Volume 3, Issue 1, January 2013 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

An Efficient Association Rule Mining Using the H-BIT Array Hashing Algorithm

L.Padmavathy[1] and V.Umarani[2]
Department of Computer Science,
Sri Ramakrishna College of Arts & Science, Coimbatore, India

***Abstract:** Association Rule Mining (ARM) finds the interesting relationship between presences of various items in a given database. Apriori is the traditional algorithm for learning association rules. However, it is affected by number of database scan and higher generation of candidate itemsets. Each level of candidate itemsets requires separate memory locations. Hash Based Frequent Itemsets - Quadratic Probing (HBFI - QP) algorithm, which is based on hashing technique for mining the frequent itemsets. In order to stay away from collisions and primary clustering in hashing process, Quadratic Probing (QP) technique is used. Though the primary clustering and collisions are eliminated, secondary clustering is formed in all cases and the hash table occupies more space than the total number of items in the database. To avoid those problems, the H-Bit Array Hashing (H-BAH) algorithm is presented in this paper. H-BAH algorithm reduces hash table size required for placing items and it also avoids hash collisions and secondary clustering. The H-Bit array that is added to the first or header bucket of the table gives the information about which buckets are hashed initially. At the time of collisions in the hashing process, the H-BAH algorithm works by finding the neighbourhood of buckets near the original hashed bucket, in order to place the collided items quickly. The H-BAH algorithm provides frequent itemsets with less computational time and memory than the existing algorithm.*

***Keywords:** Frequent itemsets, Hashing, Collisions, Linked List*

1. Introduction

Association Rule Mining [ARM] will find rules that forecast the purchasing behaviour of one item based on other items. Let A = {I_1, I_2 … I_m}, be the collection of items and T is the initial transaction database consists of the set of transactions. Every transaction 't' has a group of items, where 't' is subset of A. The two important concepts of association rule mining are **support and confidence**. User specified values are minimum support and minimum confidence.

Support: *The support [6] of an itemset X is defined as the proportion of transactions in the dataset which contains the itemset.*

$$Support\,(X) = \frac{Support\,(x)}{number\ of\ transactions}$$

ARM first generates candidate *k*-itemsets (C_k) where k starts from 1. If support of itemset is greater than or equal to minimum support then they are frequent k-itemsets (L_k). Using the set of L_k candidate *k*+1 itemsets (C_{k+1}) are formed. Searching will continue until all frequent itemsets in the database are created. From the frequent itemsets association rules are generated.

Confidence: *Confidence is a measure that detects whether the generated rules are strong or not. It is defined by using the formula*

$$Confidence\ (X \Rightarrow Y) = \frac{Support\,(X \cup Y)}{Support\,(X)}$$

If confidence of a rule is greater than or equal to the minimum confidence, they are called as strong association rules [2].

Association Rule Mining is generally a two – step process

Step 1: Find all sets of items (itemsets) whose support is greater than the user specified minimum support. Such itemsets are called frequent itemsets.

Step 2: Generate strong association rules from the frequent itemset. These rules must satisfy the user specified minimum support and minimum confidence.

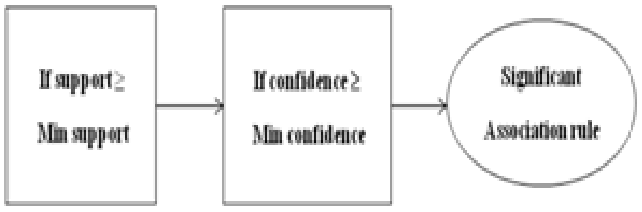

Figure 1. Basic Steps in ARM

2. APRIORI ALGORITHM

The Apriori algorithm is the well known and basic association rule mining algorithm and is used in most commercial products. Apriori algorithm is designed to operate on database containing the transactions. Apriori uses a "bottom up" approach [1], where frequent subsets are extended one item at a time (a step known as candidate generation), and groups of candidates are tested against the data. The algorithm terminates when no further successful extensions are found.
Apriori uses breadth-first search technique [2] and a tree structure to count candidate item sets efficiently.The basic idea of the Apriori algorithm is to generate candidate itemsets of a particular size and then scans the database to count whether they are large. Let Li denote the collection of large itemsets with 'i' number of items. The algorithm begins by identifying candidate 1-itemsets (C_1). Each item that has the necessary support forms the frequent 1-itemset and included in L_1 and other itemsets are dropped from consideration. This process of retaining necessary itemsets is called "pruning". The collection C_{i+1} can be constructed by considering each pair of sets in L_1. L_{i+1} are generated by comparing the support count of C_{i+1} itemset with minimum support and store the itemsets which satisfy necessary support. This procedure is continued until all frequent itemsets up to the desired maximum size have been obtained or no further pruning is possible.
Apriori suffers from two shortcomings; first, it requires multiple database scans (number equals the length of the largest pattern). Second, a huge number of candidates set need to be generated and their occurrence frequencies tested against the database. These two problems significantly limit the efficiency of this algorithm in large transactional databases. Based on Apriori algorithm, Hash-based techniques were designed with some modifications or improvements.

3. HASH-BASED TECHNIQUE

3.1. Hashing Process

The shortcomings of Apriori algorithm are gradually reduced by hashing technique [16]. Hashing technique is the very efficient method in searching to the exact data itemset in a very short time. Hashing is the process of placing each and every data item at the index of memory location for ease of usability.

The two basic things required for hashing process are hash table and hash functions. A hash table is made up of two parts: an array (the actual table where the data to be searched is stored) and a mapping function, known as a hash function. The hash function provides a way for assigning numbers to the input data such that the data can then be stored at the array index corresponding to the assigned number. In figure 2, the hash function maps the keys into corresponding bucket's index of hash table.

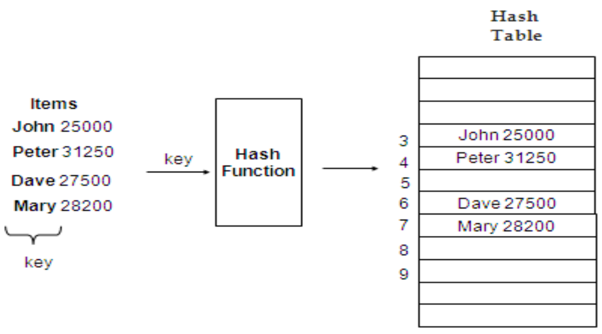

Figure 2. Example for Hashing Process

3.2 Hash-Based Technique in Association Rule Mining

A Hash table technique [3] is mainly used to reduce the size of candidate itemsets. At the time of scanning database for finding the frequent 1-itemset (L_1), all possible combinations of 2-itemset are produced. The itemsets are hashed (mapped) into the different buckets of a hash table structure based on hash functions, and increase the corresponding bucket counts. An itemset whose bucket count is less than the minimum support is detected and removed from the candidate set.

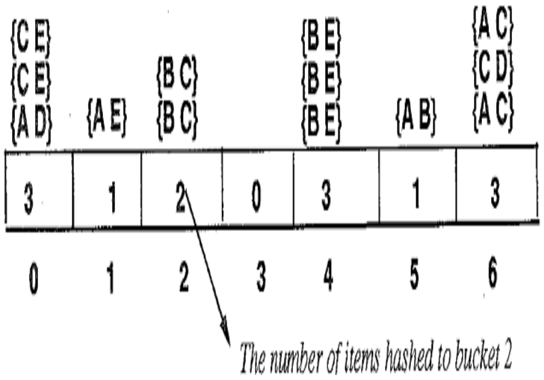

Figure 3. Hash Table with itemsets

In figure 3, if minimum support count is, say, 3, then the itemsets in buckets 1, 2, and 5 cannot be frequent and so they cannot be included in candidate 2-itemsets.

4. RELATED WORKS

Direct Hashing and Pruning [DHP] algorithm generates fast mining of association rules [4]. It utilizes a hash method for candidate itemset generation. In order to effectively reduce transaction database size, the proposed algorithm uses only a small number of candidate itemset. At the earlier stage of iterations itself database size is reduced, and therefore the computational cost is reduced.
Perfect Hashing and Pruning [PHP] algorithm generates the frequent itemset of a transaction database. The PHP [13] algorithm has some of the features of Direct Hashing and Pruning [DHP] algorithm. The PHP [8] employs hashing facility, to keep the actual count of occurrence of each candidate itemset of the database. It also prunes the transactions which do not contain any frequent items, and trims non-frequent items from the transactions at each step in order to improve the accuracy of rules.
Inverted Hashing and Pruning [IHP] algorithm [10] find out the candidate and frequent itemsets with the help of hash tables. The database is stored with three fields such as Transaction Identifier (TID) and the transaction with its corresponding hash value. Next TID Hash Table (THT) is built with items and its entries. Each entry has a number shows how many items are hashed to it. Instead of scan the database, count of items support in THT entries are added. Then items which are not satisfying minimum support are removed.
The HMFS method combines the advantages of Direct Hashing and Pruning [DHP] algorithm and Pincher-Search algorithm [17]. Hash technique of DHP algorithm is used to filter infrequent itemset in bottom-up direction. Then a top-down technique which is similar to Pincher-search algorithm is used to find the maximal frequent itemset. By combining the advantage of DHP and the Pincher-Search algorithms, number of database scan and the search space of items are reduced.
Double hashing technique [5] stores the itemset in buckets based on hash function. At the time of hash collision, double hashing technique is used to resolve them. After hashing the candidate 1-itemset, the frequent 1-itemset is calculated by using minimum support, and then candidate 2-itemset is hashed and so on. Maximal frequent items are directly calculated from frequent items itself. Finally from all the frequent items, association rules are generated.
Hash Partitioned Apriori (HPA) uses a hash function to partition candidate itemset. HPA-ELD (HPA with Extremely Large itemset Duplication) sorted itemsets based on their frequency. It selected most frequently occurring itemsets and copied it over the processors. HPA-ELD utilizes memory space efficiently and it attains good linearity on speed up ratio [14].
Hash Based Maximal Frequent Itemsets-Linear Probing (HBMFI-LP) method avoids unnecessary database scan. Database is stored in vertical format and based on hash function items are hashed into hash table [20]. To avoid hash collisions Linear Probing sequence is handled. For first level of items a linked list is created for each item. Support count of each item is stored in first node of the linked list and other nodes stored number of transactions related to corresponding item. Instead of scan the database, support count is taken from hash table and next level of items are hashed. Maximal frequent items are directly calculated from frequent items itself.
Hash Mapping Table (HMT) and Hash-Tree optimizes space complexity and time complexity in mining frequent items. HMT [18] is a one-dimensional mapping table based on hash function. HMT is mainly used to compress the dataset. Hash-Tree is constructed to hash the candidate k-itemset and the tree can decentralize support count process. All frequent itemsets are collected from the Hash-Tree.

5. EXISTING METHODOLOGY

The existing methodology, Hash Based Frequent Itemsets-Quadratic Probing (HBFI-QP) algorithm uses the hash table structure with linked list to catch on the frequent itemset [10]. HBFI-QP technique first converts the initial transactional database into vertical format (Itemset, Tidset). Itemset is the number of items in database and Tidset represents the list of transactions in which a particular item occurs [20]. The hash table is constructed based on the size 'n'.

$$n = 2(\text{Number of items}) + 1 \quad \dots (1)$$

First level itemsets in the vertical format are hashed based on the hash function, H (k) = (order of item k) mod n, where 'n' is size of hash table required for hash the items and it must be prime. The value for n is calculated as (2m + 1) and 'm' is the number of items in the database. If collisions take place, Quadratic Probing (QP) technique is employed to avoid it.

5.1 Quadratic Probing [QP]

Collisions are avoided by using Quadratic Probing technique. QP function is

$$H(k, i) = (H(k) + i^2) \bmod n \quad \dots. (2)$$

H (k) is the original hash value and probes (i) lie between 1 and table.length-1. After placing all the itemsets in hash table, linked list is created for each stored item in the table. Linked list [5] consists of nodes with support count and Tidset for every item. Instead of scan the database, frequent 1-itemset is directly captured from the hash table based on the minimum support threshold.
The second level items are hashed based on the hash function, H (k) = ((order of item X) * 10 + order of item Y) mod n. Figure 4 shows the hash table with the linked list for the second level (candidate 2-itemset) items.
Hashing with Quadratic Probing technique places all items without any collision but the probing sequence volume is high. Because of the lengthy probing sequence, it takes more time to hash the collided itemsets. Secondary clustering

occurs while using QP technique. For the same initial hash value, same sequences of numbers are repeated again, this is termed as secondary clustering.
QP technique mainly depends on load factor (ratio of the number of stored entries and the size of the table's array of buckets). Quadratic Probing happens when the load factor is less than or equal to 0.5. Even though hash table contains free space, the size of hashed itemset is not more than half the hash table’s size and so remaining memory space is idle.

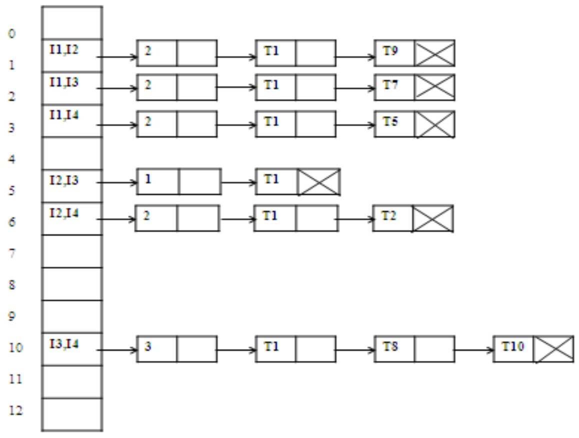

Figure 4. Hash Table for Second Level

6. Proposed Methodology

The proposed methodology, H-Bit Array Hashing (H-BAH) algorithm which is a hash-based technique mines the frequent itemsets without any collision in the hash table. The lengthy probing sequence and secondary clustering that exist with Quadratic Probing technique [10] is avoided by using the H-Bit array in the header bucket of the hash table. The H-Bit array technique also shrinks the size of hash table, which is required for placing itemsets. The initial transactional database as shown in table1 is in horizontal format and it is converted into vertical format (Itemset, Tidset).

Table 1. Horizontal Transactional Database

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

Table 2. Vertical Format of the Initial Transactional Database

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

Each and every itemset in table 2 represents the candidate 1-itemset of the database. Two required things for hashing are hash table and hash function**.** Hash table contains many buckets or slots to place the set of items and an efficient hash function determines where to place the itemsets. H-Bit array technique constructs hash table based on number of items in the database and it is defined by the following function.

n = number of items + c ….. (3)

The size of hash table (n) for the formula 3 must be the nearest prime number to the number of items in the database at each level. So the number of items can be added by any smallest integer (c) to get the nearest prime number. Hash function determines hash value and based on that value items are hashed or mapped into the hash table. First level of items (candidate 1-iitemsets) is hashed based on the following function.

H (k) = (order of item k) mod n ….. (4)

Where 'n' is the hash table size and value for n is calculated as (m + c) and it must be prime, 'm' represents total number of items in the database for the hash function 4. The table 2 contains 5 items, so the size of hash table (n) is 7 [5 + 2]. By using above hash function, locations for candidate 1-itemsets are determined.

Table 3. Locations for candidate1-itemsets

Itemset	**Locations**
I1	1
I2	2
I3	3
I4	4
I5	5

After placing all the items based on locations in table 3, an H-Bit array is added to the first bucket or slot of the hash table to store the information of all buckets. The value for 'H' is equal to the number of buckets in hash table. H-Bit array consist of bits '0' and '1'. The bit one indicates that its corresponding bucket is filled by an item and bit 0 indicates that it is free. Figure 5 shows the hash table for candidate 1-itemsets and the H-Bit array which is added to the header bucket. The number of buckets is 7 and so the 'H' value is 7.

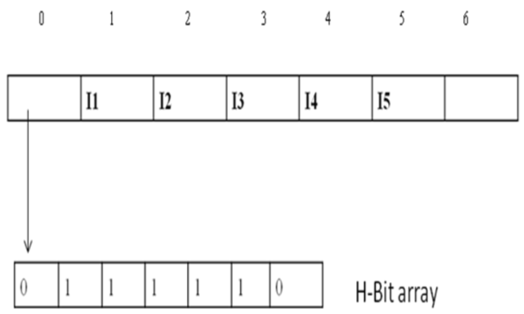

Figure 5. Hash Table for Candidate 1-itemset with H-Bit Array

The itemsets are placed and linked list [20] is created for every stored item in the hash table. Linked list comprised of a series of nodes and each node has data item and a pointer to next node. The first node in the list indicates support count of the item and Tidset for an item is directly placed in linked list below the node that has support count. Tidset in table 2 shows the list of transactions required for candidate 1-itemset. Support count is taken by adding the number of transactions that are listed in Tidset for an itemset. Figure 6 shows the hash table which is built for candidate 1-itemset.

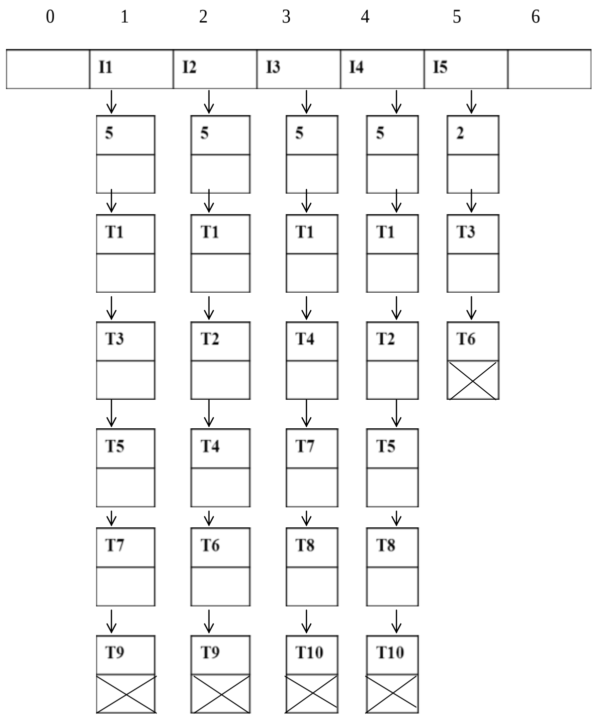

Figure 6. Hash Table Containing Linked List for Candidate 1-itemsets

Instead of scan the database, items frequencies are directly taken from the linked list and generate the frequent 1-itemset. If user specified minimum support is 3 means, the item I5 does not satisfy minimum support. So item I5 is deleted from the table and remaining items I1, I2, I3, and I4 are frequent 1-itemsets. Figure 7 represents the hash table for frequent 1-itemsets.

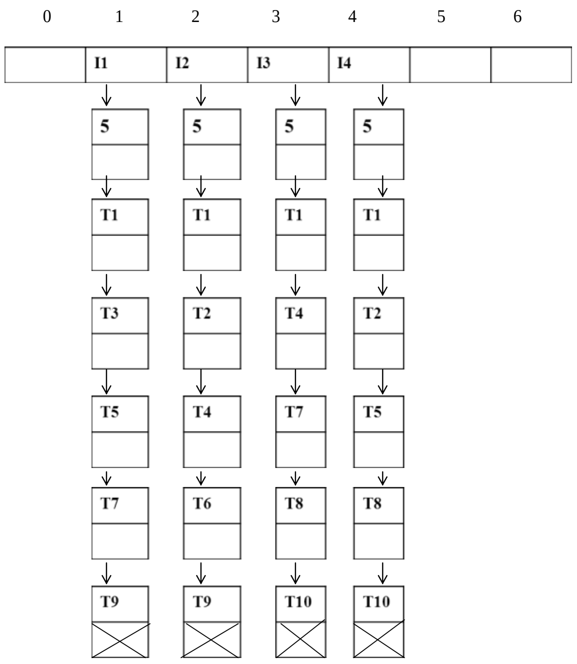

Figure 7. Hash Table for Frequent 1-itemsets

Possible combinations of frequent 1-itemsets are candidate 2-itemset and the itemsets are {I1, I2} {I1, I3} {I1, I4} {I2, I3} {I2, I4} and {I3, I4}.Taking the subset of candidate 2-itemset and search for common transactions in previous hash table. These transactions are stored as Tidset for second level (candidate 2-itemset).

Table 4. Vertical Format of the Candidate 2-itemsets

Itemset	**Tidset**
{I1, I2}	T1,T9
{I1, I3}	T1,T7
{I1, I4}	T1,T5
{I2, I3}	T1
{I2, I4}	T1,T2
{I3, I4}	T1,T8,T10

Itemsets in table 4 are hashed based on the following hash function.

H (k) = ((order of item X) * 10 + order of item Y) mod n ….. (5)

X and Y are individual items and value for n is 7 [6+1] for the formula 4.3. Collisions are unavoidable while using hash functions. For the items {I1, I4} and {I3, I4} collisions appeared. The proposed technique avoids hash collisions by searching on the added H-Bit array.

6.1 H-Bit Array Searching Technique

The non-collided items are first placed in the hash table and enhance an H-Bit array together with values 1's and 0's to the first bucket of the table. Buckets which are previously filled by the hash functions are stored with value one in the corresponding location of the array, and remaining locations are stuffed with value zero. The H-Bit array is searched in order to find empty bucket whose corresponding bit value is '0' and place the collided itemset in that locations. The searching begins at the bucket which is lastly filled by the hash function.

After placing collided itemsets in the hash table, its corresponding bit value in the array is updated as '1', so as to speed up the searching at the next iteration and subsequently the linked list is attached. Figure 8 represents the hash table with the array before placing the collided itemset.

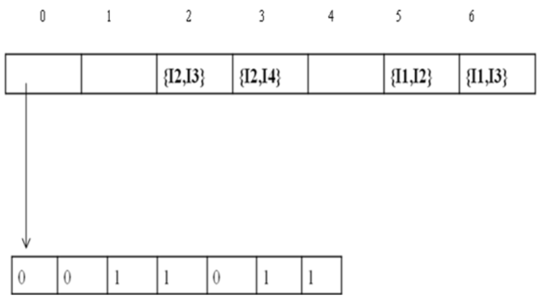

Figure 8. Hash Table before Placing Collided Items

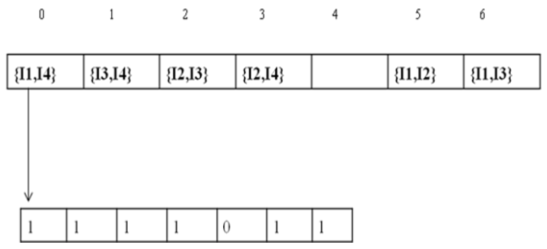

Figure 9. Hash Table after Placing Collided Items

The figure 9 shows the searching process of collided itemsets and updating the H-Bit array. The itemsets in table 4 are hashed and the linked list is created. The cross symbol in the figure 10 indicates the end of transactions.

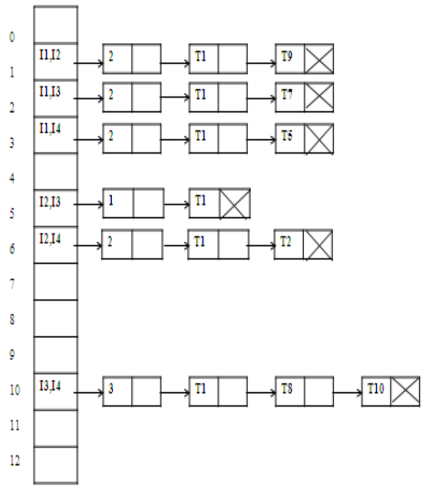

Figure 10. Hash Table for Candidate 2- itemsets

Hashing process is done for every level and it will terminate when there is no more candidate itemsets exists. Hashing technique is mainly used for its reusability. The entries in first level hash table are deleted and same locations are used to map second level itemsets and so on.

By using hash table with the linked list structures, frequent itemsets is filtered efficiently. Thus, the H-Bit array technique reduces hash table size and maps the items into hash table without any collision in a very short span of time. It also avoids lengthy probing sequence and so the secondary clustering does not takes place.

The following are the steps which have been performed using H-BAH algorithm.

STEP 1: Generate vertical format (Itemset, Tidset) of the initial transactional database.

STEP 2: Built a hash table based on number of items in the database.

n = (Number of items) + c

Where 'c' may be any integer and the hash table size (n) must be the nearest prime number to the total number of items in the database.

STEP 3: Hash the candidate 1-itemset based on the hash function.

STEP 4: Add the H-Bit array to the first slot or bucket of the hash table.

STEP 5: If collision occurs, then use H-Bit array searching technique to map collided items.

STEP 6: Place all the collided itemsets and update its corresponding bit value.

STEP 7: Create the linked list with support count and Tidset for each stored item in the hash table.

STEP 8: Scan the hash table with linked list, generate the first level frequent itemsets which satisfies minimum support threshold.

STEP 9: Form the second level of candidate itemset and create subsets for them.

STEP 10: Search for common transactions in first level hash table and they are stored as the Tidset for this level and place them into the hash table without any collisions using H-Bit array searching technique.

STEP 11: On using minimum support threshold, generate the second level frequent itemset.

The above steps are repeated until all levels of frequent itemsets are generated.

7. RESULTS AND DISCUSSION

This section gives an overview of the conducted experiments and offers the acquired results to evaluate the performance of H-BAH algorithm with the traditional Apriori and HBFI-QP algorithms in terms of time and memory consumptions.

The datasets which are used for conducting researches are Retail dataset, Generated transactional dataset, Insurance dataset, and Supermarket dataset. The number of items and number of transactions, which are taken for conducting this research, are described in the table 5.

Table 5. Dataset Details

Dataset	No.of Transactions	No.of Items
Retail Dataset	1500	2152
Generated Transactional Dataset	1000	11
Insurance Dataset	2000	37
Supermarket Dataset	785	495

Table 6 represents the number of association rules which are generated for the above datasets by the algorithms for the minimum support 20%.

Table 6. Rules Generated by the Algorithms

Dataset	Number of Rules
Retail Dataset	125
Generated Transactional Dataset	100
Insurance Dataset	377
Supermarket Dataset	70

The first set of experiment was conducted by the Retail dataset to evaluate the performance of the Apriori, HBFI-QP and the H-BAH algorithms in terms of time (in seconds) and memory (in bytes).
Apriori algorithm utilizes the locations that are equal to the number of candidate itemsets generated at all levels. HBFI-QP algorithm places each level of candidate itemsets in a hash table. The items are hashed level by level. Hash collisions are eliminated by the QP technique. H-BAH algorithm hashes the non-collided items and the H-Bit array is added to the first bucket. The array stores all the buckets information and the H-Bit array searching technique avoids the hash collisions.

Table 7. Hash Table Size Comparisons for the Retail Dataset

Number of items		Existing Method	Proposed Method
		Hash table size (buckets) for the HBFI-QP	Hash table size (buckets) for the H-BAH
Candidate 1-itemset	2152	4305	2153
Candidate 2-itemset	325	651	331
Candidate 3-itemset	560	1121	563
Candidate 4-itemset	126	253	127

Table 7 represents the number of buckets, which is the hash table size required for placing candidate itemsets of each level for the Retail dataset. The hash table size of proposed algorithm is reduced when compared to HBFI-QP algorithm. Thus, the memory usage of the propoesd algorithm is efficient than the existing methodology.

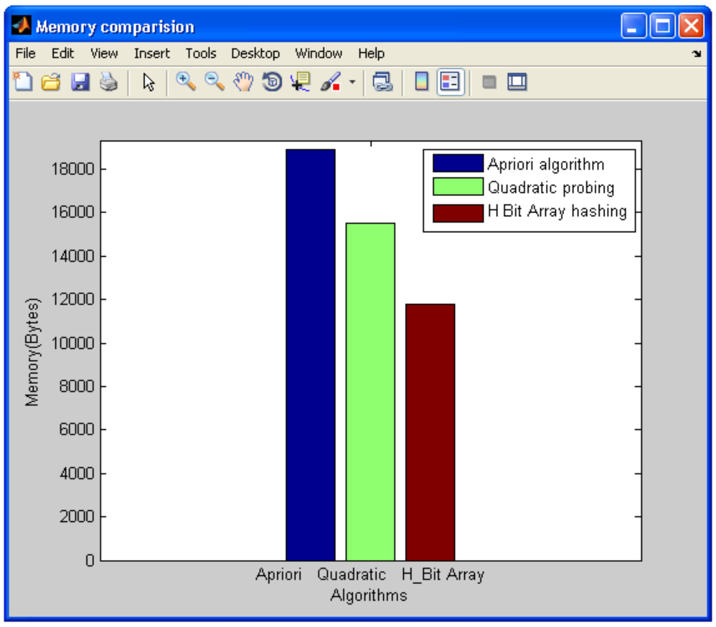

Figure 11. Graph representing the memory of the Retail Dataset

The figure 11 shows the memory requirements for placing all level of itemsets on using the Retail dataset.

Table 8. Memory Comparison for the Retail Dataset

Algorithms	**Memory (Bytes)**
Apriori	18129
HBFI-QP (Existing work)	15508
H-BAH (Proposed work)	11794

Table 8 represents the memory comparisons for the Retail dataset. On comparing the results, H-BAH algorithm performs better than Apriori and HBFI-QP algorithms.

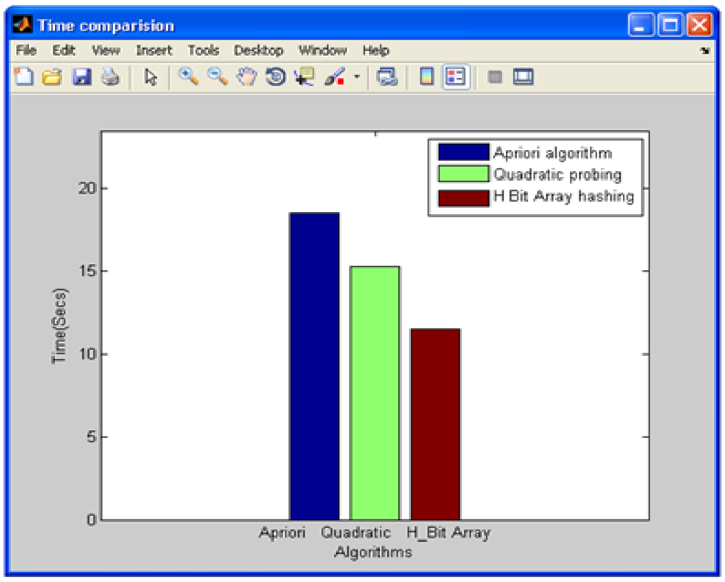

Figure 12. Graph representing the time of the Retail Dataset

The figure 12 shows the computation time, which is taken by the algorithms to generate the candidate and frequent itemsets.

8. Conclusion and Future Enhancement

Mining association rules is one of the most used functions in data mining. The traditional Apriori algorithm iteratively defines the candidate and frequent itemsets. Hash-based itemsets counting technique can improve the performance of ARM. The existing hashing algorithm, HBFI-QP is used for generating the frequent itemsets from the hash tables.

Quadratic probing technique can tie up the empty locations at the time of hash collisions for placing collided itemsets, but the technique does not use all buckets of the hash table and computation time is high.

The proposed hash-based algorithm, H-BAH places the itemsets in the hash table and an H-Bit array is added in the header bucket. If collision occurs while hashing the itemsets, H-Bit array is searched instead of lengthy probing sequences. The linked list structures are constructed with the support count and the list of transactions. The frequent itemsets are generated by scanning the linked list and hash table. A linked structure for each level of the hash table generates the list of transactions for the next higher level. The experimental results show the H-BAH algorithm is efficient when compared to the traditional Apriori and HBFI-QP algorithms in terms of time and memory requirements.

In future, N-Hash and Cuckoo hashing algorithms can be used to improve performance of hashing.

- N-Hash algorithm uses cryptographic hash functions to index the data in hash tables.
- In Cuckoo hashing algorithm, the hash table is split into two smaller tables of equal size, and each hash function provides an index into one of these two tables.
-

REFERENCES

[1] R.Agarwal, T.Imielinski, and A.Swami, (1993), "Mining Association Rules Between Sets of Items in Large Databases", In the proceedings of the ACM SIGMOD International Conference on Management of Data. pp: 207-216.

[2] R.Agarwal and Srikant.R, (1994), "Fast Algorithms for Mining Association Rules", In the proceedings of 20th International Conference on Very Large Data Bases. pp: 487-499.

[3] Chin-Chen Chang and Chin-Yang Lin, (2005), "Perfect Hashing Schemes for Mining Association Rules", The Computer Journal, Vol.48. pp: 168-179.

[4] Ebrahim Ansari Chelche, M.H.Sadreddini and G.H.Dastghaybifard, (2010), "Using Candidate Hashing and Transaction Trimming in Distributed Frequent Itemset Mining", World Applied Sciences Journal.Vol.9 No.12. pp: 1353-1358.

[5] NVB Gangadhara Rao and Sirisha Aguru, (2012), "A Hash based Mining Algorithm for Maximal Frequent Item Sets using Double Hashing", Journal of Advances in Computational Research, Vol.1 N0.1-2

[6] J.Han and M.Kamber, (2006), "Data Mining: Concepts and Techniques", second edition, Morgan Kaufmann Publishers, (ISBN: 1-55860-901-6).

[7] Hassan Najadat, Amani Shatwani and Ghadeer Objedat, (2011), "A New Perfect Hashing and Pruning Algorithm for Mining Association Rule", IBIMA Publishing, Article Id: 652178.

[8] John D.Holt and Soon M.Chung, (2002), "Mining Association Rules using Inverted Hashing and Pruning", ELSEVIER Information Processing letters. pp: 211-220

[9] Jochen Hipp, Ulrich Guntzer, and Gholamreza Nakhaeizadeh, (2000), "Algorithms for Association Rule Mining – A General Survey and comparison", ACM SIGKDD Explorations, Vol. 2 No.1, pp: 58-64

[10] M.Krishnamurthy, A.Kannan, R.Baskaran and R.Deepalakshmi, (2011), "Frequent Itemset Generation using Hashing-Quadratic Probing Technique", European Journal of Scientific Research, Vol. 50, No.4, pp: 523-532

[11] SA Ozel and HA Guvenir, (2001), "An Algorithm for Mining Association Rules using Perfect Hashing and Database Pruning", In the 10th Turkish Symposium on Artificial Intelligence.

[12] Jang Soo Park, Ming-Syan Chen and Philip S.Yu, (1997), "Using a Hash-Based Method with Transaction Trimming for Mining Association Rules", IEEE Transactions on Knowledge and Data Engineering, Vol.9 No.5.

[13] Ramakrishnan Srikant, Quoc Vu and Rakesh Agarwal, (1997), "Mining Association Rules with Item Constraints", In the proceedings of 3rd International Conference on Knowledge Discovery and Data Mining.

[14] T Shintani and M Kitsuregawa, (1996), "Hash Based Parallel Algorithms for Mining Association Rules", In the proceedings of 4th International Conference on Parallel and Distributed Information Systems. pp: 19-30.

[15] Judy C.R.Tseng, Gwo-Jen Hwang and Wen-Fu Tsai, (2006), "A Minimal Perfect Hashing Scheme to Mining Association Rules from Frequently Updated Data", Journal of the Chinese Institute of Engineers, Vol.29, No.3.

[16] K.Vanitha and R.Santhi, (2011), "Using Hash Based Apriori Algorithm to Reduce the Candidate 2-itemsets for Mining Association Rule", Journal of Global Research in Computer Science, Vol.2, No.5, (ISSN-2229-371-X).

[17] DL Yang, CT Pan and Yc Chung, (2011). "An Efficient Hash - Based Method for Discovering the Maximal Frequent Set", Computer Software and Applications Conference. pp: 511-516.

[18] Zhiyong Zeng, Hui Yang and Tao Feng, (2011), "Using HMT and HASH -TREE to Optimize Apriori Algorithm", International Conference on Business Computing and Global Informatization.

[19] Zhuang Chen, Shibang Cai, Qiulin Song, Chonglaizhu, (2011), "An Improved Algorithm Based on Pruning Optimization and Transaction Reduction", In 2nd International Conference on Artificial Intelligence, Management Science and Electronic Commerce (AIMSEC). pp: 1908-1911.

[20] A.M.J.Zubair Rahman, P.Balasubramanie and P.Venkata Krishna, (2009), "A Hash Based Mining Algorithm for Maximal Frequent Itemsets using Linear Probing" Info Comp Journal of Computer Science, Vol.8, No.1, pp: 14-19.