based Data Mining Workshop (UBDM"06), USA, Aug., 2006.

[6] J. Hu, A. Mojsilovic, "High-utility pattern mining: A method for discovery of high-utility item sets", Pattern Recognition 40 (2007) 3317 – 3324

[7] A. Erwin, R. P. Gopalan and N. R. Achuthan, "Efficient mining of high utility itemsets from large datasets," in Proc. of PAKDD 2008, LNAI 5012, pp. 554-561..

[8] S.Shankar, T.P.Purusothoman, S.Jayanthi, N.Babu, A fast agorithm for mining high utility itemsets, Proceedings of IEEE International Advance Computing Conference (IACC 2009), Patiala, India, pp.1459-1464

[9] V. S. Tseng, C.-W. Wu, B.-E. Shie and P. S. Yu, "UPGrowth:An Efficient Algorithm for High Utility Itemsets Mining," in Proc. of the 16th ACM SIGKDD Conf. On Knowledge Discovery and Data Mining (KDD 2010), pp.253-262, 2010.

[10] Christian Borgelt, "An Implementation of the FP-growth Algorithm".