Abstract - The discovery of itemsets with high utility like profits is referred by mining high utility itemsets from a transactional database. Although in recent years a number of relevant algorithms have been proposed, for high utility itemsets the problem of producing a large number of candidate itemsets is incurred. The mining performance is degraded by such a large number of candidate itemsets in terms of execution time and space requirement. When the database contains lots of long transactions or long high utility itemsets the situation may become worse. Internet purchasing and transactions is increased in recent years. Based on the interest of customer or client they search for their product in the internet. In the internet the product sellers publish their ads. Two algorithms are proposed in this paper for mining high utility itemsets with a set of effective strategies for pruning candidate itemsets, namely UP-Growth (Utility Pattern Growth) and UP-Growth. In a tree-based data structure named UP-Tree (Utility Pattern Tree) The information of high utility itemsets is maintained such that with only two scans of database candidate itemsets can be generated efficiently. The performance of UP-Growth and UP- Growth+ is compared with the state-of-the-art algorithms on many types of both real and synthetic datasets. Experimental results show that the proposed algorithms when databases contain lots of long transactions not only reduce the number of candidates effectively but also outperform other algorithms substantially in terms of runtime.

Keywords : data mining, utility mining, candidate pruning, frequent itemset, high utility itemset.

GJCST-C Classification : H.2.8

A Novel Mining Algorithm for High Utility Itemsets from Transactional Databases

By Sadak Murali & Kolla Morarjee
CMR Institute of Technology Medchal, India
A Novel Mining Algorithm for High Utility Itemsets from Transactional Databases

Sadak Murali & Kolla Morarjee

Abstract - The discovery of itemsets with high utility like profits is referred by mining high utility itemsets from a transactional database. Although in recent years a number of relevant algorithms have been proposed, for high utility itemsets the problem of producing a large number of candidate itemsets is incurred. The mining performance is degraded by such a large number of candidate itemsets in terms of execution time and space requirement. When the database contains lots of long transactions or long high utility itemsets the situation may become worse. Internet purchasing and transactions have increased in recent years. Based on the interest of customer or client they search for their product in the Internet, In the Internet the product sellers publish their ads. Two algorithms are proposed in this paper for mining high utility itemsets with a set of effective strategies for pruning candidate itemsets, namely UP-Growth (Utility Pattern Growth) and UP-Growth. In a tree-based data structure named UP-Tree (Utility Pattern Tree) The information of high utility itemsets is maintained such that with only two scans of database candidate itemsets can be generated efficiently. The performance of UP-Growth and UP-Growth* is compared with the state-of-the-art algorithms on many types of both real and synthetic datasets. Experimental results show that the proposed algorithms when databases contain lots of long transactions not only reduce the number of candidates effectively but also outperform other algorithms substantially in terms of runtime.

Keywords : data mining, utility mining, candidate pruning, frequent itemset, high utility itemset.

1. Introduction

Data Mining refers to extracting or mining knowledge from large amounts of data. In large databases finding of frequent patterns task is very important use full in many applications over the past few years. The primary goal is to discover hidden patterns, unexpected trends in the data. Data mining is concerned with analysis of large volumes of data to automatically discover interesting regularities or relationships which in turn leads to better understanding of the underlying processes. Data mining activities uses combination of techniques from database artificial intelligence, statistics, technologies machine learning. This includes bioinformatics, genetics, medicine, clinical research, education, retail and marketing research.

Utility Mining is one of the most challenging data mining tasks is the mining of high utility itemsets efficiently. Identification of the itemsets with high utilities is called as Utility Mining. The utility can be measured as per the user preferences utility can be measured in terms of cost, profit or other expressions. The limitations of frequent or rare itemset mining motivated researchers to conceive a utility based mining approach, which allows a user to conveniently express his or her perspectives concerning the usefulness of itemsets as utility values and then find itemsets with high utility values higher than a threshold. In utility based mining the term utility refers to the quantitative representation of user preference i.e. according to an itemsets utility value is the measurement of the importance of that itemset in the user’s perspective.

Mining high utility itemsets from databases refers to finding the itemsets with high profits. Itemset meaning is importance, interestingness or portability of an item to users. High utility itemsets mining has become one of the most interesting data mining tasks with broad applications and it identifies itemsets whose utility satisfies a given threshold. By using different values it allows users to quantify the usefulness or preferences of items using different values. In a transaction database this itemset consists of two aspects: First one is itemset in a single transaction is called internal utility and second one is itemset in different transaction database is called external utility. Mining high utility itemsets from databases is an important task has a wide range of applications such as website click stream analysis [13, 16, 21], online e-commerce management, mobile commerce environment planning and even finding important patterns in biomedical applications.

In Data Mining the task of finding frequent pattern in large databases is very important use full in many applications over the past few years. The goal of frequent itemset mining is to identify all frequent itemsets. The generations of association rules are straight forward, once the frequent itemsets are identified. In the real world, however, each item in the supermarket has a different importance/price and single customer will be interested in buying multiple copies of same item. Therefore, finding only traditional frequent patterns in a database cannot fulfill the requirement of finding the most valuable customers/itemsets that contribute the most to the total profit in a retail business.

Author a : Department of Computer Science and Engineering, CMR Institute of Technology Medchal, Hyderabad, Andhra Pradesh, India. E-mail : muralisadak@gmail.com

Author a : Assistant Professor, Department of Computer Science and Engineering, CMR Institute of Technology Medchal, Hyderabad, Andhra Pradesh, India. E-mail : morarjeek@gmail.com
II. BACKGROUND

All Given a finite set of items I = {i1, i2, ..., im}. Each item ip (1 ≤ p ≤ m) has a unit X is an itemset with the set of k distinct items {i1, i2, ..., ik}, where i |e|, 1 ≤ j ≤ k, and k is the length of X. An itemset with length k is called k-itemset. A transaction database D = {T1, T2, ..., Tn} contains a set of transactions, and each transaction Td (1 ≤ d ≤ n) has an unique identifier d, called TID. Each item ip in the transaction Td is associated with a quantity q (ip, Td), that is, the purchased number of ip in Td. Consider a simple database with 5 transactions and 7 items. By using different values it allows users to quantify the usefulness or preferences of items.

<table>
<thead>
<tr>
<th>TID</th>
<th>Transaction</th>
<th>TU</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>(A,1) (C,10) (D,1)</td>
<td>17</td>
</tr>
<tr>
<td>T2</td>
<td>(A,2) (C,6) (E,2) (G,5)</td>
<td>27</td>
</tr>
<tr>
<td>T3</td>
<td>(A,2) (B,2) (D,6) (E,2) (F,1)</td>
<td>37</td>
</tr>
<tr>
<td>T4</td>
<td>(B,4) (C,13) (D,3) (E,1)</td>
<td>30</td>
</tr>
<tr>
<td>T5</td>
<td>(B,2) (C,4) (E,1) (G,2)</td>
<td>13</td>
</tr>
<tr>
<td>T6</td>
<td>(A,1) (B,1) (C,1) (D,1) (H,2)</td>
<td>12</td>
</tr>
</tbody>
</table>

**Table 1 : An Example Database**

<table>
<thead>
<tr>
<th>Item</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>Profit</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**Table 2 : Profit Table**

- **Utility of an item**
  The utility of an item ip in the transaction Td is denoted as \( u(ip, Td) \) and defined as \( p(ip) \times q(ip, Td) \). For example, in Table 1, \( u({A}, T1) = 5 \times 1 = 5 \).

- **Utility of an itemset**
  The utility of an itemset X in Td is denoted as \( u(X, Td) \). For example, \( u({AC}, T1) = u({A}, T1) + u({C}, T1) + u({AC}, T1) = 5 + 1 + 6 = 12 \).

- **High utility itemset**
  An itemset is called a high utility itemset if its utility is not less than a user-specified minimum utility threshold which is denoted by Min_util. else, it is called as a low utility itemset.

- **Transaction Utility**
  The transaction utility of a transaction Td is denoted as \( TU(Td) \) and defined as \( u(Td, Td) \). For example, \( TU(T1) = u(ACD, T1) = 8 \).

- **Internal Utility**
  Internal utility value of item ip in transaction Tq, denoted as \( u(ip, Tq) \), is the value of ip in Tq. For example, in Table 1, \( u(C, T02) = 6 \).

- **External Utility**
  External utility of item ip in a transaction database, denoted as \( eu(ip) \), is the value of ip in the utility table of the database. For example, in Table 2, \( eu(C) = 1 \) and \( eu(D) = 2 \).

- **The total utility value**
  The total utility value of DB, denoted as \( Tutil(DB) \), is the sum of all transaction utility values in DB. That is, \( Tutil(DB) = \sum_{Tq\in DB} u(Tq, Tq) \). For example, \( Tutil(DB) = 96 \) as shown in Table 3.

- **Transaction-weighted utilization of an itemset**
  The transaction-weighted utilization of an itemset X is the sum of the transaction utilities of all the transactions containing X, which is denoted as \( TWU(X) \). For example, \( TWU(\{AD\}) = TU(T1) + TU(T3) = 8 + 30 = 38 \). If \( TWU(X) \) is no less than the minimum utility, X is called a high transaction-weighted utilization itemset.

- **Transaction-weighted downward closure**
  The transaction-weighted downward closure, which is abbreviated as TWDC, is stated as follows. For any itemset X, if X is not a HTWUI, then any superset of X is a low utility itemset. By this definition, the downward closure property can be maintained by using transaction-weighted utilization. For example, in Table 1, any superset of \( \{A\} \) is a low utility itemset since \( TWU(\{A\}) < Min \text{util.} \)

- **Transaction utility table**
  The transaction utility table can be calculated now for given transactional database.

<table>
<thead>
<tr>
<th>Item</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>Profit</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**Table 3 : Profit Table Transaction Utility Table**

<table>
<thead>
<tr>
<th>Item</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>Profit</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

III. LITERATURE SURVEY

R. Agrawal et al in [2] proposed Apriori algorithm, it is used to obtain frequent itemsets from the database. In mining the association rules we have the problem to generate all association rules that have support and confidence greater than the user specified minimum support and minimum confidence respectively. The first pass of the algorithm simply counts item occurrences to determine the large 1-itemsets. First it generates the candidate sequences and then it chooses the large sequences from the candidate ones. Next, the database is scanned and the support of candidates is counted. The second step involves generating association rules from frequent itemsets. Candidate itemsets are stored in a hash-tree. The hash-tree node contains either a list of itemsets or a hash table. Apriori is a classic algorithm for frequent itemset mining and association rule learning over transactional databases. After identifying the large itemsets, only those itemsets are allowed which have the support greater than the minimum support allowed. Apriori Algorithm generates lot of candidate item sets and scans database every time. When a new transaction is added to the database then it should rescans the entire database again.

J. Han et al in [11] proposed frequent pattern tree (FP-tree) structure, an extended prefix tree structure for storing crucial information about frequent patterns,
compressed and develop an efficient FP-tree based mining method is Frequent pattern tree structure. Pattern fragment growth mines the complete set of frequent patterns using the FP-growth. It constructs a highly compact FP-tree, which is usually substantially smaller than the original database, by which costly database scans are saved in the subsequent mining processes. It applies a pattern growth method which avoids costly candidate generation. FP-growth is not able to find high utility itemsets.

W. Wang et al in [23] proposed weighted association rule. In WAR, we discover first frequent itemsets and the weighted association rules for each frequent itemset are generated. In WAR, we use a twofold approach. First it generates frequent itemsets; here we ignore the weight associated with each item in the transaction. In second for each frequent itemset the WAR finds that meet the support, confidence. Weighted association rule mining first proposed the concept of weighted items and weighted association rules. However, the weighted association rules does not have downward closure property, mining performance cannot be improved. By using transaction weight, weighted support can not only reflect the importance of an itemset but also maintain the downward closure property during the mining process.

Liu et al in [15] proposes a Two-phase algorithm for finding high utility itemsets. Two-Phase algorithm, it efficiently prunes down the number of candidates and obtains the complete set of high utility itemsets. In Phase I, only the combinations of high transaction weighted utilization itemsets are added into the candidate set at each level during the level-wise search. In phase II, only one extra database scan is performed to filter the overestimated itemsets. Two-phase requires fewer database scans, less memory space and less computational cost. In Two-phase, it is just only focused on traditional databases and is not suited for data streams. Two-phase was not proposed for finding temporal high utility itemsets in data streams. However, this must rescans the whole database when added new transactions from data streams.

J. Hu et al in [12] defines an algorithm for frequent item set mining, that identify high utility item combinations. In contrast to the traditional association rule and frequent item mining techniques, which is defined as the combination of few items (rules), which satisfy certain conditions as a group and maximize a predefined objective function. The high utility pattern mining problem considered is different from former approaches, as it conducts “rule discovery” with respect to individual attributes as well as with respect to the overall criterion for the mined set, attempting to find groups of such patterns that combined contribute the most to a predefined objective function.

IV. Proposed Methods

There are three steps in the framework of proposed system. 1. To construct a global UP-Tree with the first two strategies we should scan the database twice. 2. From global local UP-Trees and UP-Tree potential high utility itemsets should be generated recursively by UP-GROWTH and 3. From the set of PHUIs identify actual high utility itemsets. To differentiate the patterns found by our methods from HTWUs we used a new term “potential high utility itemsets” as our methods are not based on traditional TWU model. From the set of HTWUs the set of PHUIs will become much smaller by our effective strategies.

a) UP-Tree

Compact tree structure, named UP-Tree (Utility Pattern Tree) is used to avoid scanning original database repeatedly and to facilitate the mining performance, to maintain the information of high utility itemsets and transactions. For reducing the size here the tree is compacting (closely packed together).

i. The Elements in UP-Tree

In a UP-Tree, each node N Consists of N.count, N.name, N.hlink, N.parent and set of child nodes. N.name is the node’s item name. The node’s support count is N.count. The node’s node utility is N.nu i.e., the utility of the node overestimated. N.parent records the parent node of N. N.hlink is a node link which points to a node whose item name is the same as N.name. A table named header table is employed to facilitate the traversal of UP-Tree. Each entry records an item name in header table, link and an overestimated utility. The last occurrence of the node is pointed by the link node which has the same item as the entry in the UP-Tree.

ii. Discarding Global Unpromising Items

With two scans of the original database the construction of a global UP-Tree can be done. Discarding global unpromising items (i.e., DGU strategy) is to eliminate the low utility items and their utilities from the transaction utilities. TU of each transaction is computed in the first scan. TWU of each single item is also accumulated at the same time. Promising and unpromising are two nodes of a node. More profits are given by selecting promising nodes and discarding unpromising nodes which gives fewer profits. High utility itemsets are only the supersets of the itemsets and the less quality is given by subsets of the item.

Transactions are inserted into a UP-Tree during the second scan of database. Unpromising items should be removed from the transaction a transaction is retrieved and also their from the transaction’s TU their utilities should also be eliminated. New TU after pruning unpromising items is called reorganized transaction utility (abbreviated as RTU). The
reorganized transaction $T$ of a RTU, is denoted as RTU ($T_r$).

iii. Decreasing Global Node Utilities

Divide-and-conquer technique is applied in the tree-based framework for high utility itemset mining in mining process. Into smaller subspaces the search space can be divided. **Discarding global node utilities** (i.e., DGN strategy) during global UP-Tree construction the node utilities which are nearer to UP-Tree root node are effectively reduced. The utilities of the nodes further reduced that are closer to the root of a global UP-Tree By applying strategy DGN. For the databases containing lots of long transactions DGN is especially suitable.

DGN and DGU are applied to construct a global UP-tree. DGU is applied after getting all promising items. By pruning the unpromising items and by sorting the remaining promising items in a fixed order, the transactions are reorganized. Any ordering can be used such as the TWU order or lexicographic, support. *The PHUI is similar to TWU*, which compute all itemsets utility with the help of estimated utility. Finally, identify high utility itemsets (not less than min_sup) from PHUIs values the global UP-Tree is constructed.

### Table 4: Reorganized Transactions and their RTUS

<table>
<thead>
<tr>
<th>TID</th>
<th>Reorganized transaction</th>
<th>RTU</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_1$</td>
<td>(C,10) (D,1) (A,1)</td>
<td>17</td>
</tr>
<tr>
<td>$T_2$</td>
<td>(E,2) (C,6) (A,2)</td>
<td>22</td>
</tr>
<tr>
<td>$T_3$</td>
<td>(E,2) (D,6) (A,2) (B,2)</td>
<td>32</td>
</tr>
<tr>
<td>$T_4$</td>
<td>(E,1) (C,13) (D,3) (B,4)</td>
<td>30</td>
</tr>
<tr>
<td>$T_5$</td>
<td>(E,1) (C,4) (B,2)</td>
<td>11</td>
</tr>
<tr>
<td>$T_6$</td>
<td>(C,1) (D,1) (A,1) (B,1)</td>
<td>10</td>
</tr>
</tbody>
</table>

b) UP-Growth

The basic method for generating PHUIs After constructing a global UP-Tree is to mine. Candidates will be generated too many. Thus by pushing two more strategies propose an algorithm **UP-Growth (Utility Pattern Growth)**. By the strategies the itemsets with overestimated utilities can be decreased and further the number of PHUIs can be reduced.

i. Discarding Local Unpromising Items

The global UP-Tree contains many sub paths. Each path is considered from bottom node of header table. This path is named as conditional pattern base (CPB). into conditional UP-Trees strategies DGN and DGU cannot be applied, a global UP-Tree actual utilities of items in different transactions are not maintained. Unless an additional database scan is performed the actual utilities of unpromising items that need to be discarded in conditional pattern bases cannot be known.

Discarding local unpromising items (i.e, DLU strategy) to discarding utilities of low utility items from path utilities of the paths. It reduce the overestimated utilities for second scan by this the complete set of PHUI are found. In the database to keep minimum item utilities for all global promising items we maintain a minimum item utility table. Here bottom entry nodes in header table are traced and nodes which are found traced to root. From the path utility of an extracted path an estimated value for each local unpromising item is subtracted. To reduce overestimated utilities locally is provided by a simple but useful schema without an extra scan of original database.

### Subroutine: UP-Growth ($T_X$, $H_X$, $X$)

**Input:** a header table $H_X$ for $TX$, A UP-Tree $TX$, an itemset $X$, and a minimum utility threshold min_util.

**Output:** All PHUIs in $TX$.

1. For each entry $ik$ in $H_X$ do.
2. Trace each node related to $ik$ via $ik$,$hlink$ and accumulate $ik$,$nu$ to nusum($ik$);
3. If nusum($ik$) < min_util, do
4. Generate a PHUI $Y = X \setminus ik$;
5. Set estimated utility of $Y$ set $pu(ik)$;
6. Construct $Y$-CPB;
7. In $H_Y$ put local promising items in $Y$-CPB
8. Apply DLU to reduce path utilities of the paths;
9. To insert paths into $TY$ with DLN apply Insert_Reorganized_Path;
10. If $TY$ is null then call UP-Growth ($TY$, $H_Y$, $Y$);  
11. End if
12. End for

*Figure 1: The Subroutine of UP-Growth*
c) An Improved Mining Method: UP-Growth$^+$

To decrease overestimated utilities of itemsets UP-Growth uses DLU and DLN than FP-Growth to achieve better performance. However, the overestimated utilities can be closer to their actual utilities by eliminating the estimated utilities that are closer to actual utilities of unpromising items and descendant nodes. For reducing overestimated utilities more effectively UP-Growth method is proposed which is more improved method. From the paths and path utilities of conditional pattern bases the local unpromising items (DNU) and their estimated Node Utilities are discarded. During the construction of global UP-Tree decreasing local Node utilities (DNN) for the nodes of local UP-Tree by estimated utilities of descendant Nodes.

In UP-Growth$^+$, minimal node utilities in each path are used to make the estimated pruning values closer to real utility values of the pruned items in database. After mining the whole UP-Tree by UP-Growth$^+$, we can obtain all PHUIs the number of PHUIs of UP-Growth$^+$ is less than that of UP-Growth. It means the overestimated utilities of itemsets as well as the number of PHUIs, as well as the overestimated utilities of itemsets, are further reduced by UP-Growth$^+$.

d) Efficiently Identify High Utility Itemsets

The third step is to identify high utility itemsets and their utilities after finding all PHUIs from the set of PHUIs by scanning original database once. This step is called phase II. Fewer candidates in phase I are generated by our method, as original database is large and it contains lots of unpromising items scanning original database is still time consuming. By this, scanning reorganized transactions high utility itemsets can be identified in our framework. Since the reorganized transactions there is no unpromising item, the execution time and I/O cost for phase II can be reduced further. When lots of unpromising items are contained in the original database this technique works well.

V. Experimental Evaluation

In this section the Performance of the proposed algorithms is evaluated. The experiments were done on a 2.80 GHz Intel Pentium D Processor with 3.5 GB memory. The operating system is Microsoft Windows 7. The algorithms are implemented in Java language. in the experiments Both real (Table 5) and synthetic datasets (Table 6) are used.

Table 5: Parameter Settings of Synthetic Datasets

<table>
<thead>
<tr>
<th>Parameter Descriptions</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>D</td>
</tr>
<tr>
<td></td>
<td>T</td>
</tr>
<tr>
<td></td>
<td>I</td>
</tr>
</tbody>
</table>

Table 6: Characteristics of Real Datasets

| Dataset         | | D | | T | | I | | Type |
|-----------------|----------------|---|---|---|---|---|---|
| Chain-store     | 1,112,949      | 7.2 | 46,086 | Sparse |
| Chess           | 3,196          | 37.0 | 75    | Dense  |

a) Evaluation on Real datasets

In this part, on three real datasets we show the performance comparison is done: Chess and Chainstore. First, Chess and Chain-store we show the result in Fig. 2. In Fig. 2 (a), we the runtime of IHUPT&FPG is the worst, followed by UPT&FPG, UPT&UPG and UPT&UPG$^+$ is the best. In Fig. 3 experimental results on real sparse databases are shown. In Fig. 3 (a) and (b) performance on Chain-store dataset is shown. In Fig. 3 (a), the runtime of IHUPT&FPG is the worst, followed by UPT&FPG, UPT&UPG and UPT&UPG$^+$ is the best. As more candidates are generated the performance of IHUPT&FPG is the worst. Among the three methods the execution time of UPT&FPG is the worst since UP-Growth$^+$ and UP-Growth efficiently prune the search space of local UP-Trees. We can observe that by comparing the performance of previous method to the performance of proposed methods substantially outperforms.

In Fig. 4 Experimental results of phase II are shown. Runtime for phase II is very long for large databases such as Chain-store so we only show the result of chess. We can observe in Fig. 4, that the runtime for phase II is not only proportional to number of candidates in phase II but also increases fiercely. Therefore in phase II the performance is highly dependent on the runtime, since the overhead of scanning databases is huge.

Figure 2: Performance Comparison on Dense Dataset
Evaluation on synthetic datasets

In this part, the results under different parameters are shown. In Fig. 5 we show the performance under varied average transaction length (T) first. On synthetic datasets TxF6.||1000.|D|100k and min_util is set to 1% these experiments are performed. In Fig. 5, with increasing T the runtime of all algorithms increases because when T is larger, transactions and databases become longer and larger. When T is larger than 25. The difference of the performance between the methods appears. The best method is UPT&UPG+ and the worst one is IH- UPT&FPG. In Fig. 5 (b), the number of candidates generated by UPT&UPG+ is the smallest. This shows that when transactions are longer by decreasing overestimated utilities that UP- Growth+ can effectively prune more candidates.

Scalability

In this subsection, The scalability of the compared methods is shown in this subsection. On synthetic datasets T10.F6.||1000.|D|100k the experiments are performed. Runtime results for phase I and II, in Fig. 5 and Table 7 number of candidates and number of high utility itemsets are shown respectively. In Fig. 5, we can observe on runtime all compared algorithms have good scalability. As shown in Fig. 5 (b), in phase I there are only minor differences for runtime. There are significant differences in runtime in

<table>
<thead>
<tr>
<th>Database</th>
<th>#HUs</th>
</tr>
</thead>
<tbody>
<tr>
<td>200k</td>
<td></td>
</tr>
<tr>
<td>400k</td>
<td></td>
</tr>
<tr>
<td>600k</td>
<td></td>
</tr>
<tr>
<td>800k</td>
<td></td>
</tr>
<tr>
<td>1000k</td>
<td></td>
</tr>
</tbody>
</table>

From the set of PHUIs, high utility itemsets are efficiently identified finally which is much smaller than HTWUIs generated by IHUP. By the reasons mentioned above, than IHUP algorithm the proposed algorithms UP-Growth and UP- Growth+ achieve better performance.

VI. Conclusions

In this paper, we have proposed two efficient algorithms named UP-Growth and UP-Growth+ for mining high utility itemsets from transactional databases. For maintaining the information of high utility itemsets a data structure named UP-Tree was proposed. With only two database scans, from UP-Tree Potential high utility itemsets can be efficiently generated. To perform a thorough performance evaluation both real and synthetic datasets were used in the experiments. Results show that the strategies considerably improved performance by reducing both the search space and the number of candidates. Moreover, the proposed algorithms, especially UP- Growth+, outperform the state-of-the-art algorithms substantially especially when databases contain lots of long transactions or a low minimum utility threshold is used.

References


11. J. Han, J. Pei, Y. Yin, “Mining frequent patterns without candidate generation,” in *Proc. of the ACM-SIGMOD Int’l Conf. on Management of Data*, pp. 1-12, 2000.


This page is intentionally left blank