International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064

An Efficient Algorithm for High Utility Mining

Mahija K C[1], Anu KS[2]

[1, 2] MTech CSE, Department of Computer Science, KMCT College of Engineering, Manassery, Calicut, Kerala, India

Abstract: ***Mining high utility itemsets from a transactional database refers to the discovery of itemsets with high utility like profits. Efficient discovery of high utility itemsets from a transactional database is a crucial and difficult task. Given a transactional database and a profit table containing profit of each item present in the database, high utility itemsets can be discovered. In recent years, several approaches have been proposed for generating high utility itemsets, but they incur the problems of producing a large number of candidate itemsets for high utility itemsets. The situation may become worse when the database contains lots of long transactions or lots of unpromising items. In this paper, we propose HUP-Tree (High Utility Pattern Tree), for maintaining information of high utility itemsets, which is then mined to produce high utility itemsets. The proposed methods reduce the number of candidate sets efficiently and improve the mining performance in terms of execution time and space requirement. This approach is quite effective especially when transactions contain lots of unpromising items***

Keywords: High utility mining, data mining, transaction utility, transaction weighted utility, utility mining**.**

1. Introduction

Discovering useful patterns hidden in a database plays an essential role in several data mining tasks, such as frequent pattern mining, association rule mining, and high utility pattern mining. Among them, frequent pattern mining [1], [14], [21] is a fundamental research topic with wide data mining applications. In market analysis, mining frequent itemsets from a transaction database refers to the discovery of the itemsets which frequently appear together in the transactions. Relative importance of each item is not considered in frequent pattern mining. To address this problem, weighted association rule mining was proposed [4], [11]. In this framework, weights of items, such as unit profits of items in transaction databases, are considered. With this concept, even if some items appear infrequently, they might still be found if they have high weights.

However, the unit profits and purchased quantities of items are not considered in the framework of frequent itemset mining or association rule mining. Hence, it cannot satisfy the requirement of the user who is interested in discovering the itemsets with high sales profits. In view of this, utility mining emerges as an important topic in data mining for discovering the itemsets with high utility like profits. Association rules mining (ARM) is one of the most widely used techniques in data mining and knowledge discovery and has tremendous applications like business, science and other domains. It is used to make the decisions about marketing activities such as promotional pricing or product placements.

The basic meaning of utility is the profitability of items to the users. The utility of items in a transaction database consists of two aspects, i.e., the profit of distinct items, and number of the items in the transaction. The utility of an item is defined as the quantity multiplied by the unit profit of that item. By an itemset we simply mean a set of items. A transactional database consists of many transactions. Each transaction represents items purchased and their respective quantities. An itemset is called a high utility itemset if its utility is no less than a user specified threshold; otherwise, the itemset is called a low utility itemset. Mining high utility itemsets from databases is an important task which is essential to a wide range of applications such as, cross-marketing in retail stores [3], [10], [19], [20] and business promotion in chain hypermarkets.

A naive approach to solve this problem is to enumerate all itemsets from the databases by the principle of exhaustion. Obviously, this approach will encounter the large search space problem, especially when databases contain lots of long transactions or a low minimum utility threshold is set. Hence, how to effectively prune the search space and efficiently capture all high utility itemsets with no miss is a big challenge in utility mining. To address this issue and to find all high utility itemsets efficiently, we are proposing a new data structure named HUP- Tree, which can be used to store information about high utility itemsets, and then can be mined to produce high utility itemsets. With the proposed approach the size candidate sets can be reduced efficiently.

2. Related Work

Extensive studies have been proposed for mining frequent itemsets [1], [2], [13], [14], [21], [22]. One of the well-known algorithms is Apriori algorithm [1], which is the pioneer for efficiently mining association rules from large databases. The tree-based approaches such as FP-Growth [14] were afterward proposed. It's widely recognized that FP-Growth achieves a better performance than Apriori-based approaches since it finds frequent itemsets without generating any candidate itemset and it scans database just twice. However, in the framework of frequent itemset mining, the importance of items to users is not considered.

Thus, the topic called weighted association rule mining was brought to attention [4], [11]. Cai et al. first proposed the concept of weighted items and weighted association rules [4]. However, since the framework of weighted association rules does not have downward closure property, mining performance cannot be improved. To address this problem, the concept of weighted downward closure property was developed. By using transaction weight, weighted support

can not only reflect the importance of an itemset but also maintain the downward closure property during the mining process. There are also many studies [6], [11] that have developed different weighting functions for weighted pattern mining. 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, some methods were proposed for high utility mining [3], [5], [10], [16], [17], [19], [20] from the databases. One popular algorithm named Two-Phase algorithm [19] proposed by Liu et al. consists of two phases. In phase I, Two-Phase algorithm employs a breadth first search strategy to enumerate HTWUIs. It generates candidate itemsets of length *k* from HTWUIs of length (*k*-1) and prunes candidate itemsets by TWDC property. In each pass, HTWUIs and their estimated utility values i.e., TWUs, are computed by scanning database. After that, the complete set of HTWUIs is collected in phase I. In phase II, high utility itemsets and their utilities are identified from the HTWUIs by scanning original database once. Two-Phase algorithm generates too many candidates for HTWUIs and requires multiple database scans. To overcome this problem, Li et al. proposed an isolated items discarding strategy, abbreviated as IIDS, to reduce the number of candidates. By pruning isolated items during the level-wise search, the number of candidate itemsets for HTWUIs in phase I can be reduced effectively. However, this approach still scans database multiple times and uses a candidate generation-and-test scheme to find high utility itemsets.

To avoid scanning database multiple times, Ahmed et al. proposed a tree-based algorithm, called IHUP [3], for mining high utility itemsets. They use an IHUP-Tree to maintain the information of high utility itemsets and transactions. Every node in IHUP-Tree consists of an item name, a support count, and a TWU value. The framework of the algorithm consists of three steps: (1) The construction of IHUP-Tree, (2) the generation of HTWUIs and (3) identification of high utility itemsets. In step 1, items in the 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 the IHUP-Tree.

IHUP and Two-Phase produce the same number of HTWUIs in phase I since they use transaction weighted utilization mining model to overestimate the utilities of the itemsets. To overcome this model Vincent et al proposed a tree structure [20] and some strategies to reduce overestimate utility values of the itemsets. But this still produces too many candidate sets. When lots of unpromising items occur together in many transactions, their transaction utility and TWU will be high and they may be included in the candidate set. Similar situation also arises when a low utility item occurs most of the times with a very high utility item; then that item will also be included in the candidate set. This model may overestimate too many low utility itemsets as HTWUIs and produce too many candidate itemsets.

Such a large number of HTWUIs degrades the mining performance in phase I in terms of execution time and memory consumption. Besides, the number of HTWUIs in phase I also affects the performance of the algorithms in phase II since the more HTWUIs are generated in phase I, the more execution time is required for identifying high utility itemsets in phase II. As stated above, the number of HTWUIs generated in phase I form a crucial problem to the performance of algorithms.

By applying the proposed strategies, the number of candidates generated in phase I can be reduced effectively and the high utility itemsets can be identified more efficiently since the number of itemsets needed to be checked in phase II is highly reduced in phase I.

3. Proposed Method

3.1 Problem Statement

Given a transaction database D and a user-specified minimum utility threshold "*minimum utility*", the problem of mining high utility itemsets from D is to find the complete set of the itemsets whose utilities are larger than or equal to *minimum utility*.

3.2 Definitions

Suppose we are given a database D as shown in Table 1. This is a transactional database containing a number of transactions.

Table 1: An example database

TID	*Transaction*	*TU*
T1	(A,1) (C,10) (D,1)	17
T2	(A,2) (C,6) (E,2) (G,5)	27
T3	(A,2) (B,2) (D,6) (E,2) (F,1)	37
T4	(B,4) (C,13) (D,3) (E,1)	30
T5	(B,2) (C,4) (E,1) (G,2)	13
T6	(A,1) (B, 1) (C,1) (D,1) (H,2)	12

Each transaction is identified by a unique identifier called TID. Each transaction consists of the items purchased and the quantities of each item. For example the transaction T_4 consists of 4 quantities of B, 13 quantities of C, 3 quantities of D and 1 quantity of E. There is another table called profit table as shown in Table 2. Profit table stores profit of each item present in the database.

Table 2: An example profit table

Item	A	B	C	D	E	F	G	H
Profit	5	2	1	2	3	5	1	1

Utility of an item '*i*' in a transaction *Td* is denoted as U(*i*,*Td*); and it is the profit of that item i in transaction *Td*. For e.g., U (A, T_1) =1* 5 = 5;

Utility of an itemset X in *Td* is denoted as u(X; *Td*) and is defined as the profit of X in *Td*; i.e., profit from all the items in itemset X in *Td*.
For eg., U(AD, T_1)= U(A,T_1) + U(D,T_1) = 5 + 2 =7;

Utility of an itemset X in the whole database *D* is denoted as U(X) and is defined as the sum of the utilities of X from all the transactions in which X is present. For eg, U (AD) = U (AD,T_1) + U (AD,T_3) + U(AD,T_6) = 7 +22 + 7 =36;

If minimum utility is set to 30, (AD) is a high utility itemset.

Transaction Utility (TU) of a transaction represents the utility or profit from that transaction. Transaction Utility of a transaction, say T_i is denoted as TU(T_i) and it is defined as the sum of the utilities of all items present in that transaction. For example TU of T_4 is the sum of the utilities of B, C, D and E; i.e., 4 * 2 + 13 * 1 + 3 * 2 + 1 * 3=30

Transaction weighted utility (TWU) of an item is the sum of the transaction utilities in which that item is present. So transaction-weighted utility of an itemset X is defined as the sum of the transaction utilities of all the transactions containing X, and is denoted as TWU (X). For example A is present in transactions T_1, T_2 and T_3. So transaction weighted utility of A can be calculated as the sum of transaction utilities of T_1, T_2, T_3 and T_6 i.e., 17 + 27 + 37 + 12 = 93.

After addressing the definitions about utility mining, we introduce the transaction-weighted downward closure (TWDC) which is proposed in [19]. An itemset *X* is called a high-transaction weighted utility itemset (HTWUI) if TWU(*X*) is no less than minimum utility. Transaction-weighted downward closure property is defined as follows: For any itemset *X*, if *X* is not a HTWUI, any superset of *X* is a low utility itemset.

Downward closure property can be maintained in utility mining by applying the transaction weighted utility. For example, TU (T_2) = U (ACEG, T_2) =27; TWU (G) = TU (T_2) + TU (T_5) = 27 + 13 = 40. If minimum utility is set to 30, {G} is a HTWUI. However, if minimum utility is set to 50, {G} and its supersets are not HTWUIs since TWU (G) must be no less than the TWUs of all {G}'s supersets.

3.3 The proposed data structure: HUP- Tree

The framework of the proposed methods consists of the following steps: 1) Scan the database to insert reduced transaction set into an HUP-Tree, 2) recursively generate PHUIs from HUP-Tree and 3) identify actual high utility itemsets from the set of PHUIs. Note that we use a new term "reduced transaction set" to distinguish the transaction set found by our methods from those using traditional method. By following our effective methods, the set of HTWUIs generated will become much smaller than the set of HTWUIs generated using traditional approach.

To facilitate the mining performance and avoid scanning original database repeatedly, we use a compact tree structure, named HUP-Tree, to maintain the information of transactions and high utility itemsets. Some methods are applied to minimize the overestimated utilities stored in the nodes of HUP-Tree. In following sections, the elements of HUP-Tree are first defined. Next, how to insert reduced transaction set into HUP tree are introduced. Finally, how the proposed methods work is illustrated in detail by a running example.

3.3.1 The Elements in HUP-Tree

In an HUP-Tree, each node N consists of N.name, N.count, N.nu, N.mq, N.parent, N.hlink 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.mq is the minimum node quantity. N.parent records the parent node of N. N.hlink is a node link which points to a node whose item name is the same as N.name. Here we have used a new term N.mq i.e., minimum node quantity, which will be explained in detail in the following section. A table named header table is employed to facilitate the traversal of HUP-Tree. In header table, each entry records an item name, an overestimated utility, and a link. The link points to the last occurrence of the node which has the same item as the entry in the HUP-Tree. By following the links in header table and the nodes in HUP-Tree, the nodes having the same name can be traversed efficiently.

The construction of an HUP-Tree can be performed by scanning the original database. In the first scan, TU of each transaction is computed. At the same time, TWU 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. An item *i* is called a promising item if TWU (*i*) >= minimum utility. Otherwise it is called an unpromising item. An item is also called a promising item if its overestimated utility is no less than minimum utility. Otherwise it is called an unpromising item. Another important property is that if *i* is an unpromising item, *i* and all its supersets are not high utility itemsets. In other words we can say that only the supersets of promising items are possible to be high utility itemsets.

Next step is removing unpromising items from the transaction and their utilities from the transaction's TU. From this new reduced set of transactions and their reduced transaction utility, TWU of each remaining promising item is calculated. If any item's TWU is less than *minimum utility*, it is also considered as an unpromising item and so that item is removed from the transactions and their utilities from TU. This process is repeated till we get no more unpromising items i.e., items whose TWU is less than *minimum utility*. This framework may require a few database scans, but produces less number of HTWUIs in phase I. If we don't do this step to reduce time spent on phase I a large number of HTWUIs will be produced and this will degrade the mining performance in phase II substantially in terms of execution time and memory consumption. That is, 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.

New transaction utility after subtracting unpromising items' utility is called reduced transaction utility (Reduced TU). New transactions after pruning unpromising items is called reduced transactions. The items in the reduced transactions are arranged in the descending order of their TWU. By using these reduced transactions, not only less information is needed to be recorded in HUP-Tree, but also smaller overestimated utilities for itemsets are generated. Here we are using reduced TU to overestimate the utilities for itemsets instead of TWU. Since the utilities of unpromising items are excluded, reduced TU must be no larger than TWU. Therefore, the number of PHUIs must be no more

than that of HTWUIs generated with TWU [3], [19]. This proposed approach is quite effective especially when transactions contain lots of unpromising items. Besides that, this approach can be easily integrated into TWU-based algorithms [3], [19], [20]. Table 3 shows the reduced transactions and their reduced transaction utilities, which are to be inserted into HUP-Tree.

Table 3: Reduced transactions

TID	*Reduced transaction*	*Reduced TU*
T_1	(C,10) (D,1) (A,1)	17
T_2	(E,2) (C,6) (A,2)	22
T_3	(E,2) (D,6) (A,2) (B,2)	32
T_4	(E,1) (C,13) (D,3) (B,4)	30
T_5	(E,1) (C,4) (B,2)	11
T_6	(C,1) (D,1) (A,1) (B, 1)	10

It is shown in [3], [20] that the tree-based framework for high utility itemset mining applies the divide-and-conquer technique in mining processes. Thus, the search space can be divided into smaller subspaces. For example, in Fig. 1, the search space can be divided into the following subspaces: {B}-Tree, {A}-Tree without containing {B}, {D}-Tree without containing {B} and {A}, {C}-Tree without containing {B}, {A}, and {D}, and {E}-Tree without containing {B}, {A}, {D}, and {C}. It can be observed that in the subspace {A}-Tree, all paths are not related to {B} since the nodes {B} are below the nodes {A} in IHUP-Tree. In other words, the items that are descendant nodes of the item i_m will not appear in {i_m}- Tree; only the items that are ancestor nodes of i_m will appear in {i_m}- Tree. From this viewpoint, we propose a method for decreasing overestimated utilities of a node by removing the utilities of descendant nodes from their node utilities in HUP-Tree. The process is performed during the construction of the HUP-Tree. By decreasing descendent node's utility, the utilities of the nodes that are closer to the root of HUP-Tree are further reduced. This method is quite effective and especially suitable for the databases containing lots of long transactions. In other words, the more items a transaction contains, the more utilities can be discarded. On the contrary, traditional TWU mining model is not suitable for such databases since the more items a transaction contains, the higher TWU is. In following sections, we describe the process of constructing an HUP-Tree.

All the reduced transactions are inserted into the tree one by one using the usual insertion procedure [3],[14], [20]. Let N_R be the root node of HUP-Tree. Let reduced transaction T_d = {i_1, i_2, i_3,…, i_n} be the first transaction to be inserted into the tree. The node for i_1, i.e., N_{i_1}, is created under N_R and its support count is updated. Then the utilities of descendant nodes under N_{i_1}, i.e., N_{i_2} to N_{i_n} are subtracted from utility of N_{i_1}. Then the second item i_2 in T_d is inserted. Like that all items in T_d is inserted. After inserting the first transaction T_d, let T_{d+1} be the next transaction to be inserted. The first item in T_{d+1} is to be inserted first. We first check if a node already exists for that item as an immediate child of N_R. If yes, its support count is incremented and corresponding utilities are updated. If such a node does not exist a new node is created under N_R with support count as 1 and it utility as that items utility minus the utility of remaining items in the transaction. This is done till all the reduced transactions are inserted in to the HUP-Tree.

Besides storing item name and its utility, minimum node quantity of that node called N_{mq} is also stored in each node of HUP-Tree. Minimum node quantity in each path is used to make the estimated pruning values closer to real utility values of the pruned items in database. Assume that Nx is the node which records the item x in the path p in a HUP-Tree and Nx is composed of the items x from the set of transactions Trans-set (Tx). The minimum node quantity of x in p is denoted as mq(x,p). The mq(x,p) is defined as minimum of all the quantity values of x in each transaction in the Trans-set. Minimum node quantity for each node is calculated during the construction of HUP-Tree. First, we add an element, namely N.mq, into each node of HUP-Tree. N.mq is the minimum node quantity of N. Initially N.mq is set to a very high value. When N is traced, N.mq keeps track of the minimum value of N.name's quantity in different transactions. If N.mq is larger than quantity (N.name, $T_{current}$), N.mq is set to quantity (N.name, T. current). Figure1 shows the HUP-Tree with N. mq in each node. In Figure 1, N.utility is the first number, N.count is the middle number and N. mq is the last number in each node.

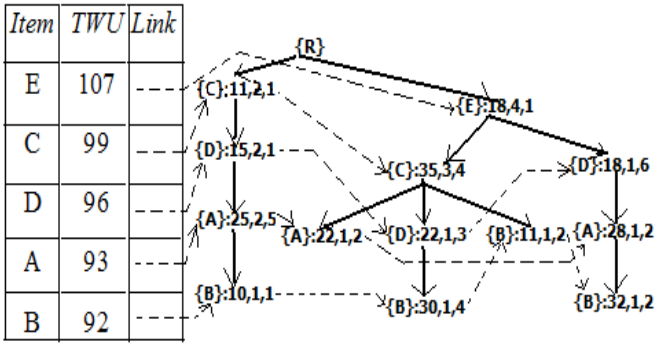

Figure 1: HUP-Tree after inserting reduced transactions in Table 3

An example is given to explain how to construct a HUP-Tree. Consider the reduced transactions in Table 3 and the profit table in Table 2. After a transaction has been reduced, it is inserted into the HUP-Tree. When T_1= {(C, 10) (D, 1) (A, 1)} is inserted, the first node N_C is created with N_C.item={C} and N_C.count=1. N.nu is increased by reduced TU(T_1) minus the minimum utilities (here minimum utilitiy of an item in a path in HUP tree is calculated as the product of N.mq and its unit profit from Table 2.) of the rest items that are behind {C} in T_1,that is, NC:nu =17-(2*1+5*1) = 10. Note that it can also be calculated as the sum of utilities of the items that are before item {D} in T_1, i.e., N_C.nu= U({C};T_1)= 10. The second node ND is created with N_D.item ={D}, N_D.count= 1 and N_D.nu= reduced TU(T_1)-U({A},T_1)=17- 5*1= 12. The third node N_A is created with N_A.item = {A}, N_A:count = 1 and N_A:nu = reducedTU(T_1) = 17. After inserting all reduced transactions by the same way, the global HUP-Tree shown in Fig. 1 is constructed. Comparing with the tree used in [3], [20], node utilities of the nodes in HUP-Tree are less.

3.4 Mining the HUP- Tree

After constructing the global HUP-tree, PHUIs can be generated by using the popular method of FP-Growth [14]. Using the proposed approach PHIs generated will be less compared to other algorithms. For the HUP-Tree in Figure1,

we can get the following PHUIs, i.e., {A}:75, {B}:83, {BD}: 60, {BDE}:56, {BE}:62 and {D}:55.

4. Conclusion

In this paper, we have proposed a tree structure named HUP-Tree for maintaining the information of high utility itemsets present in a transactional database. This HUP-Tree can be then used for mining high utility itemsets from transaction databases. PHUIs can be efficiently generated from HUP-Tree with few database scans. Moreover, we have developed an approach to decrease overestimated utility and thus enhance the performance of utility mining. This algorithm will perform well by reducing both the search space and the number of candidates. This proposed algorithm, will outperform the state of-the-art algorithms substantially especially when databases contain lots of long transactions or when a low minimum utility threshold is used.

References

[1] A. 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] R. Agrawal and R. Srikant, "Mining Sequential Patterns," Proc. 11th Int'l Conf. Data Eng., pp. 3-14, Mar. 1995.
[3] C.F. Ahmed, S.K. Tanbeer, B.-S. Jeong, and Y.-K. Lee, "Efficient Tree Structures for High Utility Pattern Mining in IncrementalDatabases," IEEE Trans. Knowledge and Data Eng., vol. 21, no. 12, pp. 1708-1721, Dec. 2009.
[4] C.H. Cai, A.W.C. Fu, C.H. Cheng, and W.W. Kwong, "Mining Association Rules with Weighted Items," Proc. Int'l Database Eng. and Applications Symp.(IDEAS '98), pp. 68-77, 1998.
[5] R. Chan, Q. Yang, and Y. Shen, "Mining High Utility Itemsets," Proc. IEEE Third Int'l Conf. Data Mining, pp. 19-26, Nov. 2003.
[6] J.H. Chang, "Mining Weighted Sequential Patterns in a Sequence Database with a Time-Interval Weight," Knowledge-Based Systems, vol. 24, no. 1, pp. 1-9, 2011.
[7] M.-S. Chen, J.-S. Park, and P.S. Yu, "Efficient Data Mining for Path Traversal Patterns," IEEE Trans. Knowledge and Data Eng., vol. 10, no. 2, pp. 209-221, Mar. 1998.
[8] C. Creighton and S. Hanash, "Mining Gene Expression Databases for Association Rules," Bioinformatics, vol. 19, no. 1, pp. 79-86, 2003.
[9] M.Y. Eltabakh, M. Ouzzani, M.A. Khalil, W.G. Aref, and A.K. Elmagarmid, "Incremental Mining for Frequent Patterns in Evolving Time Series Databases," Technical Report CSD TR#08-02, Purdue Univ., 2008.
[10] A. Erwin, R.P. Gopalan, and N.R. Achuthan, "Efficient Mining of High Utility Itemsets from Large Data Sets," Proc. 12th Pacific-Asia Conf. Advances in Knowledge Discovery and Data Mining (PAKDD), pp. 554-561, 2008.
[11] F. Tao, F. Murtagh, and M. Farid, "Weighted Association Rule Mining Using Weighted Support and Significance Framework," Proc. ACM SIGKDD Conf. Knowledge Discovery and Data Mining (KDD '03), pp. 661-666, 2003.J. Han, G. Dong, and Y. Yin, "Efficient Mining of Partial Periodic Patterns in Time Series Database," Proc. Int'l Conf. on Data Eng., pp. 106-115, 1999.
[13] J. Han and Y. Fu, "Discovery of Multiple-Level Association Rules from Large Databases," Proc. 21th Int'l Conf. Very Large Data Bases, pp. 420-431, Sept. 1995.
[14] J. Han, J. Pei, and Y. Yin, "Mining Frequent Patterns without Candidate Generation," Proc. ACM-SIGMOD Int'l Conf. Management of Data, pp. 1-12, 2000.
[15] S.C. Lee, J. Paik, J. Ok, I. Song, and U.M. Kim, "Efficient Mining of User Behaviors by Temporal Mobile Access Patterns," Int'l J. Computer Science Security, vol. 7, no. 2, pp. 285-291, 2007.
[16] H.F. Li, H.Y. Huang, Y.C. Chen, Y.J. Liu, and S.Y. Lee, "Fast and Memory Efficient Mining of High Utility Itemsets in Data Streams," Proc. IEEE Eighth Int'l Conf. on Data Mining, pp. 881- 886, 2008.
[17] Y.-C. Li, J.-S. Yeh, and C.-C. Chang, "Isolated Items Discarding Strategy for Discovering High Utility Itemsets," Data and KnowledgeEng., vol. 64, no. 1, pp. 198-217, Jan. 2008.
[18] C.H. Lin, D.Y. Chiu, Y.H. Wu, and A.L.P. Chen, "Mining Frequent Itemsets from Data Streams with a Time-Sensitive Sliding Window," Proc. SIAM Int'l Conf. Data Mining (SDM '05), 2005.
[19] Y. Liu, W. Liao, and A. Choudhary, "A Fast High Utility Itemsets Mining Algorithm," Proc. Utility-Based Data Mining Workshop, 2005.
[20] V.S. Tseng, C.-W. Wu, B.-E. Shie, and P.S. Yu, "UP-Growth: An Efficient Algorithm for High Utility Itemsets Mining," Proc. 16th ACM SIGKDD Conf. Knowledge Discovery and Data Mining (KDD'10), pp. 253-262, 2010.
[21] J. Pei, J. Han, H. Lu, S. Nishio, S. Tang, and D. Yang, "H-Mine: Fast and Space-Preserving Frequent Pattern Mining in Large Databases," IIE Trans. Inst. of Industrial Engineers, vol. 39, no. 6, pp. 593-605, June 2007.
[22] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Moal, and M.C. Hsu, "Mining Sequential Patterns by Pattern-Growth: The Prefixspan Approach," IEEE Trans. Knowledge and Data Eng., vol.16, no.10, pp. 1424-1440, Oct. 2004.

Author Profile

Mahija K C received the B.Tech in Computer Science and Engineering from Co-operative Institute of Technology, Vadakara (CUSAT) in 2003. She had worked in National institute of Technology, Calicut and AWH Engineering College, Calicut as Lecturer in the Department of Computer Science. She is currently doing last semester in M.Tech Computer Science and Engineering from KMCT College of Engineering, Calicut (Calicut University).