International Journal of Database Theory and Application
Vol.9, No.8 (2016), pp.233-246
http://dx.doi.org/10.14257/ijdta.2016.9.8.22

Fast Approach for High Temporal Utility Item Mining

Pan Yi[1], Liu Huafu[1] and Zhang Bo[2*]

[1]*Department of Computer Science and Technology, Changsha University, Changsha, 41002*
[2]*Logistics Engineering College, Wuhan University of Technology, Wuhan, 430062*
E-mail: 531129@qq.com

Abstract

High utility itemset mining algorithms should weigh itemset statistical information with their semantic importance to generate a more accurate and sensible description of the itemset utility. A novel High Utility Itemset tree (HUI-tree) structure, which is an extended prefix-tree structure for the storage of compressed utility information about itemsets, is proposed to address this issue. Moreover, FHUI-Growth, a fast approach for high utility itemset mining algorithm is developed for mining high utility itemsets. Mining efficiency is achieved with three new techniques: (1) both frequency statistic and complex itemset utility information can be compressed into the condensed HUI-tree structure, which successfully avoids multiple database rescan, (2) a part of the utility calculation process can be simplified because a tighter bottom bound pruning constrain can be obtained through the HUI-Tree, and (3) the costly tree scan operation is converted into the item conditional projection matrix row and column computation, which effectively reduces the mining process. Evaluations of the testing data set show that the execution performance and scalability are better than the classical Two-Phase algorithm.

Keywords*: high utility itemset; utility mining; conditional project matrix*

1. Introduction

Data mining involves data description and data predication. Association rule mining is one of the most popular descriptive mining methods used to discover interesting correlations among items from large databases [1]. The traditional frequent itemset mining algorithms often adopt one-fold item interest definition, which indicates that item utility is only related to statistical information, with 1 denoting the presence and 0 standing for the absence of such information. This single definition is inappropriate for real business applications [2-3]. For instance, the most common problem for managers is to identify or differentiate valuable customers from regular ones. If the system takes a traditional frequency-based association rule mining method, the managers would probably be told that customers who regularly buy daily supplies of bread, milk, tissue or soap are potential VIPs. However, based on their business sense, managers understand that customers with access to more funds and who desire luxuries can contribute more to their total profit despite their irregular visits. Other high-utility mining applications include recommendation systems and cross-selling analysis. High-utility association rule mining [2], which unites mining process with business economic objects, has recently helped industrial practitioners and researchers to discover useful and valuable itemsets across variables in large databases and has thus attracted considerable attention [4]. Item utility can be measured in terms of frequency, cost, profit or other expressions of user preferences [5]. The utility of an item can generally be divided into two: local transaction utility and external transaction utility. Local transaction utility is defined as the statistical information or quantity in a transaction, whereas the external transaction utility refers to the profit or user preferences. Multiplying the local transaction utility and external transaction utility derives the overall item utility. An item is therefore called a high-utility item if its utility is no less than a predefined threshold; otherwise, the item is called a low-

ISSN: 2005-4270 IJDTA

utility item [6]. High-utility item mining is an extension of traditional frequent item mining with the goal of identifying items that contribute a large portion of the system total utility [7]. The standard association rule mining algorithm will suffer when applied to high-utility mining in stream data because of two nontrivial problems that limit the practicality of such algorithm.

First, no direct pruning method can be used to reduce the candidate set, and handling a large number of candidate high-utility itemsets is costly. Second, the characteristic of the stream data makes the multiple databases rescan unacceptable. An item with high local transaction utility may have low external transaction, especially when the utility threshold is low or the databases have a number of long transactions, which result in the ineffectiveness of downward closure strategy. Such ineffectiveness inevitably leads to a huge temporal candidate set.

In this paper, we promote a novel algorithm, called Fast High Utility Itemset-Growth (FHUI-Growth) to solve the aforementioned problems. The proposed method develops and integrates three techniques. First, a compact structure called High Utility Item-tree (HUI-tree) is presented to save item utility. This structure adopts a prefix-tree structure to store the quantitative and semantic information about high-utility items simultaneously. Furthermore, the bottom utility boundary strategy is adopted. In this strategy, the most likely minimum itemset utilities are pre-counted. Part of the high-utility itemset can be directly identified through the bottom utility boundary. Finally, a special technique called utility item conditional projection matrix is adopted to reduce the mining time. The tree traverse computation of the item utility can be transformed into the matrix row and column operation to avoid multiple database scans. The evaluation and experimental results of the proposed algorithm

2. Related Work

A large quantity of work has been conducted for frequent itemset mining, which can be divided into three basic groups: Apriori [4], FP-Growth [6], and Eclat [8]; with most existing high-utility mining algorithms originating from the first two methods [9]. The “downward closure property” has served an important function in the traditional asocial rule mining strategy by pruning infrequent itemsets. The anti-monotone property of Transaction Weight Utility (TWU) is first adopted by the Two-Phase algorithm [2], which seeks to generate an enlarged set of high-utility item candidates by exploring the downward closure strategy. The scale of high-utility candidate itemsets remains too large to ignore despite the Transaction Weight Down Closure (TWDC) property. Li *et al.* [7] presents an isolated item pruning strategy that optimizes the candidate generation process in the first phase. The experiment results show that the high-utility candidate itemsets can be strongly reduced through this strategy at a level-wise manner. Yun [10] gives a detailed analysis on the mathematical properties of the utility function to overcome the candidate predication accuracy problem; the utility upper bound property is adopted to prune the low utility itemsets. The disadvantage of this method is that it may result in an incomplete final high-utility set because some sets have already been mistakenly pruned during the previous phase.

Many algorithms have explored the tree structure to restrict the size of the candidate set and simplify the utility computing process. For instance, the UP-Growth algorithm [6] explores a Utility Itemset Tree (UP-Tree) structure, generating a group of candidate pruning strategies, such as Discarding Local Unpromising items and Decreasing Local Node Utilities. UP-Tree can efficiently generate potential high-utility itemsets with only two scans of the database. However, UP-Growth does not address the issue of a huge potential high-utility candidate set that accompanies long transactions or the small minimum threshold.

The CTU-PRO [11] algorithm builds another data structure called Compressed Utility Itemset Tree (CUP-Tree). The individual high TWU items will be constructed to a Global CUP-Tree, after which the algorithm will construct a small projection tree for each high TWU item. The CTU-PRO method avoids rescanning the database by exploring the projection tree. The performance of the CTU-PRO algorithm is better than that of the Two-Phrase algorithm. Ahmed proposes another two tree-based algorithms called IHUP [12] and HUC_Prune [13]. The IHUP algorithm has three different kinds of HUP trees: the incremental HUP Lexicographic Tree ($IHUP_L$-Tree), the IHUP

Transaction Frequent Tree ($IHUP_{TF}$-Tree), and the IHUP-Transaction-Weighted Utilization Tree ($IHUP_{TWU}$-Tree), in which item nodes are arranged according to their lexicographic, transaction weight, or TWU value orders. Although the HUP-Tree database rescan can be traversed, the algorithm remains inefficient in compressing the huge HTWUI set without losing completeness, similar to the FP-Growth algorithm. The HUC Prune algorithm requires three database scans to identify an exact high-utility itemset.

Recently, Liu [14] discussed a high-utility itemset mining algorithm that enumerates an itemset as a prefix extension of another itemset. The upper bound on the utility prefix set of the current itemset is computed to identify if such itemset can possibly be a high-utility itemset without generating candidates. Consequently, the candidate can be pruned more efficiently by using a look-ahead strategy because the utility upper bound is more rigorous than TWU. However, the algorithm still suffers the level-wise enumerated node problem.

No proposed algorithm can directly calculate item utility. Thus, the challenge in high-utility mining lies in the itemset utility calculation and candidate pruning. In view of this challenge, we present the FHUI-Growth algorithm, which intends to integrate the advantages of the Two-Phrase algorithm, Prefix-tree structure, and matrix calculation to mine high-utility itemsets efficiently. The experiment result analysis shows that the performance of the TIFP-Growth algorithm is better and its scalability is more stable than that of the classical Two-Phrase algorithm.

3. FHUI-Tree Construction

Suppose that $I=\{i_1,i_2,\ldots,i_m\}$ is a set of items, $X\subseteq I$, and D is a transaction database, $D=\{T_1,T_2,\ldots,T_n\}$, $T_i\subseteq I$, $T_i=\{X_1, X_2,\ldots X_q\}$. We give the following definitions:

Definition 1: An inner transaction utility of item i_p in transaction T_q is denoted as $ui(i_p,T_q)$, which is the i_p's transaction frequency in T_q. As shown in Figure1a, $ui(a,T_1)$=2 and $ui(c, T_1)$=0.

Definition 2: An external transaction utility of item i_p is denoted as $ue(i_p)$, which is a weight independent of any transaction and only determined by the user. As shown in Figure1b, $ei(a)$=2, and $ei(b)$=10.

Both inner and external transaction utilities of the item are assigned a non-negative value.

Definition 3: The transaction utility of i_p contained in the transaction T_q is denoted as $u(i_p,T_q)$.

$$u(i_p, T_q)=ui(i_p, T_q)\times ue(i_p) \tag{1}$$

Definition 4: Given a k-itemset $X=(i_1, i_2,\ldots i_k)$, $X\subseteq T_q$, $1\le k\le m$, the transaction utility of X in T_q is denoted as $u(X,T_q)$, which is defined as

$$u(X,T_q) = \sum_{i_p\in X} u(i_p, T_q) \tag{2}$$

Definition 5: The transaction utility of X in database D is denoted as u(X), which is defined as

$$u(X) = \sum u(X, T_q) = \sum_{T_q\in D}\sum_{i_p\in X} u(i_p, T_q) \tag{3}$$

Definition 6: Suppose the user-defined minimum threshold is minU, we consider an itemset X as a high-utility itemset if and only if $u(X)\ge$minU; otherwise, we call it a low-utility itemset. The high-utility item is abbreviated as HUP, HUP = $\{X|X\subseteq I, u(X)\ge \text{minU}\}$. The low-utility item is abbreviated as LUP, LUP = $\{ X|X\subseteq I, u(X)<\text{minU}\}$.

Table 1a. Item External Utility

ITEM	Item external utility
A	2
B	10
C	3
D	8
E	7
F	1

Table 1b. Item Utility (Ui)

Tid	a	b	c	d	e	f
T_1	2	2	0	0	0	0
T_2	3	1	12	4	2	0
T_3	0	0	15	0	3	0
T_4	4	1	0	0	0	1
T_5	1	10	0	8	9	0
T_6	0	0	3	0	4	0

Based on the example in Table 1, suppose minU=60

$u(\{a,b\})=u(\{a,b\},T_1) + u(\{a,b\},T_5)=u(\{a\},T_1)+u(\{b\},T_1)+u(\{a\},T_5)+u(\{b\},T_5)=186$,

$\{a,b\} \subseteq$HUP

$u(\{a\})=u(\{a\},T_1) +u(\{a\},T_2) +u(\{a\},T_4)+u(\{a\},T_5)=20$,

$\{a\} \subseteq$ LUP

To simplify the interpretation process, the algorithm will give the bottom boundary, which is the minimum utility value of $u(X,T_q)$. To calculate the boundary, we need a more in-depth discussion on the item utility.

Definition 7: FHUI-Tree is a kind of prefix tree that is similar to FP-tree, except that:

The items on the FP-tree are arranged in the descending order of their support degrees; the items on the FHUI-Tree are placed according to their lexicographic order.

FP-tree retains the support degree of the current node, but FHUI-Tree records the transaction identification number information, which includes six parts: node_no (the number of the node in the FHUI-Tree), item_name (name of the item), item_utility, item_TID (the transaction identification number of the item), item_plink (directs to its super node), and item_link (directs to the nodes that contain the same item information).

Subsequently, we will construct an FHUI-tree according to the sample data contained in Table 1. After scanning the first database, all items are inserted into the header table based on their lexicographic order. Each TIFP-tree is equipped with a header table with three attributes: item_name, *ui(x,)* and f_pointer (linking to the first node in the FHUI-tree, which has the same item name). The algorithm will then re-scan the database to insert the items into the FTHUI-tree. The result is shown in Figure 1.

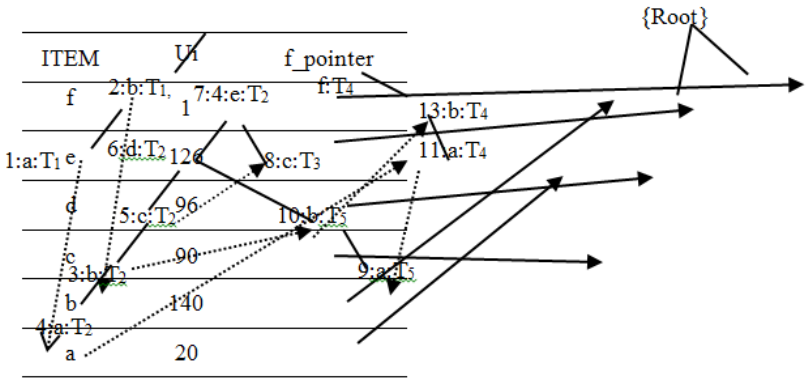


Figure 1. FHUI-Tree Example

Property 1: We can find all the paths that contain item X by traversing the tree starting from the f_pointer for each high utility item X.

Proof: As the node pointer links all the nodes with the same item_name, we can then obtain every path that contains item X by tree traverse. The utility information of each transaction can be gained based on the TID information contained in the node area. We will impose a rule that the TID of the current node should be that of the last transaction related to the item to prevent intermediate transaction information overlap. Therefore, when the current node is an intermediate transaction node, the algorithm can identify other transactions by following its item_link.

Definition 8: Given the lexicographic order "<", a prefix set of item i_p in the FTHUI-Tree is abbreviated as Pre_T(i_p), which is the item contained in the same transaction as i_p, and preserves the lexicographic order at the same time.

$$\text{Pre\_T}(i_p)=\{i_m|\forall T_q \subseteq D, i_m \subseteq T_q \wedge i_p \subseteq T_q \wedge i_m < i_p, i_m \cap i_p = \phi\}$$

Theorem 1: For a node X in the FTHUI-Tree, if $ui(X) \geq \text{minU}$, the utility sum of its Pre_T(X) denoted as $SU_{pre}(X)$, $SU_{pre}(X)$ is no less than $ui(X)$.

Proof: $SU_{pre}(X) = ui(\text{Pre\_T}(X)) + ui(X) \geq ui(X)$

4 FHUI-Growth Algorithm

4.1. Construction of Conditional Project Matrix

For most of the tree structure-based algorithm, approximately 80% CUP time is used in the tree traversing work [15], which is the major cause of low mining efficiency. However, the conditional projection matrix structure can effectively reduce the traversal time. The algorithm will begin to build the conditional project matrix (CPM) for each X item in the header table after the construction of FHUI-Tree. FHUI-Growth algorithm uses CPM(X_i) to preserve all X_i suffixed paths. Rows of CPM(X_i) denote the transaction utility of X_i in T_q, u(X_i,T_q), the first column stores the item_names of the nodes which is prefixed with X_i, whereas the rest of the columns represent a prefix path.

Definition 9: Let T be an FHUI-Tree and $X_i=\{i_1,i_2,..i_p,..i_m\}$ be the items in the header of T. The conditional project matrix array of X_i is CMP(X_i), which is a $k \times m$ matrix, in which each element of the matrix corresponds to the transaction utility of X_i in T_q, u(X_i,T_q).

Let us take the conditional matrix of item a in Figure1 as an example: the prefix paths of a including: ① b-a:T_1; ② e-d-c-b-a:T_2 ; ③ f-b-a:T_4; and ④ e-d-b-a:T_5. Nodes 1, 4, 9 or 10 do not fulfill the constraint of theorem 1. Thus, the algorithm will build a conditional matrix CPM for item a. CPM(a) is shown in Figure 2.

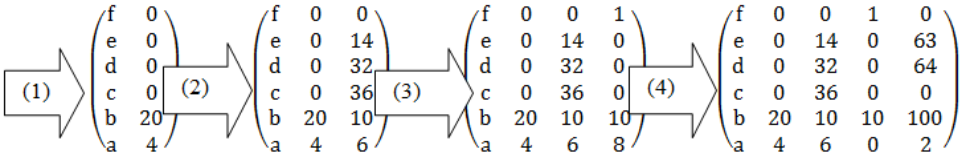


Figure 2. CPM (a)

Suppose the original candidate items size of FHUI-Tree is n, the corresponding size of CMP(X_i) is $\sum_{i=1}^{n-1} n(n-1)/2$. The size of the conditional matrix array will be released with the decrease in the size of FHUI-Tree.

The algorithm needs a two-pass database scan. During the first phase, the data set is scanned to construct the header table, items are arranged according to their lexicographic order, and at the same time, the utilities of each single item are calculated and stored in the header table. The FHUI-Tree will then be completed to facilitate the high-utility item mining in the second scan. The algorithm adopts the matrix row-column operation method, which is very efficient in utility computing, to accelerate the computing process.

Theorem 2: For each high-utility itemset X_i, CPM(X_i) contains all paths taking X_i as their suffix.

Proof: According to property 1, we can identify all paths containing X_i in the FHUI-tree. The algorithm should retain the lexicographic order of the item when inserting the transaction into the FHUI-tree. The conditional matrix would undoubtedly contain all information of all paths suffixed with X_i if we project each traversal path onto the column of the conditional matrix. The algorithm can determine all the transaction utility weights of any node X_k, $u(X_k,T_r)$ through the TID information preserved in the nodes, even if the node is not in this path, $u(X_k,T_r)$=0.

Our method uses the similarly constructed FP-Tree, which is different from LIU [14] that adopts an incompact tree structure. However, only the item frequency information is stored at the node in the FP-Tree, and frequency suppression is not a problem. The external and transaction utilities are both related to the entire item utility. Thus, the stored value in the FHUI-Tree should be adjusted when nodes are added or removed for high-utility computing. For example, two sub paths that pass through node 6(d,T_2): one is (a-b-c-d-e,T_2), and the other is (a-b-d-e,T_5), with the aggregation point at node 6. When traversing the sub path of (a,b:T_2-d,e:T_5), the algorithm will induce a transform, and the actual traversing path will become (a-b-d-e:T_5).

Suppose that minU=60, and through the computation on CPM(a), we can obtain high-utility items {{ab},{abd},{abde}}. The algorithm does not actually need to compute the utility of {abd}and {abde} because in step 4, {ab}is identified first. Thus, part of the identification work can be saved if the node utility in the FHUI-Tree is higher than minU. Pre_T{ab}={{abd},{abde}}are high-utility items on the basis of this theorem.

4.2 FHUI-Growth Algorithm

FHUI-Growth algorithm combined with the fast high-utility item identification method explores high-utility itemset mining without multiple database scanning. Furthermore, the algorithm adopts a bottom-up method to traverse the conditional matrix. Each time the algorithm focuses on the row above X_i, for the prefix item X_k of X_i, item X_k would be explored during the calculation of X_i, such that the algorithm will avoid repeated calculation mistakes.

Algorithm: FHUI-Growth
Input: Conditional project matrix $CPM_{k\times m}$, prefix of X_i, minU
Output: high utility itemset $X_{i+i'}$, and U($X_{i+i'}$)

```
1.      DO{
2.              FOR(r=k-1; r≥0; r--) {
//Only process item X_i and its Pre_T(X_i)
3.                      U_temp =0;
```


4. FOR (j=1; j≤m; j++)
//The algorithm will calculate the sum of their transaction utility only when all the itemsets are related with the same transaction.
5. IF CM[k,j] > MinU
6. THEN stop computation, Output high utility itemset (X_i)∪Pre_T(X_i) on the sub path k.
7. END IF
8. IF CM[k,j]≠0 AND CM[r,j] ≠0 THEN
9. UP_temp=UP_temp + CM[k,j]+CM[r,j];
10. END IF }
11. IF UP_temp≥minU THEN
12. Output high utility itemset, CM[k,0]∪ CM[r,0]
13. END IF }
//The itemset emerges in a bottom-up method, and the candidate itemset is gained and is suffixed with X_i.
14. CM[r,0] = CM[r,0] ∪ CM[k,0];
15. FOR (j=1; j≤m; j++)
16. CM[r,j] = CM[k,j] ∪ CM[r,j];
//deleting the k-th row.
17. k=k-1;
18. Calling FHUI-Growth($CM_{k\times m}$)
19. }
20. WHILE $CM_{k\times m}$ is not null

5. Experimental Evaluation and Performance Study

5.1. Testing Data

A variety of experiments have been conducted on both synthetic and real-world datasets to analyze the accuracy, efficiency, and scalability of the new algorithm thoroughly. The synthetic datasets T10I6D100K and T20I6D100K were provided by IBM Quest Generators [2]. T10I6100K includes 100,000 transactions with the mean transaction size of 10 items. T20I6D100K includes 100,000 transactions with the mean transaction size of 20 items. The average size of the longest frequent itemset of both T10I6100K and T20I6D100K is six. Chess [12] is a dense dataset, with the number of transactions at 3,196, distinct items at 75, and average transaction length at 37. Each transaction contained nearly more than 49% items. Table 2 shows the characteristics of the three testing datasets.

Table 2. Characteristics of Testing Datasets

Testing Dataset	Transaction numbers	Distinct Items	Average Transaction Length
T10I6D100K	100,000	10	6
T20I6D100K	100,000	20	6
Chess	3,196	75	37

The original data provided by the generator do not have special utility weight. Random utility parameters were set ranging from 1 to 10 to modify the internal transaction utility to imitate real data profit variance better. Figure 3 presents the distribution of the item external utilities. Most of the item utility values are lower than 2, which is a similar normal distribution log and coincides with the real situation that lesser numbered items are in the high-benefit group. The external utility random parameters are created ranging from 1 to 100, the utility value adopts a general normal distribution, and both database sizes are varied from 100k to 500k. The experiments were conducted on a machine with 2 GHz Intel Core i350 with 2 GB memory and implemented in C++. The performance comparison was carried out from three aspects: size of the temporal generating itemsets, mining execution time, and scale-up capability. The experiment's results are shown in Sections 5.2, 5.3, and 5.4. The

reference algorithm is the Two-Phrase [2] algorithm.

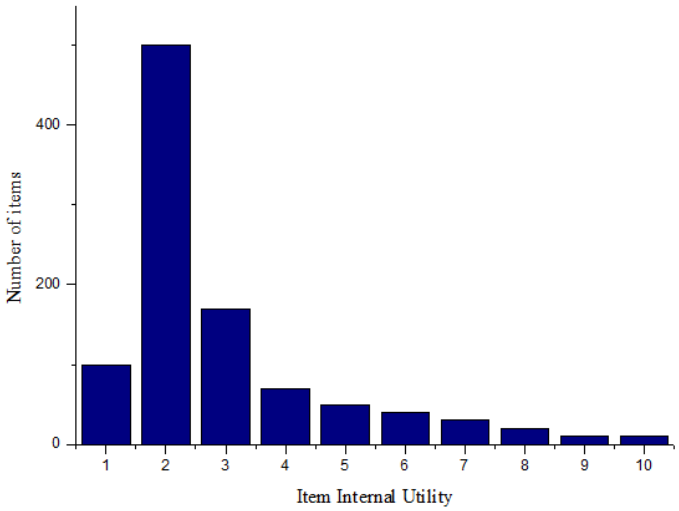


Figure 3. Distributions of Item Utility Values

5.2. Comparison Between Temporal Generating Itemsets

For the two synthetic sparse datasets, the item number was set at 1000, the minimum utility threshold of T10I6D100K and T20I6D100K ranged from 0.5% to 2% as the majority evaluation simulation for high-utility itemset mining algorithms [2,6,12]. For the dense dataset Chess, the reasonable value for the minimum threshold should be at least equal or higher than 30%; otherwise, the number of temporal candidate itemsets would be an enormous figure. The size of the temporal itemsets generated by the two algorithms is shown in Table 3. The temporal generating itemset size of FHUI-Growth is about 1/10 of that of Two-Phrase as can be seen from Table 3 and Table 4. The Two-Phrase algorithm adopts the level-wise candidate emerging and validating strategy, and it needs multiple database scans to create all possible k-candidate itemsets and ultimately identify high utility itemsets. Although part of the itemsets may never appear in the database, which could explain the unavoidable huge size of the temporal itemsets, the TWU property nevertheless effectively restricts its search space. The extended pattern growth method is integrated with the FHUI-Tree data structure in our FHUI-Growth algorithm instead. The anti-monotone property of item utility calculation has been partly considered in the bottom utility boundary strategy, the main benefit of which is the capability to forecast the true high-utility itemsets. The denseness of the dataset affects the efficiencies of the two algorithms. Based on Table 5, the level-wise methodology is clearly inappropriate for the dense dataset because scanning the whole dataset to prune low-utility itemsets will be costly.

Table 3. Number of Candidates on T10I6D100K

Minimum Utility	FHUI-Growth	Two-Phrase
0.2%	16,320	361,675
0.3%	3,218	303,810
0.4%	2,893	258,840
0.5%	1,536	226,290
0.6%	1,003	182,710
0.7%	642	130,280
0.8%	538	129,286
0.9%	497	101,253
1.0%	401	98,790

Table 4. Number of Candidates on T20I6D100K

Minimum Utility	FHUI- Growth	Two-Phrase
0.2%	26,372	401,856
0.3%	10,915	371,953
0.4%	4,461	337,431
0.5%	3,217	312,067
0.6%	2,346	278,631
0.7%	749	231,462
0.8%	704	229,503
0.9%	573	191,060
1.0%	529	183,921

Table 5. Number of Candidates on Chess

Minimum Utility	FHUI- Growth	Two-Phrase
30%	583,432	4,320,145
40%	322,104	3,563,207
50%	13,003	3,203,412
60%	8,321	1,614,293
70%	2,205	627,428

5.3. Evaluation of Execution Efficiency

The graphical representation of the comparison of the execution time of both algorithms is shown in Figure 4. As explained in Section 5.2, the Two-Phrase is similar to the Apriori algorithm in that it is a candidate pruning-based high-utility item-mining algorithm. Suppose the current high-utility itemset is $X_c=\{x_{c1},x_{c2},\ldots,x_{cn}\}$, the algorithm takes all of its supersets as the potential candidate set X_{c+1}. Nevertheless, X_{c+1} might have low utility or might not exist in the dataset. By contrast, in the FHUI-Growth algorithm, the system will not calculate the utility value of X_c if it does not appear in the present dataset. Furthermore, the algorithm can directly identify a true item utility value, then decide whether or not it should be added to the high-utility itemsets through the matrix row and column operation. Hence, no time is wasted on generating and testing the temporal candidate high utility set. The Two-Phrase algorithm also performs an additional scan on the itemsets to prune low-utility itemsets from the candidate sets. The total dataset scanning number of the Two-Phrase would be $M+1$ when the maximum length of the candidate itemsets is M. Meanwhile, two dataset scans independent of the maximum high-utility item length are sufficient for the FHUI-Growth algorithm. Therefore, Figure 4 shows that when the minimum utility threshold decreases, the execution time of the Two-Phrase algorithm increases sharply compared with that of FHUI-Growth. This finding can be better proven by the experiment result from Figure 6. The Chess dataset is a typical dense dataset wherein the size of the candidate high utility itemset is much larger than that of the two sparse synthetic datasets, thereby taking more work for the Two-Phrase algorithm to perform a dataset scan. Conversely, our algorithm adopts a projection matrix calculation to store and compute the utility information, without requiring additional space to store the candidate itemsets because it is not a generating and testing method.

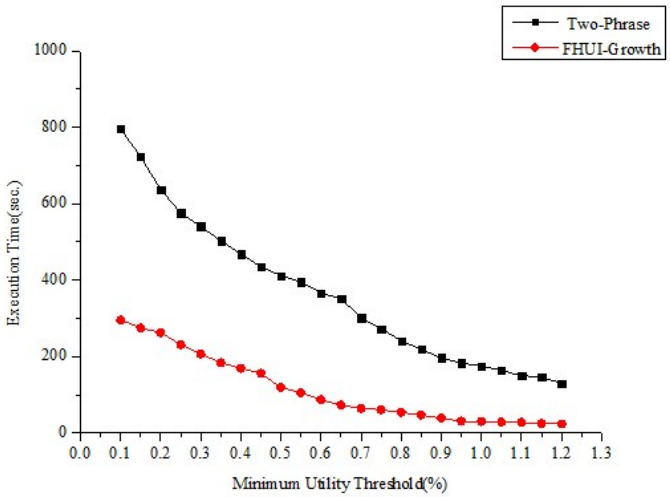


Figure 4a. Execution Time on T10I6D100K

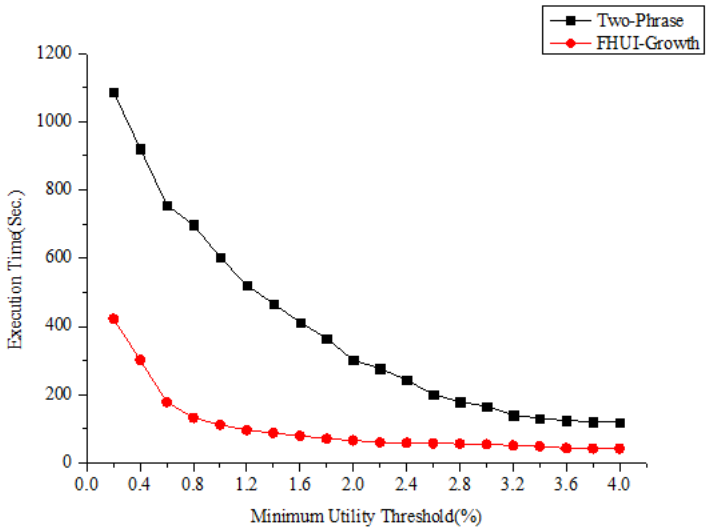


Figure 4b. Execution Time on T20I6D100K

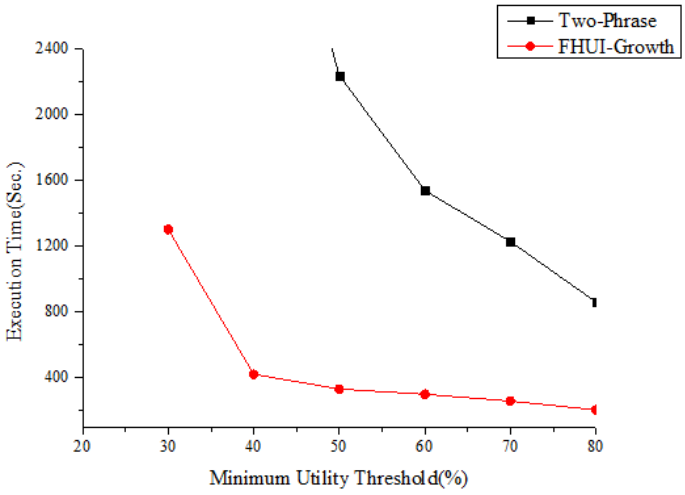


Figure 4c. Execution Time on Chess

5.4. Evaluation of Memory Consumption

High-utility itemsets contained in the same transactions overlap because the HUI-Tree adopts a prefix sharing strategy. This method is an efficient way of storing frequent itemsets using minimal memory without losing utility information, especially for a dense dataset. The number of tree nodes is significantly smaller than the potential candidate itemsets generated in the Two-Phrase algorithm. The pattern growth algorithm does not require generating the potential candidate itemsets in advance, the combination of the prefix tree data structure and the tree traverse strategy can effectively condense the memory into a small size. The comparison the memory consumption of the two algorithms on Chess dataset is presented in Figure 5.

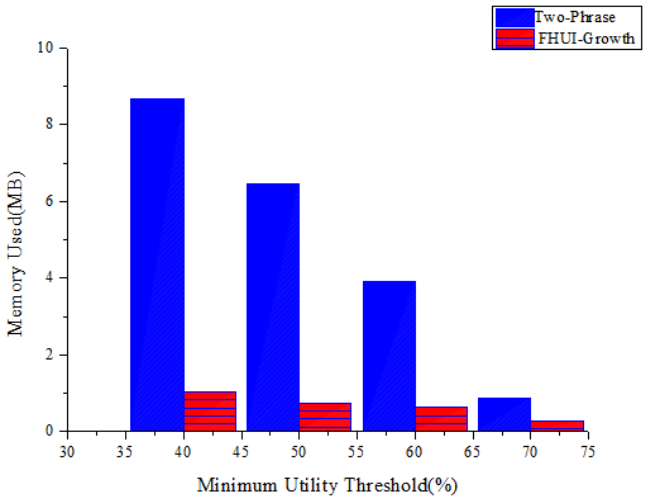


Figure 5. Memory Used on Chess

5.5. Evaluation of Scalability

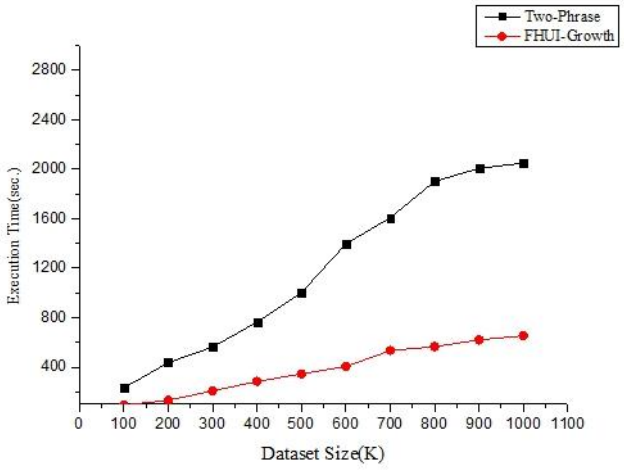


Figure 6. Evaluation of Scalability on T10I6D100K

The evaluation of scalability was also explored by testing varying dataset sizes in an incremental manner. We choose the T10I6D100K as the basic dataset, the minimum utility threshold was fixed at 0.6%, the dataset size ranges from 100K to 1000K. From the experimental results shown in Figure 6, we can see that the execution time for the two algorithms both increase with the expansion of the database size. The transaction size has a

significant influence on the Two-Phrase algorithm than on the FHUI-Growth algorithm. Based on the proposed tree construction and traversing strategies, the identification of itemset utility by the new algorithm is more direct and efficient than that of the Two-Phrase algorithm because of the lack of an additional expense on scanning of unpromising itemsets. Therefore, the scalability of the new algorithm is better than that of the Two-Phrase algorithm.

6. Conclusions

Incorporating utility considerations in data-mining tasks has recently gained popularity. A novel algorithm has been presented in this paper to address the problem of high utility mining. A new FHUI-Tree structure is presented. Aside from general information, including statistical support and transaction number, the item utility information is also stored in the FHUI-Tree, which requires less memory to compress the itemset utility information. Multiple database scans can be reduced to only two dataset scans. In addition, three other strategies are presented to accelerate the calculation process: First, the algorithm chooses the itemset utility value, rather than calculating the transaction utility value, as the constraining basis of candidate pruning, thus decreasing the expenses spent on hopeless itemsets. Second, if an item utility bottom is higher than the minimum utility threshold, an item utility bottom boundary is generated, then the item can be labeled as a high-utility item, and the identification process can be accelerated without affecting the mining accuracy. Third, the computation of item utility can be transformed into the matrix row and column operation by incorporating the conditional matrix array technique into the FHUI-Growth method. Thus, multiple database scans can be avoided. The experiment result analysis shows that the performance of the TIFP-Growth algorithm is better and its scalability is more stable than that of the classical Two-Phrase algorithm, despite the decreasing user-defined threshold.

Acknowledgements

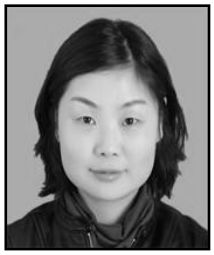
This study was supported by Natural Science Foundation of China(No. 61173049), Scientific Research Fund of Hunan Provincial Education Department of China(No.13A114, No.14A016).

References

[1] D. T. Larose, “Discovering knowledge in data: an introduction to data mining”, John Wiley & Sons, Hoboken, **(2014)**.
[2] Y. Liu, W. K. Liao and K. Choudhary, “A Fast high utility itemsets mining algorithm”, Proceedings of the 1st international workshop on Utility-based data mining, Chicago, USA, **(2005)**.
[3] H. F. Li, H. Y. Huang and Y. C. Chen, “Fast and memory efficient mining of high utility itemset in data stream”, Eighth IEEE International Conference on Data Mining, Pisa, Italy, **(2008)**.
[4] V. S. Tseng, B. E. Shie and C. W. Wu, “Efficient Algorithms for Mining High Utility Itemsets from Transactional Databases”, IEEE Transactions on Knowledge and Data Engineering, vol. 8, no. 25, **(2013)**.
[5] S. Venkateswari and R. M. Suresh, “Association Rule Mining in E-commerce: A Survey”, International Journal of Engineering Science and Technology, vol. 4, no. 3, **(2011)**.
[6] V. S. Tseng, B. E. Shie and C. W. Wu, “Efficient algorithms for mining high utility itemsets from transactional databases”, IEEE Transactions on Knowledge and Data Engineering, vol. 8, no. 25, **(2013)**.
[7] Y. C. Li, J. S. Yeh and C. C. Change, “Isolated items discarding strategy for discovering high utility itemset”, Data and Knowledge Engineering, vol. 1, no. 64, **(2008)**.
[8] A. Salam and M. S. H. Khayal, “Mining top−k frequent patterns without minimum support threshold”, Knowledge and information systems, vol. 1, no. 30, **(2012)**.
[9] B. Le, H. Nguyen and B. Vo, “An efficient strategy for mining high utility itemsets”, International Journal of Intelligent Information and Database Systems, vol. 2, no. 5, **(2011)**.
[10] U. Yun and J. Kim, “A fast perturbation algorithm using tree structure for privacy preserving utility mining”, Expert Systems with Applications, vol. 3, no. 42, **(2015)**.
[11] J. Pillai and O. P. Vyas, “Overview of itemset utility mining and its applications”, International Journal of Computer Applications, vol. 11, no. 5, **(2010)**.
[12] C. F. Ahmed, S. K. Tanbeer and B. S. Jeong, “Efficient tree structures for high utility itemset mining in incremental databases”, IEEE Transactions on Knowledge and Data Engineering, vol. 12, no. 21, **(2009)**.
[13] C. F. Ahmed, S. K. Tanbeer and B. S. Jeong, “HUC-Prune: an efficient candidate pruning technique to mine high utility itemset”, Applied Intelligence, vol. 2, no. 34, **(2011)**.

[14] J. Q. Liu, K. Wang and B. Fung, "Direct discovery of high utility itemset without candidate generation", IEEE 12th International Conference on Data Mining, Brussels, Belgium, **(2012)**.
[15] B. Vo, T. P. Hong and B. Le, "DBV-Miner: a dynamic bit-vector approach for fast mining frequent closed itemsets", Expert Systems with Applications, vol. 8, no. 39, **(2012)**.

Authors



PanYi, She received her M.S. degree (2000) and Ph.D degree (2005) in Computer Software and Theory from Huazhong University. Now she is a associate professor in Department of Mathematics and Computer Science of Changsha University. Her current main research interests include time synchronization, data mining.

Liu Huafu, He is a professor in Department of Mathematics and Computer Science of Changsha University. His main research interests include the design and implementation of wireless sensor networks, pattern recognition and *etc.*

Zhang Bo, He received his M.S. degree (2008) in Mechanical Manufacturing and Automation from Wuhan University of Technology. Now he is majoring in Logistics Management of Wuhan University of Technology. His current main research interests include big data processing and analysis. (Corresponding author of the paper.)

