Modified Frequent Itemset Mining using Itemset Tidset pair

[1]Dr. Jitendra Agrawal, [2]Dr. Shikha Agrawal
[1]School of Information Technology, Rajiv Gandhi Proudyogiki Vishwavidyalaya, Bhopal, India
[2]Dept. of CSE, UIT, Rajiv Gandhi Proudyogiki Vishwavidyalaya, Bhopal, India

Abstract
Many algorithms have been proposed to mine all the Frequent Itemsets in a transaction database. Most of these algorithms were based on apriori-like candidate set generation-and-test approach to generate the association rules from the transactional database. The apriori-like candidate approach can suffer from two nontrivial costs: it needs to generate a huge number of candidate sets, and it may need to repeatedly scan the database and check a large set of candidates by pattern matching. This paper thus attempts to propose a new data-mining algorithm "Modified FIMIT" (MFMIT) for mining all the Frequent Itemsets in a transaction database using Vertical Transaction Database format. To improve the performance of the FIMIT in terms of time, MFIMIT uses sentinel arrangement on the cost of storage. The proposed MFIMIT is efficient and scalable over large databases, and is faster than the previously proposed methods.

Keywords
Association Rule; Data Mining; Frequent Itemset; Vertical Database Format

I. Introduction

Association rule mining [2-3], one of the most important applications of data mining [1], is used to identify relationships among a set of items in a database. These relationships are not based on inherent properties of the data themselves (as with functional dependencies), but rather based on co-occurrence of the data items. Association Rule Mining is a two-step process.

1. Find all Frequent Itemsets: Each of these itemsets will occur at least as frequently as a pre-determined minimum support count.
2. Generate strong Association Rules from the Frequent Itemsets: These rules must satisfy minimum support and minimum confidence.

The second step is the easier of the two. The overall performance of mining association rules is determined by the first step. Many algorithms have been proposed to mine all the Frequent Itemsets in a transaction database. These algorithms differ from one another in the method of handling the candidate sets and the method of reducing the number of database passes.

The Apriori algorithm [3] proposed by Agrawal and Srikant, is the most popular algorithm to find all the frequent itemsets. It uses Horizontal Database format and Breadth First Search strategy and needs a database scan in each iteration. Another algorithm, Apriori-Tid [3] is also same as Apriori except that it creates a candidate transaction database after each iteration. But both require a pruning step to prune the candidate sets. These algorithms waste a lot of time in multiple scaning a database. The Partition algorithm [4] partitions the original database. It scans the original database only twice, to discover all frequent itemsets. But this Partition algorithm requires the Apriori method in the first phase of the algorithm. The Pincer-Search algorithm [5] reduces the number of database scan required to mine all Frequent Itemsets. But this also follows the Apriori procedure. The Dynamic Itemset Counting (DIC) algorithm [06] requires four different structures Dashed Box, Dashed Circle, Solid Box, Solid Circle. The DIC algorithm reduces the number of database scan required to mine all Frequent Itemsets. But this algorithm also requires multiple database passes.

The FP-Tree Growth algorithm [7-8], proposed by Han, maintains a Frequent Pattern Tree (FP-Tree) of the database. This algorithm also uses Horizontal Database format. This algorithm needs to scan the database only once to create the FP Tree of the database. This method does not generate any set of candidate itemsets. It does not require pruning step. It does not require checking a large set of candidate itemsets by Pattern Matching. But the algorithm requires creating and maintaining FP-Tree structure of the database and for this the FP-Tree algorithm requires two scan of the whole database in Phase-I of the algorithm. We observe that all the algorithms have two important and common steps:

(i). Candidate Generation and
(ii). I/O Operation.

In the MFIMIT, We use the Vertical Database Format to mine all the Frequent Itemsets in a transaction database. Moreover, the proposed method does not require generation of set of Candidate Itemsets and the Pruning Step.

II. Proposed Work

This is a modified version of FIMIT algorithm[12]. Like FIMIT, the algorithm MFIMIT, is a level-wise algorithm and iterative in nature. It uses Breadth First Search strategy like Apriori algorithm.

To reduce the number of comparisons in the join step of FIMIT algorithm, as part of MFIMIT, we eliminate the join step with a sentinel arrangement in the three lists (i.e. primary, secondary and next list).

Here k-itemsets are used to explore (k+1)-itemsets. first the set of frequent 1-Itemsets is found. This set is denoted by L1. L1 is used to find L2, the set of frequent 2-Itemsets, which is used to find L3 and so on until no more frequent k-Itemsets can be found. It uses vertical transaction database or first creates a vertical database from given horizontal database.

Sentinel Arrangement: This arrangement provides a way to find union of two itemsets IP and IS which are joinable.

Rest of the methodology of MFIMIT is similar to the method FIMIT.

MFIMIT Algorithm

Objective: Find all frequent itemsets using an iterative levelwise approach without candidate generation using the concept of (Itemset : Tidset) pair.

Input: Database TDBH of transactions (i.e. horizontal database), minimum support count threshold σmin.

Output: L' will be the list of all frequent itemsets in database (with their support counts).

Methodology:

1. Convert horizontal transaction database TDBH into vertical transaction database TDBV.
2. Find I, set of all items in TDBV (or assume given).
3. Find T, set of all transaction Ids i.e. Tids in TDBV (or assume given) where TDBV = all the 1-Itemsets with their Tidsets.
4. For k=1 where k represents the iteration number. Count the support of 1-Itemsets from the database TDBV, to determine L1, by calculating the size of list of Tids or size of Tidsets, for each 1-Itemset.
5. Arrange the TDBV such that 1-Itemsets are in ascending order of their support count and their Tidsets are in ascending order of their Tids. [Optional Step]
   L1 = ({Frequent 1-Itemsets} : {Their Tidsets})
   L'1 = ({Frequent 1-Itemsets} : Support Count)
6. Take only 'Itemsets' and |Tidsets| for support count for each pair (Itemset : Tidset) in L1, to find L'1. Ignore Tidset for each Itemset in L'1 and also ignore both temporary and permanent sentinel elements.
7. From 2nd iteration i.e. k=2 onwards, continue finding L'k iteratively until L'k's value is not null.

for (k=2; L'k-1 ≠ ∅; k++) {
Lk = GetNextLevelFrequentItemsetTidsetPairs2 (Lk-1, σmin);
L'k = Take only 'Itemset' and |Tidset| for support count, for each pair (Itemset : Tidset) of Lk. Ignore 'Tidset' for each 'Itemset' in L'k and also ignore both temporary and permanent Sentinel elements in L'k.
}

8. Return L' = Sum of all L'k

Procedure GetNextLevelFrequentItemsetTidsetPairs2 (Lk-1, σmin)

This procedure generates all frequent itemsets with their support counts.

Assume, Lk-1 is frequent (k-1)-Itemsets and σmin is Minimum Support Count Threshold, then the procedure looks like:

Procedure GetNextLevelFrequentItemsetTidsetPairs2 (Lk-1, σ_{min})
{
Lk = (∅ : ∅);
// Primary Pointer P.
P ← 1 i.e. P points to Ist Itemset-Tidset Pair of Lk-1
// Secondary Pointer S.
S ← (P+1) i.e. S points to (P+1)th Itemset-Tidset Pair of Lk-1
// P ≠ End of Lk-1 i.e. Perm. Sentinel i.e. Last Itemset-Tidset Pair of Lk-1
while (P ≠ Permanent Sentinel) {
while (S = = Temp. Sentinel or S = = Perm. Sentinel or P = = Temp. Sentinel) {
if (S = = Temp. Sentinel or S = = Perm. Sentinel) then {
// Result: All Frequent Itemsets with their Support Counts.
Add a Temp. Sentinel to Lk;
P++;
S ← (P+1); }
if (P = = Perm. Sentinel) then {
Add a Perm. Sentinel to Lk;}
goto return Statement; }
// Intersection of Tidsets at P and S
Find { $T_P \cap T_S$ };
// Removal of Infrequent Itemsets.
if ($|TP \cap TS| >= \sigma min$) then {
// Join Step : Union of Itemsets at P and S.
// IP ∪ IS = IP[1], IP[2], IP[3], …….. IP[k-1], IS[k-1]
Find { IP ∪ IS };
// Frequent (Itemset : Tidset) Pair.
IN : TN = ($I_P \cup I_S$: $T_P \cap T_S$);
Lk = L_k ∪ (IN : TN)
}
S++; } // while closes.
return Lk;
} // Procedure Ends.

Note:

1. Each element of list Lk is of the form (Itemset : Tidset) pair like each element or node in primary list i.e. ({Itemset} : {Tidset}). It has both temporary and permanent Sentinel Arrangement.
2. Each element of List L'k is of the form (Itemset : Support Count) pair like each element or node in F.I. Reservoir List i.e. ({Itemset} : Support Count). It does not have Sentinel Arrangement. Which means Lk is without sentinels and Tidsets = L'k.
3. We can get Support Count of an itemset by finding magnitude or size of its Tidset i.e. Support Count = $|$ Tidset $|$
4. Temporary Sentinel = ({$} : {0}).
   Permanent Sentinel = ({$} : {-1}).

We have chosen '$' as an item in itemset for both sentinels and '0' for temporary and '-1' for permanent sentinel, as Tid in Tidset. Thus, we have these two (Itemset : idset) pair for sentinel elements or nodes in the list Lk. No transactions are assigned IDs as these integers '0' and '–1' in the given database. Also '$' is not any item in the database.

IP = Itemset pointed by Pointer P.
IS = Itemset pointed by Pointer S.
TP = Tidset pointed by Pointer P.
TS = Tidset pointed by Pointer S.
So, the Procedure

GetNextLevelFrequentItemsetTidsetPairs2 (L_{k-1}, σ_{min}) can be rewritten as follows:

Procedure GetNextLevelFrequentItemsetTidsetPairs2 (L_{k-1}, σ_{min})

Let Lk-1 be the Frequent (k-1)-Itemsets and σmin be the minimum support count threshold, then, the algorithm looks like:

Procedure GetNextLevelFrequentItemsetTidsetPairs2 (L_{k-1}, σ_{min})
{
Lk = (∅ : ∅);
// Primary Pointer P.
P← 1 i.e. P points to Ist Itemset-Tidset Pair of Lk-1
// Secondary Pointer S.
S ← (P+1) i.e. S points to (P+1)th Itemset-Tidset Pair of Lk-1
// P ≠ End of Lk-1 i.e. Perm. Sentinel i.e. Last Itemset-Tidset Pair of Lk-1
while (TP ≠ -1)
{
while (TS = = 0 or TS = = (-1) or TP = = 0)

{
if (TS = = 0 or TS = = (-1)) then {
Add a Temp. Sentinel to LK i.e. LK = LK ∪{$} : {0});
}
P++;
S ← (P+1);
if (TP = = -1) then {
Add a Perm. Sentinel to Lk i.e. Lk = Lk ∪ ({$} : {-1});
goto return Statement; }
}
// Intersection of Tidsets at P and S
Find { $T_P \cap T_S$ };
// Removal of Infrequent Itemsets.
if ($|T_P \cap T_S| >= \sigma min$) then {
// Join Step : Union of Itemsets at P and S.
// IP ∪ IS = IP[1], IP[2], IP[3],…….. IP[k-1], IS[k-1]
Find { IP ∪ IS };
// Frequent (Itemset : Tidset) Pair.
IN : TN = (IP ∪ IS : TP ∩ TS);
$Lk = L_k \cup (I_N : T_N)$;
}
S++;
} // while closes.
return Lk;
} // Procedure Ends.

Explanation of the MFIMIT with example

The algorithm MFIMIT works as follows:

Given : A horizontal transaction database TDBH contains a set of transactions as shown in Table 3.3, a set of Items I = {a, b, c, d, e} and a set of Transaction IDs i.e. Tids T = {1, 2, 3, 4, 5, 6, 7, 8, 9} and a minimum support threshold (i.e. minimum support count=2) σ_{min}. The transaction database contains a set of transactions.

Table 3: Horizontal Transaction Database for MFIMIT

Transaction ID or Tid	List of Items in the transaction
1	a, b, e
2	b, d
3	b, c
4	a, b, d
5	a, c
6	b, c
7	a, c
8	a, b, c, e
9	a, b, c

Prerequisites: The method MFIMIT uses vertical database or it first converts a given horizontal database into vertical database to form a list of Tids of all items in I. The method requires only a single pass (scan) of the given horizontal transaction database TDBH to know the Tidset of each item in I i.e., to construct the vertical transaction database TDBV. This vertical transaction database TDBV is a list of all 1-Itemsets with their Tidsets as shown in Table 4.

Table 4 : Vertical transaction database for MFIMIT

Items	Tids of Transactions containing the Item or Tidsets.
A	1, 4,5,7,8,9
B	1,2,3,4,6,8,9
C	3,5,6,7,8,9
D	2,4
E	1,8

First Pass:

First pass of MFIMIT is same as first pass of FIMIT. So, the steps involved in first pass are:

1. Using above vertical database TDBV, first find support of each 1-Itemset by finding the size of Tidset for each 1-Itemset.
2. Arrange all the 1-Itemsets in lexicographical order of their items and Tidsets in ascending order of their Tids(i.e. Transaction IDs).
3. Compare the support of each 1-Itemset with the minimum support count and find all frequent 1-Itemsets directly.
4. Keep all the frequent 1-Itemsets with their Tidsets in a list named Primary List (or Lk). It is used for further processing.
5. Put all these frequent 1-Itemsets with their support count in a separate list, which will store all the frequent itemsets generated by the algorithm as it proceeds. We call this list as Frequent Itemsets Reservoir (or F.I. List). This list contains the result.

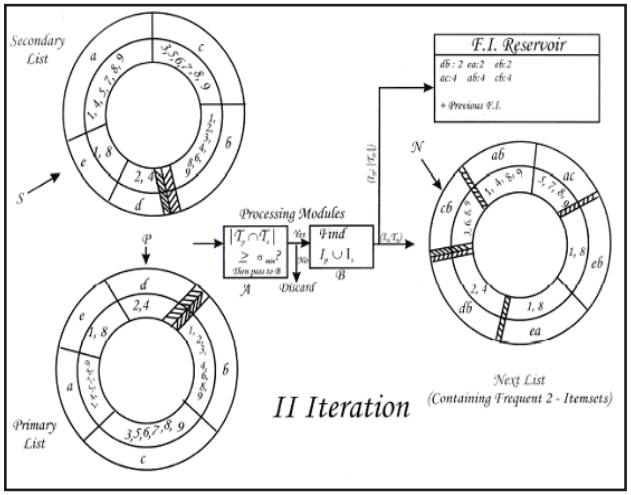
After the First Pass, subsequent passes are iterative in nature.

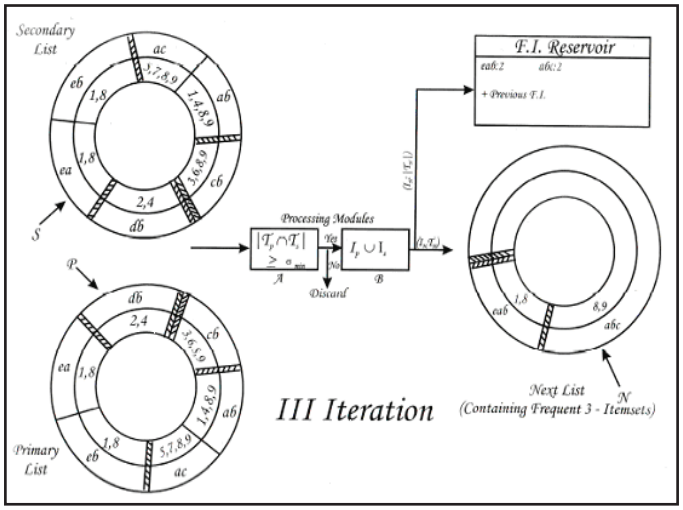
Second Pass:

Now to explain and understand the functioning of the algorithm in the subsequent iterations, let's consider the following pictorial views (Fig. 5, Fig. 6 and Fig. 7)

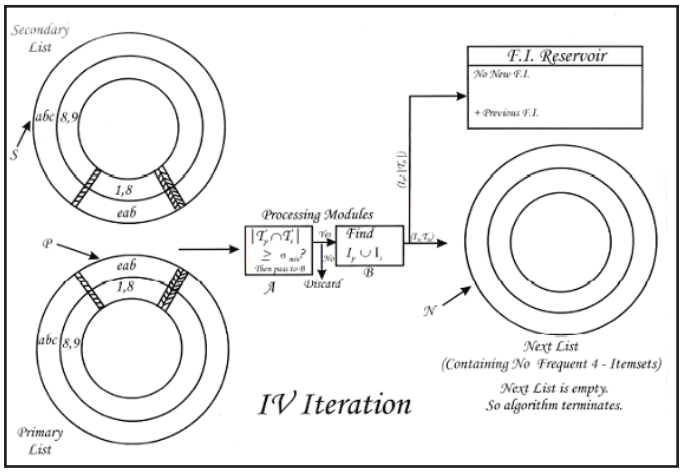
Pictorial View contains:

- Primary List(Lk-1)
- Secondary List(Lk-1)
- Next List(Lk)
- Two processing modules A & B
- Two pointers P and S, one in each Primary and Secondary List
- A pointer N in the Next List and
- A frequent itemsets reservoir or List (F.I. List).





The algorithm uses three lists:



1. Primary List (Lk-1),
2. Secondary List (Lk-1), which is a copy of Primary list and
3. Next list (Lk).

Each element of each list Primary, Secondary or Next List has two fields Itemset and Transaction IDs or Tidset, in which this itemset is present.

- A pointer P called Primary Pointer points to Primary list
- A pointer S called Secondary Pointer points to Secondary list
- A pointer N called Next Pointer points to Next List.

Processing Modules are shown only to give an expression of the necessary processing used in the method. There are two modules:

Processing Module A: The steps involved are:

1. Find the Intersection of the Tidsets of the Itemsets pointed by pointers P and S.
2. Then it find the magnitude (or size) of the intersection set to know the Support Count.
3. Then, compare the size of the intersection set with the given minimum Support Threshold Count i.e. σ_{min}. If the size of intersection set is greater or equal to the given σ_{min}, then the intersection set and its size is passed to the module B; otherwise it is discarded and no further processing is done in module B.

The steps can be represented as:

Find $(T_P \cap T_S)$
Find $|T_P \cap T_S|$
IF $|T_P \cap T_S| >= \sigma_{min}$ THEN pass to module B.

Processing Module B: The steps involved are:

(a). If the intersection set is not discarded in module A, then further processing is done in module B. It combines both itemsets pointed by pointers P and S by finding Union of the Itemsets. Thus, module B performs the Join Step. Here it is not necessary to find whether the itemsets pointed by P and S are joinable (join compatible) or not because pointers P and S always point to joinable itemsets in this method.

(b). It then passes the union of the itemsets and the intersection of the Tidsets, pointed by pointers P and S to the Next List (Lk)

(c). It also passes the union of the Itemsets and magnitude of the intersection set of the Tidsets to the frequent itemset reservoir (F.I. List).

The steps can be represented as:

Find $(I_P \cup I_S)$
$(I_N : T_N) = (I_P \cup I_S : T_P \cap T_S)$ to Next List.
$(I_N : |T_N|) = (I_P \cup I_S : |T_P \cap T_S|)$ to F.I. Reservoir

Sentinal Arrangement: This method uses a Sentinel Arrangement. There are two types of sentinels used in this method:

1. Temporary Sentinel
2. Final Sentinel.

The arrangement of the two sentinels (mainly the temporary sentinel) eliminates the Join Steps of method FIMIT and eliminates unnecessary processing of Itemsets to find union of two k-Itemsets to form a (k+1)-Itemsets. It also eliminates the need to find whether two k-Itemsets are joinable or not. That is, it improves performance of the method in terms of time on the cost of storage. The permanent sentinel simply indicates the end of any list Primary, Secondary or Next List.

Steps involved in second pass are:

1. When the Primary List is formed after the Ist pass, a permanent sentinel is added at the end of the Primary List (as well as Secondary List).
2. Before starting any iteration, the primary pointer P is set to point the first element of the Primary List (Lk-1).
3. Secondary pointer S is set to point the element, next to the element pointed by primary pointer P, in the secondary list (Lk-1).
4. Pointer N is set to point the first empty location in the Next List (Lk).
5. The elements pointed by pointers P and S are processed in the processing modules and then the two pointers are updated as follows:

(a). If Temporary Sentinel or Permanent Sentinel is encountered by the secondary pointer S, then a temporary sentinel is added to the Next List, at empty location pointed by pointer N.

The primary pointer P is increment by 1 and the secondary pointer S is set to point to the element, next to the element pointed by primary pointer P.

(b). If Temporary Sentinel is encountered by the primary pointer P then the Pointer P is increment by 1and the secondary pointer S is set to point to the element, next to the element pointed by primary pointer P.

(c). If Permanent Sentinel is encountered by the primary pointer P then a permanent sentinel is added to the Next List at empty location pointed by pointer N and the current iteration is completed.

6. When the Permanent Sentinel is encountered by primary pointer P then the Primary List (as well as Secondary List) is processed completely and the Next List is ready for further

processing in the (subsequent) next iteration.

7. Then the Primary List and Secondary List are removed from memory. It means after complete processing of Primary List (and also Secondary List), the Next List is copied to both the Primary List and Secondary List, because Secondary List is a copy (or image) of the Primary List.
8. The Next List becomes the Primary List and Secondary List and we make a new empty Next List for Next iteration.
9. Now, all the frequent 2-Itemsets are in the F.I. Reservoir with their support counts and also the Next List contains all the frequent 2-Itemsets only with their Tidsets.

We can have a sentinel element in the lists, by putting a $ (dollar) symbol in the itemset and putting a 0 (zero) in the Tidset for temporary sentinel and putting a -1 (minus one) in the Tidset for Final Sentinel because $ is not any item in the set I, i.e. $ $\notin$ I and 0 & -1 are not IDs of any transaction, i.e. {0, -1} $\notin$ T.

In the pictorial view I and II, temporary sentinel is represented by a single hatched block and Permanent Sentinel is represented by a double hatched block.

Next Pass:

1. The Primary List (Lk-1) and Secondary List (Lk-1) are processed as above (as previous iteration) recursively.
2. This process continues until the Next List (Lk) formed in current iteration becomes empty i.e., until it does not contain any valid Itemset or element, except the sentinel elements. Then, we will have all the Frequent Itemsets with their support count in the F.I. reservoir.
3. If the Next List (Lk) formed in current iteration is empty, then the algorithm terminates.

Advantages of MFIMIT

Since MFIMIT is a modification of FIMIT, most of the advantages of MFIMIT are similar to the FIMIT, except some additional advantages of MFIMIT over FIMIT, which are described here explicitly. The advantages of MFIMIT are as follows:

1. In MFIMIT, the arrangement of two sentinels (mainly the temporary sentinel) eliminates the join step of method FIMIT and eliminates unnecessary processing of itemsets to find union of two k-Itemsets to form a (k+1)-Itemset.
2. It means it eliminates the need to find whether the two k-Itemsets are joinable or not. Hence, it improves performance of the algorithm in terms of time on the cost of storage.
3. In MFIMIT, the calculation of intersection set of two Tidsets and calculation of its size for checking with the minimum support count is done before the calculation of union of the two itemsets, so it eliminates the need of unnecessary calculation of union of each two k-Itemsets to find (k+1)-Itemsets as compared to FIMIT.

III. Conclusion

In this paper, we proposed new scheme for extracting association rules from transactional datasets. Here we have proposed modification to FIMIT and named it as MFIMIT(Modified FIMIT). MFMIT is iterative, level-wise algorithms, based on breadth first search strategy like Apriori and uses vertical transaction database format to mine all the frequent item-sets. This algorithm skips generation of set of candidate item sets and the pruning step and also reduces number of database scans. Their main strength is not the speed, but the simplicity of its structure and ease of implementation.

Refrences

[1] Usama Fayyad, Gregory Piatetsky-Shapiro, Padhraic Symth,"From Data Mining to Knowledge discovery in databases ", AI MAGAZINE, American Association for Artificial Intelligence, 1996.

[2] R. Agarwal, T. Imielinski, A. Swami,"Mining association rules between set of items in large databases", In Proceeding of the ACM SIGMOD International conference on Management Data, Washington, DC, May 1993, pp. 207-216.

[3] R. Agrawal, R. Srikant,"Fast algorithms for mining association rules", In Proceeding of the International . Conference on Very Large Databases (VLDB'94), Santiago, Chile, pp. 487-499, 1994.

[4] A. Savasere, E. Omiecinski, S. Navathe,"An efficient algorithm for mining association rules in large databases", In Proceeding of the International. Conference Very Large Data Bases (VLDB'95), Zurich, Switzerland, pp. 432-443, 1995.

[5] D. I. Lin, Z. M. Kedem,"Pincer-search: A new algorithm for discovering maximum frequent set. In Sixth International Conference on Extending Database Technology, pp. 84-92, 1998.

[6] S. Brin, R. Motwani, J. D. Ullman, S. Tsur,"Dynamic itemset counting and implication rules for market basket analysis", In Proceeding of the ACM-SIGMOD International Conference on Management of Data (SIGMOD'97), Tucson, AZ, pp. 255-264, 1997.

[7] Qiankun Zhao, Sourav S. Bhowmick," Association Rule Mining : A Surve", Singapore, pp. 1-20.

[8] J. Han, J. Pei, Y. Yin,"Mining frequent patterns without candidate generation", In Proceeding of the ACM-SIGMOD International Conference on Management of Data (SIGMOD'00), Dallas, TX, pp. 1-12, 2000.

[9] Mohammed J. Zaki, K. Gouda,"Fast Vertical Mining using Diffsets", In Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining, New York, USA, pp. 326-335, 2003.

[10] P. Shenoy, J. R. Haritsa, S. Sudarshan,"Turbo-charging vertical mining of large databases", In Proceeding of the International Conference on Management of Data, 2000.

[11] M. J. Zaki, C. J. Hsiao,"CHARM : An efficient algorithm for closed itemset mining", In Proceeding of the 2nd SIAM International Conference on Data Mining, Arlington, VA, pp. 457–473, 2002.

[12] J. Agrawal, R.C. Jain,"An efficient algorithm for mining frequent itemsets", International Conference on Advances in Computing, Control, and Telecommunication Technologies, Kerala, India, pp. 179-183, 2009.

Dr. Jitendra Agrawal was born in 1974, is an Assistant Professor in the Department of Computer Science & Engineering at the Rajiv Gandhi Proudyogiki Vishwavidyalaya, MP, India. He earned his Master Degree from Samrat Ashok Technology Institute, Vidisha (M.P.) in 1997 and awarded Doctor of Philosophy in Computer & Information Technology in 2012. He has published more than 50 publications in International Journals and Conferences. He has published two books named Data Structures and Advanced Database Management System. He is the recipient of the Best Professor in Information Technology award by the World Education Congress in 2013. He is a senior member of the IEEE (USA), Life member of Computer Society of India (CSI), Life member of Indian Society of Technical Education (ISTE).

Dr Shikha Agrawal is an Assistant Professor in Department of Computer Science & Engineering at University Institute of Technology, Rajiv Gandhi Proudyogiki Vishwavidyalaya, Bhopal (MP) India. She obtained B.E., M.Tech. and Ph.D in Computer Science & Engineering from Rajiv Gandhi Proudyogiki Vishwavidalaya Bhopal. She has more than twelve years of teaching experience. She has been awarded as "Young Sciencetist" by Madhya Pradesh Council of Science and Technology, Bhopal in 2012. Her other extraordinary achievements include "ICT Rising Star of the Year Award 2015" and "Young ICON Award 2015". Her area of interest is Artificial Intelligence, Soft Computing and Particle Swarm Optimization and Database. She has published more than 30 research papers in different reputed international journals and 9 chapters. She is a Madhya Pradesh State Students' Coordinator, Computer Society of India (CSI), 2016-2017.