Abstract:
Association rule mining is used to stumble on the patronize item sets in substantial database. In the data mining field, utility mining materialize as an essential topic that to excavation the high value item sets from databases which consign to verdict the item sets with elevated profits. The vast number of high utility item sets formulate a challenging hitch for the mining accomplishment, awaited to generate more potential high utility item sets. So it expends higher progression in outsized database and diminish the mining efficiency. They proposed two novel approaches such as UP growth and UP-growth+ in addition to compact UP Tree data structure, which is used for well-organized discovering high utility item sets from huge transactional databases. In existing system indiscriminate memory allocation is used to squirrel away the candidate that conduct to high potential I/O operations. Also this method is time devour and requires high memory storage. In sequence to solve this problem in proposed system, we used taxonomy with R-hashing technique for the memory allocation. Hence the applicant items are stored with their particular memory in UP tree. The experimental result illustrates that the proposed system is additional successful than the existing techniques according to the memory space and the amount of candidate item sets initiation and input and output operations.

Keywords: candidate sets, frequent Item set, high utility item set, utility mining, data mining.

I. Introduction: The purpose of our proposed systems is in relation to finding high utility item set. Data mining is the method of retrieving item sets as of database. Here, the meaning of item set utility is interestingness, profitability and importance of an item to users. An item set is described as high utility item set if its utility is no less than a user individual minimum utility threshold; or else, it is called a low-utility item set. The goal of frequent item set mining is to find items that co occur in a transaction database above a user given frequency threshold, without considering the quantity or weight such as profit of the items. However, quantity and weight are significant for addressing real world decision problems that require maximizing the utility in an organization. The high utility item set mining problem is to find all item sets that have utility larger than a user specified value of minimum utility. In existing System, HUP Algorithm is used to mining High Utility Item sets from database but there are some disadvantages like, it generates huge set of PHUIs. In our project we use UP-Growth Algorithm. Main advantages of his Algorithm are, it scan database only two times and it generates less set of PHUIs. The rest of the paper is organized as follows. Proposed algorithms are explained in section II. Experimental results are presented in section III. Concluding remarks are given in section IV. However, mining high utility item sets from the databases is not an easy task since the downward closure property [1] used in frequent item sets mining cannot be applied here. In other words, pruning search space for high utility item sets mining is difficult because a superset of a low utility item sets may be a high utility item sets. A naïve approach for this problem is to enumerate all item sets from the databases by the principle of exhaustion.

Obviously, this approach will encounter the large search space problem, especially when databases contain lots of long transactions or a low threshold is set. Hence, how to effectively prune the search space and efficiently capture all high utility item sets with no miss is a big challenge in utility mining.

II. Proposed Algorithm:

- Firstly, store transaction dataset and profit table into database.
- Generate Global UP-Tree by using DGU and DGN.
- Use UP Growth Algorithm for mining HUIs.
- Generate Local UP Tree by using DLU and DLN.
- Identify the HUIs.
Precondition:

- Select Transaction dataset and profit table.
- Calculate Transaction Utility of Item set(TU) by using following formula
  \[ TU(Td) = u(Td, Td) \]

Algorithms:

A. Discarding Global Unpromising Items algorithm

1. Compute Transaction Weighted Utility (TWU) of Item sets. e.g. TWU(A) = \( u(T1, T1) + u(T2, T2) + u(T3, T3) \)
2. Remove unpromising items from each transaction, Rearrange items in a descending order of TWU and calculate Reorganized transaction utility. e.g. RTU(T2) = Previous TU(T2) - [Q(G)*P(G)]

B. Decreasing Global Node utilities algorithm

1. Select the items from which item our transaction is started.
2. Insert these items into Up-Tree under root node.
3. From these node select next items and insert it into UP-Tree
4. Then recursively call above steps until all nodes inserted into Tree.
5. In this way construct the Global UP-Tree.

C. UP-Growth algorithm

1. Trace each node related to specific item via link and calculate sum of node utility, start from leaf node.
2. If sum is less than minimum threshold then remove that node from tree.
3. If sum is greater than minimum threshold then calculate highest utility node, select that node and generate Path from this node.
   1. Apply DLU to reduce path utilities of the paths.
   2. Insert this reorganized path into tree by using DLN.
   3. If node from local tree is not completed then recursively call this method.

D. Discarding Local Unpromising items algorithm

1. Compute local unpromising items.
2. Remove local unpromising items from the path and recalculate path utility.

E. Decreasing Local Node utilities algorithm

1. Construct Local UP-Tree.
2. Calculate node utility.
3. Identify High Utility Item sets.

R-hashing Algorithm:

Implements a S4 hash class in R similar to hashes / associated arrays / dictionaries in other programming languages. Where possible, the hash class uses the standard R accessors: 

\[ \text{hash \ class: R} \]

Methods

HASH ACCESSORS:

- Slice Replacement
- Slice replacement with interpolation
- Single key replacement
- Single key lookup with interpolation
- Single key replacement no interpolation
- Single key lookup no interpolation

Manipulation:

- Remove all key-value pairs from hash
- Remove specified key-value pairs from hash
- Test for existence of key
- Test if no key-values are assigned
- Return number of key-value pairs from the hash
- Retrieve keys from hash
- Retrieve values from hash
- Make a copy of a hash using a new environment
- Internal function for displaying hash

Note
HASH KEYS must be a valid character value and may not be the empty string "."
HASH VALUES can be any R value, vector or object.
PASS-BY REFERENCE. Environments and hashes are special objects in R because only one copy exists globally. When provide as an argument to a function, no local copy is made and any changes to the hash in the functions are reflected globally.
PERFORMANCE. Hashes are based on environments and are designed to be exceedingly fast using the environments internal hash table. For small data structures, a list will out-perform a hash in nearly every case. For larger data structure, i.e. >100-1000 key value pair the performance of the hash becomes faster. Much beyond that the performance of the hash far outperforms native lists.
MEMORY. Objects of class hash do not release memory with a call to rm. clear must be called before rm to properly release the memory.

III. Related Works
A frequent item set is a set of items that appears at least in a pre-specified number of transactions. Formally, let \( I = \{ i_1, i_2, \ldots, i_m \} \) be a set of items and \( DB = \{ T_1, T_2, \ldots, T_n \} \) a set of transactions where every transaction is also a set of items (i.e. item set).
Given a minimum support threshold \( \text{minSup} \) an item set. Frequent item set mining is the first and the most time consuming step of mining association rules. During the search for frequent item sets the anti-monotone property is used.

3.1 Problem Statement
Pruning search space for high utility item set mining is difficult because of random memory allocation in an UP - Tree. With this reason the time taken to scan the candidate item set UP-Tree is high and also it requires high potential I/O operations. So we are in need of an effective algorithm to overcome these problems.

3.2 High Utility Item sets
A high-utility item set mining model was defined by Yao, Hamilton and Butz [13]. It is a generalization of the share mining model [3, 4]. The goal of high utility item set mining process is to find all item sets that give utility greater or equal to the user specified threshold. The following is the set of definitions given in [13] which we shall illustrate on a small example.

**Definition 1:** The external utility of an item \( ip \) is a numerical value \( yp \) defined by the user. It is transaction independent and reflects importance (usually profit) of the item. External utilities are stored in a utility table. For example, external utility of item B in Table 2 is 10.

**Definition 2:** The internal utility of an item \( ip \) is a numerical value \( xp \) which is transaction dependent. In most cases it is defined as the quantity of an item in transaction. For example, internal utility of item E in transaction T5 is 2 (see Table 1).

<table>
<thead>
<tr>
<th>TID</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>18</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>6</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>3</td>
<td>0</td>
<td>25</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>6</td>
<td>2</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

In an example Table 2 shows that the profit of Utility mining given below,

<table>
<thead>
<tr>
<th>Item</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>Profit($)</td>
<td>3</td>
<td>10</td>
<td>1</td>
<td>6</td>
<td>5</td>
</tr>
</tbody>
</table>

**Definition 3:** Utility function \( f \) is a function of two variables: \( f(x, y) : (R+,R+) \rightarrow R+ \). The most common form also used in this paper is the product of internal and external utility: \( xp \cdot yp \).

**Definition 4:** The utility of item \( ip \) in transaction \( T \) is the quantitative measure computed with utility function from Definition 3 \( u(ip, T) = f(xp, yp), ip, T \).
For example: utility of item E in transaction T5 is \( 2 \times 5 = 10 \).

3.3 Fast Algorithm for Mining Utility-Frequent Item sets
In the fast mining algorithm, 2P-UF utility-frequent item sets mining algorithm described in Section 2 is prove to find all utility-frequent item sets. However, due to the monotone property of quasi attribute support measure it has a few disadvantages which render it unusable for mining of large datasets. The first weak point is the reversed way of candidate generation. 2P-UF algorithm wastes time checking long item sets that are highly unusual to be utility-frequent.
For example, when mining a database with 1000 distinct items (attributes) 2P-UF algorithm first generates and checks all item sets of length 999, then item sets of length 998 etc. Short item sets which have fairly large probability to be utility-frequent come at the very end. Candidate generation function is also slow and inefficient as it
computes intersection of every pair of candidates in each iteration. Moreover, computation of quasi support measure is also inefficient because special data structures (hash trees) cannot be used and we have to scan database once for every candidate. Finally, the two-phase form of the algorithm is space consuming since we have to store all quasi utility-frequent candidates from the first phase to filter them in the second phase. It is possible to avoid this waste of space by merging both phases and filter non utility-frequent candidates in every iteration of the algorithm.

IV. Performance Evaluation
Performance about different sorting methods which were shows the results on Accidents dataset including following sorting methods on UPT&UPG+: lexicographical order (LEX), support descending order (SUP), TWU descending (TWU(D)) and ascending (TWU(A)) orders, and real utility descending (UD) and ascending (UA) orders. For UD and UA, we use the order of actual utilities of 1-PHUIs after the first time database scan. To show the performance of the proposed algorithms, we compared several compared methods and give them new notations as follows: IHUPTWU algorithm, which is proposed in [3] and composed of IHUPTWU-Tree and FP-growth, is denoted as IHUPT&FPG. In the existing algorithms, there are two methods UPT&UPG and UPT&UPG+, that are composed with R-Hashing method.

To compare the performance of FP-Tree and UP-Tree, a method called UPT&FPG is also used with the proposed technique. It generates PHUIs from UP-Tree by FP-Growth directly, in other words, only DGU and DGN are applied. Since R-Hashing technique is used for following purposes. First, less memory required to store the candidate items. Second, less time consuming process while scanning the candidate item in the UP – Tree. Finally low potential I/O operations.

V. Experiment And Result
The experiments were performed on a 2.60 GHz Intel Pentium D Processor with 3.4 GB memory. The operating system is Microsoft Windows 7. The algorithms are implemented in Java language.

The proposed system is tested using foodmart transaction dataset. Following are some reasons why our system outperforms the state of the art algorithms:
1. Utilities of nodes in the global UP-Tree are much less than TWUs of nodes in IHUP-Tree because DGU and DGN algorithms effectively decrease overestimated utilities while constructing the global UP-Tree.
2. UP-Growth algorithm generates less number of candidate item sets than FP-Growth algorithm because UP-Growth algorithm uses DLU and DLN methods for constructing local UP-Tree.

VI. Conclusion
In this paper, we have proposed an efficient algorithm R-Hashing technique for mining high utility item sets from transaction databases. A data structure named UP-Tree is used for maintaining the information of high utility item sets. though, the potential high utility item sets can be efficiently generated from the UP-Tree with only two scans of the database, proposed method decreases the scanning process. In the experiments, both of synthetic and real datasets are used to evaluate the performance of our algorithm. The mining performance is improved significantly since both the search space and the number of candidates are effectively reduced by the proposed strategies. The experimental results show that R-Hashing structure performs the state-of-the-art algorithms significantly in large transactional database.

VI. References


Authors:

**Kona Gunamani** is a student of Computer Science Engineering from Aditya Institute of Technology And Management, Tekkali. Presently pursuing M.Tech (CSE) from this college.

**Vishnu Murty Sivala** is a Sr. Asst. Professor of Aditya Institute of Technology And Management, Tekkali. He is pursuing PH.D in JNTU Kakinada University.