IJCSE International Journal of Computer Sciences and Engineering Open Access
Research Paper Volume-5, Issue-8 E-ISSN: 2347-2693

Comparative Study of Top 10 Algorithms for Association Rule Mining

B. Nigam[1*], A. Nigam[2], P. Dalal[3]

[1*]Dept. of Information Technology, IET DAVV, Indore, India
[2]Ideavate Pvt. Ltd., Indore, India
[3]Dept of Computer Engineering, DAVV, Indore, India

**Corresponding Author: bhawnanigam@gmail.com*

Available online at: www.ijcseonline.org

Received: 06/Jul/2017, Revised: 19/Jul/2017, Accepted: 15/Aug/2017, Published: 31/Aug/2017

Abstract: We live in a world where each day tons and tons of data is generated from millions of sources. Companies and organizations thrive for this data in order to acquire valuable information that helps them understanding their customer needs and demands. This valuable insight collaborates in improving the services and products – thus enhancing the overall business and profits. Filtering out such significant information thus requires employing some data mining algorithms. Data mining is a wide area of study that is further developing day by day and is very useful in deriving important information and coherence from large and raw datasets. When we talk about one very well-known field of business, known as Market Basket Analysis – data mining has significantly affected this sector. As the name suggests, it is an analysis of shopper's basket at a mart. We generally see various items arranged on the shelves in the malls and supermarkets; we also observe certain products recommended to us when we shop online. All of this is worked in the back-end with the help of data-mining algorithms that provide a proper analysis of customer buying patterns and hence it makes the relations and suggests them to the customers, which in turn results in an enhanced sales. Technically, association rule mining and frequent itemset mining is done for such analysis. These algorithms are also used in designing various games and in recommendation systems. In this paper we are thus understanding these algorithms and compare the efficiency of the most common ones on the basis of factors such as time, support and memory consumed.

Keywords- Data mining, Association rules, Apriori, FP-Growth, Eclat, dEclat

I. INTRODUCTION

As discussed market-basket analysis is a very significant stream that makes use of data mining techniques like association rule mining and frequent itemset mining in order to understand customer buying patterns. So when we talk about association rules, as the name says, these are if and then statements that help in discovering the relationships between the objects that are used together, quite frequently. These unfold the relationships between unrelated data that is stored either in a non-relational or relational database or may belong to some other information repository. And the task of association rule mining can be broken into two steps: first Frequent itemset generation and secondly Confidence Rules Generation. In this paper we are majorly focusing upon the techniques used in frequent itemset mining. Basically, frequent itemset mining focuses at sequences of actions or events. Here, the database takes the form of sets of transactions where each transaction has a number of items. And then there is a minimum threshold support value and user-specified minimum confidence, define as:

Support: Support(S) of an association rule is defined as the percentage or the fraction of records that contain X∪Y to the total number of records in the database. Suppose the support of an item is 0.2%, it means only 0.2 percent of the transaction contains purchase of this item.

Confidence(C):- Confidence(C) of an association rule can be defined as the percentage or the fraction of the number of transactions that contain X∪Y to the total number of records that contain X. Confidence [3] basically measures the strength of the association rules, for an instance, if the confidence of the association rule X⇒Y is 80%, then it means that 80% of the transactions containing X also contain Y together.

$$Rule: X \Rightarrow Y \quad \begin{cases} Support = \dfrac{frq(X,Y)}{N} \\ Confidence = \dfrac{frq(X,Y)}{frq(X)} \end{cases}$$

II. RELATED WORK

There are many algorithms for mining frequent itemsets. Each uses a different technique, has different intermediate steps and structures – that defines the efficiency of these algorithms. The intermediate steps and structures may include candidate itemset generation, creation of tidsets and diffsets, hash-tree formations, etc. So let us focus on the 10

mostly used algorithms for frequent itemsets mining [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21]:

[A]. Apriori Algorithm
[B]. AprioriTid Algorithm
[C]. FP Growth Algorithm
[D]. FP Max Algorithm
[E]. Charm Bitset Algorithm
[F]. HMine Algorithm
[G]. LCM Algorithm
[H]. Pascal Algorithm
[I]. Eclat Algorithm
[J]. dEclat Algorithm

A. Apriori Algorithm: Apriori Algorithm proceeds by identifying the frequent individual items in the database and then extends them to larger and larger item sets as long as those item sets appear sufficiently often in the database. It uses a "bottom up" approach [1], where frequent subsets are extended one item at a time (a step known as candidate generation) [11], and groups of candidates are tested against the data. Apriori uses breadth-first search and a Hash tree structure to count these candidate item sets efficiently. The algorithm terminates when no further successful extensions [15] are found. This level-wise approach helps to reduce the number of itemsets that are required to count the support. The general idea is to use k-itemsets to explore (k+1)-itemsets.

1. Firstly find the set of frequent 1-itemsets, Lk: this is done by scanning the database and accumulating the count for each item and then keeping those that meet the minimum support in a new set called Lk.
2. Next, Lk is used to find Ck+1 (the set of candidate 2-itemsets):- This is a two-step process that first generates Ck+1 based on Lk and then prunes Ck+1 by getting rid of those Ck+1 itemsets.
3. To find Lk+1, support count for all the itemsets in Ck+1 is found and those below the minimum support are rejected.
4. The steps 2&3 are continued until no new frequent (k+1)-itemsets are found.

Apriori Algorithm is very easy to understand and implement. It can be used on large item sets, however, when it is required to find a large number of candidate rules, it can become computationally expensive. Moreover, multiple database scans is time and memory consuming as well. In order to improve this algorithm, AprioriTid was introduced.

B. AprioriTid Algorithm: Apriori TID has the same candidate generation function as Apriori, however, it does not use database for counting support after the first pass. Instead, an encoding of the candidate item sets which was used in the previous pass is used again here. Eventually, in the later passes the size of encoding [1,12] can become much smaller than the database – thus saving reading effort. Following is the example that sums up the working of AprioriTid Algorithm:

1. Firstly, one should keep in mind that database is not used at all for counting the support of candidate itemsets after the first pass.
2. The candidate itemsets are generated in similar fashion as Apriori algorithm.
3. Another set C' is generated and each member has the TID of each transaction along with the large itemsets present in this transaction. This set is actually used to count the support of each candidate itemset.

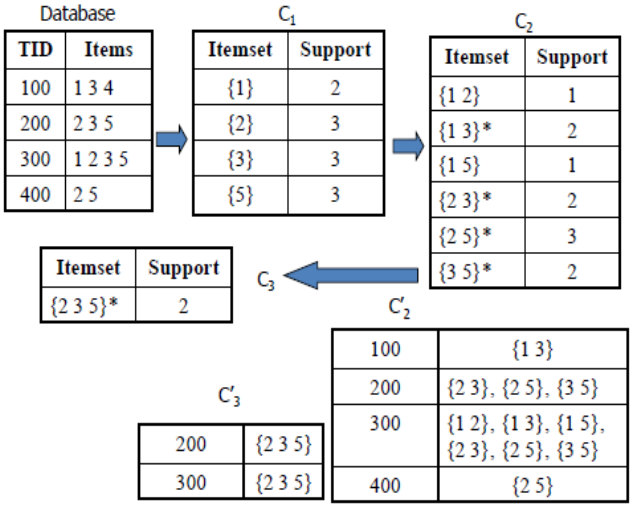

Figure 1. Example of AprioriTid Algorithm

So what we observe is that the number of entries in C' have decreased since than the number of transactions in the database, especially in the later passes. Thus, making this algorithm more efficient and fast.

C. FP-Growth Algorithm: This is another important frequent pattern mining method, which generates frequent item set without candidate generation. It uses a divide and conquer strategy while creating a tree based structure. It works by generating a prefix-tree data structure known as FP-tree from two scans of the database. This method involves two phases. First phase needs two database scans for generating the FPtree [2, 12] by compressing that input database and creating an FP-tree instance that represent frequent items. After this first step it divides the compressed database into a set of conditional databases, each one associated with one frequent pattern. And then finally, each such database is mined separately. However, this phase doesn't need any scan over database and it uses on FP-tree to generate frequent item set.

This way FP-Growth reduces the search costs looking for short patterns recursively and then concatenating them in the long frequent patterns, that offers good selectivity. However, for large databases, it is sometimes difficult to hold an FP-

Tree in the main memory and thus, this we move on to an enhanced version of this algorithm.

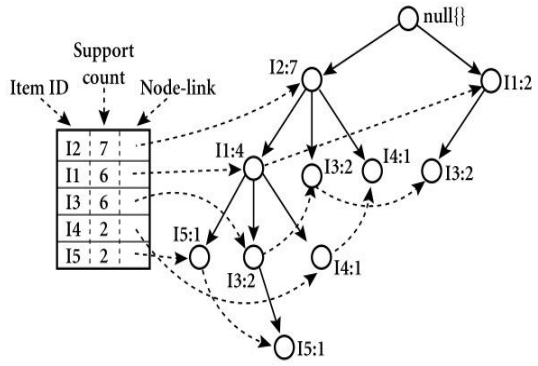

Figure 2. Example of an FP-Tree Structure

D. FPMax Algorithm: FPMax is based on the very famous FPGrowth algorithm and is used to discover frequent maximal item sets [2, 3] in a transaction database. It includes several strategies that mines maximal item sets efficiently while it prunes the search space. We already know that a frequent item set is an item set that appears in at least min support transactions from the transaction database. While a frequent closed item set is a frequent item set that is not included in a proper superset having the same support. Having known that, we define a frequent maximal item set [14, 18] as a frequent item set that is not included in a proper superset (i.e. already a frequent item set). The set of frequent maximal item sets is thus a subset of the set of frequent closed item sets, which is a subset of frequent item sets. The reason we use the frequent maximal item sets is that they are usually much smaller than the set of frequent item sets and also smaller than the set of frequent closed item sets. Thus this is a very efficient algorithm over FP-Growth.

E. Charm Algorithm: Charm is an efficient algorithm for enumerating the set of all frequent closed itemsets. It explores both the item set space and transaction space [4, 5], over a novel IT-tree (itemset - tidset tree) search space unlike the other algorithms that exploit only the itemset search space. It employs a highly efficient hybrid search method that skips many levels of the IT-tree and quickly identifies the frequent closed itemsets, instead of enumerating many possible subsets. Using a fast hash-based approach, it can eliminate non-closed itemsets and with a novel vertical data representations called as diffsets [19, 20, 21], it can track the differences in tids of candidate pattern from its prefix pattern. The introduction of diffsets, drastically cut downs the size of memory required to store the intermediate results. One of the most efficient and simplest algorithms, it works the best for large databases by saving time and memory.

F. HMine Algorithm: HMine algorithm takes advantage of a novel hyper-linked data structure known as H-Struct [5, 18] that dynamically adjusts links in the mining process. This algorithm features a limited and precisely predictable space overhead that can run really fast in memory-based setting. Moreover, it can be scaled up to very large databases by database partitioning. Also, for dense data sets HMine integrates with FP-tree structures – thus making it a more versatile algorithm.

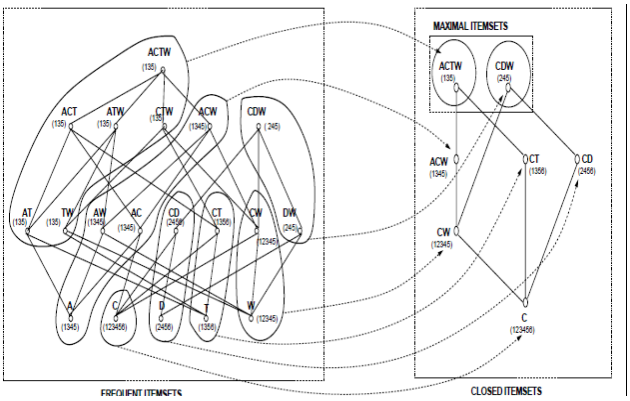

Figure 3. Frequent Closed and Maximal Item sets

G. LCM Algorithm: Precisely called as LCM, Linear Time Closed Item set Miner enumerates (output, or count) all frequent itemsets, all maximal frequent itemsets, or all frequent closed itemsets in a given transaction database for given a support. LCM features quite a stable performance for both computation time and memory usage [6, 11]. The basic idea of this algorithm is depth-first search. LCM in its first step computes the frequency of each item set composed of one item. If that item set is frequent, then enumerate frequent item sets are obtained by adding one item to it. This way, recursively, LCM enumerates all frequent itemsets.

H. Pascal Algorithm: Pascal is a two-way algorithm that is used to discover frequent itemsets and at the same time identify which ones are generators [8, 16] in a transaction database. Pascal is an Apriori-based algorithm. It uses a special pruning property that can avoid counting the support of some candidate itemsets. This property is based on the fact that if an itemset of size k is not a generator, then its support is the support of the minimum support of its subsets of size k-1. The Pascal algorithm is considered to be more or less as efficient as Apriori algorithm since it is an Apriori-based algorithm. Pascal utilizes pruning strategies [13] that are supposed to make it faster by avoiding counting the support of some candidates.

I. Eclat Algorithm: It is a depth first search based algorithm. Eclat algorithm uses a vertical database layout i.e. instead of explicitly listing all transactions [9, 17]; each item is stored together with its cover (also called tidlist) and uses the intersection based approach to compute the support of an item set. It requires less space than Apriori [10] if item sets are small in number .It is suitable for small datasets and

requires less time for frequent pattern generation than other Algorithms.

J. dEclat Algorithm: dEclat is a variation of the Eclat algorithm that is implemented using a structure called "diffsets" rather than "tidsets". *Diffset* [9, 10] only keeps track of differences in the tids of a candidate pattern from its generating frequent patterns. They drastically cut down (by orders of magnitude) the size of memory required to store intermediate results. The initial database stored in diffset format, instead of tidsets can also reduce the total database size. Since the diffsets are a small fraction of the size of tidsets, intersection operations are performed blazingly fast! We have also made use of these diffsets in Charm, which has proved to be the most efficient of all the algorithms.

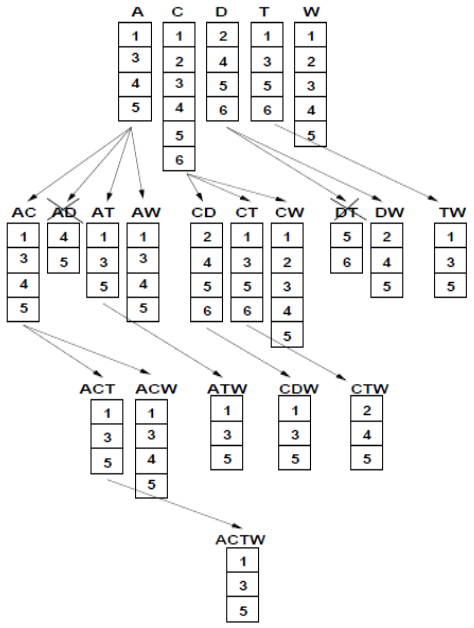

Figure 4. Tid Intersections for Pattern Counting

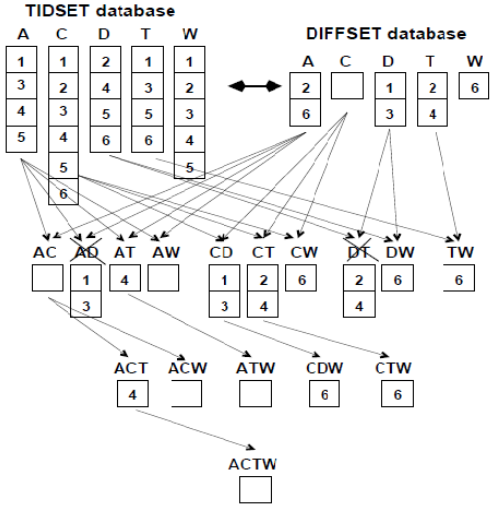

Figure 5. Diffsets for Pattern Counting

III. RESULTS AND DISCUSSION

As learnt and discussed, we can observe that each algorithm follows a different approach and has different intermediate structures [20] formed – which affect their efficiency. Also, a few algorithms follow a hybrid approach that leads to faster and better results. Thus, in order to understand, compare and evaluate the performance of these ten algorithms we have applied them on a large transactional database consisting of 75,000 transactions. These transactions belong to a bakery shop having customer transactions of each day, for which we are trying to perform the market-basket analysis by extracting frequent item sets and then later using them to generate the various association rules.

Following three are the graphs on different minimum support values that compare these ten algorithms on the basis of the time taken by them:

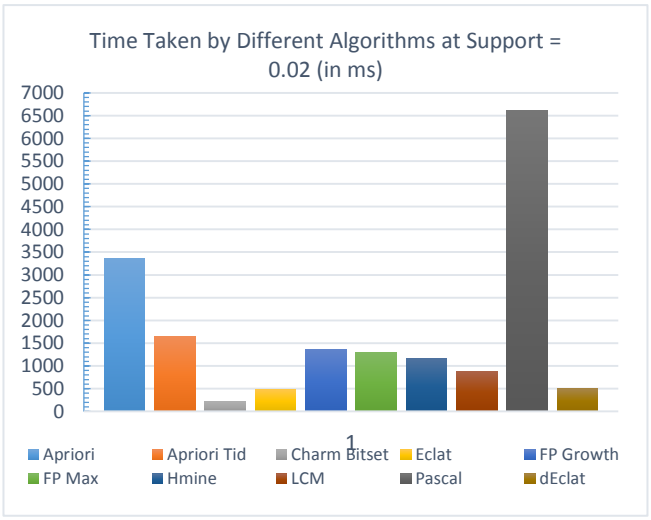

Figure 6. Time Taken at Support = 0.02

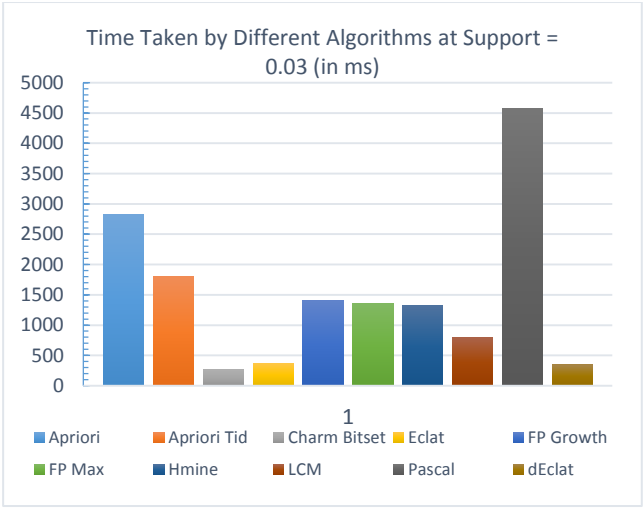

Figure 7. Time Taken at Support = 0.03

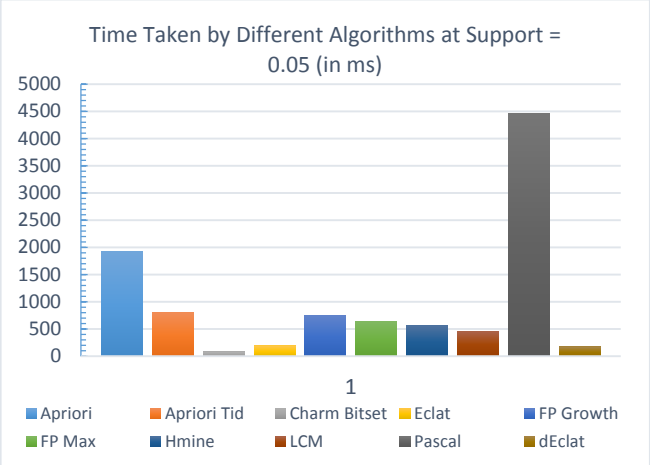

Figure 8. Time Taken at Support = 0.05

Following three are the graphs on different minimum support values that compare these ten algorithms on the basis of the memory consumed by them:

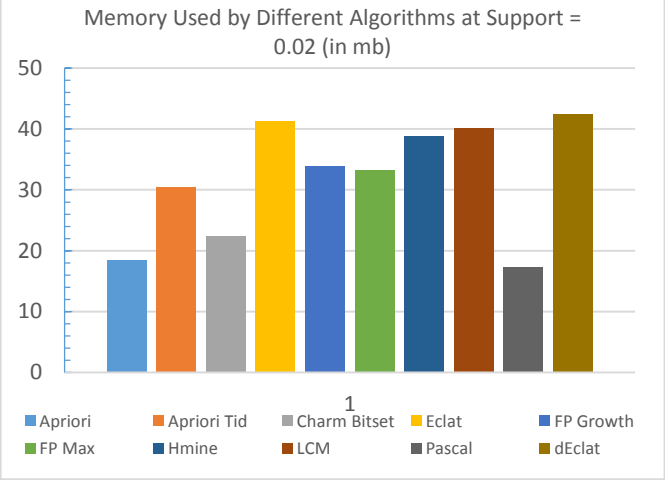

Figure 9. Memory Consumed at Support = 0.02

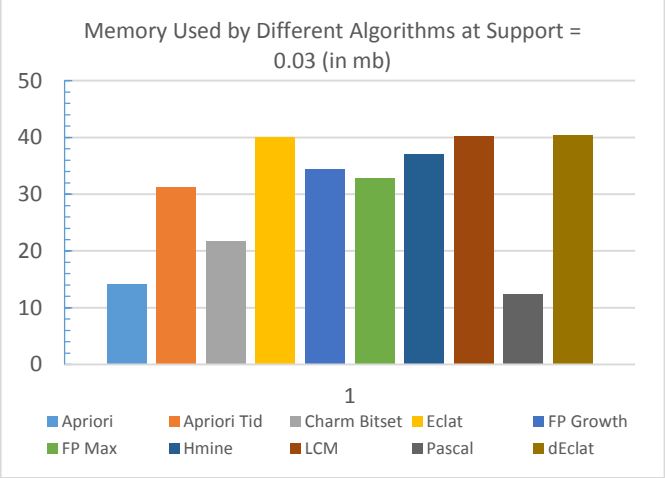

Figure 10.Memory Consumed at Support = 0.03

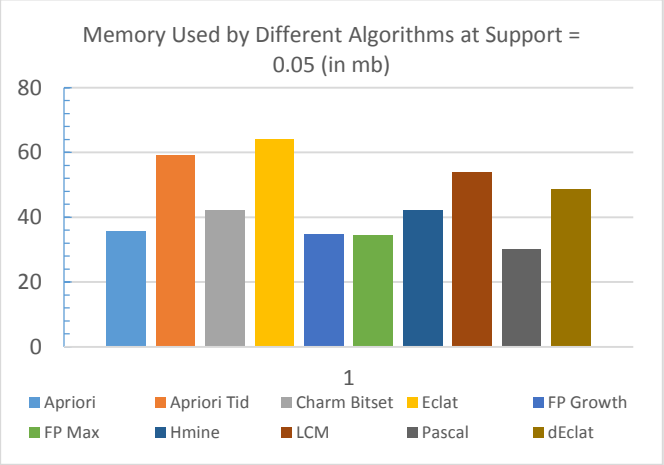

Figure 11.Memory Consumed at Support = 0.05

IV. CONCLUSION

Of all the ten algorithms where different approaches are employed, Charm, dEclat and Eclat have proven to be the best and suitable for large sized databases. These algorithms can be scaled up and extended easily without much efforts and will run fast, while utilizing minimum memory. However, when we talk about smaller datasets, we may find that primary algorithms like Apriori or FP-Growth are suitable as well, since they follow a simpler approach of candidate generation and tree creation. On the other hand when we need some more details such as identifying the generators, Pascal proves to be the best choice as it will take up almost the similar time as Apriori and provide us with two results. Furthermore, AprioriTid, FP Max and dEclat are the other algorithms which are basically the enhanced versions of the previously proposed algorithms namely, Apriori, FPGrowth and Eclat, respectively. Lastly, when we talk about the Charm, it certainly has its own charm in proving us tremendous results by employing a hybrid approach towards generating the frequent item sets.

REFERENCES

[1]. Agrawal R, Srikant R., "*Fast algorithms for mining association rules*", InProc. 20th int. conf. very large data bases, VLDB, Vol. 1215, pp. 487-499, 1994.

[2]. Han J, Pei J, Yin Y, Mao R., "Mining frequent patterns without candidate generation: A frequent-pattern tree approach", Data mining and knowledge discovery, Vol.8, Issue.1, pp.53-87, 2004.

[3]. Akilandeswari. S and A.V.Senthil Kumar, "*A Novel Low Utility Based Infrequent Weighted Itemset Mining Approach Using Frequent Pattern*", International Journal of Computer Sciences and Engineering, Vol.3, Issue.7, pp.181-185, 2015.

[4]. Lei Wen, "An efficient algorithm for mining frequent closed itemset," Fifth World Congress on Intelligent Control and Automation, Vol. 5, pp. 4296-4299, 2004.

[5]. M. J. Zaki and K. Gouda, "Fast vertical mining using diffsets," Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '03, PP.pp. 326-335, 2003.

[6]. J. Pei, J. Han, H. Lu, S. Nishio, S. Tang, and D. Yang. "*H-Mine: Fast and space-preserving frequent pattern mining in large databases*", IIE Transactions, Vol.39, Issue 6, pp.593-605, 2007.
[7]. Takeaki Uno, Masashi Kiyomi and Hiroki Arimura, "*LCM ver. 2: Efficient Mining Algorithms for Frequent/Closed/Maximal Itemsets*",. Proc. IEEE ICDM Workshop on Frequent Itemset Mining Implementations Brighton, UK, November 1, 2004.
[8]. Bastide, Y., Taouil, R., Pasquier, N., Stumme, G., &Lakhal, L., "*Mining frequent patterns with counting inference",* ACM SIGKDD Explorations Newsletter, Vol.2, Issue.2, 66-75, 2000.
[9]. P. Sagar, M. Goyal, "*A Review: Comparative Analysis of various Data Mining Techniques*", International Journal of Computer Sciences and Engineering, Vol.4, Issue.12, pp.56-60, 2016.
[10]. JHS Tomar, JS Kumar, "*A Review on Big Data Mining Methods*", International Journal of Scientific Research in Network Security and Communication, Vol.4, Issue.3, pp.7-14, 2016.
[11]. R.J. Bayardo, "*Efficiently Mining Long Patterns from Databases*", ACM SIGMOD Conf. Management of Data, June 1998.
[12]. M.Manigandan and K. Aravind Kumar, "*Secure Multiparty Protocol for Distributed Mining of Association Rules*", International Journal of Scientific Research in Computer Science and Engineering, Vol.3, Issue.1, pp.6-10, 2015.
[13]. R. Agrawal and R. Srikant, "*Fast Algorithms for Mining Association Rules",* Proc. 20th Very Large Data Base Conf., Sept. 1994.
[14]. D. Gunopulos, H. Mannila, and S. Saluja, "*Discovering All the Most Specific Sentences by Randomized Algorithms*", Int'l Conf. Database Theory, pp. 215-29, Jan. 1997.
[15]. S. Brin, R. Motwani, J. Ullman, and S. Tsur, "*Dynamic Itemset Counting and Implication Rules for Market Basket Data*", Proceedings of the 1997 ACM SIGMOD international conference on Management of data - SIGMOD '97, 1997.
[16]. M. J. Zaki and C.-J. Hsiao., "*CHARM: An efficient algorithm for closed association rule mining*", Technical Report 99-10, Computer Science Dept., Rensselaer Polytechnic Institute, October 1999.
[17]. D. Cheung, J. Han, V. Ng, A. Fu, and Y. Fu, "*A Fast Distributed Algorithm for Mining Association Rules*", Fourth Int'l Conf. Parallel and Distributed Information Systems, Dec. 1996.
[18]. B.A. Davey, H.A. Priestley, "*Introduction to Lattices and Order*", Cambridge Univ. Press, 1990.
[19]. D. Eppstein, "*Arboricity and Bipartite Subgraph Listing Algorithms*", Information Processing Letters, Vol.51, pp. 207-211, 1994.
[20]. Philippe Fournier-Viger, "*An Open-Source Data Mining Library*", www.philippe-fournier-viger.com
[21]. M.R. Garey and D.S. Johnson, "*Computers and Intractability: A Guide to the Theory of NP-Completeness*", Revista Da Escola De Enfermagem Da USP, Vol.44, Issue.2, pp.340, 1979.

