Journal of Computer Science 10 (9): 1881-1889, 2014
ISSN: 1549-3636
© 2014 Science Publications
doi:10.3844/jcssp.2014.1881.1889 Published Online 10 (9) 2014 (http://www.thescipub.com/jcs.toc)

A COMBINTORIAL TREE BASED FREQUENT PATTERN MINING

[1]Yamuna Devi, N. and [2]J. Devi Shree

[1]Department of MCA, Coimbatore Institute of Technology, Coimbatore, India
[2]Department of EEE, Coimbatore Institute of Technology, Coimbatore, India

Received 2014-02-06; Revised 2014-02-07; Accepted 2014-05-06

ABSTRACT

Frequent pattern mining is a process of extracting frequently occurring itemset patterns from very large data storages. These frequent patterns are used to generate association rules which define the relationship among items. The strength of the relationship can be measured using two different units namely support value and confidence level. Any relationship that satisfies minimum threshold of support value is known as frequent pattern. There are several methods and algorithms suggested to mine frequent patterns from large databases. Most of the methods can be assessed for its complexity based on the number of processing levels and number of candidate sets with subsets that are generated in each level. In this study, the combinatorial approach which generates minimal number of combinations using a tree structure and automatically filters infrequent itemsets and mine frequent patterns is suggested. It scans input database once and carries out minimized intersections to count the support value. The complexity is based on the number of transactions and the maximum length of transactions. The new approach purely depends on the size of input transaction database. The combinatorial approach does not depend on the unknown number of processing levels and there is nocandidate sets and subsets generation. The proposed method makes minimal number of combinations when compared to number of candidate sets and subsets in other methods. The method is compared with number of existing legendary methods for its performance.

Keywords: Association Rule Mining, Frequent Item Set Mining, Combinatorial Approach, Tree Structure Based Combinations

1. INTRODUCTION

In recent years, extracting interesting patterns from a huge volume of data is necessary since the new technologies such as cloud computing, mobile applications, social networks cause a huge amount of data generation in many ways. These data are to be stored, maintained and integrated to get useful information from them by analyzing in various ways. This extraction process is an essential part of knowledge discovery which is also known as data mining. Among many techniques in data mining, Association rule mining is a key technique which defines the dependency between any two itemsets. Association rules are generated using algorithms by finding frequent patterns as an initial step. The frequent patterns are mined using minimum support threshold and further minimum confidence threshold is used to generate association rules. Mining frequent patterns from large scale databases is a hot research area in which many techniques have been implemented. Apriori algorithm is the most widely used oldest algorithm to find frequent patterns and association rules. Many researchers improved the efficiency of Apriori algorithm using various techniques and implementations were done. After careful analysis, it is found that the main deficiencies in almost all Apriori-based algorithms suffered are, too many scans of the transaction database, large amount of unnecessary candidate itemsets and subsets generation and pruning process.

Corresponding Author: Yamuna Devi, N., Department of MCA, Coimbatore Institute of Technology, Coimbatore, India

Many methods have been suggested which scan the database only once, still they generates more number of candidate sets and subsets in pruning process. It is absolute necessary for new ideas that can reduce the number of scans, number of candidate sets and subsets generation in pruning process. In this study, a new method is suggested that scans the database only once. It also avoids pruning process and hence candidate sets and subsets generation. Instead, it uses combinatorial method to generate combinations of itemsets in each transaction which is less in number when comparing to pruning process.

2. RELATED WORKS

The Apriori algorithm is the first and foremost method to mine frequent patterns. The limitations of Apriori algorithm are suggested as the number of scans and generation of huge quantity of candidate sets. The algorithm takes a stretched duration to generate candidate sets and pruning process. The pruning process generates in turn a large quantity of subsets for each candidate set in every k^{th} level and compared with the candidate sets in k-1^{th} level. This also extends the execution time of the algorithm. As an improvement, the Vertical data format method is suggested in ECLAT algorithm with only one database scan and transforms the input database into {Itemset, Tid} form. This method is profitable than Apriori because it does not scan the database more than once. Further instead, it does intersection with the {Itemset, Tid} sets. It is a monotonous task in vast databases to prune the candidate sets using apriori property at each level. Zhang (2012) proposed a method that reduces the number of scans and hence the candidate set generation.

Following that many methods and techniques have been suggested with improvements. Among them, frequent pattern tree growth algorithm eliminates candidate large itemset generation. But the process of generating tree data structure and the pruning process using the tree structure is considered as lacking part. Agrawal and Ramakrishnan (1994) developed AprioriTid algorithm which uses the set $\bar{C}_{k-1}$ (frequent sets in k-1^{th} level) to prune candidate sets in C_k (candidate sets in k^{th} level) and produces $\bar{C}_k$ (frequent sets in k^{th} level). In this algorithm scans the database once but the huge candidate sets are generated as in Apriori method. The AprioriHybrid method is suggested by same authors that combines the Apriori algorithm and AprioriTid algorithm. In AprioriHybrid, during the initial passes, the method follows Apriori algorithm and AprioriTid method is followed in latter passes. This further reduces execution time since Apriori takes more time in latter passes and AprioriTid takes the same through initial passes.

As further improvements, Goswami *et al*. (2010) proposed a new algorithm using record filter approach. In this approach the transactions that are not having number of items that is equal to or greater than k (k-itemset) are rejected for scanning. The probability concept is used in Apriori algorithm by Sunil *et al*. (2012). Jaishree *et al*. (2013) explained transaction reduction method to improve efficiency. Jnanamurthy *et al*. (2013) discussed mining maximal frequent item sets using subset creation. In all the above said and new improvements, the thing which cannot be avoided is the generation of huge volume of candidate sets. Any algorithm that avoids or reduces the generation of candidate sets will further improves the performance of frequent pattern mining. Vijayarani and Sathya (2013) proposed the implementation of ECLAT algorithm over data streams. The implementation of prefix tree to mine frequent sets was given by Grahne and Zhu (2003). Tohidi and Ibrahim (2011) introduced an algorithm to generate frequent patterns without generating a tree based on Prime Factor Miner (PFM). Venkatesan and Ramraj (2011) proposed a Bit search method instead of depth first and breadth first search techniques (Venkatesan and Ramraj, 2011). To improve the performance of Apriori algorithm, sorting and clustering technique was used by Jha and Borah (2012). Another improvement was done by Nagesh *et al*. (2013) using fully organized candidate generation and viper algorithm. The candidate set size is considered for improvement in the work proposed by Sheila (2012). The probability theory is used by Smythe and Goodman (1992). Sunil *et al*. (2010) suggested a method with dynamic function applied on transposition of the database. This study suggests a new way of using combinatorial method to mine frequent patterns which avoids generation of candidate sets and pruning process.

3. FREQUENT PATTERNS

Association Rules are generated in two steps. As first step, generate frequent patterns. The frequent patterns are those itemsets whose occurrences exceed a predefined threshold support value in the database. The second step is to generate association rules from those large frequent itemsets with the constraints of minimal confidence. The first step can be done in turn two sub-steps. They are, candidate itemsets generation and frequent itemsets generation by pruning process. Here,

the generation of candidate large itemsets and pruning process are focused for improvement. Formally, Jiawei *et al*. (2012) defined Association rule mining problem as follows. D = $\{T_1, T_2, \ldots, T_N\}$ is a database of N transactions. Each transaction consists of subset of I, where I = $\{i_1, i_2, \ldots, i_m\}$ is a set of all items. An association rule is an implication of the form $A \Rightarrow B$, where A and B are itemsets, $A \subseteq I$, $B \subseteq I$, $A \cap B = \phi$. In support-confidence framework, each association rule has support and confidence to confirm the validity of the rule. The support denotes the occurrence rate of an itemset in D and the confidence denotes proportion of data items containing B in all items containing A in D. Defined in terms of equations:

$$\text{Sup}(I) = \text{Count}(I)/\text{Count}(D)$$
$$\text{Sup}(A \Rightarrow B) = \text{Sup}(A \cup B)$$
$$\text{Conf}(A \Rightarrow B) = \text{Sup}(A \cup B)/\text{Sup}(A)$$

An itemset with k elements is called a k-itemset. An itemset is frequent if its support is greater than a support threshold, originally denoted by min_support. The frequent itemset mining problem is to find all frequent k-itemset, 1< = k< = m, in a given transaction database D. Assume that the items are from an ordered set and the transactions in D contain sorted itemsets.

4. PROPOSED ALGORITHM

The proposed Direct-vertical algorithm mines the frequent patterns in a different way using combinatorial method. It generates all possible k-itemset frequent patterns corresponding to each transaction on the fly while the transaction is read from the input database. The algorithm works in stages as, it reads the current transaction and generates all possible ordered combinations of items in that transaction. Then these combinations are verified for minimum support using intersection method. All combinations that satisfy minimum support count are considered as frequent itemsets and are stored in *frequent itemset table*. This process is repeated for each transaction. Finally, the algorithm constructs a *1-itemset table* for 1-itemset frequent sets and *frequent itemset table* for k-itemsets where k> = 2 in vertical form as {itemset, Tid}.

The proposed algorithm reads one transaction at a time. While reading a transaction, based on the minimum support, the frequent 1-itemsets alone are considered from the current transaction to fabricate ordered combinations of k-itemsets, where k> = 2. The support value for each combination itemset is calculated using intersection method. The intersection is performed using *1-itemset table*. The intersection process results TID-set for each combination. The absolute support count for each itemset is the length of the TID-set of the corresponding combination. The combination which satisfies minimum support threshold is considered as frequent set.

This algorithm requires only one scan of the transaction database to generate the set of all frequent itemsets without generating any candidate sets and subsets and hence there is no pruning process. All infrequent itemsets will be filtered on the fly. This qualifies the efficiency of the proposed algorithm. The algorithm works by calculating ordered combination of items in each transaction Ti. The proposed algorithm is given in **Figure 1**.

Example

The Direct-vertical algorithm generates all k-itemset frequent sets on the fly while reading the transaction database. **Figure 2** depicts some steps in execution of the proposed algorithm by considering the **Fig. 2a** as simple transaction database with minimum support 43%. Read the first three transactions T100, T200 and T300, enter into the 1-itemset table as given in **Fig. 2b** which is *1-itemset table*. Here, there are two items B, E that satisfy minimum support. So, the combination BE goes to 2-itemset frequent set with the transactions T100, T200 and T300. While reading T400, there are four combinations for frequent items which include three 2-itemsets and one 3-itemset. After performing intersection for each combination, include combinations that satisfy the minimum support as given in **Fig. 2c**. If any k-item*set al*ready exists in the table, then its support count alone is increased. When all transactions are read, **Fig. 2d**, *frequent itemset table*, is generated which shows all frequent k-itemsets.

4.1. Data Structures

The algorithm reads one transaction at time and generates all frequent itemsets from that transaction. An extra field is attached with each item in both 1-itemset table and frequent itemset table to maintain and update the support count. While reading each transaction, all combinations are generated using frequent 1-itemsets alone in that specific transaction. The algorithm generates combinations using tree data structure which is advantageous when compared to other ways. The approach given by Shant and Choueiry (2010) is implemented in this proposed Direct-Vertical algorithm to improve the efficiency. It uses the divide-and-conquer technique to further reduce the complexity.

```
Algorithm Direct_Vertical (D, min_Support)
Initialize i=1, support=0;
while (i<= n) do       //for each transaction Ti in D & n is
                       //the total number of transactions in D
{
  read transaction Ti;
  for each item Ij ∈ Ti,                    //1<=j<=m, m is the number of different items
    append Ti in 1-itemset table against Ij and increment
    support of Ij;
    count all Ij in Ti that have support >= minSupport and
    move all Ij to frequent list S;
    if count<2 then  {i=i+1; Break;}        //eligibility to make combinations
    else {
        Ck = Produce_Combinations (S) // Refer Section 4.1
//Ck is the set of all possible ordered combinations of //frequent 1-itemsets from S, where 2<=k<=count
          for each C ∈Ck  {
            if C already exist in frequent itemset table, then
            append Ti against C;
                        //To increment the support value
            else {
                    S1 = result of intersection of
              corresponding TID-set of individual item in C;
                  if count(S1) >= minSupport then
                include C in frequent itemset table
                and append Ti against C;
           //inclusion of new frequent itemset
                }
        }
    }
 i=i+1;
}
```

Fig. 1. Direct vertical algorithm

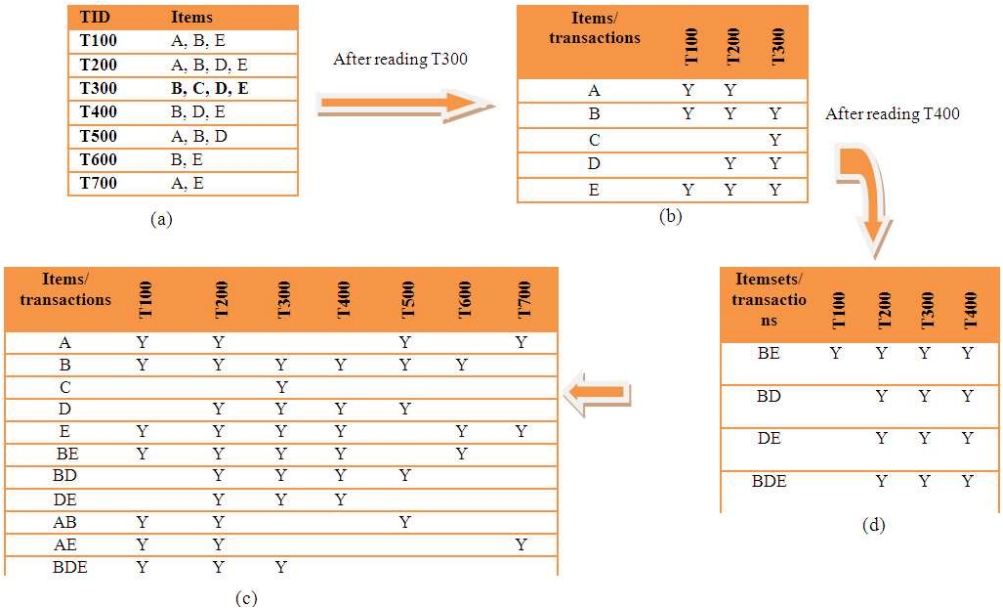

Fig. 2. Example for execution of proposed algorithm

4.1.1. Generating k-Combinations

The algorithm Produce_Combinations generates all possible combinations for the elements of given non-negative set S. This algorithm in turn calls Produce_Combi_Tree on c and s = |S| to generate the combination tree where 2 < = c < = s. The algorithms are given in **Fig. 3**. The elements of S are stored in an array and the index values of the array are passed to generate tree. The algorithm Poduce_Combi_Tree is a divide-and-conquer algorithm that solves the problem by generating a tree. Given a root node and a non-negative integer s, it divides the problem into (s-c-bal + 1) sub-problems. The sub-problems are solved by making a recursive call. The recursion ends when c = 0. The final solution is constructed by traversing the tree in depth wise manner start from root through each and every path. This solution set gives the combinations of positions of elements. These positions are mapped to the corresponding element in the set S and the combinations of all the elements are generated.

As an example, the execution of the above said algorithms is explained using **Fig. 4** with the initial call of Produce_Combinations(2, S) where S = {I1, I2, I3, I4}. The tree is generated by lexicographical order of the labels specified for nodes in the **Fig. 4**. The tree is traversed in depth wise and set of combinations generated are {I1, I2}, {I1, I3}, {I1, I4}, {I2, I3}, {I2, I4}, {I3, I4}. The time and space complexity of the algorithm Produce_Combinations are $\binom{s}{c}$, where s = |S|, 2 < = c < = s.

4.2. Evaluation of Proposed Algorithm

For most of the existing algorithms, the complexity can be defined based on the number of levels (l), number of candidate sets generated in each level (m) and the number of subsets of each candidate set in all k-1 levels. The total number of candidate sets and subsets generated can be calculated as:

$$\sum_{k=2}^{l}\sum_{j=1}^{m}\binom{p}{k-1}+lm$$

where, p = C_{kj} (Each candidate itemset).

So, the complexity is about defined O(lm). The AprioriHybrid in addition involves the cost of intermediate method switching. But in the proposed Direct-vertical algorithm, the complexity can be defined based on number of transaction and number of ordered combinations generated. The total number of combinations calculated as:

```
Algorithm Produce_Combinations(S)
root←ϕ ;
s← |S|
for c ← 2 to s
    Produce_Combi_Tree(c, 0, root, s)
end
Combi_set ← Read_Tree_Depthwise(root)
Element_Combi_Mapping ← Read_Element_Mapping(Combi_set)
return Element_Combi_Mapping
Algorithm Produce_Combi_Tree(c, bal, root, s)
if c = 0 then return
for  x← bal to (s − c) do
        new ←  x
        add new as a child to root
Produce_Combi_Tree((c − 1), (x + 1), new, s)
end
```

Fig. 3. Algorithms to generate ordered combinations using tree structure

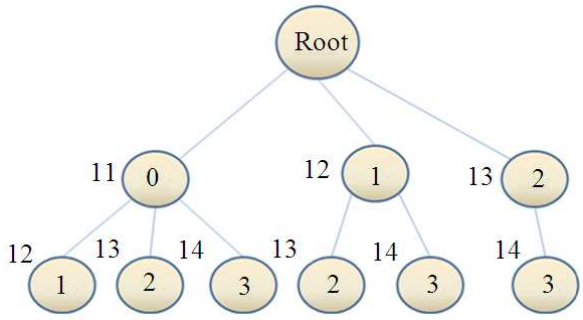

Fig. 4. Tree structure for 2-itemset combinations

$$\sum_{i=1}^{T_n}\left[\frac{n_i!}{(n_i-c_j)!c_j}\right]$$

where,
T_n = Number of transactions
n_i = Number of frequent items in transaction T_i
C_j = The order of combinations and j = 2,3,…$|n_i|$

The above formula produces totally 2^{ni}-n_i-1 * Tn. combinations. So, the complexity of the proposed algorithm can be defined as O(2^{ni} * Tn). This is less in count when compared to other methods and hence the complexity is reduced. These combinations are calculated using a tree data structure.

5. PERFORMANCE

5.1. Proposed Algorithm Vs Algorithm Apriori

The Apriori algorithm is the first and the foremost association rule mining algorithm which generates all

frequent itemsets in first phase. There are n levels and in each k^{th} level, k-itemsets frequent sets (L_k) are generated. Each L_k is used to generate candidate itemsets C_{k+1} in next level which is formed as $L_k \bowtie L_k$. A huge set of candidate sets are generated at each level. For each candidate set, a number of subsets are generated for pruning process. Each level requires one database scan. The Apriori property is used to reduce the search space which eliminates some of the candidate itemsets by pruning technique. The complexity of the algorithm depends on the number of levels(n), number of candidate sets generated in each level (C_k) and the number of subsets generated in each level to check Apriori property (S_k*C_k). So, variably the complexity can be defined $O(S_k*C_k)$, 1< = k< = n.

The proposed Direct-Vertical algorithm does not generate candidate sets and in turn subsets. Instead, in generates combinations which is less in count than set of candidate sets and subsets. The complexity of the proposed algorithm is $\binom{s}{c}$, where s is number of frequent items in each transaction and 2 < = c < = s. The proposed algorithm depends on the number of transactions in the database (one time scan) and the maximum length of the transactions.

5.2. Proposed Algorithm Vs ECLAT Algorithm

In ECLAT algorithm, transaction database is transformed to vertical data format as <item, {TID}> where item is the name/id of the item and {TID} is the set of transaction identifiers containing the item. After one scan of transaction database for transformation, it follows the procedure of Apriori algorithm by generating candidate sets and subsets. The support value of each candidate itemset is counted by intersecting the sets of {TID} of every pair of frequent single items instead of database scan. This algorithm produces a huge number of candidate sets and subsets. So, the space complexity remains equal to Apriori algorithm as $O(S_k*C_k)$, 1< = k< = n.

The proposed algorithm follows the vertical data format representation and intersection process as in ECLAT. But, it is totally different in reading the input transaction database and generation of ordered combinations instead of candidate sets. ECLAT takes one scan of transaction database initially for complete transformation. The proposed algorithm reads one transaction at a time for whole process. An itemset combination is verified for support count using intersection method at first occurrence. The second occurrence of the combination is considered as 'exist' category combination which is not required intersection process. In this case, the current transaction id is appended to that existing combination. This proves the reduced number of intersections in proposed algorithm when compared to ECLAT algorithm.

5.3. Proposed Algorithm Vs AprioriTid

The AprioriTid algorithm also generates candidate itemsets in each level like Apriori algorithm and ECLAT. The appreciated thing in AprioriTid algorithm is it does not scan the database after the first level. During first level, it reads the transactions and transforms the individual items as separate set in the same transaction. This form is known as $\bar{C}_k$. This $\bar{C}_k$ is used for counting support value of each candidate itemsets in C_{k+1}. Each member of the set C_k is of the form < TID; $\{X_k\}$ >, where each X_k is a potentially large k-itemset present in the transaction with identifier TID. It also checks whether the candidate itemsets in C_{k+1} are contained in the transaction with identifier TID by taking subsets.

While comparing this algorithm, the proposed algorithm does not generate any candidate sets and subsets and produces ordered combinations which are less in count. There is no dependency of previous level results in proposed algorithm. For each transaction, it finishes generation of all possible frequent itemsets. It proves the better performance over AprioriTid algorithm.

5.4. Proposed Algorithm Vs AprioriHybrid

AprioriHybrid is a good algorithm which mines the frequent itemsets. It is a combination of Apriori algorithm and AprioriTid algorithm. AprioriHybrid follows exactly Apriori algorithm for certain passes after which it follows AprioriTid algorithm. This is because during initial passes Apriori algorithm takes much less time than AprioriTid algorithm. In later passes, AprioriTid beats Apriori algorithm. The reason for this is Apriori and AprioriTid use the same candidate generation procedure. In the later passes, the number of candidate itemsets reduces. On the other hand, rather than scanning the database, AprioriTid scans $\bar{C}_k$ for obtaining support counts and the size of $\bar{C}_k$ has become smaller than the size of the database. So, it is a good idea to use Apriori in initial passes and AprioriTid in later passes. When the size of $\bar{C}_k$ is enough to fit in memory, there the switching takes place. There is a cost involved for this switching.

In general, AprioriHybrid is advantageous over Apriori based on the decrease in the size of the $\overline{C}_k$ set in the later passes. On the other hand, if there is a gradual decline in the size of C_k, a significant improvement can be obtained in the execution time. The cost of switching must also be considered. While considering these constraints, the proposed algorithm does not have any uncertain situations and there is no extra cost involved for any process.

6. RESULTS

To make the comparison between the algorithms based on the number of subsets, number of candidate sets and number of intersections, a real time surveyed numeric database is used. The database consists of 5000 transactions includes 30 different items. The implementations were modified to specify the count of number of subsets, candidates and interactions. The execution was done with various support counts. **Figure 5** shows comparison between number of ordered combinations and number of subsets generated in the proposed algorithm and others respectively. The comparison between number of ordered combinations generated in proposed method with total number of subsets and candidate sets generated in other methods is shown in **Fig. 6**.

The intersection method is followed in ECLAT and AprioriTid of above discussed algorithms. The proposed algorithm is completely different in intersection process in terms of the itemsets chosen for intersection in which the number of intersections is considerably reduced. It is shown in **Fig. 7** that direct vertical algorithm performs less number of intersections compared to ECLAT. The same is compared with subset verification in AprioriTid technique.

The proposed method consumes very less execution time when compared to Apriori, ECLAT, AprioriTid and AprioriHybrid methods. To compare the relative performance of the algorithms, the experiments were performed on the Adult dataset from UCI machine learning database repository (Blake *et al*., 1998). The Adult dataset contains 48842 records and 14 columns. The relative performance is analyzed for complexity based on number of combinations generated and subsets generated. The comparison of execution time between all these methods is shown in **Fig. 8**.

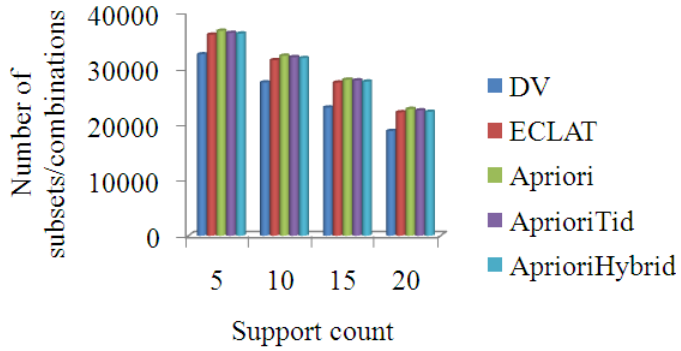

Fig. 5. Count on subsets Vs combinations

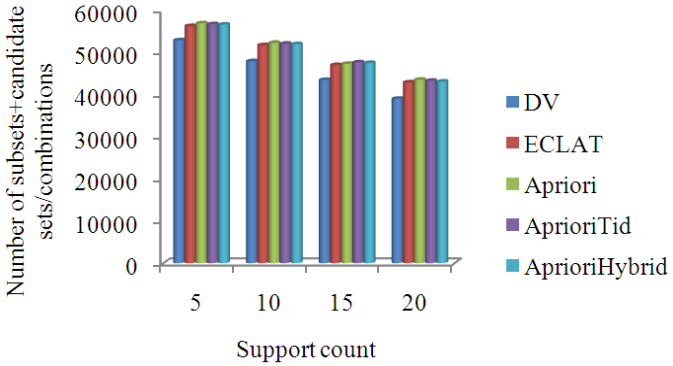

Fig. 6. Count on subsets + candidate sets Vs combinations

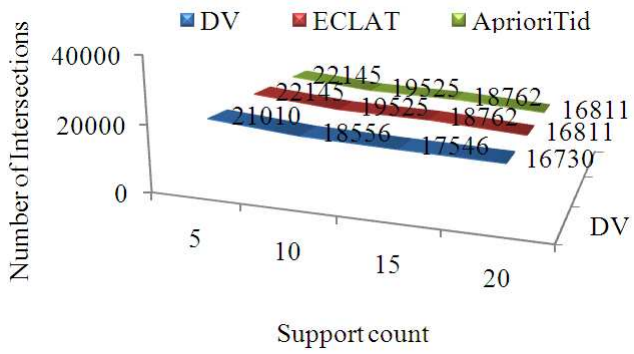

Fig. 7. Count on intersections Vs subset comparison

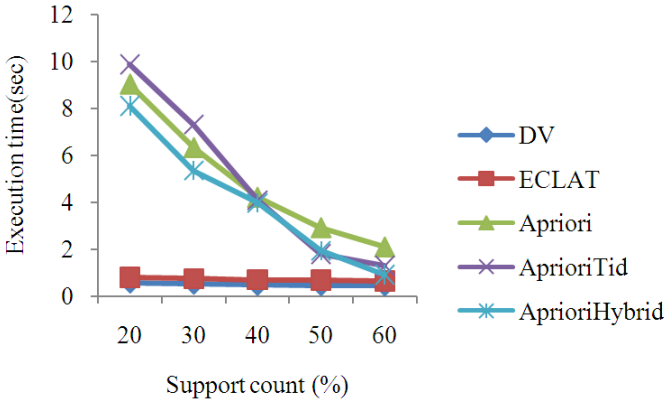

Fig. 8. Execution time for various support count

7. CONCLUSION

A new direct-vertical algorithm using combinatorial approach is proposed to mine frequent patterns in a large scale databases. The proposed algorithm differs from other methods in the way of reading the transaction database and generating combinations and filtering the infrequent combinations. After compared with some existing legendary algorithms, it is proved that the proposed algorithm outperforms others in terms of execution time and memory usage. The experiments were conducted with many synthetic datasets while only one dataset is used to compare the performance in this study. It is observed that the increase in execution time with the size of transaction database is linear and gradual. The experiments help to decide the feasibility of the proposed algorithm to mine frequent patterns in efficient manner by overcoming the bottlenecks in existing algorithms. This algorithm can be further improved by including the probability to find maximum possible number of combinations.

8. REFERENCES

Agrawal, R. and S. Ramakrishnan, 1994. Fast algorithms for mining association rules in large databases. Proceedings of the 20th International Conference on Very Large Data Bases, (LDB' 94), San Francisco, CA, USA pp: 487-499.

Blake, C.L., D.J. Newman and C.J. Merz, 1998. UCI repository of machine learning databases. Department of Information and Computer Science. University of California. Irvine. CA. USA.

Goswami, D.N., C. Anshu and C.S. Raghuvanshi, 2010. An algorithm for frequent pattern mining based on apriori. Int. J. Comput. Sci. Eng., 2: 942-947.

Grahne, G. and J. Zhu, 2003. Efficiently using prefix-trees in mining frequent itemsets. Proceedings of the Frequent Itemset Mining Implementations, (FIMI' 03), Melbourne. Florida, pp: 1-12.

Jaishree, S., H. Ram and J.S. Sodhi, 2013. Improving efficiency of apriori algorithm using transaction reduction. Int. J. Sci. Res. Public., 3: 1-4.

Jha, I.N. and S. Borah, 2012. Efficient association rule mining using improved apriori algorithm. Int. J. Scientific Eng. Res., 3: 1-4.

Jiawei, H., M. Kamber and J. Pei, 2012. Data Mining: Concepts and Techniques. Morgan Kaufmann, USA.

Jnanamurthy, H.K., H.V. Vishesh, V. Jain, P. Kumar and R.M. Pai, 2013. Discovery of maximal frequent item sets using subset creation. Int. J. Data Min. Know. Manag. Process, 3: 27-38.

Nagesh, H.R., M.B. Kumar and B. Ravinarayana, 2013. Improved implementation and performance analysis of Association rule mining in large databases. Proceedings of the 3rd International Conference, Jan. 18-19, Mumbai, India, pp: 94-104. DOI: 10.1007/978-3-642-36321-4_9

Shant, K. and B.Y. Choueiry, 2010. Tree-based algorithms for computing k-combinations and k-compositions. Department of Computer and Engineering, University of Nebraska-Lincoln.

Sheila, A.A., 2012. Association rule mining based on Apriori algorithm in minimizing candidate generation. Int. J. Sci. Eng. Res., 3: 1-4.

Smythe and Goodman. 1992. An information theoretic approach to rule induction from databases. IEEE Trans. Know. Data Eng., 4: 301-316. DOI: 10.1109/69.149926

Sunil, J., R.S. Jadon and R.C. Jain, 2010. An implementation of frequent pattern mining algorithm using dynamic function. Int. J. Comput. Appli., 9: 37-41. DOI: 10.5120/1410-1904

Sunil, K.S., K.S. Shyam, K.C. Akshay, A. Prabhu and K.M. Bharathraj, 2012. Improved aprori algorithm based on bottom up approach using probability and matrix. Int. J. Comput. Sci., 9: 242-246.

Tohidi, H. and H. Ibrahim, 2011. Using unique-prime-factorization theorem to mine frequent patterns without generating tree. Am. J. Econ. Bus. Admin., 3:58-65. DOI: 10.3844/ajebasp.2011.58.65

Venkatesan, N. and E. Ramraj, 2011. High performance bit search mining technique. Int. J. Comput. Applic., 14: 15-21. DOI: 10.5120/1817-2371

Vijayarani, S. and P. Sathya, 2013. Mining frequent item sets over data streams using eclat algorithm. Proceedings of the International Conference on Research Trends in Computer Technologie, (TCT' 13), New York, USA, pp: 27-31.

Zhang, S.L., 2012. A new mining algorithm of association rules and applications. Proceedings of the 7th International Conference on Intelligent Computing, Aug. 11-21, Zhengzhou,China, pp: 123-128. DOI: 10.1007/978-3-642-24553-4_18