International Journal of Database Theory and Application
Vol.10, No.3 (2017), pp.31-44
http://dx.doi.org/10.14257/ijdta.2017.10.3.03

Mining High-Utility Itemsets Based on Multiple Minimum Support and Multiple Minimum Utility Thresholds

Md Fazla Elahe and Kun Zhang

School of Computer Science and Engineering, Nanjing University of Science and Technology, China
lotus_xclusive@yahoo.com, zhangkun@njust.edu.cn

Abstract

Mining high utility itemsets from a transactional database refer to the discovery of high utility itemsets that generate high profit and several approaches have been proposed for this task in recent years. Algorithms like HUIM-MMU and MHU-Growth overcome the limitation of using a single threshold for the whole database. However, they still generate a large number of candidate itemsets and thus it degrades the performance of the algorithms. In this paper, we address this issue by combining two different kinds of thresholds used by HUIM-MMU and MHU-Growth. By using these two thresholds we propose two algorithms namely HUIM-MMSU and HUIM-IMMSU. HUIM-MMSU is a candidate generation and retest based algorithm, which relies on sorted downward closure (SDC) property. On the other hand, HUIM-IMMSU uses a tree-like data structure. Experiment result shows that the proposed two algorithms can effectively discover high utility itemsets from the transactional database.

Keywords*: High utility itemset (HUI); Multiple minimum support; Multiple minimum utility*

1. Introduction

Among many of the data mining research topics, Frequent Itemsets Mining (FIM) [1], [2] is one of the fundamentals. The most popular application of Frequent Itemsets Mining (FIM) is market basket analysis which refers to mine the itemsets that have bought together in a frequency greater than specified by the user. The traditional frequent pattern mining approaches treat all items with the same importance and it assumes that every item in a transaction is binary (whether it is present or absent in a transaction ignoring the number of occurrence in the transaction) but in real world applications, the frequency of a pattern may not be the only indicator of meaningful pattern. The reason behind it is an item in a transaction can have a different degree of importance (Weight). Several weighted rule mining technique [3,4] has been proposed to solve this problem. However, the weighted rule mining technique only considers weights of an item such as unit profit for a transaction database. Therefore it does not satisfy user requirement since the profit composed with the purchase quantity for a transaction database and the weighted rule mining does not consider purchase quantity.

To solve the above limitation of weighted association rule mining, utility mining [5]–[14] was proposed as an interesting expansion of frequent itemsets mining and weighted association rule mining. In real life, utility value can be labeled as profit, cost and preference value from the user. In a word, utility means how useful an itemset is. Since the utility mining had proposed, high utility (e.g: profits) itemsets mining become an important topic in data mining. High utility itemsets mining refers to mine itemsets those have high utility. For this paper, we will mainly focus on mining high utility itemsets. There are a lot of real life applications of high utility itemsets mining [10], [12], [15]–[17]. For example, somebody may be

ISSN: 2005-4270 IJDTA

interested in finding itemsets that generate large revenue or somebody wants to make a decision based on the high value of costs.

The problem of mining all the high utility itemsets efficiently is much more complex because the utility of an itemset is neither monotonic nor anti-monotonic, that means high utility itemset may have a superset or subset with a low, equal to high utility value [1]. The widely used downward-closure for frequent itemsets mining states that the support of an item is anti-monotonic, that means subsets of a frequent itemset are frequent and supersets of an infrequent itemset are infrequent. Downward-closure property is very powerful to prune the infrequent itemsets and reduce the search space but we cannot apply this technique directly for mining high utility itemsets.

The problem of mining high utility itemsets has been carried out by many studies [13], [15]–[18]. Mining high utility itemsets in two phases using transaction weighted downward-closure property is the most popular approach. The algorithms that use this technique are Two-phase [19], IHUP [11], UP-Growth [7] and so on. In the first phase, these algorithms generate potential high utility itemsets by overestimating their utility. Then in the second phase, it calculates the exact utility and prunes the low utility itemsets. A more efficient algorithm was proposed named HUI-Miner [20] that mine high utility itemsets in a single phase and it does not require candidate generation. HUIM-MMU [21] and MHU-Growth [22] algorithms use a new technique for discovering high utility itemsets. HUIM-MMU uses multiple minimum utility thresholds to reduce the search space and on the other hand, MHU-Growth uses multiple minimum support thresholds. However, there is a fundamental gap between these two algorithms. Therefore it is still challenging to design an efficient algorithm for mining high utility itemsets.

Based on the observation we propose two algorithms named HUIM-MMSU and HUIM-IMMSU. Both the algorithms use two different kinds of thresholds: multiple minimum support threshold and multiple minimum utility thresholds. By using these two thresholds search space is reduced so as the execution time and cost. HUIM-MMSU uses candidate generation and retest based technique and relies on the sorted downward-closure property, on the other hand, HUIM-IMMSU uses a novel tree structure. We compare the performance of HUIM-MMSU and HUIM-IMMSU using real life dataset. Results show that HUIM-IMMSU performs better in comparison with HUIM-MMSU in terms of execution time and memory consumption.

The remainder of this paper is organized as follows. In section II we will introduce preliminary and related work. Algorithm HUIM-MMSU and HUIM-IMMSU are presented in section III. Experiments are shown in section IV and conclusions are given in section V.

2. Background

This section is divided into two parts. In the first part, we define some definitions and discuss the problem of utility mining. In the second part, we introduce related work.

2.1. Preliminary

Let I be a finite set of items where I= {i1, i2, i3,….., im}, each item ip($1 \le p \le m$) has a unit profit pr(ip). A transaction database D= {T1, T2, T3, ….., Tn} where Tr $\in$ D ($1 \le r \le n$) is a subset of I and has a unique identifier r called Tid. Each item ip in transaction Tr is associated with a quantity q(ip, Tr), that is the purchased quantity of item ip in transaction Tr.

Definition 1(utility of an item/itemset in a transaction): utility u of an item ip in a transaction Tr is denoted by u(ip, Tr) and defined as pr(ip) × q(ip, Tr). The utility u of an

itemset X, where X is a group of items and X ⊆ I in a transaction Tr is denoted by u(X, Tr) and defined as $\sum_{ip \in X} u(ip, Tr)$. For example, the utility of an item a in T1 is u(a, T1)= 2×2 = 4 and utility of an itemset {a, d} in T1 is u({a, d}, T1) = 2×2+1×3=7 (from table 1 and 2).

Table 1. Transaction Database

TID	Transaction (item, Quantity)	Transaction Utility (TU)
T_1	a:2, d:3, e:1	11
T_2	b:1, c:2, d:1	14
T_3	a:3, b:2	12
T_4	a:1, f:1	3
T_5	b:1, d:3	6
T_6	b:2, d:2, e:1	12
T_7	a:2, b:3, c:2	23
T_8	b:2, c:3	21
T_9	a:1, f:4	6
T_{10}	a:2, c:2	14

Table 2. Property of Items

Item	Profit	MMU	MMS	Support	Utility	TWU
a	2	12	4	6	22	69
b	3	20	3	6	33	88
c	5	25	3	4	45	72
d	1	10	2	4	9	43
e	4	20	2	2	8	23
f	1	15	2	2	5	9

Definition 2 (utility of an itemset in a database): utility of an itemset X in database D is denoted by u(X) and defined as $\sum_{X \in Tr \wedge Tr \in D} u(X, Tr)$. For example utility of an itemset {a, c} in database D is u({a, c})= u({a, c}, T7) + u({a, c}, T10) = 2×2+5×2+2×2+5×2=28 (from table 1 and 2).

Definition 3 (transaction utility of a transaction): transaction utility (TU) of a transaction Tr is denoted by TU(Tr) and defined by $\sum_{ip \in Tr} u(ip, Tr)$. For example transaction utility of T1 is TU(T1)= u({a, d, e}, T1) = 11 (from table 1 and 2).

Definition 4 (minimum support of an itemset): If an itemset X consists of items such that X={i1, i2, i3, …, ik}then minimum support if itemset X refers to least MIS values of items in X. Minimum support of itemset X is denoted by MIS(X) and defined as min[MIS(i1), MIS(i2), MIS(i3),….. MIS(ik)] where ip∈ X and 1≤p≤k. For example minimum support of an itemset {a, b, c} is MIS({a, b, c })= 3 (from table 1 and 2).

Definition 5 (minimum utility of an itemset):): If an itemset X consists of items such that X={i1, i2, i3, …, ik} then minimum utility of itemset X refers to least MU values of items in X. Minimum utility of itemset X is denoted by MU(X) and defined as min[MU(i1), MU(i2), MU(i3), ….. MU(ik)] where ip∈ X and 1≤p≤k. For example minimum utility of an itemset {a, b, c} is MU({a, b, c })= 16 (from table 1 and 2).

Definition 6 (high utility itemset): If the support of an itemset X is no less than the minimum support of X and utility of itemset X is greater than equal to the minimum utility of X then itemset X is called high utility itemset. For example itemset {b, c} is an high utility itemset because SUP({b, c})=3 which is equal to MIS({b, c}) and U({b, c})= 53 which is greater than MU({b,c}) (from table 1 and 2).

Definition 7 (transaction weighted utility): Transaction weighted utility of an itemset X

refers to the sum of all the transaction utilities that contain X. Transaction weighted utility of itemset X is denoted by TWU(X) and defined as $\sum_{X \in T_r \wedge T_r \in D} TU(T_r)$. For example, transaction weighted utility of a is TWU(c) = TU(T2)+ TU(T7)+ TU(T8)+ TU(T10) (from table 1).

2.2. Related Work

Before the problem of mining high utility itemset mining was proposed formally, extensive studies have been proposed for mining frequent itemsets. Apriori [1] was the first well-known frequent itemsets mining algorithm which relies on a property called downward-closure property [1]. A more efficient frequent itemsets mining algorithm named Fp-Growth [2] was then proposed. Fp-Growth uses tree-like data structure and it does not require to generate candidates to mine frequent itemsets. The rest of frequent itemsets mining algorithms are either based on Apriori or Fp-Growth.

Considering the importance of items to the user, weighted association rule mining [3,4] was proposed. Since the proposal of weighted association rule, a lot of techniques have been proposed by researchers. By considering the non-binary transaction of items utility mining [5]–[14] was then proposed and become a significant research topic in data mining. Two phase algorithm [19] is proposed by Liu et al. which composed of two mining phases. IHUP [11] was then proposed by Ahmed et al. to efficiently mine high utility itemsets and it uses a tree-like data structure. Some other widely studied high utility itemsets mining algorithms are HUP-tree [6] by Lin et al. UP-growth and UP-growth+ [7] by Tseng et al. MHU-Growth [22] for mining high utility itemsets with multiple minimum support was first proposed by Ryanga et al. HUIM-MMU [21] for Mining high utility itemsets with multiple minimum utility thresholds was then proposed by Lin et al. Our study aims to remove the fundamental research gap between MHU-Growth and HUIM-MMU and use multiple minimum support and multiple minimum utility thresholds to efficiently discover all high utility itemsets.

3. Proposed Algorithms

In this section, we first define some common definitions that are used in both HUIM-MMSU and HUIM-IMMSU. Then we discuss the pruning techniques. At last, we propose HUIM-MMSU and HUIM-IMMSU algorithm.

Definition 8 (high transaction weighted utilization itemset): High transaction weighted utilization is denoted by HTWUI and defined by HTWUI ←{X|TWU(X) ≥MU(X)}. For example {b,c} is a HTWUI because its TWU is 58 which is greater than its minimum utility of the itemset.

Definition 9 (least minimum utility): Least minimum utility is denoted by LMU and defined by LMU = min{MU(i1), MU(i2),….. MU(im)}, where m is the total number of items in MMU table. For example, LMU of items in table 1 is 10 (from table 2).

Definition 10 (least minimum support): Least minimum support is denoted by LMS and defined by LMS = min{MIS(i1), MIS(i2),….. MIS(im)}, where m is the total number of items in MMU table. LMS of items in table 1 is 2 (from table 2).

There are five pruning conditions used by both of our proposed algorithms. These pruning conditions are used to prune unwanted itemsets and reduce the search space to make the algorithms more efficient.

Pruning condition 1: (TWU(i_j) < LMU) Transaction weighted utility of an item is less than the least minimum utility. For example, LMU of transactions in Table 1 is 10 so item f will be pruned because its TWU is 9 which is less than 10. It means that item f is unable to generate any high utility itemsets so thus it will be pruned.

Pruning condition 2: (SUP(i_j) < LMS) Support of an item is less than least minimum

support. Assume that LMS of transactions in Table 1 is 3 so items e and f will be pruned because of less support than LMS. It means that items e and f are unable to generate any high utility itemsets so thus they will be pruned.

Pruning condition 3: (TWU(X) < MU (X)) Transaction weighted utility of an itemset is less than the minimum utility of that itemset. For example, If TWU of itemset X is 15 and MU of X is 20 then X will be pruned.

Pruning condition 4: (U(X) < MU (X)) Utility of an itemset is less than the minimum utility of that itemset. For example, If utility of an itemset X is 15 and its MU is 20 then X will be pruned.

Pruning condition 5: (SUP(X) < MIS (X)) Support of an itemset is less than the minimum support of that itemset. For example, if support of an itemset X is 3 and its MIS is 5 then X will be pruned.

3.1. Proposed HUIM-MMSU Algorithm

As we discussed earlier that the famous pruning technique for frequent itemsets mining, downward closure property cannot be applied for mining high utility itemsets. Transaction weighted downward closure property (TWDC) was introduced by Liu et al. [19] for pruning the search space to get better performance. Lin et al. [21] have shown that TWU is not downward-closed in some cases. To address this issue Lin et al. [21] proposed a new technique called sorted downward closure property (SDC property). According to downward closure property if we sort items in ascending order of their MU values then the subset of a HTWUI is also a HTWUI. Thus the sorted downward closure property guarantees anti-monotonicity for HTWUI. But if we use sorted downward closure property directly to produce 1-itemsets HTWUI then some items may miss mining. Instead of using SDC to produce 1-itemsets HTWUI we use least minimum utility (LMU) and least minimum support (LMS) to get the complete set of high utility itemsets.

Our proposed HUIM-MMSU algorithm generates the complete set of high utility itemsets in two phases. In the first phase, HTWUIs are identified by generating and retest based technique. If TWU of an itemset is greater than or equal to minimum utility (MU) of that itemset then the itemset is a HTWUI and rest of the itemsets are pruned because only the HTWUIs are able to generate HUI. In the second phase, an additional database scan is required to get the exact utility of HTWUIs. The algorithm then checks whether the utility and support values of HTWUIs are greater than or equal to MU and MIS values of those itemsets. If it is then the algorithm store those itemsets as HUI and the rest of the itemsets are pruned. The pseudo-code of the algorithm is given in algorithm 1.

Algorithm 1: HUIM-MMSU

Input: D- a transactional database, an external utility table, MMU- user-specified multiple minimum utility threshold table, MMS- user-specified multiple minimum support threshold table

Output: The set of complete high-utility itemsets.

1. Calculate LMU from MMU- table and LMS from MMS-table;
2. Scan D to calculate transaction weighted utility, transaction weighted utility of an item is the sum of transaction utility that contains that item according to the definition we can get the TWU by using the following formula;
   $TWU(i_j) = \sum_{i_j \in T_r \wedge T_r \in D} TU(T_r)$;
3. **for** each $i_j \in D$ **do**
4. **if** $TWU(i_j) \geq LMU$ && $SUP(i_j) \geq LMS$ **then**……(according to the pruning condition 1 and 2)

5. $HTWUI^1 \leftarrow HTWUI^1 \cup i_{j;}$
6. sort $HTWUI^1$ in ascending order of mu values;
7. set K←2;
8. **While** $HTWUI^{k-1} \neq$ null **do**
9. C_k = generate-candidate($HTWUI^{k-1}$);
10. **for** each K itemset X in C_k **do**
11. scan D to calculate TWU(X);
12. **if** TWU(X) ≥ MU(X) **then** ……(according to the pruning condition 3)
13. $HTWUI^k \leftarrow HTWUI^k \cup X$;
14. set K ←K+1;
15. HTWUIs ←U $HTWUI^k$;
16. **for** each itemset X in HTWUIs **do**
17. scan D to calculate U(X) and SUP(X);
18. **if** U(x) ≥ MU(X) && SUP(X) ≥ MIS(X) **then**……(according to the pruning condition 4 and 5)
19. HUIs← HUIs U X;
20. **return** HUIs;

The HUIM-MMSU algorithm first calculates LMU from MMU-table and LMS from MMS-table. Then it scans the database to get TWU and support of each item. Pruning condition 1 and 2 are applied on line 4. By applying these two pruning conditions we get HTWUI1 on line 5. According to the previous discussion, we then order $HTWUI^1$ in ascending order of their mu values. From line 8 to line 14 candidates of different levels are generated and $HTWUI^k$ is identified by the by the pruning condition 3. Then we scan the database for the last time to get exact utility and support information of HTWUIs. Pruning condition 4 and 5 are then applied to get HUIs on line 18 and 19.

3.2. Proposed HUIM-IMMSU Algorithm

Our proposed HUIM-IMMSU algorithm discovers all the high utility itemsets in three steps. In the first step, a global tree is constructed from the database. In the second step, some pruning techniques are used to prune unwanted items and restructure the global tree. Finally, the mining technique comes to the action.

First step: IMMSU-tree is constructed by a single scan and used to maintain the information about transactions and some other important information about the items such as minimum utility, minimum item support. Each node N_i in IMMSU-tree consists of several elements such as $N_{i.name}$ is for storing the name of the item, $N_{i.MIS}$ for storing the minimum item support of the item, $N_{i.MU}$ for storing minimum utility of the item, $N_{i.count}$ is for support count of the node, $N_{i.nu}$ for TWU, $N_{i.parent}$ to store parent info of the node and $N_{i.nodelink}$ for storing the information about the node that has the same name as $N_{i.name}$. To facilitate tree traversal in restructure and mining process IMMSU-tree also includes a header table consisting of the item name, utility value, TWU value, support count and a node link. To insert transaction from the database a function named Insert_Transaction(Tree T, MMU Table, MMS Table) is called. The pseudo code for IMMSU-tree construction is given in Algorithm 2. The fully constructed IMMSU-tree for the transaction database Table I is given in figure 1.

Algorithm 2: IMMSU-tree

Insert_Transaction (Tree T, MMU Table, MMS Table)

1. N←the root node of Tree
2. **reorder** items in T by Support descending order
3. **calculate** transaction utility of T, TU(T)

4. **for** each item i in transaction T
5. **if** N has not a child N_i such that $N_{i.name}$=i
6. **create** a new child N_i under N where $N_{i.name}$=i, $N_{i.MIS}$ = MIS(i), $N_{i.MU}$=MU(i)
7. **increase** $N_{i.count}$ by 1, $N_{i.nu}$ by TU(T)
8. **if** header has not an entry for i
9. **create** a new entry E_i, where $E_{i.nu}$ = 0 , $E_{i.count}$ =0, $E_{i.MIS}$=MIS(i), $E_{i.MU}$ = MU(i)
10. **increase** $E_{i.nu}$ by TU(T), $E_{i.count}$ by 1
11. N←N_i

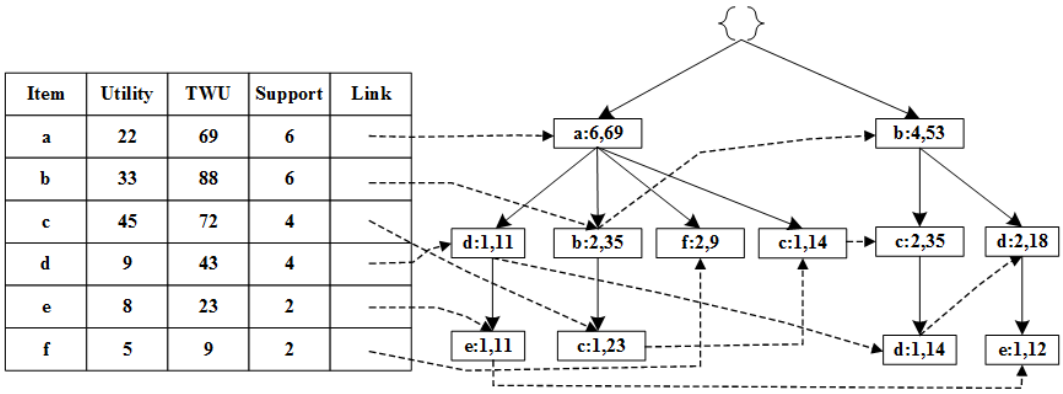

Figure 1. IMMSU Tree

Second step: Once the IMMSU-tree is constructed it is ready for restructuring by pruning unimportant items and reduce the search space. Pruning conditions 1 and 2 are applied in the second step. The pseudo code of the second step is given in algorithm 3. It consists of three functions: Restructure_Tree, Prune, and Merge. If TWU of any item is less than LMU or support of an item is less than LMS then Prune function is called from the Restructure_Tree function. To remove all the items that satisfy the pruning condition, link and node link are used. If node N_i where $N_{i.name}$ =i is removed from the IMMSU-tree if N_{child} is the child node of Ni then it is necessary to change $N_{child.parent}$ to N_{parent}. If N_{parent} already has a child node $N_{j.child}$ such that $N_{j.child.name}$ = $N_{child.name}$ then need to be merged. If the nodes are merged then node utility and support of $N_{j.child}$ are increased by the corresponding value of N_{child}. The restructured IMMSU-tree is given in figure 2.

Algorithm 3: Restructure IMMSU-tree

Restructure_Tree(Tree, MMU Tree, MMS Table)

1. **calculate** LMU from MMU table
2. **calculate** LMS from MMS table
3. **for each** entry of an item i in header of Tree /*Bottom Up*/
4. **if** TWU(i) < LMU then **call** prune (Tree, i)…(according to the pruning condition 1)
5. **else if** SUP(i) < LMS then **call** prune (Tree, i)..(according to the pruning condition 2)

Prune(Tree, i)

1. **for each** node N_i in tree such that $N_{i.name}$=i
2. **if** N_i has children then **call** Merge($N_{i.parent}$, $N_{i.children}$)
3. **remove** N_i from Tree

Merge(Parent, Children)

1. **for each** child node C in Children
2. **if** Parent has a child N_{child} such that $N_{child.name}$ =

$C._{name}$

3. **increase** $N_{child.count}$ by $C._{count}$, $N_{child.nu}$ by $C._{nu}$
4. **call** Merge(N_{child}, $C._{children}$)
5. **else** $C._{parent}$ ← Parent

Third step: The last step of HUIM-IMMSU algorithm is used to mine high utility itemsets by bottom-up traversal of restructured IMMSU-tree. The header table is used to follow the node link for mining all high utility itemsets efficiently. The mining process starts from the bottommost item in the header table and for each item, a conditional pattern base is created by extracting all paths from the item to root. Support and TWU of

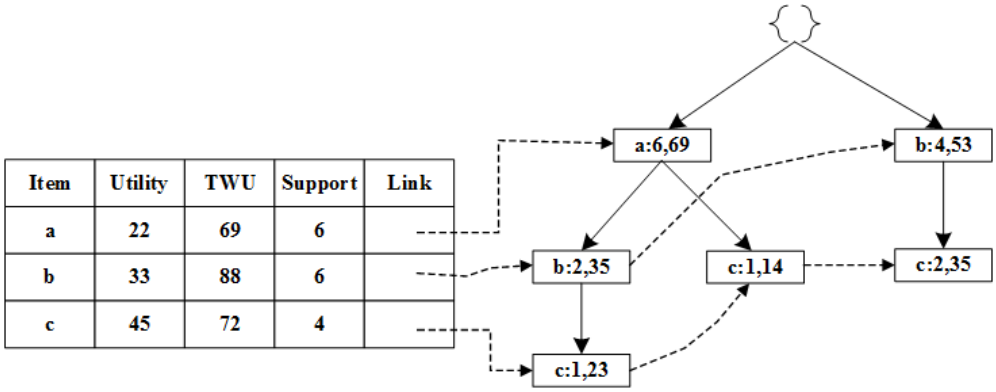

Figure 2. Restructured IMMSU tree

each item in the extracted conditional tree is also calculated. The pseudo code of the third step of HUIM-IMMSU algorithm is given in algorithm 4. Figure 3 describes the mining process of the last item of header table.

Algorithm 4: IMMSU-Growth

IMMSU-Growth (Tree)

1. **for each** entry of an item i in header of tree /*Bottom up*/
2. **for each** node N_i such that $N_{i.name}$ = i in the tree
3. **extract** a path p consists of nodes from root node to N_i such that each node $N_{ik.MIS} \geq N_{i.MIS}$ /*MIS of every item should be greater than equal to MIS of the itemsets */
4. **for each** item node N_{ik} in the extracted path p
5. **accumulate** $N_{ik.count}$ to support, $N_{ik.nu}$ to TWU, $N_{ik.MU}$ to MU and $N_{ik.MIS}$ to MIS
6. **if** TWU(i) ≥ MU(X) then **generate** {i} /* X is the nodes in the extracted path p */……(according to the pruning condition 3)
7. **create** Prefix tree for i, T_i
8. **prune** items from T_i such that U < MU(X) and SUP < MIS(X) /* X is the nodes in the extracted path p */ …..(according to the pruning condition 4 and 5)
9. **call** IMMSU-Growth (T_i, {i}, MIS, MU)

IMMSU-Growth (T_x, X, MIS, MU)

1. **for each** entry of an item i in header T_x /*Bottom up*/
2. **for each** node N_i such that $N_{i.name}$=i in T_x
3. **extract** a path p consists of nodes from root node to N_i such that each node $N_{ik.MIS} \geq N_{i.MIS}$ /*MIS of every item should be greater than equal to MIS of the itemsets */

4. **for each** item node N_{ik} in the extracted path p
5. **accumulate** $N_{ik\cdot count}$ to support, $N_{ik.nu}$ to TWU, $N_{ik\cdot MIU}$ to MU and $N_{ik\cdot MIS}$ to MIS
6. **If** Sup(i) ≥MIS and TWU (i) ≥ MU then generate X U {i}
7. X←X U {i}
8. **create** prefix tree for {X}, Tx
9. **prune** items from Tx such that U < MU(X) and SUP < MIS(X)
10. **call** IMMSU-growth(Ti, {i}, MIS, MU)

According to the pruning condition 1 and 2 items d, e, f are pruned from IMMSU-tree shown in figure 1 and the fully restructured tree is given in figure 2. As the mining process progresses conditional pattern base for c is given in Table 3. It is seen from Table 3 that there are three paths that satisfy initial conditions to construct prefix tree for c (Figure .3).

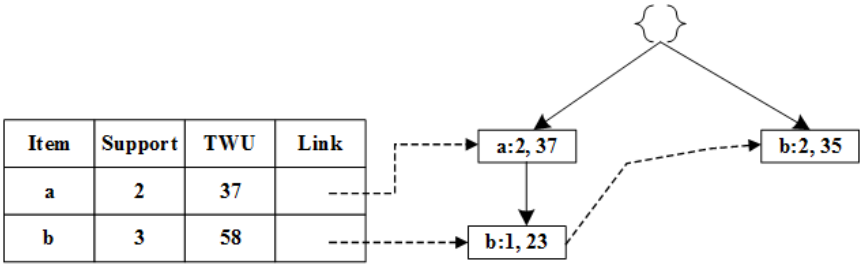

Figure 3. Prefix Tree for c

Although item A has greater utility value than its minimum utility value but according to the pruning condition 5, it is pruned from the prefix tree of c. On the contrary, item b is not pruned as it has greater support and utility value than its minimum support and utility value.

Table 3. Conditional Pattern base for C

Path	Support	Utility
<a,b>	1	23
<a>	1	14
<b>	2	35

The conditional tree for C is given in the figure 4.

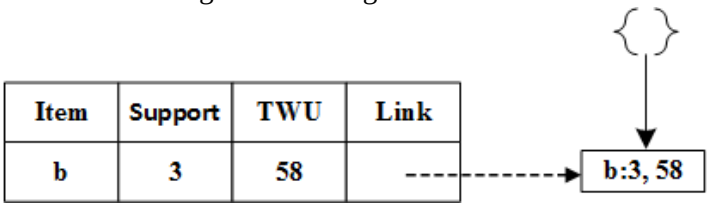

Figure 4. Conditional Tree for c

4. Experimental Results

The Performance of our proposed algorithms is evaluated in this section. Experiments were performed on a 1.80 GHz Intel core i3 processor with 4 gigabytes of main memory and running on windows 8.1. Java language is used to implement

the algorithms. No other studies have been done on the topic of high utility itemsets mining that uses multiple minimum support and multiple minimum utility thresholds at the same time. MHU-Growth and HUI-MMU are used to verify the effectiveness of our proposed algorithm which can provide the benchmark.

A real life dataset named retail is used in the experiment to validate the effectiveness of the proposed algorithms. The retail dataset is about product sales information in a retail store. The characteristics of the dataset are given in the table. A uniform distribution in [1, 10] is used to generate the internal utility values. A Gaussian (normal) distribution is used to generate the external utility values.

$$MIS(i) = Max[\beta \times Sup(i), LS] \quad (1)$$

$$MU(i) = Max[\alpha \times pr(i), GLMU] \quad (2)$$

Table 4. Characteristics of Retail Dataset

Dataset	Transactions	Items	Avg. length	Max. Length	Dense/sparse ratio
retail	88,162	16,470	10.3	76	0.625

Furthermore, based on the discussion, we assign MIS and MU values to each item using the equation 1 and equation 2. In the equation 1, the parameter β is used to control how the MIS values are related to their frequencies where $0 \leq \beta \leq 1$. If $\beta=0$ then a single MIS value that is LS is assigned to every item. In the equation 2, pr(i) refers to the external utility of item i and to ensure the randomness of MU values the value of α set to [1000, 10000] for the retail dataset. If we set $\alpha=0$ then a single MU value that is GLMU is assigned to each item.

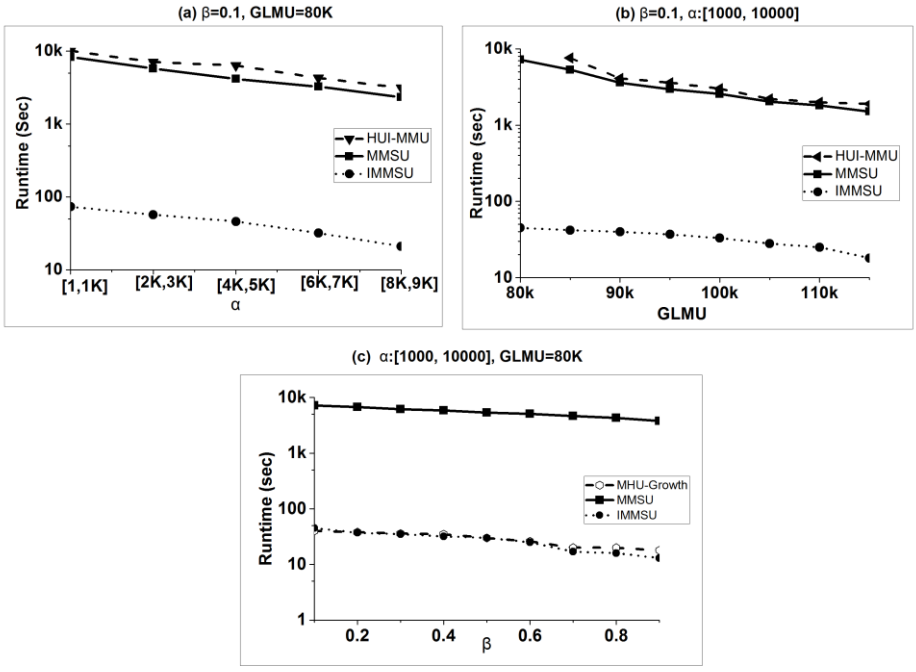

Figure 5. Runtime of Algorithms

4.1. Runtime Analysis

The Performance of HUIM-MMSU and HUIM-IMMSU in terms of runtime are compared in this part. Figure 5 shows the experimental results on the retail dataset and least minimum support (LS) set to 200 in this experiment. Runtimes of algorithms are compared within various interval under fixed β, GLMU (figure 5(a)), various GLMU under fixed β and fixed interval (figure 5(b)), various β under fixed interval and GLMU (figure 5(c)). It is seen that runtime decreases with the increase of the value of parameter α, β and GLMU (figure 5). It can be observed from Figure 5(a) and 5(b) that HUIM-IMMSU outperforms HUIM-MMSU and HUI-MMU. Because HUIM-IMMSU algorithm uses tree-like data structure and it does not require to generate candidates while the other two generate candidates. It is also seen that HUIM-MMSU requires less time than HUI-MMU. The region behind that HUIM-MMSU uses some extra pruning conditions to reduce the search space and a number of generated itemsets. From Figure 5(c) it can be seen that MHU-Growth and HUIM-IMMSU require the same execution time.

4.2. Generated Itemsets Analysis

In this experiment, number of high utility itemsets generated by algorithms are compared. Figure 6 shows the experimental results on the retail dataset. Least support (LS) is set to a value of 200. As we are using same pruning conditions, number of generated high utility itemsets for HUIM-MMSU and HUIM-IMMSU are same. From the figure 6, it is seen that number of HUIs decreases with the increase of the values of α, β, and GLMU. It is also seen that number of HUIs generated by HUIM-MMSU is less than the baseline algorithm HUI-MMU and MHU-Growth. The region behind it is HUIM-MMSU uses some extra pruning conditions in compare to HUI-MMU and MHU-Growth. In real world applications traditional algorithms for mining high utility itemsets may suffer from “rare item problem” that is if minimum utility threshold is set to a low value then a lot of useless itemsets will generate and if it is set to high value then many important itemsets may miss mining. Our proposed algorithms can avoid the “rare item problem” by setting the parameter values based on the actual requirement. Thus the proposed algorithms can be acceptable in the real-world application.

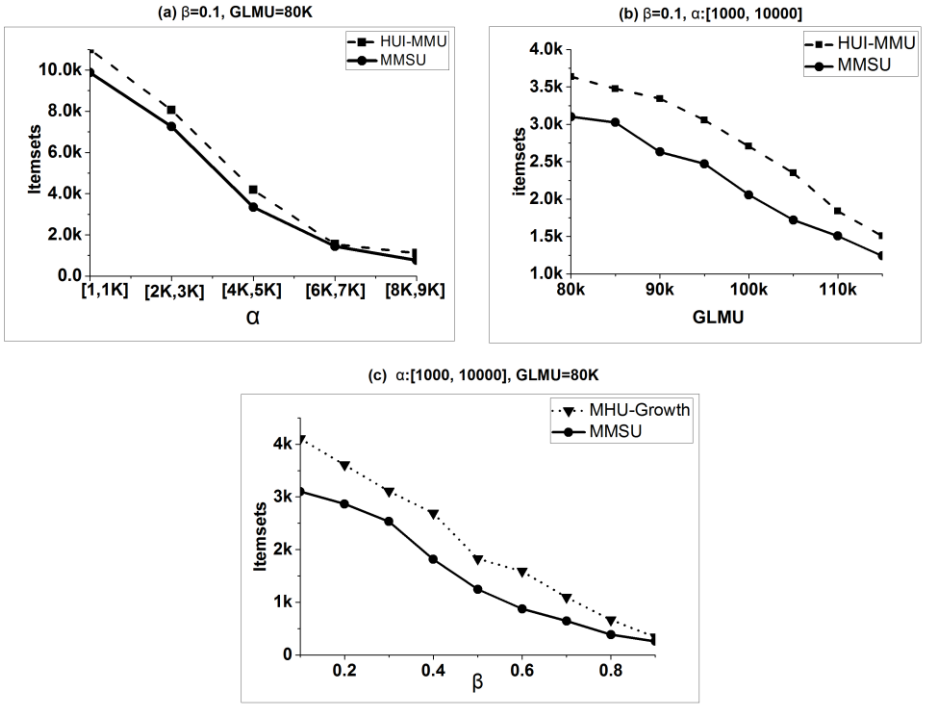

Figure 6. Number of Generated Itemsets by the Algorithms

4.3. Memory Usage Analysis

We compared the memory usage of proposed algorithms with a baseline HUI-MHU while α and GLMU are variable and MHU-Growth while β is variable. It can be seen from figure 7(a) and 7(c) that HUI-MMU and HUIM-MMSU algorithms use

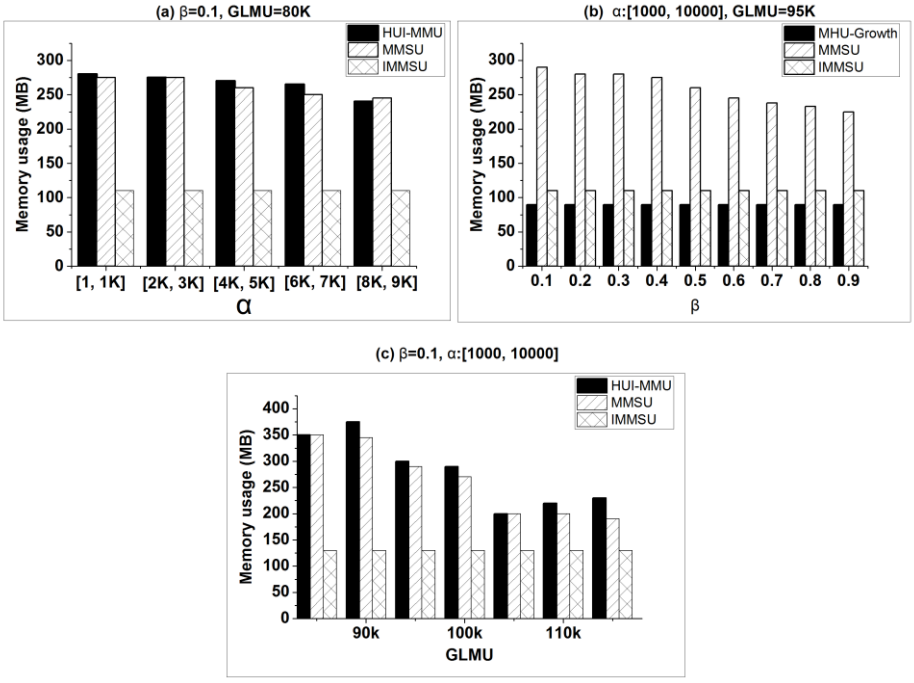

Figure 7. Memory Usage by Algorithms

almost same memory because both of the algorithms uses the same kind of data structure but HUIM-IMMSU uses less memory because it does not generate candidates and uses a tree-like data structure. Although MHU-Growth and HUIM-IMMSU use the same tree-like data structure but HUIM-IMMSU uses more memory because each node in the tree store some extra information in compare to MHU-Growth.

5. Conclusion

In this paper, we proposed two efficient algorithms named HUIM-MMSU and HUIM-IMMSU for mining high utility itemsets. For mining high utility itemsets efficiently and reduce the search space we use five pruning conditions. HUIM-MMSU is a candidate generation and retest based algorithm which relies on sorted downward closure property and HUIM-IMMSU uses a tree-like data structure and it does not need to generate candidates to discover high utility itemsets. Experiments are conducted to compare the effectiveness of both the algorithms and the experiment results show that the proposed algorithms can mine high utility itemsets effectively in terms of runtime, the number of itemsets and memory consumption. Moreover, the proposed algorithms can effectively avoid the 'rare item problem' and give aid to experts to make the correct decision based on the items generated.

References

[1] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules," Proceeding VLDB '94 Proc. 20th Int. Conf. Very Large Data Bases, vol. 1215, **(1994)**, pp. 487–499.
[2] J. Han, J. Pei, and Y. Yin, "Mining frequent patterns without candidate generation," ACM SIGMOD Rec., vol. 29, no. 2, **(2000)**, pp. 1–12.
[3] K. Sun and F. Bai, "Mining Weighted Association Rules without Preassigned Weights," vol. 20, no. 4, **(2008)**, pp. 489–495.
[4] U. Yun and J. J. Leggett, "WIP : mining Weighted Interesting Patterns with a strong weight and / or support affinity," Sixth SIAM Int. Conf. Data Min., **(2006)**, pp. 623–627.
[5] C. W. Lin, G. C. Lan, and T. P. Hong, "An incremental mining algorithm for high utility itemsets," Expert Syst. Appl., vol. 39, no. 8, **(2012)**, pp. 7173–7180.
[6] C. W. Lin, T. P. Hong, and W. H. Lu, "An effective tree structure for mining high utility itemsets," Expert Syst. Appl., vol. 38, no. 6, **(2011)**, pp. 7419–7424.
[7] V. S. Tseng, B. Shie, C. Wu, and P. S. Yu, "Efficient Algorithms for Mining High Utility Itemsets from Transactional Databases," IEEE Trans. Knowl. Data Eng., vol. 25, no. 8, **(2013)**, pp. 1772–1786.
[8] P. Fournier-Viger, C. W. Wu, S. Zida, and V. S. Tseng, "FHM: Faster high-utility itemset mining using estimated utility co-occurrence pruning," Lecture Notes in Computer Science, vol. 8502, **(2014)**, pp. 83–92.
[9] S. Zida, P. Fournier-viger, J. C. Lin, C. Wu, and V. S. Tseng, "EFIM : A Highly Efficient Algorithm for High-Utility Itemset Mining," Lecture Notes in Computer Science, vol. 9413**, (2015)**, pp. 530–546.
[10] V. S. Tseng, C.-W. Wu, B.-E. Shie, and P. S. Yu, "UP-Growth: An Efficient Algorithm for High Utility Itemset Mining," in the 16th ACM SIGKDD international conference, **(2010)**, pp. 253–262.
[11] C. F. Ahmed, S. K. Tanbeer, B. Jeong, and Y. Lee, "Mining High Utility Patterns in Incremental Databases," Proc. 3rd Int. Conf. on Ubiquitous Information Management and Communication, vol. 21, no. 12, **(2009)**, pp. 656–663.
[12] T. Liang and T. Liang, "An efficient algorithm for mining temporal high utility itemsets from data streams," Journal of Systems and Software, Vol. 81 no. 7, **(2008)**, pp. 1105–1117.
[13] Y. Li, "Isolated items discarding strategy for discovering high utility itemsets Isolated items discarding strategy for discovering high utility itemsets," Data and knowledge Engineering, Vol. 64 No. 1, **(2008)**, pp. 198–217.
[14] Y. Wang, J. Yeh, and Y. Wang, "Efficient algorithms for incremental utility mining," Proc. of the 2nd Int. conf. on Ubiquitous information management and communication, **(2008)**, pp. 212-217.
[15] B. Shie, V. S. Tseng, and P. S. Yu, "Online Mining of Temporal Maximal Utility Itemsets from Data Streams." Proc. of the 2010 ACM Symposium on Applied Computing, **(2010)**, pp. 1622-1626.
[16] C. Ss, C. Ss, H. J. Hamilton, and C. Ss, "A Unified Framework for Utility Based Measures for Mining Itemsets," ACM SIGKDD Second Int. Workshop on Utility Based Data Mining, **(2006)**, pp. 28-37.
[17] B. Shie, H. Hsiao, V. S. Tseng, and P. S. Yu, "Mining High Utility Mobile Sequential Patterns in Mobile Commerce Environments." Proc. of the 16th Int. Conf. on Database systems for advanced applications, **(2011)**, pp. 224-238.

[18] V. S. Tseng, C. Wu, B. Shie, and P. S. Yu, "UP-Growth : An Efficient Algorithm for High Utility Itemset Mining," Proc. of the 16th ACM SIGKDD Int. Conf. on Knowledge discovery and data mining, **(2010)**, pp. 253–262.
[19] Y. Liu, W. Liao, and A. Choudhary, "A Two-Phase Algorithm for Fast Discovery of High Utility Itemsets," **(2005)**,pp. 689–695.
[20] M. Liu and J. Qu, "Mining High Utility Itemsets without Candidate Generation," Proc. 21st ACM Int. Conf. Inf. Knowl. Manag., **(2012)**, pp. 55–64.
[21] J. C. Lin, W. Gan, P. Fournier-viger, and T. Hong, "Mining High-Utility Itemsets with Multiple Minimum Utility Thresholds." Proc. of the Eighth Int. Conf. on Computer Science & Software Engineering, **(2015)**, pp. 9-17.
[22] H. Ryang, U. Yun, and K. Ho, "Discovering high utility itemsets with multiple minimum supports," Intelligent Data Analysis, vol. 18, **(2014)**, pp. 1027–1047.

