Discovery of Frequent Itemsets Based on Minimum Quantity and Support

Preetham Kumar preetham.kumar@manipal.edu
Senior Lecturer
Department of Information and Communication Technology
Manipal Institute of Technology
Manipal University
Manipal, 576104, India

Ananthanarayana V S anvs@nitk.ac.in
Professor & Head
Department of Information Technology
National Institute of Technology Karnataka,
Surathkal, 575025, India

Abstract

Most of the association rules mining algorithms to discover frequent itemsets do not consider the components of transactions such as total cost, number of items or quantity in which items bought. In a large database it is possible that even if the itemset appears in a very few transactions, it may be purchased in a large quantity for every transaction in which it is present, and may lead to a very high profit. Therefore the quantity and the total cost of the item bought are the most important components, lack of which may lead to loss of information. Our novel method discovers all frequent itemsets based on items quantity, in addition to the discovery of frequent itemsets based on user defined minimum support. In order to achieve this, we first construct a tree containing the quantities of the items bought, as well as the transactions which do not contain these items in a single scan of the database. Then, by scanning the tree, we can discover all frequent itemsets based on user defined minimum quantity as well as support. This method is also found to be more efficient than the Apriori and the FP-tree, which require multiple scans of the database to discover all frequent itemsets based on user defined minimum support.

Keywords: Confidence, Minimum total cost, Number of items, Quantity, Support.

1. INTRODUCTION

Data mining is the extraction of the hidden predictive information from large databases. It is a powerful new technology with great potential to analyze important information stored in large volumes of data. It is one of the steps in knowledge discovery in databases. The goal of knowledge discovery is to utilize the existing data to find new facts and to uncover new relationships that were previously unknown, in an efficient manner with minimum utilization of space and time. There are several data mining techniques [2]. One of them is Association Rules Mining. Among the areas of data mining, the problem of deriving association [1, 2] from data has

received a great deal of attention. This describes potential relations among data items (attribute, variant) in databases. Agarwal et al [1] formulated the problem in 1993. In this problem, we are given a set of items and a large collection of transactions, which are subsets of these items. The task is to find relationship between the presences of various items within this database. An item is a thing that is sold in each transaction. For every item, its item number appears in a particular transaction. Some transactions also contain relevant information of the transactions such as, quantity in which it is bought, cost of the item, customer's age, salary etc. An itemset [1] (this term is used for item set in all books and papers on data mining) is a set of all items. Let A = $\{I_1, I_2, \ldots I_m\}$ be a set of items. Let D, the transaction database, be a set of transactions, where each transactions t is a set of items. Thus, t is a subset of A. A transaction t is said to support an item I_i , if I_i, is present in t. Further, t is said to support a subset of items X contained in A, if t supports each item in X. An itemset X contained in A has a support s in D, if s% of transactions in D support X. For a given transaction database D, an association rule[1] is an expression of the form X->Y, where X and Y are the sets of A and where $X \subseteq A$, $Y \subseteq A$, and $X \cap Y = \Phi$. The intuitive meaning of such a rule is that the transaction of the database which contains X tends to contain Y. The rule X->Y has support s in a given transaction database D if s% of transactions in D support XUY (i.e., both X and Y). This is taken to be the probability, $P(X \cap Y)$. The rule X->Y has confidence c in the transaction database D if c percentage of transactions in D containing X that also contain Y. This is taken to be the conditional probability, P(Y|X). That is, Support(X->Y)=$P(X \cap Y)$=s, Confidence(X->Y)=P(Y|X) = Support(X->Y)/Support (X)=c.

Let D be the transaction database and s be the user specified minimum support. An itemset X contained in A is said to be frequent in D with respect to s, if support value of X is greater than or equal to s. Every frequent itemset satisfies downward closure property, i.e every subset of frequent itemset is frequent.

Mining of association rules is to find all association rules that have support and confidence greater than or equal to the user-specified minimum support and minimum confidence respectively [2,3,4,6]. This problem can be decomposed into the following sub problems:

a) All itemsets that have support above the user specified minimum support are discovered. These itemset are called frequent itemsets.

b) For each frequent itemset, all the rules that have user defined minimum confidence are obtained.

There are many interesting algorithms proposed recently and some of the important ones are the candidate generation based Apriori and its variations and non candidate generation based algorithms such as FP-tree, PC- Tree algorithms.

The algorithm Apriori called as level-wise algorithm operates in a bottom-up, breadth-first search method. It is the most popular algorithm to find all frequent sets, proposed in 1994 by Agrawal et al [7]. It makes use of the downward closure property. The nicety of the method is that before reading the database at every level, it graciously prunes many of the sets, which are unlikely to be frequent.

The number of database passes is equal to the largest size of the frequent itemset. When any one of the frequent itemsets becomes longer, the algorithm has to go through several iterations and, as a result, the performance decreases.

The FP- Tree Growth algorithm is proposed by Han et al [2, 3]. It is a non candidate generation algorithm and adopts a divide and conquers strategy. The following steps are used. (i) Compress the database representing frequent items into FP-tree, but retain the itemset association information (ii) Divide such a compressed database into a set of conditional databases, each associated with one frequent item (iii) Mine each such database separately.

This algorithm requires two scans of the database to discover all frequent sets. The main idea of the algorithm is to maintain a Frequent Pattern Tree of the database.

A frequent pattern tree is a tree structure consisting of an item-prefix-tree and a frequent-item-header table. The FP-Tree is dependent on the support threshold. For different values of threshold the trees are different. Also, it depends on the ordering of the items. The ordering that is followed is the decreasing order of the support counts.

When the database is large, it is sometimes unrealistic to construct a main memory based FP-tree. An interesting alternative is to first partition the database into a set of projected databases,

and then construct an FP-tree and mine it in each projected databases. Such a process can be recursively applied to any projected databases if its FP-tree still can not fit in main memory.
A study on the performance of the FP-growth method shows that it is efficient and scalable for mining both long and short frequent patterns, and is about an order of magnitude faster than the Apriori algorithm.
The Pattern Count tree (PC-tree) is the contribution of V. S. Ananthanarayana et al [5] which is a complete and compact representation of the database. PC-tree is a data structure, which is used to store all the patterns occurring in the tuples of a transaction database, where a count field is associated with each item in every pattern, which is responsible for compact realization of the database.
Each node of the PC-tree consists of (i) item-name (ii) count and (iii) two pointers called child pointer(c) and sibling pointer(s).
In the node, the item-name field specifies the item that the node represents, the count field specifies the number of transactions represented by a portion of the path reaching this node, the c-pointer field represents the pointer to the following pattern and the s-pointer field points to the node which indicates the subsequent other patterns from the node under consideration.
It is proved that, construction of PC-tree and generation of all large itemsets requires a single database scan. Because of compactness of PC-tree, the algorithms based on PC-tree are scalable. Also, in ordered to discover large itemsets a unique ordered FP-tree, called Lexicographically Ordered FP-tree is constructed from a PC-tree without scanning the database.

1.1 Motivation

One of the key features of most of the existing algorithms is that they assume underlying database size is enormous, and involves either a candidate generation process or a non-candidate generation process. The algorithms with candidate generation process require multiple passes over the database and are not storage efficient. In addition, the existing algorithms discover all frequent itemsets based on user defined minimum support without considering the components such as quantity, cost and other attributes which lead to profit.
Consider for example a sample database given in Table 1 in which every element of each transaction represents either the quantity of the respective attribute or the item.

TABLE 1: Sample Database

TID/Attributes	A	B	C	D
1	10	5	0	0
2	0	0	3	0
3	4	0	4	0
4	5	2	5	0
5	0	0	0	10

If an itemset appears in a very few transactions but in a large quantity, then it is possible that buying of this itemsets leads to profit, will not qualify as frequent itemset based on user defined minimum support. This results in a loss of information. In the sample database given in Table 1, if the user defined minimum support is 2 transactions, then an item D is not frequent and will not appear in the set of frequent itemsets, even though it is bought in a large quantity and leads to more profit than other frequent items. This motivated us to propose the following method which discovers all frequent itemsets based on user defined minimum quantity. With this method it is also possible to discover frequent itemsets based on user defined minimum support.

1.2 Frequent itemset and quantity based Frequent itemset or weighted minimum support

If an itemset satisfies user defined minimum support then we call it a frequent itemset. If an itemset satisfies user defined minimum quantity then we say that it is a quantity based frequent itemsets. The weight of an itemset is the ratio of the quantity in which the itemset is bought to the number of transactions in which it is present. For example, if a user defined minimum quantity is equal to 4 then the items A, C, D become quantity based frequent one itemets.

2. PROPOSED METHOD

Our novel method uses the concept of tree called Q-TID Tree. The Q-TID tree consists of information regarding items, the quantity in which the items have been purchased and the TIDs in which these items are not present**.** The structure of this tree is as follows.

2.1 Structure of the Q-TID

The Q-TID tree has three different nodes and is shown in Figure 1.

(i) The first type of node is labeled with item name or attribute name and two pointers, one pointing to the nodes containing quantity, and another is a child pointer pointing to the next attribute. This node represents the head of that particular branch.

(ii) The second type of node labeled as Quantity1, Quantity2 etc, indicates which particular item is purchased. This node has only one pointer pointing to the next object having this particular attribute.

(iii) The third type of node, appears as the last node in every branch of the tree, and is similar to the second type, but consists of information corresponding to the transactions ids(TIDs), which do not contain the particular item. This information represents a whole number which is obtained, forming prime product of prime numbers. If this product is factorized, then it is possible to obtain all the TIDs which do not contain the particular item.

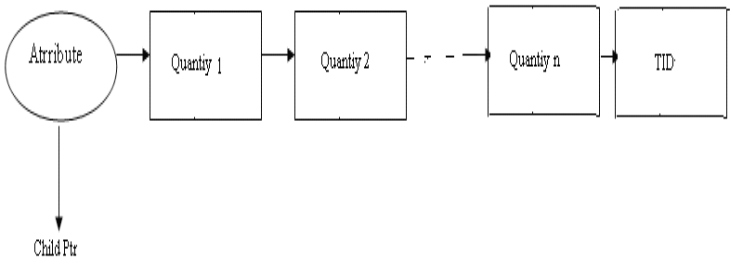

FIGURE 1: A Branch of Q-TID Tree

2.2 Forming the product corresponding to the TIDs

In this step mapping of every transaction which does not contain the particular item to a prime number, and then its product, is obtained with the already existing number in the last node of every branch of the tree. The prime numbers assigned to the TID are shown in Table 2. For example, the first prime number 2 is assigned to 1st TID and a second prime number 3 is assigned to the 2nd TID and so on.

TABLE 2: Prime Number Table

Positive integers	1	2	3	4	5	6	7	8	9
Prime numbers	2	3	5	7	11	13	17	19	23

The prime numbers are used to make our process very simple because of their following property.
If a, b are any two prime numbers with multiplicity m and n then their product is $a^m b^n$. With this product, it is possible to obtain the original prime numbers used for forming the product.
For example, consider the prime numbers 2, 3, 5 and its product is 30. With this product it is possible derive all prime numbers involved in forming the product. Therefore one can see that the prime numbers 2, 3 and 5 are involved in forming the product. In our case, these numbers represent the TIDs 1, 2 and 3. The Q-TID tree corresponding to the sample database given in Table 1 is shown in Figure 2.

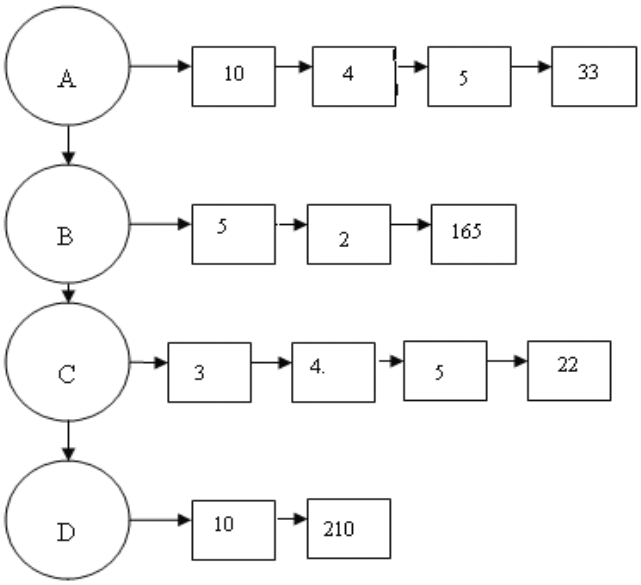

FIGURE 2: Q-TID tree for Table1

The Q-TID tree algorithm involves following steps. They are

- **A** Construction of Q-TID Tree.
- **B** Removal of infrequent items based on user defined minimum quantity of Q-TID Tree
- **C** Discovery of frequent itemsets based on user defined minimum quantity
- **D** Discovery of frequent itemsets based on user defined weighted minimum support

A Algorithm for constructing Q-TID Tree

Input: The database D
Output: Q-TID tree
Create an attribute or item node labeled with respective item and its corresponding TID node labeled with 1
for each item I in a transaction t € D
do begin
create a node labeled with its quantity and add
these nodes to the respective item or attribute
node.
If I is not in t
find t^{th} prime number and its product with
already existing prime number in the last node TID of
the branch corresponding to I.
end

B Algorithm for Reducing the Q-TID Tree

Input : weighted_min_qsup =user specified minimum quantity

n = number of transactions in which a particular item is present

Output: Q-TID tree contains only frequent items based on user defined quantity in addition to the information corresponding to the TIDs which do not contain a particular item.

```
for each item or attribute in a Q-TID tree
do begin
        If sum (quantities of all nodes except a last node/n) < weighted_min_qsup then
        remove those nodes corresponding to the quantities from the branch which
        originates from the attribute node labeled with I
 end
```

C Algorithm to discover all frequent itemsets based on quantity

Input: A Reduced Q-TID tree, weighted_min_qsup=user specified minimum quantity.
F= {set of all frequent one itemsets based on quantity}
P=set of all non empty subsets of F excluding the sets containing one attribute.

Output: set of all frequent itemsets =F_Q .

```
begin
   F_Q = { set of all frequent attributes or one itemset based on quantity }
   for each f in P do
        begin
            T = { TIDs of first attribute in f }
              for each m in f other than first attribute do
              begin
                    T=T ∩ { TIDs of m}
              end
             If T is non empty then
               If (sum of quantities of T /|T| ) >= weighted_min_qsup
                    F_Q = F_Q U f
        end
```

D Algorithm to discover all frequent itemsets based on minimum support.

Input: A Reduced Q-TID tree, min_sup=user specified minimum support, N =Total number of transactions.
L= {set of all frequent one itemsets based on minimum support}
P(L) ={set of all non empty subsets of L excluding the sets containing one attribute or one item }.
F_M = {set of all frequent attributes or one itemset based on minimum support}
Output: set of all frequent itemsets. F_M

```
for each f in P
do begin
    T = {prime factors corresponding to TIDs which do not contain the first attribute in f }
    for each m in f other than first attribute
    do begin
            T=T U { prime factors of corresponding TID s which do not contain m}
    end
    If T is non empty
        If ( N - | T | ) >= min_sup
            F_M = F_M U f
end
```

Illustration:

Consider for example, a sample database given in Table 1,
If we consider weighted_min_qsup = 4 then the attributes A, C and D will be frequent in the database. The Reduced Q-TID is shown in Figure 3.

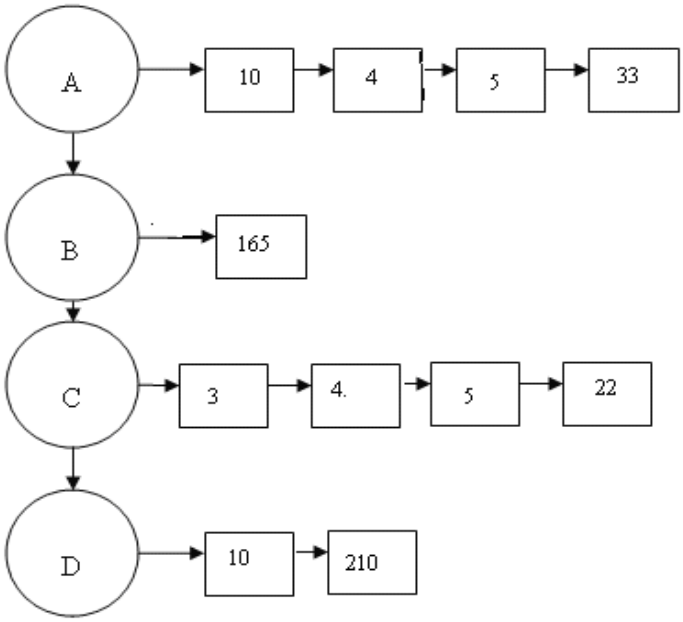

FIGURE 3: A Reduced Q-TID Tree

we observe that $\mathbf{F_Q}$ ={A, C, D} . Therefore
P = {{A, C}, {A, D}{C,D}, {A, C, D} }
Consider a set {A, C}, which appears in transactions 3 and 4.
i.e T={3,4}.
The TIDs are obtained by using the following procedure .
The last node of the item A corresponding to the TID contains 33, which involves prime numbers 3 and 11. Similarly, the last of C contains 22, which involves prime numbers 2 and 11. The union of these numbers will give us set {2, 3, 11}. These numbers correspond to the transaction 1, 2, 5. Hence both A and C will occur only in transactions 3 and 4. Further, the sum of their corresponding quantities is equal to 9+9=18 and weighted support of {A,C} = 18/2 =9.
Hence {A, C}is frequent.
By similar arguments, we found that the sets {A, D}, {C, D} and {A,C, D} are infrequent sets.
Hence $\mathbf{F_Q}$ = { {A}, {C}, {D},{ A, C}}
Now if user defined minimum support is equal to 2 then the items A, B, C will be frequent one itemsets.
Therefore **L =** {A, B, C} . Therefore
P(L) = { {A, B}, {A,C}, {B,C}, {A, B, C} }. Now applying above algorithm, we see that $\mathbf{F_M}$ = { {A}, {B}, {C}}.
It can be observe from the Q-TID tree that itemsets
{A, B} is not present in transactions 2, 3 and 5.
{A, C } is not present in transactions 1, 2 and 5.
{B, C} is not present in transactions1, 2, 3 and 5.
{A, B, C} is not present in transactions 1, 2, 3 and 5.
Since N =5, we have supports of {A, B} = 2, {A, C } = 2, {B, C}= 2 and {A, B, C} =1. This shows that the itemset {A, B, C } is not frequent.
Now $\mathbf{F_M}$ = { {A}, {B}, {C}, {A, B}, {A,C}, {B,C}}.

Note : If an item is present in every transaction, then the node corresponding to the TIDs which do not contain that particular item will not appear in the Q-TID Tree.

The following example illustrates this situation. Consider a sample database given in Table 3 which has 3 attributes and 3 tuples.

TABLE 3: Sample Database

TID	A1	A2	A3
T1	3	2	1
T2	6	2	1
T3	2	2	4

The Q-TID tree corresponding to Table 3 is given in Figure 4. It does not contain nodes corresponding to the TIDs since every item is present in every transaction.

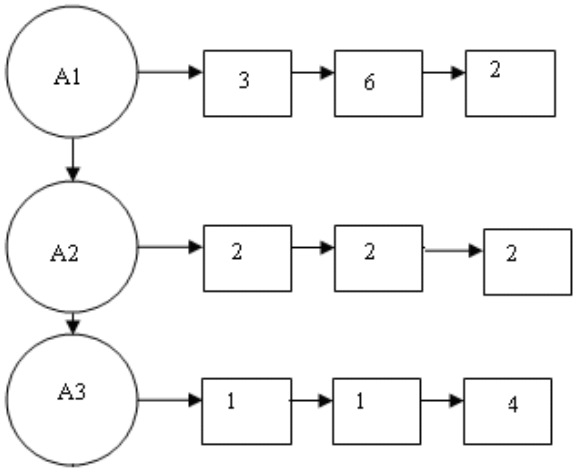

Figure 4: Q-TID Tree of Table3

3. PERFORMANCE ANALYSIS

Theoretical Analysis

The algorithm consists of four steps

A Construction of Q-TID Tree

If the given database contains N transactions then this can be done in O(N) time.

B Removal of infrequent items based on user defined weighted minimum support of Q-TID Tree

If there are m items which are not frequent then all the nodes containing quantity information of these m attributes are deleted. If there are on an average g nodes for every attribute then this step is in O(mg).

C Discovery of frequent itemsets based on user defined weighted minimum support

The major step is the process of finding power set P. If there are n frequent 1-item attribute sets then this step is in $O(2^n)$.

D Discovery of frequent itemsets based on user defined minimum support

The major step is the process of finding P(L). If there are n frequent 1-item attribute sets then this step is in $O(2^n)$

4. EXPERIMENTAL ANALYSIS

For the performance analysis, a simulation of buying patterns[10] of the customers in retail patterns was generated and in the data set which we used every attribute value of the transaction

was considered as quantity of the corresponding attribute in the database**.** We compared our algorithm with FP- tree and Apriori and found that ours was time efficient.
The above algorithm is implemented and used for discovering frequent itemsets based on quantity as well as minimum support for data sets containing the transactions 100, 500, 1000, 5000 with 20 attributes or items. The time required to discover all frequent itemsets is shown Figure 5 and Figure 6 respectively.

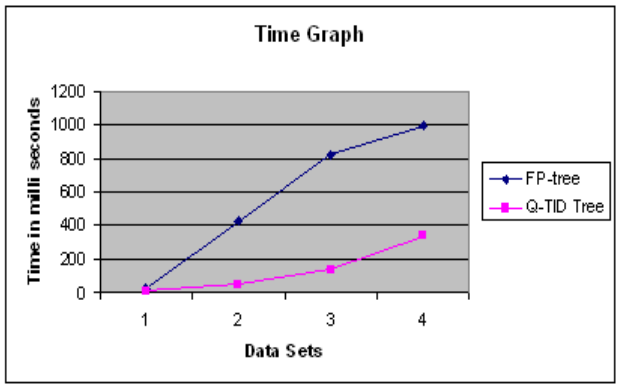

FIGURE 5: Q-TID Vs FP-Tree

The proposed algorithm is superior to the FP-tree algorithm in following ways:

1. Scans database only once.
2. No sorting of each item of the transaction.
3. No repeatedly searching the header table for maintaining links, while inserting a new node into tree.

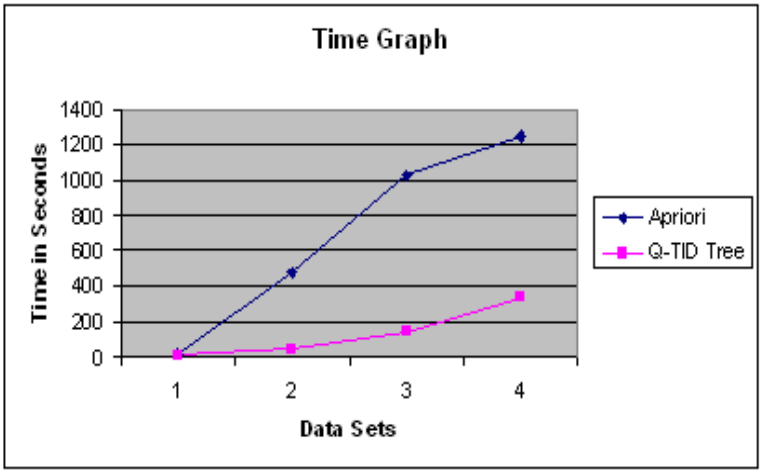

FIGURE 6: Q-TID Vs Apriori

The proposed algorithm is superior to the Apriori algorithm in the following ways:

1. Scans database only once.
2. It does not involve candidate generation method.

5. CONCLUSION

The Q-TID Tree Algorithm is a new method for finding frequent itemsets based on user defined quantity and minimum support. It is found that this algorithm is time efficient when compared to the FP-tree and Apriori. Since we have used small data sets, this algorithm can be fine tuned with large databases. This method may be modified further to store TIDs more efficiently.

6. REFERENCES

[1] Jiawei Han Micheline Kamber, Data Mining Concepts and Techniques .Morgan Kaufman, San Francisco, CA, 2001

[2] Han, J., Pei, J., Yin, Y. “Mining Frequent Patterns without Candidate Generation”, Proc. of ACM-SIGMOD International Conference Management of Data. Dallas, 2000, TX, 1-12.

[3] Han J, Pei Jian, Runying Mao ” Mining Frequent Patterns without Candidate Generation: A Frequent Pattern Tree Approach” , Data Mining and Knowledge Discovery, Kluwer Academic Publishers, Netherland” 2004, pp 53-87.

[4] R. Hemlata, A. Krishnan, C. Scenthamarai, R. Hemamalini. ”Frequent Pattern Discovery based on Co-occurrence Frequent Tree”. In Proceeding ICISIP-2005.

[5] Ananthanarayana V. S, Subramanian, D.K., Narasimha Murthy M. “Scalable, distributed and dynamic mining of association rules” In Proceedings of HIPC'00. Springer Verlag Berlin,Heidelberg,2000, 559-566.

[6] R. Srikant and R. Agarwal. “Mining generalized association rules”, Proceedings of International Conference on Very Large Data Bases 1995, pages 407-419.

[7] Rakesh Agrawal, Tomasz Imielinski, Arun Swami, “Mining Association Rules between Sets of Items in Large databases”, Proceedings of the 1993 ACM SIGMOD Conference Washington DC, USA, May 1993

[8] Rakesh Agarwal, Ramakrishnan Srikant, “Fast algorithms for mining Association Rules”, In proceedings of the 20th International Conference on Very Large databases, Santigo, 1994, pp 478-499.

[9] S.Y.Wur and Y.Leu, “An effective Boolean Algorithm for mining Association Rules in large databases”, The 6th International conference on Database systems for Advanced Applications, 1999, pp 179-186.

[10] IBM/Quest/Synthetic data.