IJST EXPLORE International Journal of Science and Technology
CENTRE OF PROFESSIONAL RESEARCH PUBLICATIONS
ISSN 2049-7318

International Journal of Science and Technology Volume 5 No. 11, November, 2016

Mining High Utility Item set Using Hash-Map without Candidate Generation

Sadhana G, Prof. Muralidhar A
Vit University.Chennai Campus

ABSTRACT

High utility itemsets mining is used to extract useful profitable items from a set of transactions. Though there are algorithms like Apriori and Association rule mining, these algorithms generate only the frequent itemsets, which enables to discover only interesting patterns and help in decision making. High Utility Itemsets mining on the other hand, puts light on profitable set of frequent itemsets. In order to extract these high utility patterns, several algorithms generate large number of candidate itemsets, most of which are not frequent. This not only increases the memory space for storage but also takes longer time to compute. In this paper, we extract high utility itemsets without candidate generation with the help of an algorithm called HUI-Extractor (High Utility Itemsets Extractor) implemented with the help of hash-map. The data structure hash-map exclusively stores the information about each item and its corresponding utility value in the form of key-value pair. This provides easy access to every item individually, as when compared to list data structure. Experiments are conducted to evaluate the performance of proposed algorithm in terms of memory occupied and the number of high utility itemsets generated.

Keywords: *High Utility Itemsets, Total Utility, Transaction Weighted Utility, Hash-Map Structure*

1. INTRODUCTION

High utility Itemsets mining is useful in the process of decision making for several industries like retail industries, web services, supply chain industries as items in these industries vary in many aspects in real-world. Apriori is an algorithm used to find the frequent itemsets. It first generates candidate itemsets by joining the database with itself to obtain possible combination of itemsets. Once the candidate itemsets are obtained, interestingness measures called the Support and Confidence are applied to the obtained itemsets. Those candidates that satisfy the minimum threshold are retained, the rest are discarded. This algorithm has complexity in exponential terms as it generates subsets of items in candidate generation. However, apriori algorithm can extract only frequent itemsets, which is not sufficient to answer all the questions such as, what would be the profit after the sale of frequent itemsets? Though, knowing the frequent itemsets in the transaction over a period of time can help in handling the demand-supply balance, to identify profit we will need different measures. Thus, introducing high utility itemsets. The high utility itemsets considers the frequent itemsets as well as the utility value associated with them. Utility value signifies the profit obtained when one quantity of the item occurs in a transaction. For example, when a customer buys ten pens and twenty pencils at 15 rupees and 5 rupees each respectively, the total amount the customer pays in this transaction is 250 rupees. In the same store, let another customer buy a book for 1000 rupees. In these two transactions the frequently occurring item would be the pens and pencils, but they do not account for profit. Whereas in the second transaction although the quantity is one, the item is not included in the frequently occurring itemsets, the profit is more as compared to the previous transaction. This is High Utility Itemsets.

Algorithms used to implement high utility itemsets so far have used two important steps. First, the algorithms mine the entire database to obtain up-to-date high utility itemsets. Since this is similar to apriori like algorithm, it scans the entire database once to get the support count of each item. Next in the second phase, a list structure is used to organize the transactional data. With the help of this list, the itemsets are pruned without candidate itemsets generation. This has definitely increased the processing of mining algorithms. But, one short coming of list structure is it would be difficult to map the profit or utility of each item to its corresponding term. Specially, it is not efficient to use list in frameworks like R and map-reduce. Thus, in this paper, we have proposed a method of using hasp-map. Hash-map is a data structure that stores any element in terms of 'key and value' pair. Thus, we store the items as key and its corresponding utility value as value. This enables us to access the terms more easily as compared to list. Hasp-map can be used in any platform as easily as used in JAVA platform. There are inbuilt packages in R, that enables to build data in hash-map format.

We show the implementation of extracting high utility itemsets from a sample dataset called retail industry dataset. The following sections of the paper briefly explain about the common terms used for calculation in the algorithm. Third section explains the algorithm, followed by the fourth section that shows the result and analysis of the performed experiment. The last section describes the conclusion and references.

2. RELATED TERMINOLOGY

2.1. Problem Statement

Let *i*={i1,i2,i3,...in*}* be a set of items involved in transactions, ד be a set of transactions and ט be the utility value for each item in *i*. Each item in *i* has a utility value ט in the utility table. Each transaction ד contains an itemset that is a subset of items in *i*. When an itemset contains 'k' items in it, it called k-itemset.

Statement 1 - The global utility **g**[υ] of an item *i*, is the profit of the item in the corresponding utility set, represented as **g**[*i*]

Statement 2 - The local utility **l**[υ] of an item *i* is the quantity of the item in the transaction ד to which it belongs, represented as **l**[i]

Statement 3 - The utility of an item can be calculated by multiplying the local utility **l** and global utility **g**, represented as, υ [*i*]=**l**[*i*]*__g__[*i*]

Statement 4 - The utility of an itemset in a transaction ד , represented as υ [*i*, ד], is the sum of all the item utilities in the transactions that contain *i*, υ [*i*, ד]=∑ υ [*i*, ד] }

Statement 5 - The utility of an itemset, represented as υ [*i*], is the sum of utilities of *i* in all transactions containing *i*, ∑ υ [*i*]

Statement 6 - The total utility of a transaction, tυ, is the sum of all the items in ד , ∑ υ [*i*,ד] where *i* is the set of items in the entire transaction

Items	TU	Profit
3 5 1 2 4 6	30	1 3 5 10 6 5
3 5 2 4	20	3 3 8 6
3 1 4	8	1 5 2
3 5 1 7	27	6 6 10 5
3 5 2 7	11	2 3 4 2

Fig.1 Transactional Database with Items and corresponding Utility values

Once these information is obtained, the high utility itemsets can be obtained. An itemset *i*is said to be an high utility itemset if and only if the utility of the itemset, υ [*i*] is greater than the user specified minimum threshold level. This minimum threshold *minThresh* acts as a filter to extract only the high utility itemsets. Given a database and *minThresh*, task is to extract the high utility itemsets.

For example, consider the transactional database as shown in figure1. Every row indicates a new transaction. The first column indicates the items purchased in transaction 1. The corresponding profit or utility of the items is shown in the third column of the table. That is, for example, in the first transaction the local utility **l**[3]=1. The global utility **g**[3]= 1. Utility of an item, for instance, item 5 is the product of **l**[5] and **g**[5], which is 1*3 = 3 for transaction 1. When a combination of itemset is considered, for example, {3,5,2}, the utility of an itemset can be calculated as the sum of utilities of individual items in the set. The utility of this itemset {3,5,2} is 1+3+10 =14 in transaction 1. The total utility of a transaction is also required as to generate the high utility itemsets. This tυcan be calculated as, say for transaction 4, 6+6+10+5 = 27, which is indicated by the second column in the figure1.

2.2. Related Work

For frequent itemsets with utilities greater than *minThresh*, are generally said to have interestingness. Previous work in algorithms like apriori have shown downward closure property. But, for mining high utility itemsets in very interactive way, this property cannot exist. When an item is added to an itemset one-by-one, the support of the itemset continuously goes down or does not change. When utility of an itemset is considered, it varies in a random fashion. For example, the support count of the itemsets {1},{1,2},{1,2,5} and {1,2,5,6} are 8, 6, 4 and 3 respectively, their utilities are 23, 8,27 and 18 and assuming a *minThresh* of 20. Then, the high utility of {1,2,5} contains both high utility {1} and low utility {1,2}. Thus, pruning strategy does not work for mining high utility itemsets.

For high utility itemset mining, [2] Liu has proposed important property to prune the mining problem of high utility itemsets.

Statement 7 - Transaction-weighted utility of an itemset *i*, denoted as twu(*i*), is the sum of the utilities of all the transactions containing *i*, where twu(*i*) = ∑ tυ}

Statement 8 - If twu(*i*} is less than a given "minThresh", all supersets of X are not high utility itemsets.

For high-utility itemsets mining, the database is scanned twice. the input is obtained in the form shown in Fig 1. The input is initially stored in a hash-map structure in the form of buckets, which becomes easy to access the product and corresponding utility value. From this hash table of 1-itemsets, the database is scanned once to extract items satisfying "minThresh" and sorted in ascending order, based on the transaction weighted utility(twu).In the second scan, itemsets of k-order are obtained by avoiding candidate generation, as discussed further.

The algorithms like FUP Mining, generates a set of frequent items in two passes. The database is scanned once while building the tree. Every item in the list, in its corresponding transaction are put in the tree. After the tree is built, if the itemset is likely to be high utility itemset then a conditional FP-Prefix tree is constructed for these items.In the next step, the database is again scanned to obtain a set of high utility item candidates. This surely reduces the number of scans and candidate sets as compared to traditional apriori algorithm, and outperforms in quality and complexity. Still, FUP Mining process generates way too large amount of candidate itemsets.

3. HASH-MAP STRUCTURE

Hash-map data structure is efficient in storing data that has 2 parts. Every element has a *key* and a *value* pair. For mining high utility itemset, we require an item and its corresponding profit. Previous work have used list for storing the item-profit. It becomes a little complex to identify the exact profit of an item in list, though it is most accurate, it is not faster in

computation. Hash map on the other hand, stores the information in key value pair. This increases the speed of access every time the database is scanned.

This hash-map structure has a hashing based technique for storing and retrieving the key-value pairs. This acts as an index for the values to be retrieved from the hash-map. The Hash-map structure contains three main concepts: hash-keys, hash-values and hash-buckets. The hash-key here is referred to be the product number, the hash value is the corresponding profit or utility value the product. This key-value pair is stored in the hash-bucket which is the number of transactions. The number of hash-buckets 'k' is equal to the number of transactions, ד .

	1	**10**
T1	2	6
	3	3
	4	5
	5	5
	6	1
T2	2	8
	3	3
	4	6
	5	3
T3	1	5
	3	1
	4	2
T4	1	10
	3	6
	5	6
	7	5
T5	2	4
	3	2
	5	3
	7	2

Fig 2. Initial Hash Table indicating hash key, hash value and hashbucket.

3.1. HASH-MAP Structure with k-itemsets

Once the hash-map with 1-itemset is constructed, the database is scanned once, the first scan. In this scan the items whose twu (Transaction Weighted Utility) is more than minThresh is retained, the rest of the items are discarded. The twu of an item can be calculated as, for example consider item 3, twu(3) = 30+20+8+27+11 = 96 and twu(1)=65. Similarly, the transaction weighted utility of all items are calculated. Consider an assumption that the "minThresh" is 50. Then the items 2 and 6 are discarded. The rest of the items in the hash table are sorted in an ascending order as per their twu values. That is, 2<1<5<3, as shown in figure 3. The hash table shows the 1-itemset key-value pair.

T1	2	6
	1	10
	5	5
	3	3
T2	2	8
	5	3
	3	3
T3	1	5
	3	1
T4	1	10
	5	6
	3	6
T5	2	4
	5	3
	3	2

Fig 3. Sorted Hash Table.

Once the 1-itemsets are formed in hashed table, the 2-itemsets and 3-itemsets are formed (for this example, upto 3-itemsets). Since the table formed in 1-itemset hash table is sorted, to obtain higher order k-itmesets is easier.

Statement 9 - The items that occur in a transaction ד , in the sorted hash table after a set of items *i*, can be represented as ד |*i*.}

The database is scanned once again, the second scan, to find the k-itemsets. The items that are encountered after an itemset *i*, are summed up to get the differential utility measure. As the k-itemsets are formed, each set will contain three parts namely: transaction ID, *i*util and ρutil. *i*util is the utility of the itemset in the transaction ד , and ρutil represents the differential utility. For example, in the 1-itemset hash table, the *i*util[2,T2] = 8 and ρutil [2,T2]= υ [5,T2] + υ [3,T2]=3+3=6. The set of 1-itmesets in Fig 3 can be similarly calculated for *i*util and ρutil as shown in Fig 4. The obtained information is also stored in the form of hash-map structure. Since hash-map can be called a type of indexing, it is very efficient to identify and locate an item in the dataset in less time as compared to lists or other data structures.

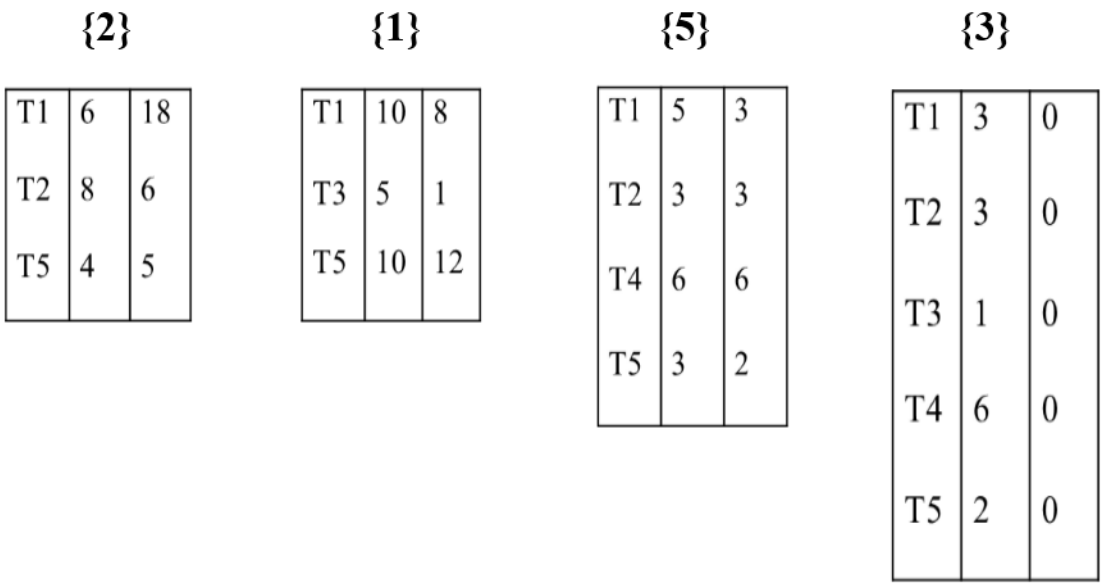

Fig 4. Initial utility values indicating *i*util and ρutilfor 1-itemsets.

3.2. HASH-TABLE for 2-itemsets:

The 1-itemsets are joined with itself to produce high-order itemsets. While joining, the subsets whose transaction weighted utility(twu) is less than the defined "minThresh" is discarded. This follows the property of pruning. That is, itemsets whose items are not frequent to be nominated for high-utility item, is not used while performing join operation. All these operations take place during the second scan, an extra scan is not required for identifying 2-itemsets. An item, say *{2}* is combined with every other item in the hash-table. The 2-itemsets can be generated by directly intersecting with the common transactions that are encountered, with the items being subset. For each common transaction in the two items hash-table, the items are added to the itemset. There is no need for scanning the database once again because, if the length of hash-table for 1-itemsets of one item is 'm' and the length of hash-table for other item is 'n', then at most (m+n) reads of database are required because the items are ordered, thus reducing the complexity. For example, when *{2}* is compared with *{1}* then the transactions are compared as shown in Fig 5(a), and the utilities of 2-itemsets produced are shown in Fig 5(b).

When an identical transaction is traversed, the information of the items are combined. This 2-itemset hash-table also has three parts, the transaction ID, *i*util and ρutil. The transaction ID is common to both the items, hence the same name is retained. The *i*util field is the sum of the utilities of the items, and the ρutil is the value that is same the second element being combined.

For example, as shown in Fig 5., the item prefixed is *{2}*. Fig 5(a) shows that *{2,1}* is obtained by identifying the similar transactions in both the hash-tables. Item *{2}* has transaction IDs *{*T1,T2,T5*}* and item *{1}* has transaction IDs *{*T1,T3,T4*}*. The identical transaction is T1. Thus, the itemset *{2,1}* can be obtained by adding the utility values of both the items within the itemset. That is, *i*util*{2,1}* = *i*util*{2}*+*i*util*{1}* = 6+10 =16. The ρutil *{2,1}* is the ρutil value of *{1}* = 8. Similarly, the item *{2}* is combined with other items in the hash-table.

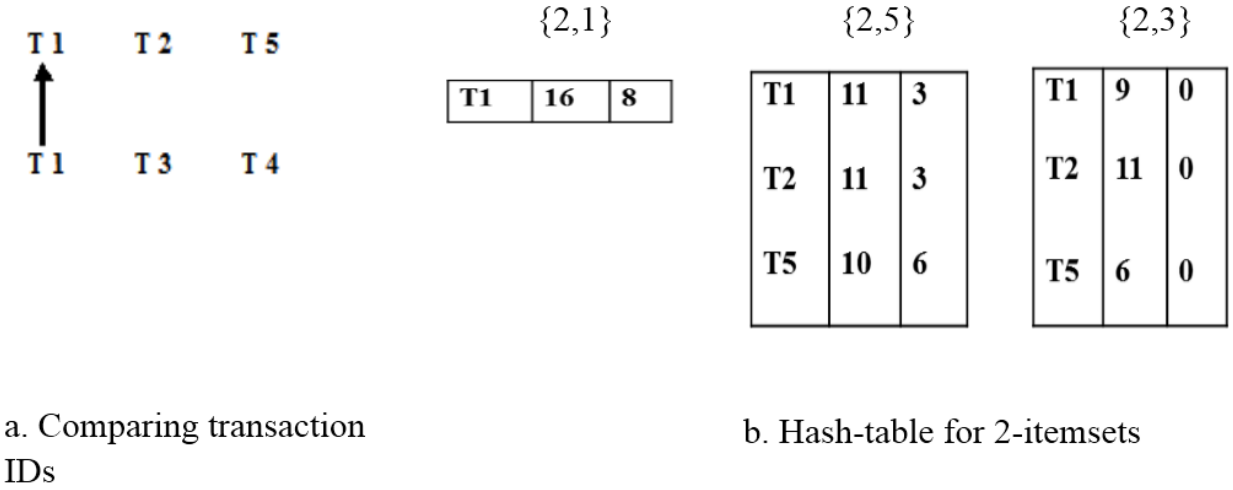

Fig 5. 2-itemset hash-table

3.3. HASH-TABLE for k-itemsets for k>3:

The 3-itemsets hash-table is obtained in a similar way as 2-itemset hash-table was constructed. The items match their common transactions, and calculate the *i*util value and ρutil value. The prefix is kept constant and is used to combine with other items. Here the prefix takes the size of 2, since the items are taken from the 2-itemset hash-table. For example, consider the 2-itemset *{*2,1*}* and *{*2,5*}*. As in forming the 2-itemset, the identical transactions are identified. In this case, the identical transaction is T1. When calculated the *i*util, it leads to 27 and ρutil leads to 3. But when the total utilities of the items are considered from Fig 2, the item utilities vary. This conflict occurs due to direct calculation. Thus, to avoid the conflict the total utilities of the items are directly considered from the initial hash-table. Thus, as shown in Fig 6. the 3-itemset *{*2,1,5*}* can be obtained by adding the utility values of *{*2,1*}* and *{*2,5*}*, which leads to 21. ρutilis the same as *{*2,5*}*, which is 3. In this method, the hash-table that calculated by directly adding the item utility is miscalculation because, the itemsets *{*2,1*}* and *{*2,5*}* both contain the utility values of *{*2*}*. Therefore, it is right to calculate the *i*util value for 3-itemset, in this example, as *i*util*{*2,1,5*}* in T1 is υ*{*2,1*}*[T1]+υ*{*2,5*}*[T1]-υ*{*2*}*[T1]=16+11-6=21. By this access to itemsets *{*2,1*}*,*{*2,5*}* and *{*2*}* in 3-itemsets hash-table is easy. The 3-itemsets formed are then passed to the next phase, for extraction of high utility itemsets. As in apriori algorithm the rules are generated of the frequent itemsets based on measures such as support and confidence, the high utility itemsets are extracted from the hash-tables by applying the HUI-Extractor algorithm as described in further sections.

So far, the process of constructing hash-tables for utility itemsets was described. The next step is to formulate an algorithm such that it decides whether or not the itemsets in the hash-table be considered as High-utility Itemset, that is illuminated in the next section.

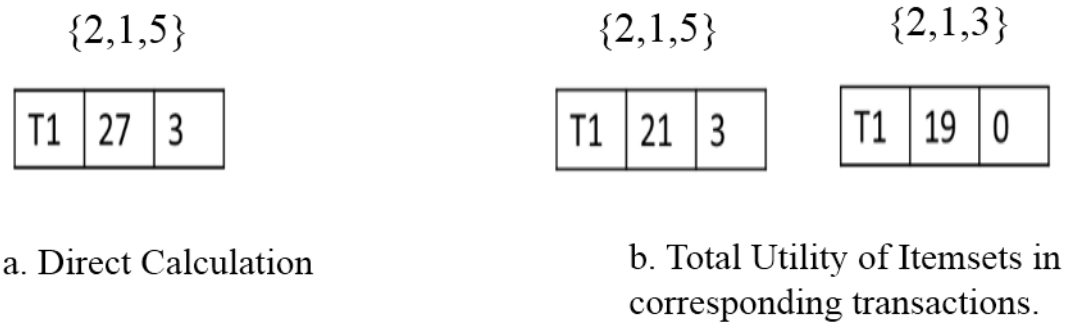

Fig 6. 3-itemset hash-table

4. HIGH UTILITY ITEMSET EXTRACTOR

Algorithms like Apriori and FP-Growth are used to extract frequent itemsets from the database. These algorithms though do not provide any information related to profit of the items, the procedure adapted to extract the items provides a methodology to extend the same to obtain utility information about the frequent itemsets. In this section, the algorithm used to extract the high utility itemsets from the hash-table is described. The pruning strategy used in apriori and FP-Growth algorithms is used in this algorithm to reduce the search space complexity. Usually, a database with 'n' transactions take 2^n comparisons, as the subsets also are considered. Since, in this paper, hash-tables are constructed that contain hash-key and hash-value, the subsets of itemsets are not considered, thereby reducing the complexity considerably.

5. ALGORITHM

Input. Hash-table of k-itemsets and their corresponding utility values v. A set of 1-itemset hash-tables ”minThresh” value, minimum Threshold value.
Output. High utility Itemsets, HUI

```
1: foreach hash-table itemsets do
2: if SUM(_util)_ minThresh then
3:     output the prefix associated with i;
4: endif
5: if SUM(iutil)+SUM(ρutil) ≥ minThresh then
6:     g[υ] = NULL;
7:     foreach hash-table i+1 after i in i do
8:       g[υ] = g[υ]+Hash-Table(i,i,i+1);
9:     endfor
10:    HUI-Extractor(i, g[υ]s, minThresh);
11: endif
12: endfor
```

5.1. HUI EXTRACTOR ALGORITHM

In order to improve the efficiency the utility mining process, the calculation of *i*util and ρutil can be exploited in the itemsets in hash-table. The algorithm above shows the pseudo code of the HUI-Extractor algorithm. The algorithm takes as input a set of itemsets as constructed during the two scans of the database. A set of itemsets within the hash-table, which gives the items and their corresponding profit value. A *minThresh* value, that is used as a threshold filter, based on which the algorithm segregates the itemsets of high utility. The algorithm begins by checking for each itemset *i*in the hash-table. If the itemset satisfies the *minThresh* then the prefix that is associated with

that itemset is outputted. That is, in the example shown above, for the itemset *{*2,1,5*}*,the itemset has a prefix *{*2*}*. Also, while considering the *minThresh*, the sum of all *i*utils and ρutils must exceed the threshold value that is specified to process further. As discussed earlier, the items in initial hash-table are sorted according to the transaction weighted utility. thus, once the high utility itemsets are extracted, the obtained items are also ordered as the initial hash-table items are. Thus, to explore the search space, the algorithm intersects itemset i+1 that is followed by i in the hash-table. The Hash-Table procedure in the algorithm in line 8, is the procedure as discussed above, to find the twu and prepare the initial hash-table. Although no explicit algorithm is mentioned in this paper to create the hash-table, the process of construction of the hash-table is described. Finaaly, all the itemsets are prefixed with an item and recursively processed. With the given set of data and a *minThresh* this algorithm will output all the high utility itemsets.

4. EXPERIMENT

In order to perform experiment on this idea of hash table, a dataset of retail was considered. The reatil dataset is a set of transactions, that indicate the items purchased in one transaction and their corresponding profits or utility. The dataset contains 88,162 transactions, with a total of 16,740 items and their corresponding profit values. The average transaction length is 10 and the maximum length of the transaction is 76. The process of constructing hash-table and extracting high utility itemsets was carried out in RStudio tool version 3.2.3. RStudio is a tool that works by importing packages. It is simple and efficient to carry out mining algorithms. It can handle data up to few gigabytes.

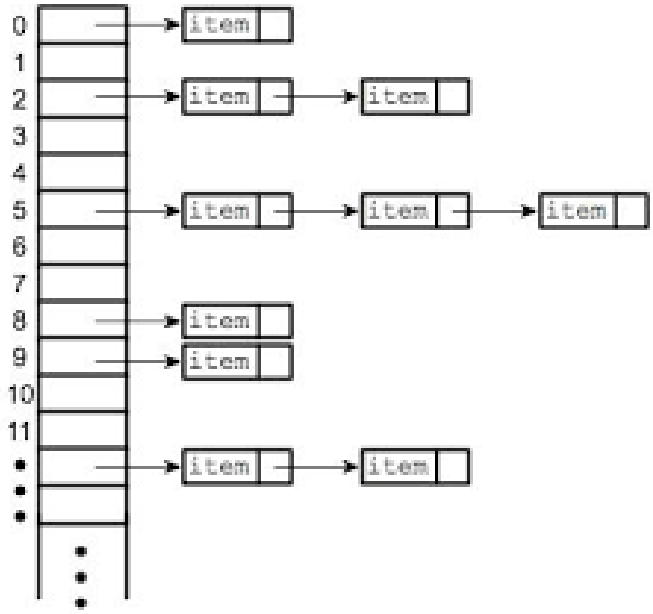

Fig 7. Hash Table structure

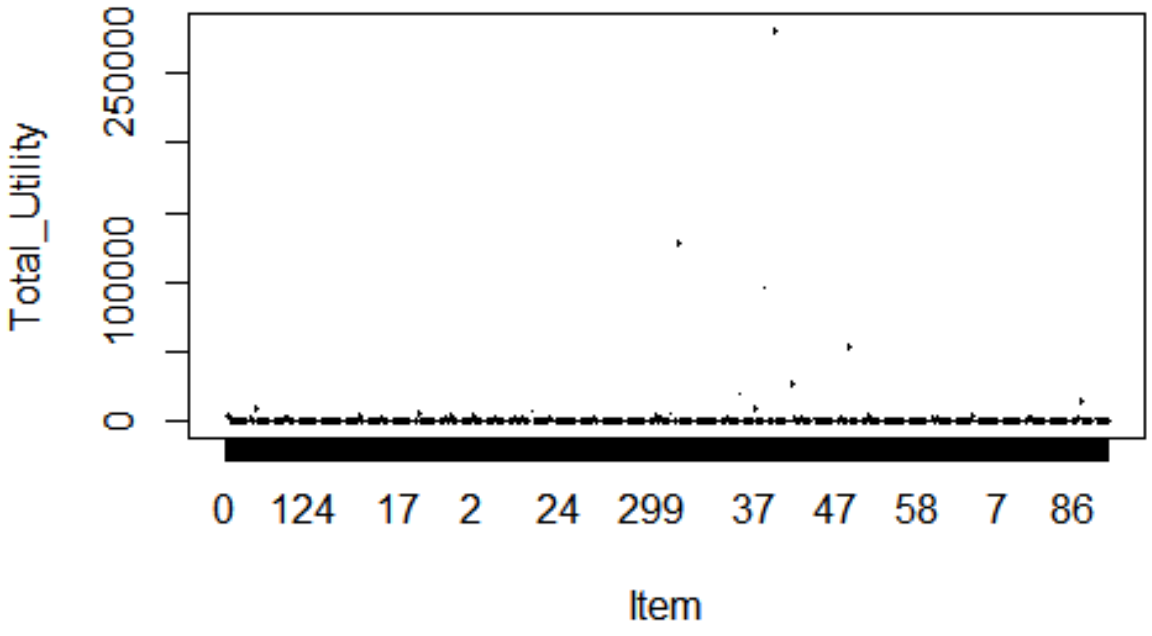

Fig 8. Graph for Item

versus TotalUtility

The figure shown in Fig7. represents how the item is stored in hash-table. The numbers 0,1,2,3,... represent the transaction IDs, each transaction ID points to a hash-bucket, that contains all the items present in that transaction. In the retail dataset, the items are initially in the format item\:TU\:profit. This data is then split to contain items corresponding to its profit value in the form of hash table. Once the data is ready, RStudio provides packages and built-in functions to make calculations required for the algorithm.

The Total Utility of an item is the sum of utility value in every transaction it occurs. A plot of Item and its corresponding total utility is as shown in Fig 8. RStudio is preferred because, before applying the algorithm the data can analyzed by plotting various graphs and charts, which provides various insights.

6. CONCLUSION

In this paper, a novel structure for storage and retrieval of items and its profit called hash-map structure was proposed along with an algorithm to extract the high utility itemsets. Utility itemsets extracted out of the data, not only provides information about how frequently the item was purchased, but also the profit associated with the item. Also, other algorithms used in mining process generate large number of candidate itemsets during the second scan of the database. HUI-Extractor avoids the generation of candidate itemsets as the items can be directly retrieved using hash-map structure. This algorithm also avoids heavy calculations as in other algorithms. One possible future work would be to extend the work by implementing in a distributed framework using Big Data tools, which will open up several factors for improvement. Thus, to conclude, HUI-Extractor algorithm is efficient in way it uses the hash-map structure for retrieval.

REFERENCES

Jerry Chun-Wei Lin, Wensheng Gan, Tzung-Pei Hong and Vincent S. Tseng. Efficient algorithms for mining up-to-date high utility itemsets, ELSEIVER, Advanced Engineering Informatics, June 2015.

Y. Liu, W.-K. Liao, and A. Choudhary. A fast high utility itemsets mining algorithm. In Proc. Utility-Based Data Mining Workshop, pages 9099, 2005.

Liu, H. Lu,W. Lou, Y. Xu, and J. X. Yu. Efficient mining of frequent patterns using ascending frequency ordered prefixtree. Data Mining and Knowledge Discovery, 9(3):249274, 2004.

Y. Liu, W.-K. Liao, and A. N. Choudhary. A two-phase algorithm for fast discovery of high utility itemsets. In Proc. Pacic- Asia Conf. Knowledge Discovery and Data Mining, pages 689695, 2005.

V. S. Tseng, C.-W. Wu, B.-E. Shie, and P. S. Yu. Upgrowth: An efficient algorithm for high utility itemset mining. In Proc. ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining, pages 253262, 2010.

Y.-C. Li, J.-S. Yeh, and C.-C. Chang. Direct candidates generation: A novel algorithm for discovering complete sharefrequent itemsets. In Proc. Fuzzy Systems and Knowledge Discovery, pages 551560, 2005.

P. Fournier-Viger, C.W. Wu, S. Zida, V.S. Tseng, FHM: faster high-utility itemset mining using estimated utility co-occurrence pruning, Found. Intell. Syst. 8502 (2014) 8392.

R. Agrawal, R. Srikant, Fast algorithms for mining association rules in large databases, in: International Conference on Very Large Data Bases, 1994, pp. 487499.

S.B. Kotsiantis, Supervised machine learning: a review of classification techniques, in: The Conference on Emerging Artificial Intelligence Applications in Computer Engineering: Real Word AI Systems with Applications in eHealth, HCI, Information Retrieval and Pervasive Technologies, 2007, pp. 324.B. Barber and H.J.Hamilton: Extracting share frequent itemsets with infrequent subsets.Data Mining and Knowledge Discovery, 7(2) (2003), 153-185

Subramanian, K., Kandhasamy, P., Subramanian, S.: A Novel Approach to Extract High Utility Itemsets from Distributed Databases. Computing and Informatics 31, 15971615 (2012)

Fan,W., Bifet, A.: Mining Big Data: Current Status, and Forecast to the Future. SIGKDD Explorations 14(2), 15 (2012)