A Hash Based Frequent Itemset Mining using Rehashing

Sirisha Aguru[1]
Department of Computer Scince and Engineering[1],
Sri Vasavi Engineering College
Pedatadepalli,India.
[1]*sirib4u1@gmail.com*

Batteri Madhava Rao[2]
Department of Computer Applications[2]
Sri Vasavi Engineering College
Pedatadepalli,India.
[2]*madhavbatteri@gmail.com*

Abstract—Data mining is the use of automated data analysis techniques to uncover previously undetected relationships among data items. Mining frequent item sets is one of the most important concepts of data mining. Frequent item set mining has been ahighly concerned field of data mining for researcher for over two decades. It plays an essential role in many data mining tasks that try to find interesting itemsets from databases, such as association rules, correlations, sequences, classifiers and clusters . In this paper, we propose a new association rule mining algorithm called Rehashing Based Frequent Item set (RBFI) in which hashing technology is used to store the database in vertical data format. To avoid hash collision and secondary clustering problem in hashing, rehashing technique is utilized here. The advantages of this new hashing technique are easy to compute the hash function, fast access of data and efficiency. This algorithm provides facilities to avoid unnecessary scans to the database.

Keywords-- *Data Mining ;Maximal frequent item set; Frequent Item set; Collision; Rehashing ;Hashing; Double hashing*

*****

I. INTRODUCTION

Data mining is the process of discovering meaningful new and interesting correlation, patterns and trends by sifting through large amounts of data, by using pattern recognition technologies as well as statistical and mathematical technique [63]. Now a days Data mining has been widely used and unifies research in various fields such as computer science, networking and engineering, statistics, databases, machine learning and Artificial Intelligence etc. There are different techniques that also fit in this category including association rule mining, classification and clustering as well as regression [1]. Finding association rules is the core process of data mining and it is the most popular technique has been studied by many researchers.. It is mining for association rules in database of sales transactions between items which is important field of the research in dataset [2]. The benefits of these rules are detecting unknown relationships, producing results which can used as a basis for decision making and prediction.

Frequent itemset mining has wide applications. The research in this field is started many years before but still emerging. This is a part of many data mining techniques like association rule mining, classification, clustering, web mining and correlations. The same technique is applicable to generate frequent sequences also. In general, frequent patterns like tree structures, graphs can be generated using the same principle. There are many applications where the frequent itemset mining is applicable. In short, they can be listed as market-basket analysis, bioinformatics, networks and most in many analyses. Agarwal et. al [4] is the first person to state this problem. Later many algorithms were introduced to generate frequent itemsets.

A. Frequent Item sets

Let I = { I1, I2, I3, …, Im} be a set of items. Let D be the transactional database where each transaction T is a set of items such that T ⊆ I. Each transaction is associated with an identifier TID. A set of items is referred as item set. An item set that contains K items is a K-item set. The number of transactions in which a particular item set exists gives the support or frequency count or count of the item set. If the support of an itemset I satisfies the minimum support threshold, then the item set I is a frequent itemset. Association rules are usually required to satisfy a user-specified minimum support and a user-specified minimum confidence at the same time. Association rule generation is usually involving two steps :

(1)Finding out all the frequent item sets which are greater than or equal to user-specified minimum support threshold

(2)Generating association rules from frequent item sets[5].

II. LITERATURE SURVEY

Methods for finding the maximal elements include All-MFS [6], which works by iteratively attempting to extend a working pattern until failure. A randomized version of the algorithm that uses vertical bit-vectors was studied, but it does not guarantee every maximal pattern will be returned. MaxMiner [7] is another algorithm for finding the maximal elements. It uses efficient pruning techniques to quickly narrow the search. MaxMiner employs a breadthfirst traversal of the search space; it reduces database scanning by employing a lookahead pruning strategy DepthProject [8] finds long itemsets using a depth first search of a lexicographic tree of itemsets, and uses a counting method based on transaction projections along its branches. It returns

a superset of the **MFI** and would require post-pruning to eliminate non-maximal patterns. FPgrowth [9] uses the novel frequent pattern tree (FP-tree) structure, which is a compressed representation of all the transactions in the database. Mafia [10] is the most recent method for mining the **MFI**. Mafia uses three pruning strategies to remove non-maximal sets. The first is the look-ahead pruning first used in MaxMiner. The second is to check if a new set is subsumed by an existing maximal set. Apriori is the first efficient algorithm that performs onlarge databases which was proposed by Agrawal and Srikant [11] and Mannila et. al [12] independently at the same time. They proposed their cooperative work in [13] MaxMiner [7] performs a breadth-first traversal of the search space as well, but also performs lookaheads to prune out branches of the tree. The lookaheads involve superset pruning, using apriori in reverse (all subsets of a frequent itemset are also frequent). In general, lookaheads work better with a depth-first approach, but MaxMiner uses a breadth-first approach to limit the number of passes over the database. DepthProject [8] performs a mixed depth-first traversal of the tree, along with variations of superset pruning. Instead of a pure depth-first traversal, DepthProject uses dynamic reordering of children nodes. With dynamic reordering, the size of the search space can be greatly reduced by trimming infrequent items out of each node's tail. Also proposed in DepthProject is an improved counting method and a projection mechanism to reduce the size of the database. The other notable maximal pattern methods are based on graph-theoretic approaches. MaxClique and MaxEclat [14] both attempt to divide the subset lattice into smaller pieces ("cliques") and proceed to mine these in a bottom-up Apriori-fashion with a vertical data representation. The VIPER algorithm has shown a method based on a vertical layout can sometimes outperform even the optimal method using a horizontal layout [15]. Other vertical mining methods for finding **FI** are presented by Holsheimer [17] and Savasere et al. [18]. The benefits of using the vertical tid-list were also explored by Ganti et al. [16].

III. PAPER ORGANIZATION

The remainder of the paper is organized as follows: Section-IV presents the different hashing techniques used for frequent itemsets. Section-V presents the proposed work with algorithm. Section-VI presents the example of the proposed work. Section-VII, deals with the experimental results graphically. Section-VIII gives the conclusion.

IV. HASHING TECHNIQUES

A. AprioriAlgorithm using hashing

Our hash based Apriori implementation, uses a data structure that directly represents a hash table. This algorithm proposes overcoming some of the weaknesses of the Apriori algorithm by reducing the number of candidate k-item sets. In particular the 2-itemsets, since that is the key to improving performance. This algorithm uses a hash based technique to reduce the number of candidate itemsets generated in the first pass. It is claimed that the number of item sets in C2 generated using hashing can be small so that the scan required to determine L2 is more efficient.

For example, when scanning each transaction in the database to generate the frequent 1-itemsets, L1, from the candidate 1-itemsets in C1, we can generate all of the 2-itemsets for each transaction, hash(i.e) map them into the different buckets of a hash table structure, and increase the corresponding bucket counts . A 2-itemset whose corresponding bucket count in the hash table is below the support threshold cannot be frequent and thus should be removed from the candidate set. Such a hash based apriori may substantially reduce the number of the candidate k-item sets examined.

B. Hashing Techniques

A hash table (hash map) is a data structure used to implement an associative array, a structure that can map keys to values. A hash table uses a hash function to compute an *index* into an array of *buckets* or *slots*, from which the correct value can be found..Hash functions are primarily used in hash tables, to quickly locate a data record given its search key . Specifically, the hash function is used to map the search key to an index; the index gives the place in the hash table where the corresponding record should be stored. Ideally, the hash function will assign each key to a unique bucket, but this situation is rarely achievable in practice (usually some keys will hash to the same bucket). Instead, most hash table designs assume that *hash collisions*—different keys that are assigned by the hash function to the same bucket—will occur and must be accommodated in some way.

1) Separate Chaining

In separatechaining, each bucket is independent, and has some sort of list of entries with the same index. The time for hash table operations is the time to find the bucket (which is constant) plus the time for the list operation. (The technique is also called *open hashing* or *closed addressing*.)

2) Open addressing

In another strategy, called open addressing, all entry records are stored in the bucket array itself. When a new entry has to be inserted, the buckets are examined, starting with the hashed-to slot and proceeding in some *probe sequence*, until an unoccupied slot is found. When searching for an entry, the buckets are scanned in the same sequence, until either the target record is found, or an unused array slot is found, which indicates that there is no such key in the table.[12] The name "open addressing" refers to the fact that

the location ("address") of the item is not determined by its hash value. (This method is also called closed hashing; it should not be confused with "open hashing" or "closed addressing" that usually mean separate chaining.)
Well-known probe sequences include:

- Linear probing, in which the interval between probes is fixed (usually 1)
- Quadratic probing, in which the interval between probes is increased by adding the successive outputs of a quadratic polynomial to the starting value given by the original hash computation
- Double hashing, in which the interval between probes is computed by another hash function

A drawback of all these open addressing schemes is that the number of stored entries cannot exceed the number of slots in the bucket array. In fact, even with good hash functions, their performance dramatically degrades when the load factor grows beyond 0.7 or so. For many applications, these restrictions mandate the use of dynamic resizing, with its attendant costs.Open addressing schemes also put more stringent requirements on the hash function: besides distributing the keys more uniformly over the buckets, the function must also minimize the clustering of hash values that are consecutive in the probe order. Using separate chaining, the only concern is that too many objects map to the *same* hash value; whether they are adjacent or nearby is completely irrelevant.

3) Rehashing

Rehashing is a technique used in hash tables to resolve hash collisions, when two different values to be searched for producing the same hash key. It is a popular collision resolution technique used on hash tables.Like linear probing ,it uses one hash value as a starting point and then repeatedly steps forward an interval ,until the desired value is located; an empty location is reached , or the entire table has been searched. .In Linear probing, Quadratic probing and Double hashing, we have to guess the number of elements we need to insert into a hash table. Whatever our collision policy is, the hash table becomes inefficient when load factor is too high. The load factor is a measure of how full the hash table is allowed to get before its capacity is automatically increased..Rehashing technique resolves the collisions that are encountered during various collision resolution techniques used in open addressing starategy.This is done by increasing the size of a hash table, and restoring all of the items into the hash table using the hash function h(k)=k%m where m is the new length of the hash table after increasing it..

V. PROPOSED WORK

In general the structure of the transactional database may be in two different ways – Horizontal data format and Vertical data format. In this paper, transactions of database are stored in the vertical format. In vertical data format, the data is represented as item-tidset format, where item is the name of the item and Tidset is the set of transaction identifiers containing the item.

In this paper, a new Rehashing Based Frequent Item set(RBFI) generation algorithm of the vertical data format for the transactional database is proposed. In this first, the data is represented as an item and Transaction id set (Tidset) format.To avoid collisions, primary clustering problem encountered in linear probing[19] and secondary clustering problem encountered in quadratic probing[20] rehashing technique is used.

1) RBFI Algorithm

Input: D, a database of transactions where all are represented as vertical hash table.
Process logic: Finding the frequent item sets.
Output: Generating the frequent item sets.

begin
m=0;k=0;
Get minimum support,min_sup;
Generate the new database in (Items, Tidset) format
For all Items I ε D_k do
Increment m;
n=2*m+1;
D_k=D;
do
begin
Make a hash table of size n. Map items on to the buckets. If collision occurs then use Rehashing technique. Create a linked list for the k^{th} level to maintain the transaction from the database Dk .

for all Items I ε D_k do
begin
Generate a subset of items.
end.
Find common transaction between the subsets in the kth level.
Eliminate the subset <=min_sup.
Dk = Items >=min_sup.
Increment k.
end until frequent item set is found.
End.

VI. EXAMPLE OF THE PROPOSED WORK

Consider Table 1.Initial Transaction Database.For our convinience, Let us replace these real time items Benoquin,Dialyte,Ibuprofen,Nutradrops,Veetids with I1,I2,I3,I4,I5 respectively.

TABLE 1.INITIAL TRANSACTION DATABASE

TransactionID	Itemsets
T1	Benoquin,Dialyte,Ibuprofen,Nutradrops Dialyte, Nutradrops
T2	Benoquin,Veetids
T3	Benoquin, Dialyte,Ibuprofen
T4	Benoquin,Nutradrops
T5	Dialyte,Ibuprofen
T6	Benoquin,Ibuprofen
T7	Dialyte,Nutradrops
T8	Benoquin,Dialyte
T9	Ibuprofen,Nutradrops
T10	

TABLE 2. TRANSACTION DATABASE

TransactionID	Itemsets
T1	I1,I2,I3,I4
T2	I2,I4
T3	I1,I5
T4	I1,I2,I3
T5	I1,I4
T6	I2,I3
T7	I1,I3
T8	I2,I4
T9	I1,I2
T10	I1,I3

By taking minimum support count=3 , the following table shows Tid for all 5 items

TABLE-3: VERTICAL FORMAT OF TRANSACTIONAL DATABASE

Itemsets	TransactionID
I1	T1,T3,T4,T5,T7,T9
I2	T1,T2,T4,T6,T8,T9
I3	T1,T4,T6,T7,T10
I4	T1,T2,T5,T8,T10
I5	T2,T3

The items in the transaction are hashed based on the hash function:

$$h(k) = (\text{order of item } k) \bmod n.$$

The n value is determined by using the formula (2m + 1) where m is the number of items in the database. The transaction in which I1 are present is connected in the form of linked list and the first node denotes the number of occurrences of the item in the transactions. It can be observed from Figure .1 that I1 is hashed to 1st location and it is determined using the hash function. Similarly all items are hashed into the hash table. The cross symbol indicates the end of the items in the list. Here, the linked list is created based on the item set and not on the transactions because the transactions are more so that it occupies more memory and it is very difficult to access the items. There is a link between a transactions in each item sets. The linked list is created for all levels of frequent item set generation. In the next higher level, the item subsets become low and it is easy to find frequent item sets of that level. The process continues until the exact frequent item set is found.This is shown in the below figure.1

TABLE -4: VERTICAL FORMAT OF THE TRANSACTIONAL DATABASE IN THE SECOND LEVEL

ITEM SET	TID
{I1, I2}	T1, T4, T9
{I1, I3}	T1, T4, T7
{I1, I4}	T1, T5
{I2, I3}	T1, T4, T6
{I2, I4}	T1, T2, T8
{I3, I4}	T1, T10

TABLE 5 VERTICAL FORMAT OF THE TRANSACTIONAL DATABASE IN THE THIRD LEVEL

Item set	Tid set
{I1,I2,I3}	T1,T4
{I1,I2,I4}	T1
{I2,I3,I4}	T1

The item set in the second level from Table.4. are hashed based on the hash function,

$$h(k)=((\text{order of X})*10+\text{order of Y}) \bmod n.$$

Here , the item sets are mapped to 1,2,3,1,2,1.Here , there is a collision for {I1,I2}{I2,I3};they are mapped to 1 and {I1,I2}{I3,I4} are mapped to 1 and {I1,I3}{I2,I4} are mapped to 2.Rehashing technique is used to overcome this collision.

Let h(k) be a hash function that maps element k to an integer in [0,j+1] , where $j = 2*m+1$ and m is the size of the table.

 ISSN: 2321-8169
4198 - 4204

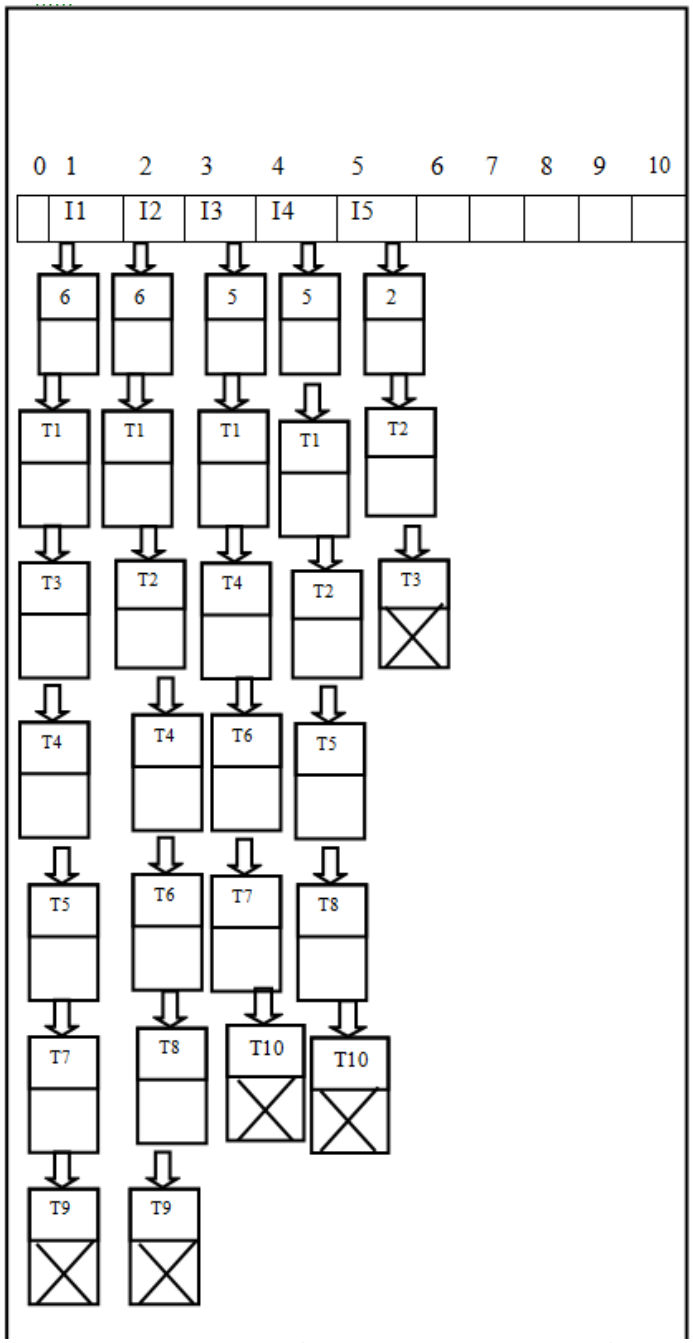

Fig. 1.: Hash table including links for the transactional database in the first level.

Here whenever collisions occur after mapping the frequent item sets then an immediate check for the number of buckets still vacant in the hash table must be done. If it is observed that the hash table is either half –filled or is more than half of the size of the hash table is occupied then it isappropriate to apply rehashing technique using which we can double the size of the hash table thus providing enough buckets for all frequent item sets without any collisions.

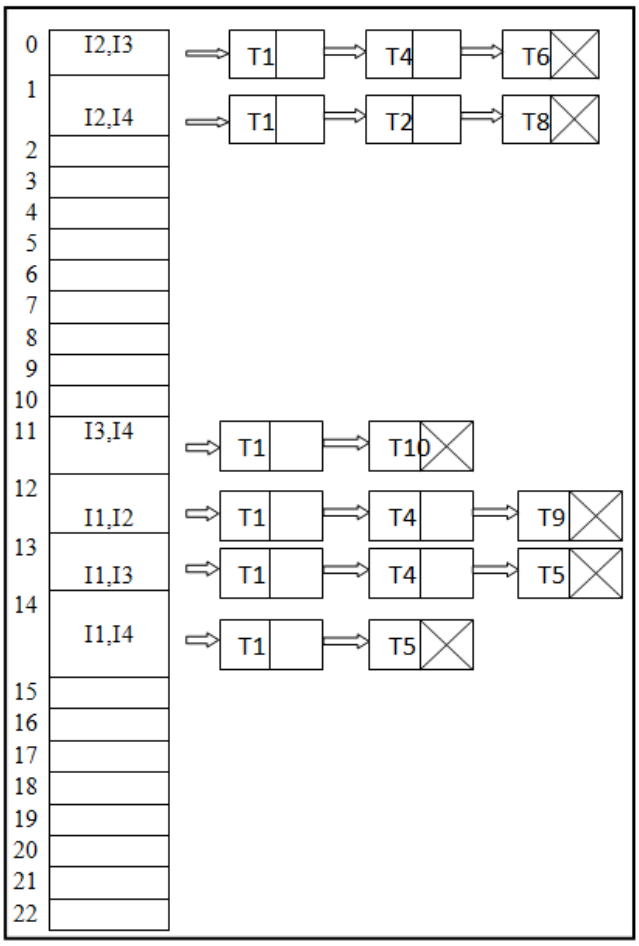

Figure .2.Hash table including links for the transaction database at second level

Here , we increase the size of the hash table by doubling the actual size , so that the resulting hash table size is also a prime number. Thus the size of the hash table after increasing is j= (2*m+1), where m=11 (initial hash table size) .Therefore , $j=23$.Now,we apply the hash function.

$$h(k)=((\text{order of X})*10+\text{order of Y}) \bmod j$$

It better avoids primary ,secondary clustering problems and some collisions that may still occur using Double hashing technique also.

After rehashing the collision is resolved and the 2-itemsets {I1,I2} ,{I1,I3},{I2,I3},{I2,I4},{I3,I4} ,{I1,I4} are mapped to 12 ,13,0,1,11,14 buckets respectively as shown in figure.2. In the second level, item sets {I1,I2} ,{I1,I3},{I2,I3},{I2,I4},{I3,I4} whose support counts are greater than or equal to 3 are said to be frequent itemsets .It can be observed from Table 4 and Figure.2. The 3-item sets are generated from frequent itemsets of second level as shown in the Table .5

The itemsets in the third level are hashed based upon the hash function

 ISSN: 2321-8169
4198 - 4204

H(k)=((order of X)*100+(order of Y)*10+order of Z)mod j.

Using this hash function the itemsets {I1,I2,I3},{I1,I2,I4},{I2,I3,I4} are mapped to location 8,9 and 4 respectively as shown in the Figure.3.

The proposed algorithm (RBFI) performs better because Frequent Itemset is calculated in a simplest way. The structure of transactional database is vertical data format. This makes easy to perform several tasks. In this format, support also need not be calculated separately. In this case, support is directly given by the number of transactions in the Tidset of each FI or it can be obtained from the count value in the header node of the corresponding linked list. It is about 2 to 3 times faster than other hash based technique. It quickly finds an empty location in the hash table to map the items. The RBFI performs better with large number of transactions and long item sets.

Here, this algorithm doesn't require performing separate pruning. Hash data structure can be maintained to store the database.

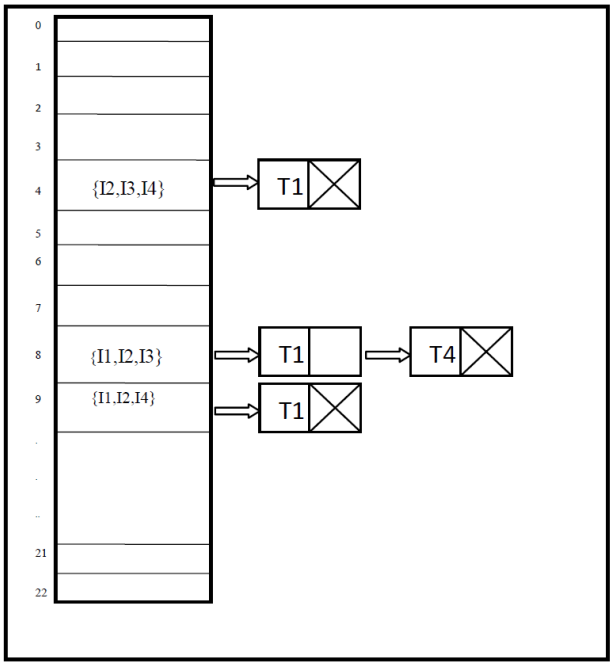

Figure .3.Hash table including links for the transaction database at third level

VII. Experimental Results

Figure 4. Shows a time comparison between Apriori algorithm,HBFI-DH and RBFI algorithms for various values of thresholds. From the diagram it can be seen that the time taken for RBFI is considerably reduced. In this method the time taken to hash items in to vertical hash table is comparatively very low. For various support counts the time taken to find a frequent item set is less when compared with Apriori and HBFI-DH.\

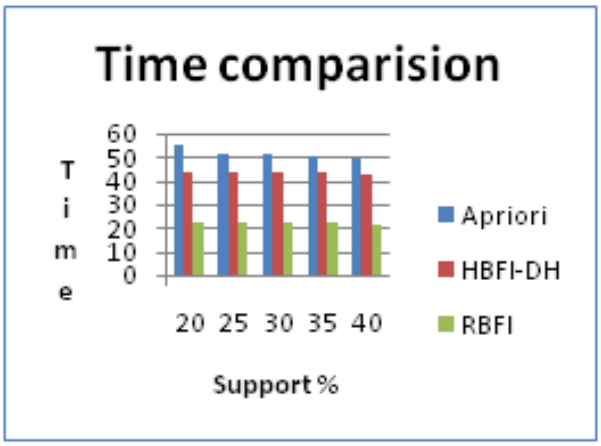

Figure .4 .Time Comparisions between Apriori,HBFI-Dh and RBFI

VIII. Conclusion

In this paper, an effective algorithm for the initial candidate set generation has been proposed. Our experimental results demonstrate that it is better than Apriori and HBFI-DH.HBFI-DH is a hash based algorithm and it is very effective for the generation of candidate item sets and it eliminates the items which are not needed for the generation of frequent item sets before the generation of candidate 2-item sets. The algorithm works well but it suffers from collisions (secondary clustering) problem.

We presented RBFI, an algorithm for finding frequent item sets. Our experimental results demonstrate that RBFI is better than Apriori and other hash based methods because it efficiently map the item sets in the hash table and it also avoids the primary clustering problem and secondary clustering.The vertical data format representation of the database leads to the easy manipulations on hash data structure.RBFI uses all the bins and hence the phenomenon of secondary clustering will not occur with Rehashing.

References

[1] The Gartner Group, www.gartner.com.

[2] M.S.V.K. Pang-Ning Tan, "*Data mining, in Introduction to datamining"*, Pearson International Edition, 2006, pp.2-7.

[3] J. Han, M. Kamber, "*Data Mining: Concepts and Techniques 3rdedition*", Morgan Kaufmann Publishers, 2013.

[4] R. Agrawal, T. Imielienski and A. Swami, "Miningassociation rules between sets of items in large databases. In P. Bunemann and S. Jajodia, editors, Proceedings of the 1993 ACM SIGMODConference on Management of Data, Pages 207-216, Newyork, 1993, ACM Press.

[5] R. Agrawal, T. Imielinski, and A. Swami, "Mining Association RulesBetween Sets of Items in Large

ISSN: 2321-8169
4198 - 4204

Databases," Proc. ACM SIGMOD, May1993, pp. 207–216.

[6] D. Gunopulos, H. Mannila, and S. Saluja, "Discovering all the most specific sentences byrandomized algorithms", In Intl. Conf. onDatabase Theory, Jan. 1997.

[7] Roberto Bayardo, "Efficiently mining long patterns from databases", in ACM SIGMOD Conference 1998.

[8] R. Agarwal, C. Aggarwal and V. Prasad, "A tre projection algorithm for generation of frequent itemsets", Journal of Parallel and Distributed Computing, 2001

[9] J. Han, J. Pei, and Y. Yin. "Mining frequent patterns without candidate generation", In ACM SIGMOD Conf., May 2000

[10] Burdick, D., M. Calimlim and J. Gehrke, "MAFIA: A maximal frequent itemset algorithm for transactional databases", In International Conference on Data Engineering, pp: 443 – 452, April 2001, doi = 10.1.1.100.6805

[11] Heikki Mannila, Hannu Toivonen, and A. Inkeri Verkamo, "Efficient Algorithms for discovering association rules", in Usama M. Fayyad and Ramasamy Uthurusamy, editors, AAAI Workshop on Knowledge Discovery on Databases (KDD-94),pages 181-192, Seattle, Washington, 1994, AAAIPress.

[12] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules", in Proceedings of the 20th International Conference on Very Large Databases (VLDB'94), Santiago de Chile, September 12-15, pages 487-499, Morgan Kaufmann, 1994.

[13] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. I. Verkamo, "Fast discovery of association rules", Advances in Knowledge Discovery andData Mining, pages 307-328, MIT Press, 1996.

[14] M. J. Zaki, "Scalable Algorithms for Association Mining", IEEE Transactions on Knowledge and Data Engineering, Vol. 12, No. 3, pp 372-390, May/June 2000.

[15] P. Shenoy, J. R. Haritsa, S. Sudarshan, G. Bhalotia, M. Bawa, and D. Shah, "Turbo-charging Vertical Mining of Large Databases", SIGMOD Conference 2000: 22-33.

[16] V. Ganti, J. E. Gehrke, and R. Ramakrishnan, "DEMON: Mining and Monitoring Evolving Data", ICDE 2000: 439-448

[17] M. Holsheimer, M. L. Kersten, H. Mannila, and H.Toivonen, "A Perspective on Databases and Data Mining", KDD 1995: 150-155.

[18] A. Savasere, E. Omiecinski, and S. Navathe, "An efficient algorithm for mining association rules in large databases", 21st VLDB Conference, 1995.

[19] A.M.J. Md. Zubair Rahman, P. Balasubramanie and P. Venkata Krihsna "A Hash based Mining Algorithm for Maximal Frequent Itemsets using Linear Probing". Infocomp Journal of Computer Science 2009, Vol.8, No.1, pp.14-19.

[20] M. Krishnamurthy, A. Kannan , R. Baskaran, R. Deepalakshmi "Frequent Item set Generation Using Hashing-Quadratic Probing Technique" "European Journal of Scientific Research ISSN 1450-216X Vol.50 No.4 (2011), pp. 523-532.