© 2017 IJSRSET | Volume 3 | Issue 5 | Print ISSN: 2395-1990 | Online ISSN : 2394-4099
Themed Section: Engineering and Technology

An Efficient Approach to Mine High Utility Itemsets

Rayhan Ahmed Simanto, Ahmed Abdal Shafi Rasel, Rianon Zaman, Md. Towhidul Islam Robin
Stamford University Bangladesh, Dhaka, Bangladesh

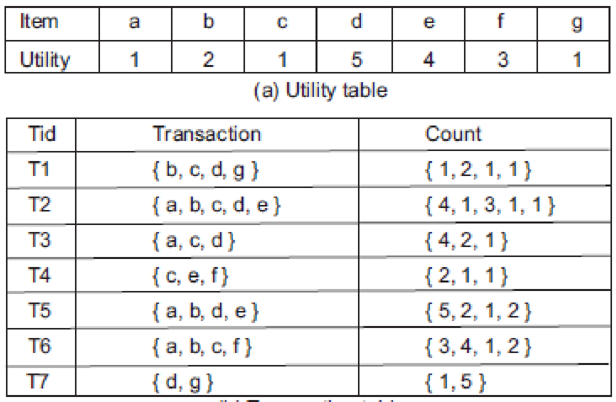
ABSTRACT

High utility itemsets refer to the sets of items with high utility like profit in a database, and efficient mining of high utility itemsets plays a crucial role in many real life applications and is an important research issue in data mining area. To identify high utility itemsets, most existing algorithms first generate candidate itemsets by overestimating their utilities, and subsequently compute the exact utilities of these candidates. These algorithms incur the problem that a very large number of candidates are generated, but most of the candidates are found out to be not high utility after their exact utilities are computed. In this paper, we propose an algorithm, called HUI-Miner (High Utility Itemset Miner), for high utility itemset mining. HUI-Miner uses a novel structure, called utility-list, to store both the utility information about an itemset and the heuristic information for pruning the search space of HUI-Miner. By avoiding the costly generation and utility computation of numerous candidate itemsets, HUI-Miner can efficiently mine high utility itemsets from the utility lists constructed from a mined database.

Keywords: Frequent itemset, Association rule mining, Utility set, Cluster based mining.

I. INTRODUCTION

The rapid development of database techniques facilitates the storage and usage of massive data from business corporations, governments, and scientific organizations. How to obtain valuable information from various databases has received considerable attention, which results in the sharp rise of related research topics. Among the topics, the high utility itemset mining problem is one of the most important, and it derives from the famous frequent itemset mining problem [7, 8]. Mining frequent itemsets is to identify the sets of items that appear frequently in transactions in a database. The frequency of an itemset is measured with the support of the itemset, i.e., the number of transactions containing the itemset. If the support of an itemset exceeds a user-specified minimum support threshold, the itemset is considered as frequent. Most frequent itemset mining algorithms employ the downward closure property of itemsets [4]. That is, all supersets of an infrequent itemset are infrequent, and all subsets of a frequent itemset are frequent. The property provides the algorithms with a powerful pruning strategy. Inthe process of mining frequent itemsets, once an infrequent itemset is identified, the algorithms no longer check all supersets of the itemset. For example, for a database withn items, after the algorithms identify an infrequent itemset containing k items, there is no need to check all of its supersets, i.e., $2(n-k) - 1$ itemsets. Mining of frequent itemsets only takes the presence and absence of items into account. Other information about items is not considered, such as the independent utility of an item and the context utility of an item in a transaction. Typically, in a supermarket database, each item has a distinct price/profit, and each item in a transaction is associated with a distinct count which means the quantity of the item one bought. To calculate support, an algorithm only makes use of the information of the first two columns in the transaction table, the information of both the utility table and the other columns in the transaction table are discarded. However, an itemset with high support may have low utility, or vice versa. For example, the support and utility of itemset {bc} appearing in T1, T2, and T6 are 3 and 18 respectively. appearing in T2 and T5 are 2 and 22. In some applications, such as market analysis, one may be more interested in the utility rather than support of itemsets. Traditional frequent itemset mining algorithms cannot evaluate the utility information about itemsets. Like frequent itemsets, itemsets with utilities not less than a user-specified minimum utility threshold are generally valuable and interesting, and they are called "high utility itemsets". To mine all high utility itemsets from a database is very intractable, because the downward closure property of itemsets no longer holds for high utility itemsets. When items are appended to an itemset one by one, the support of the itemset monotonously decreases or remains unchanged, but the utility of the itemset varies irregularly. For example, for the database in Fig. 1, the supports of {a}, {ab}, {abc}, and {abcd} are 4, 3, 2, and 1, but the utilities of these itemsets are 16, 26, 21, and 14, respectively. Suppose the threshold is 20, and then high utility {abc} contains both high utility {ab} and low utility {a}. Therefore, the pruning strategy used in the frequent itemset mining algorithms becomes invalid. Recently, a number of high utility itemset mining algorithms have been proposed [25, 18, 14, 5, 23, 22]. Most of the algorithms adopt a similar framework: firstly, generate candidate high utility itemsets from a database; secondly, compute the exact utilities of the candidates by scanning the database to identify high utility itemsets. However, the algorithms often generate a very large number of candidate itemsets and thus are confronted with two problems: (1) excessive memory requirement for storing candidate itemsets; (2) a large amount of running time for generating candidates and computing their exact utilities.



Item	a	b	c	d	e	f	g
Utility	1	2	1	5	4	3	1

(a) Utility table

Tid	Transaction	Count
T1	{ b, c, d, g }	{ 1, 2, 1, 1 }
T2	{ a, b, c, d, e }	{ 4, 1, 3, 1, 1 }
T3	{ a, c, d }	{ 4, 2, 1 }
T4	{ c, e, f}	{ 2, 1, 1 }
T5	{ a, b, d, e }	{ 5, 2, 1, 2 }
T6	{ a, b, c, f }	{ 3, 4, 1, 2 }
T7	{ d, g }	{ 1, 5 }

(b) Transaction table

Figure 1: Sample Itemsets

II. PROBLEM FORMULATION

Let I={i1, i2, i3, . . . , in} be a set of items and DB be a database composed of a utility table and a transaction table. Each item in I has a utility value in the utility table. Each transaction T in the transaction table has a unique identifier (tid) and is a subset of I, in which each item is associated with a count value. An itemset is a subset of I and is called a k-itemset if it contains k items. Definition 1. The external utility of item i, denoted as (i), is the utility value of i in the utility table of DB. denoted as u(i, T), is the product of iu(i, T) and eu(i), where u(i, T) = iu(i, T) × eu(i). For example, in Fig. 1, eu(e) = 4, iu(e, T5) = 2, and u(e, T5)= iu(e, T5) × eu(e) = 2 × 4 = 8. Definition 4. The utility of itemset X in transaction T, denoted as u(X, T), is the sum of the utilities of all the items in X Σ in T in which X is contained, where u(X, T) = i ∈ X ∧ X⊆T u(i; T). Definition 5. The utility of itemset X, denoted as u(X), is the sum of the utilities of X in all the transactions containing X in DB, w here u(X) =

Σ T∈DB∧X⊆T u(X; T).

For example, in Fig. 1,

{ae}, T2) = u(a, T2) + u(e, T2)

= 4 × 1 + 1 × 4 = 8, and u({ae}) = u({ae}, T2) + u({ae}, T5) = 8 + 13 = 21.

III. UTILITY-LIST STRUCTURE

Table In our HUI-Miner algorithm, each itemset holds a utilitylist. Initial utility-lists storing the utility information about a mined database can be constructed by two scans of the database. Firstly, the transaction-weighted utilities of all items are accumulated by a database scan. If the transaction-weighted utility of an item is less than a givenminutil, the item is no longer considered according to Property 1 in the subsequent mining process. For the items whose transaction-weighted utilities exceed the minutil, they are sorted in transaction-weighted-utility-ascending order.

For the database in Fig. 1, suppose the minutil is 30, and then the algorithm no longer takes items f and g into consideration after the first database scan. The remaining items are sorted: e<c<b<a<d. 1: Sample transactions with itemsets.

IV. SEARCH SPACE

The search space of the high utility itemset mining problem can be represented as a set-enumeration tree [19]. Given a set of items I = {i1, i2, i3, . . . , in} and a total order on all items (suppose i1 < i2 < · · · < in), a set-enumeration.

Tree representing all itemsets can be constructed as follows. Firstly, the root of the tree is created; secondly, then child nodes of the root representing n 1-itemsets are created, respectively; thirdly, for a node representing itemset {is · · · ie} (1 ⩽ s ⩽ e < n), the (n−e) child nodes of the node representing itemsets {is · · · iei(e+1)}, {is · · · iei(e+2)}, ...,{is · · · iein} are created. The third step is done repeatedly until all leaf nodes are created. For example, given I = {e, c, b, a, d} and e < c < b < a < d, a set-enumeration tree representing all itemsets of I is depicted in Fig. 2.only small portion of items). In our experiments we chose different dataset with different properties, to prove the efficiency of the algorithms, Table 3 shows the datasets and the characteristics such as length and type.

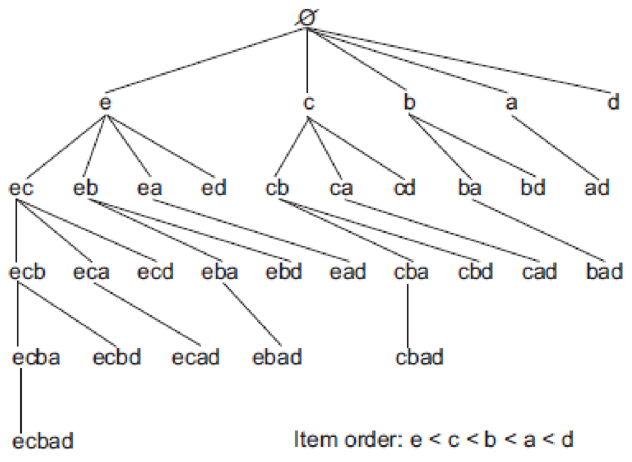


Figure 2: Set-Enumeration Tree

Therefore, the iutil of the element associated with T2 in the utility-list of {eba} is: u({eba}, T2) = u({eb}, T2) + u({ea}, T2) - u({e}, T2) = 6 + 8 - 4 = 10. That associated with T5 is: u({eba}, T5) = u({eb}, T5) + u({ea}, T5) - u({e}, T5) = 12 + 13 - 8 = 17. The values of u({eb}, T), u({ea}, T), and u({e}, T) can be accessed from the utilitylists of {eb}, {ea}, and {e}, respectively. Suppose itemsets Px and Py are the combinations of itemset P with items x and y (x is before y.), respectively, and P.UL, Px.UL, and Py.UL are the utility-lists of itemsets P, Px, and Py. Algorithm 1 shows how to construct the utility-list of itemset Pxy. The utility-list of a 2-itemset isconstructed when P.UL is empty, namely when P is empty,and the utility-list of a k-itemset (k ⩾ 3) is constructed whenP.UL is not empty. Note that element E in line 5 can always be found out when P.UL is not empty, because the tid sets in both Px .UL and Py. UL are subsets of the tid set in P.UL. The utility-lists of all the itemsets with {eb} as prefix constructed by Algorithm 1 are showed in Fig. 7(b). Thus far, we have illustrated how to construct the utility list of an itemset. When does HUI-Miner construct the utility-list of an itemset and how does HUI-Miner judge whether or not the utility-list of an itemset should be constructed, which will be illuminated in the next section.

V. PROPOSED ALGORITHM

Following algorithm shows the pseudo-code of HUI-Miner. For each utility-list X in ULs (the second parameter), if the sum of all the iutils in X exceeds minutil, and then the extension associated with X is high utility and outputted. According to Lemma 1, only when the sum of all the iutils and rutilsin X exceeds minutil should it be processed further. When the initial utility-lists are constructed from a database, they are sorted and processed in transaction-weighted utility-ascending order. Therefore, all the utility-lists in ULs are ordered as the initial utility-lists are. To explore the search space, the algorithm intersects X and each utility-list Y after X in ULs. Suppose X is the utility-list of itemset Px and Y that of itemset Py, and then construct(P.UL, X, Y) in line 8 is to construct theutility-list of itemset Pxy as stated in Algorithm Finally, the set of utility-lists of all the 1-extensions of itemset Pxis recursively processed. Given a database and a minutil, The sums of the iutils and rutils in the utility-list of an itemset can be computed by scanning the utility-list. To avoid utility-list scan, in the process of constructing a utilitylist, HUI-Miner simultaneously accumulates the iutils and rutils in the utility-list. In addition, there is also no need to bind each itemset to its utility-list. The itemsets represented by all child nodes of a node in a set-enumeration tree have the same prefix itemset. Therefore, for a 1-extension, its extended item can be separated from its prefix itemset. We lightly modify the utility-list structure when implementing HUI-Miner. For example, the utility-implemented as those on the first line in a utility-list stores the extended item and the sums of the iutils and rutils, and the prefix itemset is stored independently.

```
Input: P.UL, the utility-list of itemset P, initially
       empty;
       ULs, the set of utility-lists of all P's
       1-extensions;
       minutil, the minimum utility threshold.
Output: all the high utility itemsets with P as prefix.
1  foreach utility-list X in ULs do
2      if SUM(X.iutils)≥minutil then
3          output the extension associated with X;
4      end
5      if SUM(X.iutils)+SUM(X.rutils)≥minutil then
6          exULs = NULL;
7          foreach utility-list Y after X in ULs do
8              exULs = exULs+Construct(P.UL, X, Y);
9          end
10         HUI-Miner(X, exULs, minutil);
11     end
12 end
```

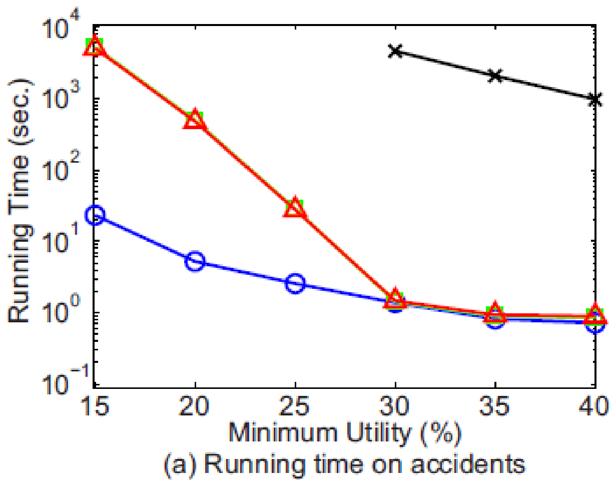
Another important detail is the processing order of items. In previous algorithms, such as IHUPTWU and UP-Growth, items are sorted transaction-weighted-utility-descending order, which can reduce the size of prefix-trees used in these algorithms.

VI. RUNNING TIME

We considered the following factors for creating our new scheme, which are the time and the memory consumption, these facto The running time of the four algorithms on all databases is depicted in Running time was recorded by the "time" command, and it contains input time, CPU time, and output time. The output results of the four algorithms .We terminated a mining task, once its running time exceeds 10000 seconds.

When measuring running time, we varied the minutil for each database. The lower the minutil is, the largerthe number of high utility itemsets is, and thus the more the running time is. For example, for database chain in when the minutils are 0.004% and 0.009%, the numbers of high utility itemsets are 18480 and 4578, and= the running times of HUI-Miner are 580.9 seconds 445.1 seconds, respectively. In addition, the curve for UPGrowth almost totally overlaps the curve for UP-Growth+ in the running time of IHUPTWU for any minutil exceeds 10000 seconds for database chess, and thus there isno curve for IHUPTWU.

For almost all databases and minutils, HUI-Miner performs the best. HUI-Miner is almost two orders of magnitude faster than the other algorithms for dense databases. For example, the running times of HUI-Miner and UP-Growth+ are 35.8 seconds and 6302.3 seconds for database mushroom when the minutil is 2%. HUI-Miner is slower than UP-Growth+ for high minutils, and we found out in this case that UP-Growth+ generates very few candidate itemsets (only 2007 candidates when the minutil is 0.6%); however, for low minutils, HUI-Miner is even an order of magnitude faster than UP-Growth+ (UP-Growth+ generates 178128 candidates when the minutil is 0.35%.).



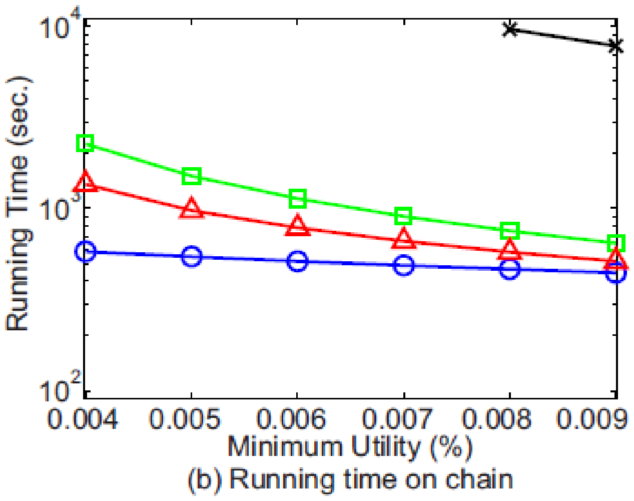
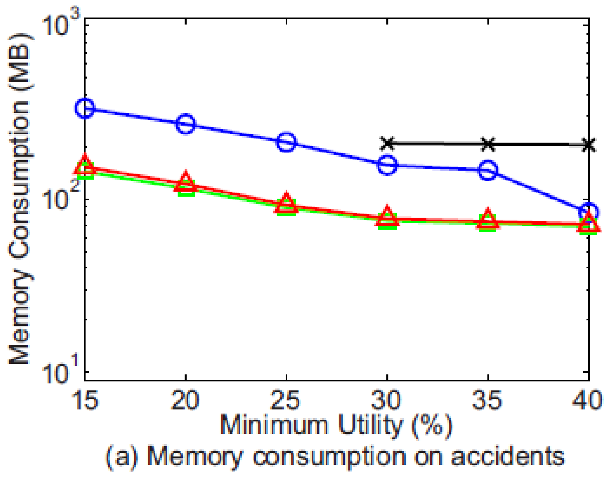


Figure 3: Running time on different Dataset

VII. MEMORY CONSUMPTION

Generally, the memory consumption of these algorithms is proportional to the number of candidate itemsets they generate. For example, for database T10I4D100K, IHUPTWU generates 3826202 candidate itemsets and consumes 109.0 MB of memory while UP-Growth+ generates 1007150 candidate itemsets and consumes 50.22 MB of memory, when the minutil is 0.005%. The number of high utility itemsets is only 313509 for the mining task. HUI-Miner neither generates nor stores candidate itemsets, and thus it consumes only 23.62 MB of memory.



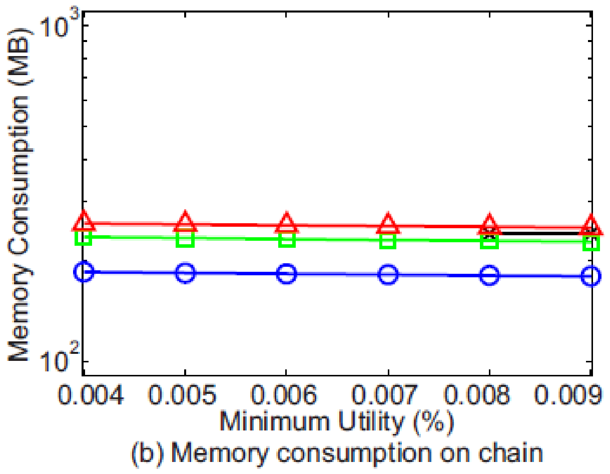


Figure 4: Memory Consumption on different Dataset

VIII. CONCLUSION

In this paper, we have proposed a novel data structure, utility-list, and developed an efficient algorithm, HUIMiner, for high utility itemset mining. Utility-lists provide not only utility information about itemsets but also important pruning information for HUI-Miner. Previous algorithms have to process a very large number of candidate itemsets during their mining processes. However, most candidate itemsets are not high utility and are discarded finally. HUI-Miner can mine high utility itemsets without candidate generation, which avoids the costly generation of time and memory consumption.

IX. REFERENCES

[1]. Frequent Itemset Mining Dataset Repository. http://fimi.ua.ac.be/, 2012.

[2]. NU-MineBench: A Data Mining Benchmark Suite. http://cucis.ece.northwestern.edu/projects/DMS/MineBench.html, 2012.

[3]. Valgrind: A GPL-d System for Debugging and Profiling Linux Programs. http://valgrind.org/, 2012.

[4]. R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In Proc. Int'l Conf.Very Large Data Bases, pages 487-499, 1994.

[5]. C. F. Ahmed, S. K. Tanbeer, B.-S. Jeong, and Y.-K.Lee. Efficient tree structures for high utility pattern mining in incremental databases. IEEE Transactions on Knowledge and Data Engineering, 21(12):1708-1721, 2009.

[6]. B. Barber and H. J. Hamilton. Extracting share frequent itemsets with infrequent subsets. Data Mining and Knowledge Discovery, 7(2):153-185, 2003.

[7]. A. Ceglar and J. F. Roddick. Association mining. ACM Computing Surveys, 38(2), 2006.

[8]. J. Han, H. Cheng, D. Xin, and X. Yan. Frequent pattern mining: Current status and future directions. Data Mining and Knowledge Discovery, 15(1):55-86, 2007.

[9]. J. Han, J. Pei, Y. Yin, and R. Mao. Mining frequent patterns without candidate generation: A frequent pattern tree approach*. Data Mining and Knowledge.

[10]. J. Hu and A. Mojsilovic. High-utility pattern mining: A method for discovery of high-utility item sets. Pattern Recognition, 40(11):3317 - 3324, 2007.

[11]. Y.-C. Li, J.-S. Yeh, and C.-C. Chang. Direct candidates generation: A novel algorithm for discovering complete share-frequent itemsets. In Proc. Fuzzy Systems and Knowledge Discovery, pages 551-560, 2005.