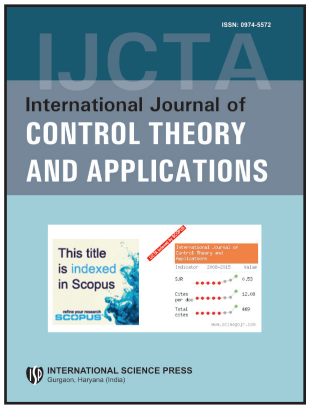


International Journal of Control Theory and Applications

ISSN : 0974-5572

© International Science Press

Volume 10 • Number 16 • 2017

A Novel Approach for Mining Related High Utility Itemsets with Database Reduction

J. Wisely Joe[a] and S.P. Syed Ibrahim[b]

[a]*Research Scholar, SCSE, VIT University (Chennai Campus), India*
[b]*Professor, SCSE, VIT University (Chennai Campus), India*

Abstract: Data mining techniques are the act of making decisions from databases. Recently, there has been a very high focus in designing high utility itemset mining algorithms for large databases and many papers are being published. High utility itemsets are set of items with high utility, importance and profit. We propose a new HUI mining algorithm which uses utility list to represent transactions and it considerably reduces the running time of finding high utility itemsets from large databases. The high utility itemsets mined using this algorithm are Related High Utility Itemsets (RHUI's) which are not exact HUI's generated from base algorithms but a similar itemsets with little variation with that of exact HUI's. RHUI's are generated in lesser time by applying utility list approach with transaction merging. Related transactions are merged together to reduce the size of database. Merging requires an extra DB scan and this is balanced by reducing the necessity of constructing utility lists for all items present in DB. Which guarantees, the related high utility items are found in lesser time and they are profitable when compared with existing algorithms.

Keywords: Data mining, utility mining, high utility itemset, utility list, RHUI.

1. INTRODUCTION

Association analysis[1], [2], [7] in data mining handles a collection of original transactions to find association between the items present in the database DB using item's occurrences in transactions. Association between the items are expressed in the form of association rules and frequent patterns. These rules are much used in making business decisions in the fields like market analysis, healthcare systems, banking, census data and fraud detection analysis. But not all rules generated by this method are giving interesting solutions. Also the number of candidate itemsets generated during the rule construction is very high.

Frequent pattern growth algorithm [5],[19] also does the same association mining but without candidate itemset generation. This algorithm uses a compact and compressed tree structure for memory representation of transactions. The number of scans involved in the construction of frequent patterns is less when compared with previous approach and divide and conquer partitioning used in this method leads to less usage of search space. This algorithm works well even when the database contains combination of long and short transactions . This approach is used in many frequent pattern methods like maximal and closed frequent pattern algorithms.

Association rule mining algorithm and FP-Growth algorithm generate many rules and frequent patterns but they may not be profitable. Previous two algorithms assume that each item cannot appear more than once in each transaction . Importance or usability of item in the transactions are not taken into consideration. They use only frequency of items or itemsets as a measure to generate rules or patterns. Profit is an integer value given to every item participating in transactional DB based on their importance or chances of making profit in the business. Profit of item can also be referred as weight of item. There are numerous algorithms to generate weighted frequent itemsets with weight or profit as a factor [19]. In all the above algorithms, an user defined threshold value is used to filter rules or patterns or weighted rules. Itemsets or patterns with frequency value greater than that threshold will be considered as weighted itemsets or frequent itemsets.

To address the issues related to quantity, high utility itemset (HUIM) mining [3][6][9] has been defined. Quantity is units of item purchased, which differs in every transaction depends on customer need. High utility itemset mining is to extract the itemsets with utility value higher than the specified minimum utility threshold. Threshold is set by considering profitability, weight, season, importance, quantity and cost of item or itemsets. High utility itemset mining algorithms proposed in papers [3], [6], [9], [10], [12], [13], [14] resolve all the issues present in high utility itemset mining algorithms .The main challenge or limitation of HUI mining algorithm is its performance. A large number of non-profitable candidate itemsets generated by HUI mining algorithms may degrade the performance of mining process because most of the rules generated by these algorithms are not useful to take successful business decisions. To avoid the problem of candidate generation, HUI-Miner (High Utility Itemset Miner) algorithm[18], and FHM[20] were proposed to mine HUIs in one phase in a faster manner. HUI-Miner algorithm uses a structure called utility list to store the node utility information [18] and it generates no candidate itemsets. Though these algorithms do only one or two scans of database to mine HUIs, data processing tasks take more execution time. So it is necessary to have a more efficient algorithm to mine HUIs. An one-phase algorithm named EFIM (Efficient high-utility Itemset Mining)[16] has a data reduction method to reduce the size of database in turn to minimize data processing time. It merges all identical transactions in a database. However, the number of identical transactions in a database will not be very high. To overcome the challenges in this paper we propose a novel method for reducing the size of the database by merging the related transactions based on transaction utility TU, which leads to the reduction in the cost of database scans and data processing time .

The rest of this paper is organized as follows. Sections 2, 3, 4 and 5 respectively presents the problem definition, the related works, the proposed algorithm and the conclusion in order.

2. BACKGROUND

A. Problem Definition

In this paper, we propose a new method to mine all related high utility itemsets (RHUIs) by database reduction. Database reduction is done through related transactions merging which will reduce the processing time. RHUI mined from this method may or may not be an HUI but the execution time is reduced once this RHUI is included into or excluded from the list of HUIs. This inclusion or exclusion will not affect the efficiency of algorithm because the effect of this is negotiable. Here we have listed all required definitions in detail.

Take a set of m available items denoted as $I = \{I_1, I_2, \ldots, I_m\}$, a transactional database of size n and n denotes number of transactions. $DB = \{T_1, T_2, ..., T_n\}$ such that each transaction T_c, $T_c \subseteq I$ and T_c has a unique identifier called its Transaction identifier Tid. Each item $i \in I$ is associated with a positive number $p(i)$, which represents the profit or weight of item i and is called external utility of i. For every item i in transaction T_c, a positive number $ut(i, T_c)$ is given and is called the internal utility of i, which represents the purchase quantity of i in transaction T_c. For example, Table 1 shows a transaction database containing fifteen transactions (T_1, T_2, ..., T_{15}), which will be used as running example. In this running example transaction T_5 indicates that items A, D, H and G appear in this T_5 with internal utility of 4, 8, 3, and 7. Table 2 gives the external utilities of the items present in database. External utilities of the items present in T_5 are 10, 15, 25 and 5 respectively. A positive integer used for pruning itemsets is called minimum utility threshold MT and is set according to the user's preference.

Definition 1: Utility of an item util(i, T_c)

The utility of an item i in a transaction T_c is denoted as

util(i, T_c) and defined as $p(i) \times ut(i, T_c)$ if $i \in T_c$.

Definition 2: Utility of an itemset in a transaction util(X, T_c)

The utility of an itemset X in a transaction T_c is represented as util(X, T_c) and defined as util(X, T_c) = $\sum_{i \in X} u(i, T_c)$ if $X \subseteq T_c$

Definition 3: Utility of an itemset util(X)

The utility of an itemset is represented as util(X) and defined as util(X) = $\sum_{Tc \in s(X)} u(X, T_c)$, where $s(X)$ is the set of transactions containing X in the database.

For example, the utility of item H in T_5 is util(H, T_5) = 3 × 25 = 75. The utility of the itemset {D, H} in T_5 is util({D, H}, T_5) = util(D, T_5) + util(H, T_5) = 8 × 15 + 3 × 25 = 195. The utility of the itemset {C, D} is util({C, D}) = util({C, D}, T_1) + util({C, D}, T_4) + util({C, D}, T_9) = ((5 × 2) + (7 × 15)) + ((4 × 2) + (7 × 15)) + ((4 × 2) + (2 × 15)) = 266

Table 1
A Transaction Database

TID	*Transaction*
T_1	(A, 3) (B, 3) (C, 5) (D, 7) (G, 10)
T_2	(A, 1) (B, 1) (D, 7) (E, 5)
T_3	(B, 4) (D, 3) (E, 7) (F, 12) (H, 1)
T_4	(B, 2) (C, 4) (D, 7) (E, 8) (F, 9)
T_5	(A, 4) (D, 8) (H, 3) (G, 7)
T_6	(A, 2) (B, 9) (E, 2) (G, 9)
T_7	(B, 3) (C, 2) (F, 8) (G, 3) (H, 2)
T_8	(C, 4) (E, 3) (G, 3) (H, 1)
T_9	(A, 4) (B, 2) (C, 4) (D, 2) (E, 1) (F, 9) (G, 7)
T_{10}	(A, 1) (B, 1) (D, 7) (E, 5)
T_{11}	(A, 6) (B, 7) (D, 9) (E, 4)
T_{12}	(A, 4) (B, 3) (C, 2) (G, 5)
T_{13}	(A, 3) (B, 2) (C, 5) (G, 10)
T_{14}	(E, 12) (F, 7) (G, 6)
T_{15}	(C, 4) (F, 3) (G, 9)

Table 2
Profit table

Item	A	B	C	D	E	F	G	H
Profit	10	6	2	15	1	1	5	25

Definition 4 : Transaction utility TU

The transaction utility of a transaction T_c is represented as $TU(T_c)$, and defined as the sum of utilities of items present in the transaction T_c. For example, transaction utility of some transactions in Table 1 are TU(T1) = 30 + 12 + 10 + 105 + 50 = 207, $TU(T_4) = 242$, $TU(T_9) = 135$, $TU(T_{12}) = 87$ and $TU(T_{15}) = 56$. TU of all transactions in the running examples are listed in Table 3.

Table 3
List of TUs

TID	*TU*
T_1	207
T_2	126
T_3	103
T_4	242
T_5	270
T_6	121
T_7	95
T_8	51
T_9	135
T_{10}	126
T_{11}	141
T_{12}	87
T_{13}	102
T_{14}	49
T_{15}	56

Definition 5: Total Utility ToU

The total utility of a database DB is represented as ToU, and defined as the sum of utilities of all transactions.

In our running example ToU = 207 + 126 + 103 + 242 + 270 + 121 + 95 + 51 + 135 + 126 + 241 + 87 + 102 + 49 + 56 = 2011

Definition 6: Minimum Utility Threshold MT

Minimum utility threshold is a positive integer set by the user. Only if the itemset utility satisfies minimum utility threshold MT, itemset is considered as a high utility itemset and selected for further calculation else will be discarded . In this running example user defined MT is 40%.

Definition 7: Transaction weighted Utility twu

Transaction weighted Utility of an item/itemset in a database DB is the sum of transaction utilities of all transactions containing that item/itemset. For example in Figure 1, twu (D) = $TU(T_1) + TU(T_2) + TU(T_3) + TU(T_4) + TU(T_5) + TU(T_9) + TU(T_{10}) + TU(T_{11}) = 1450$.The TWU's of all 1-itemsets are listed below.

Table 4
Transaction weighted utility

Item	A	B	C	D	E	F	G	H
TWU	1415	1585	975	1450	1194	680	1173	519

3. RELATED WORKS

Property 1: If twu(X) is less than given "MT", all supersets of X are not high utility itemsets.In our example MT is 40%. That is, the itemsets having more utility than 40% of total utility, ToU are high utility itemsets and others are of low utility.

40% of ToU = 2011 * 0.4 = 804. twu(E) = 1194 which is greater than 804.Then E is an high utility itemset . twu(H) = 519 which is lesser than 804. Then H is not considered for further calculations and it is discarded after first scan. Based on Property 1, any super set of a non high utility itemset can not be an high utility itemset (ie) super sets of item H can not be an HUI as item H is not an HUI.

A. Utility – list Structure

Definition 8: Given an itemset X and a transaction T_i where $X \subseteq T_i$, all the items after X in T_i is denoted as T_i/X.

Consider the view in Figure 5, $T_9/\{G, A\} = \{D, B\}$ and $T_{12}/\{C, G\} = \{A, B\}$.

Definition 9: In the transaction remaining utility of itemset X is represented as Rutil(X, T_i), is the sum of the utilities of all the items in T_i/X in T_i, where Rutil(X, T_i) = $\sum_{i \in (Ti/X)}$ util(i, T_i).

In our example, Rutil((G, A), T_9) = $(2 \times 15) + (2 \times 6) = 42$.

Every element in utility list of itemset X has three fields: T_{id}, I_{util}, and R_{util}.

- T_{id} – Identifier of transactions which contains X.
- I_{util} – Utility of X in Transaction, i.e., $u(X, T_i)$.
- R_{util} – Remaining utility of X in Transaction,

i.e., R_{util} (X, T_i).

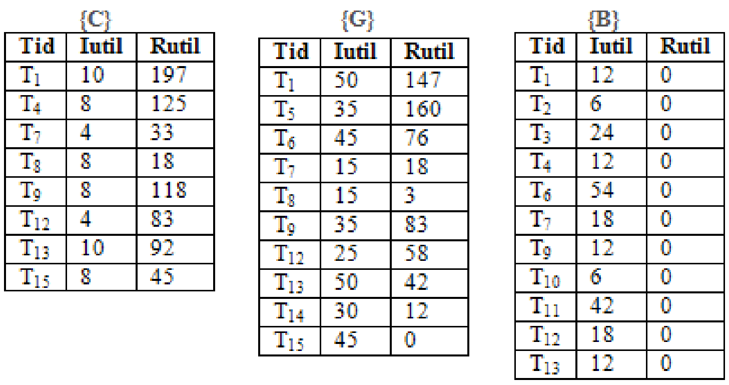
To mine high utility itemsets, most of the algorithms directly perform methods on the entire database. Though we use utility-list structure to maintain the utility information of item/itemset, the process of mining HUI may be tedious, and time consuming one .Because we need to build utility list for each item/itemset by two database scans. In the first scan TU, twu are calculated and if the transaction weighted utility of an item is lesser than a user specified threshold MT, the item is not considered for further actions according to Property 1 in the following mining process. The items whose transaction weighted utility exceed the MT will remain in the transaction/database. Items in the transactions are sorted based on twu from lowest to highest. For the database in Figure 1 with MT - 40%, the algorithm discards items F and H. The remaining items are reordered: C < G < E < A < D < B.

Definition 10: A transaction is reordered after deleting the items whose twu-s are lesser than a user given MT from the transaction. The reordered transactions are listed in Table 5.

During the second database scan, the algorithm scans each reordered transaction again for constructing initial utility-lists. Some of them are shown in Figure 1.

Table 1
A Transaction Database

TID	*Transaction*
T_1	(C, 5) (G, 10) (A, 3) (D, 7) (B, 2)
T_2	(E, 5) (A, 1) (D, 7) (E, 1)
T_3	(E, 7) (D, 3) (B, 4)
T_4	(C, 4) (E, 8) (D, 7) (B, 2)
T_5	(G, 7) (A, 4) (D, 8)
T_6	(G, 9) (E, 2) (A, 2) (B, 9)
T_7	(C, 2) (G, 3) (B, 3)
T_8	(C, 4) (G, 3) (E, 3)
T_9	(C, 4) (G, 7) (E, 1) (A, 4) (D, 2) (B, 2)
T_{10}	(E, 5) (A, 1) (D, 7) (B, 1)
T_{11}	(E, 4) (A, 6) (D, 9) (B, 7)
T_{12}	(C, 2) (G, 5) (A, 4) (B, 3)
T_{13}	(C, 5) (G, 19) (A, 3) (B, 2)
T_{14}	(G, 6) (E, 12)
T_{15}	(C, 4) (G, 9)



{C}

Tid	Iutil	Rutil
T_1	10	197
T_4	8	125
T_7	4	33
T_8	8	18
T_9	8	118
T_{12}	4	83
T_{13}	10	92
T_{15}	8	45

{G}

Tid	Iutil	Rutil
T_1	50	147
T_5	35	160
T_6	45	76
T_7	15	18
T_8	15	3
T_9	35	83
T_{12}	25	58
T_{13}	50	42
T_{14}	30	12
T_{15}	45	0

{B}

Tid	Iutil	Rutil
T_1	12	0
T_2	6	0
T_3	24	0
T_4	12	0
T_6	54	0
T_7	18	0
T_9	12	0
T_{10}	6	0
T_{11}	42	0
T_{12}	18	0
T_{13}	12	0

Figure 1: 1-Itemsets-Utility-List

Consider the Utility list of {C}:

In T_1, Iutil(C, T_1) = 10, Rutil(C, T_1) = util(G, T_1) + util(A, T_1) + util(D, T_1) + util(B, T_1)

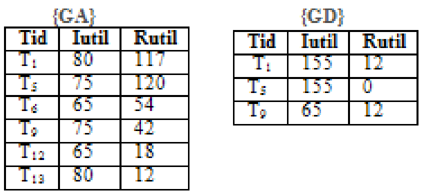
= 50 + 30 + 105 + 12 = 197.

In T_7, Iutil(C, T_7) = 4, Rutil(C, T_7) = util(G, T_7) + util(B, T_7) = 15 + 18 = 33.

For the rest of 1-itemsets, utility lists can be done in the same manner. To find utility list of k-itemsets, it is not necessary to scan the database again. By finding the intersection of the utility list of $\{x\}$ and $\{y\}$, we can find the utility list of $\{x, y\}$. We use transaction identifiers Tids as key to find the transactions which are in both the sets. Number of comparisons for finding common Tids is less, because all Tids in utility lists are ordered.

For each common transaction t, the algorithm will generate a new entry E in the utility list of $\{x, y\}$ with same Tid . The Tid of new entry E is same as the Tid of t. The Iutil of E is the sum of the Iutils associated with transaction t in the utility lists of $\{x\}$ and $\{y\}$. Suppose y is after x, and then Rutil associated with t in the utility list of $\{y\}$ is assigned as the Rutil of entry E . Utility lists of 2-itemsets with itemset {G} as prefix are shown in Figure 2.

For example, to construct the Utility list of itemset {G, A}, this method finds the intersection of utility lists of {G} {T_1, T_5, T_6, T_7, T_8, T_9, T_{12}, T_{13}, T_{14}, T_{15}} and of {A} {T_1, T_2, T_5, T_6, T_9, T_{10}, T_{11}, T_{12}, T_{13}} and it results in {T_1, T_5, T_6, T_9, T_{12}, T_{13}}. That is {G, A} is found in transactions T_1, T_5, T_6, T_9, T_{12} and T_{13}. In T_1, Iutil({G, A}, T_1) = Iutil(G, T_1) + Iutil(A, T_1) = 50 + 30 = 80, and Rutil({G, A}, T_1) = Rutil(A, T_1) = 117. Similarly, for other transactions in the resultant set should be calculated.



{GA}

Tid	Iutil	Rutil
T_1	80	117
T_5	75	120
T_6	65	54
T_9	75	42
T_{12}	65	18
T_{13}	80	12

{GD}

Tid	Iutil	Rutil
T_1	155	12
T_5	155	0
T_9	65	12

Figure 2: 2-Itemsets-Utility-List

B. Search Space

The search space of this HUI mining problem is given in the form of set-enumeration tree [16]. For the set of items I = {I_1, I_2, …, I_n} and total order of all items ($I_1 < I_2 < ... < I_n$), a set-enumeration tree is constructed . The set of items {I} in our running example is given. I = {A, B, C, D, E, G} and total order is C < G < E < A < D < B, a rough set-enumeration tree of all items I is given in Figure 3.

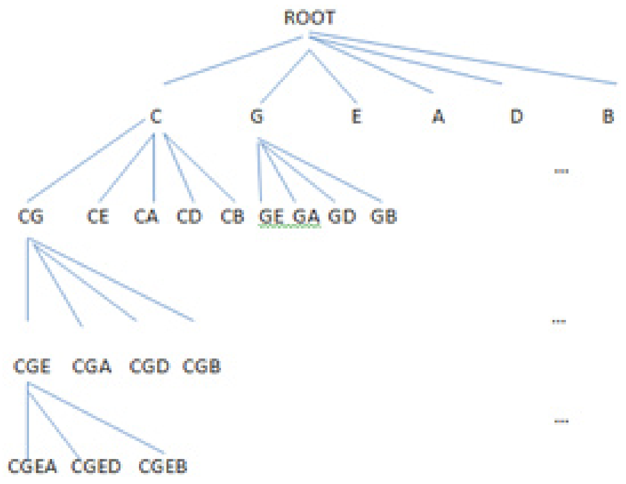


Figure 3: A set-enumeration tree

HUI-Miner algorithm[18] recursively processes only the promising extensions from the set-enumeration tree and leaves up others. This algorithm also clearly explains how to construct the utility list of *k*-itemsets and with whom. The 2-itemset utility lists are constructed when the parent list is empty and the *k*-itemsets utility lists ($k \geq 3$) are constructed when the parent list is not empty.

Pruning strategy in HUI-Miner algorithm is more time-consuming process, because the number of items are more for large databases and the size of set- enumeration tree will also be very high.Our motto is to reduce the size of search space so that HUIs can be found in lesser time. That is, time spent on Utility list construction and pruning should be less when compared with existing algorithm.

4. THE PROPOSED METHOD

Our proposed RHUI-Miner algorithm is a single phase HUI generating algorithm, which gives a new approach to reduce the time and memory required in the process of finding HUIs from large databases. The following sections present and explain how RHUI–Miner algorithm reduces the time and memory required, how the database projection is done and how cost of DB scan is reduced.

A. Database Reduction

Initial steps of this algorithm are same as existing HUI-Miner algorithm. It performs a database scan to calculate the transaction utility(TU) of transactions and transaction weighted utility(twu) of 1-itemsets. Only those items having twu higher than the user given minimum utility threshold (MT) are selected for further steps and others are discarded. Remaining items in every transactions are sorted selected items, find reordered transactions which is sorted based on twu.

Algorithm 1: *DBReduction* Algorithm

Input: *TU[ ]*,Transaction utility of all transactions

|*T*|, Number of transactions

Output: *RDB*, Revised Database

```
1.  Find minimum $MIN$ and maximum $MAX$ element from $TU[\,]$ ;
2.  $D := MAX - MIN$;
3.  Range Value $R := D/(|T|/2)$;
4.  //Split $|T|$ Transactions into $|T|/2$ groups of range R
5.  Do insert every transaction $T_i$ into respective group.
6.  Concatenate transactions in each group into one Reduced Transaction $RT$
7.  Form new $DB$ $RDB$ with half the size of old $DB$.
8.  Return $RDB$
```

Apply RHUI-Miner algorithm on the reordered transactions. All the transactions are to be put in different bins of different ranges based on TU. Number of different ranges can be found by minimum and maximum transaction utility, minTU, maxTU. D is the difference between maxTU and minTU. If the number of transactions in the database is *k* then *k*/2 number of bins are used to hold the database .When dividing D by *k*/2, we get range value. With that, we can form *k*/2 ranges or bins. Scan the TUs alone from memory and find appropriate bin for placing the transaction[11]. After placing we may find that some bins are empty, some are heavily loaded and some have few number of transactions stored in it. Merge the transactions in a bin into one new transaction. That is, every bin makes a new transaction RT except empty bins .Merging transactions is because the list of transactions in a bin may be more or less equal to each other, (ie) they may have same set of items. In our running example, total number of transactions is 15. This may be reduced to 8. Thus the database is reduced to a database having only 8 transactions .

$$\text{minTU} = 49 : \text{maxTU} = 270 : \text{D} = 270 - 49 = 221 : \text{R} = (221/8) = 27.$$

Value of maxTU is divided into 8 ranges and they are 49 to 76,77 to 104, 105 to 132, …, 245 to 270. As per our running example, now we have 8 ranges and the bins 1 to 8 have 3, 4, 3, 2, 0, 1, 1, 1 number of transactions respectively. In reorganized database we have at most *k*/2 number of transactions.

Eg. In range 1 we have <T_8 T_{14} T_{15}>

where, T_8<(C,4) (G,3) (E,3)>, T_{14}< (G,6) (E,12)>, T_{15}<(C,4) (G,9)>.From this we can easily spot out only three items C,G,E are repeatedly present in all the three transactions which are in a bin 1. New transaction RT_1 =<(C,4 + 4) (G, 3 +6 + 9)(E, 3 +12)> = <(C,8) (G,18) (E,15). List of new transactions RT_1 to to RT_7 are shown in Table 6.

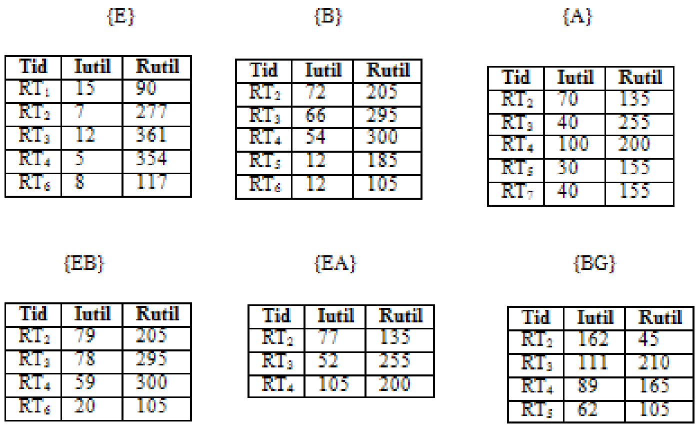
Algorithm 1 explains how the database size reduced to half and now we need to scan the new database which has only reorganized transactions RT. But this time the size of DB is reduced to half and time consumption for finding new TU and twu is reduced to half. The biggest challenge here we face is minimum utility threshold(MT). As the quantity or utility of items in transactions are increased after transaction merging or database projection, we may find most of the itemsets are HUIs for the same MT .

Table 1
Merged Transactions

TID	*Transaction*
RT_1	(C, 8) (E, 15) (G, 18)
RT_2	(C, 9) (E, 7) (D, 3) (B, 12) (G, 18) (A, 7)
RT_3	(E, 12) (A, 4) (D, 14) (B, 11) (G, 9)
RT_4	(C, 7) (G, 7) (E, 5) (A, 10) (D, 11) (B, 9)
RT_5	(C, 5) (G, 10) (A, 3) (D, 7) (B, 2)
RT_6	(B, 2) (C, 4) (D, 7) (E, 8)
RT_7	(A, 4) (D, 8) (G, 7)

Most of the itemsets utility will be higher than that of MT. To solve this issue we use level wise minimum utility threshold [4] [8] [15] [17]. Few utility lists with new transactions are shown in Figure 4.

At every level of pruning, minimum utility threshold MT will be increased based on user preference. Though the time taken for DB scan is increased in RHUI-Miner algorithm, efficiency of the algorithm is improved than the existing approach. Because in existing method more time is spent in construction of Utility lists. In the proposed method, number of Utility lists constructed are limited as the DB size is reduced to half. Thus proposed method is more efficient than existing algorithms.



{E}

Tid	Iutil	Rutil
RT_1	15	90
RT_2	7	277
RT_3	12	361
RT_4	5	354
RT_6	8	117

{B}

Tid	Iutil	Rutil
RT_2	72	205
RT_3	66	295
RT_4	54	300
RT_5	12	185
RT_6	12	105

{A}

Tid	Iutil	Rutil
RT_2	70	135
RT_3	40	255
RT_4	100	200
RT_5	30	155
RT_7	40	155

{EB}

Tid	Iutil	Rutil
RT_2	79	205
RT_3	78	295
RT_4	59	300
RT_6	20	105

{EA}

Tid	Iutil	Rutil
RT_2	77	135
RT_3	52	255
RT_4	105	200

{BG}

Tid	Iutil	Rutil
RT_2	162	45
RT_3	111	210
RT_4	89	165
RT_5	62	105

Figure 4: A set-enumeration tree

B. Pruning Strategy

For every calculated utility list of an itemset, sum of all the Iutils and Rutils in the utility list is calculated .This sum is the key information to decide whether the itemset should be pruned or not. Itemsets having lesser utility value than the MT will be pruned away .Only the itemsets with greater utility value than MT will be used for finding next level Utility lists. This is given in Algorithm 2.

Algorithm 2: *RHUI-Miner* Algorithm

Input: *R.UL*, the utility list of itemset *R*, and initially

empty

*UL*s, set of utility list of all *R*'s 1-itemsets

Min_Util, Initial minimum utility threshold

v, User defined increase in *Min_Util* at every level

Output: *RHUI*s, all related high utility itemsets with *R* as

Prefix

```
1.   for each utility list L in ULs do
2.        Find the sum of iutils and if sum ≥
          Min_Util then
3.             List the extensions of L
4.   end
5.        if SUM(L.iutils)+SUM(L.rutils)≥Min_Util then
6.             exULs = NULL;
7.             for each utility list Y after L in ULs do
8.                Construct utility list and add with exULs
9.             end
10.            RHUI-Miner(L, exULs, Min_Util + v);
11.       end
12.  end
```

Repeat the same procedure as above to mine all Related High Utility Itemsets (RHUIs) . In running example, number of HUIs is only Six <E, G, A, D, GA, AD> when we apply existing approach. Where as number of RHUIs is sixteen <C, E, B, A, G, D, CE, CB, EB, EA, BA, BG, AG, BD, AD, GD> only in first two levels and the count will get increased when we continue further levels. At this point we have an new and important problem to get resolved .That is number of HUIs generated by the new algorithm. As the total utility in utility lists are getting increased, number of generated HUIs will also get increased. To avoid that, we can increase minimum utility threshold MT at every level by users preference[15]. When MT is increased in level 2 by 5% or 10 % or 15% or 20 %, number of RHUIs will be 10, 8, 6 or 3.

5. CONCLUSION

This paper proposes a new technique to mine all Related High Utility Itemsets (RHUIs) from large databases in lesser time. This method uses a novel approach of merging transactions in database based on their transaction utility so as to reduce database size. As the number of transactions are very much reduced, the time spent on the process of mining RHUIs will also be reduced. Even though the mined RHUIs are not exactly same as the list of HUIs, the profitability of RHUIs are more or less equal to HUIs. When we consider the execution time of both algorithms, RHUI-Miner effectively works on it. This is explained above with example set of transactions .This RHUI-Miner algorithm will outperform existing HUI-Miner algorithm on real and synthetic datasets.

REFERENCES

[1] Agrawal, R., Imielinski, T., and Swami, " Mining association rules between sets of items in large databases". In Proc. of 1993 ACM SIGMOD International Conf on Management of Data, P. Buneman and S. Jajodia, Eds. Washington, 207-216.

[2] Agrawal R. and Srikant R., "Fast algorithms for mining association rules". In Proc. of the 20th International Conf. on, VLDB "94, pp. 487-499

[3] 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.

[4] Jiawei Han, Yongjian Fu, "Mining Multiple-Level Association Rules in Large Databases", TKDE '99.

[5] Jiawei Han, Jian Pei, and Yiwen Yin ."Mining Frequent Patterns without Candidate Generation" In Proc. of SIGMOD'2000 Paper ID: 196.

[6] C.-W. Lin, T.-P. Hong, and W.-H. Lu, "An effective tree structure for mining high utility itemsets," In Proc. of Expert Systems. Appl., Vol. 38, No. 6, pp. 7419–7424, 2011.

[7] M. Klemettinen, H. Mannila, P. Ronkainen, H. Toivonen, and A.I. Verkamo. "Finding interesting rules from large sets of discovered association rules". In Proc. of the 3rd International Conf. on Information and Knowledge Management, pages 401-408, Nov. 1994.

[8] M.C. Tseng and W.Y. Lin, "Mining generalized association rules with multiple minimum supports". Proc. of International Conf. on Data Warehousing and Knowledge Discovery (2001) 11-20.

[9] B. Le, H. Nguyen, T. A. Cao, and B. Vo, "A novel algorithm for mining high utility itemsets," in Proceedings of 1st Asian Conference Intelligent Information Database Syst., 2009, pp. 13–17.

[10] Y. Liu, W. Liao, and A. Choudhary, "A fast high utility itemsets mining algorithm," in Proc. Utility-Based Data Mining Workshop, 2005, pp. 90–99.

[11] Shichao, Z. & Jingli, Lu. Chengqi, Z. "A fuzzy logic based method to acquire user threshold of minimum-support for mining association rules", Information Sciences, 1-15, 2003.

[12] 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," in Proceedings of IEEE International Confeference on Data Mining, 2008, pp. 881–886.

[13] Vincent S. Tseng, Cheng-Wei Wu, Bai-En Shie, and Philip S. Yu." UP-Growth: An Efficient Algorithm for High Utility Itemset Mining", In Proceedings of KDD'10, July 25–28, 2010.

[14] R. Chan, Q. Yang, and Y. Shen, "Mining high utility itemsets," in Proceedings of IEEE International Confeference on Data Mining ., 2003, pp. 19–26.

[15] Wen-Yang Lin, Ming-Cheng Tseng, "Automated support specification for efficient mining of interesting association rules," Journal of Information Science., 2005.

[16] Souleymane Zida, Philippe Fournier-Viger, Jerry Chun-Wei Lin, Cheng-Wei Wu, Vincent S. Tseng, "EFIM: A Highly Efficient Algorithm for High-Utility Itemset Mining".

[17] Bing Liu, Wynne Hsu and Yiming Ma, "Mining Association Rules with Multiple Minimum Supports", KDD-99.

[18] Mengchi Li and Junfeng Qu, "Mining High Utility Itemsets without Candidate Generation", in the proceedings of CIKM'12, Maui, HI, USA.

[19] Feng Tao, Fionn Murtagh, Mohsen Farid, "Weighted Association Rule Mining using weighted support and significance framework", KDD'03.

[20] Philippe Fournier-Viger, Souleymane Zida, Cheng-Wei Wu, Vincent S. Tseng, "FHM: Faster High-Utility Itemset Mining using Estimated Utility Co-occurrence Pruning", ISMIS' 14, Springer, LNAI, pp 83-9.