ISSN : 0976-8491 (Online) | ISSN : 2229-4333 (Print)

Performance Improvement of Association Rule Mining Algorithms Through ASRMOLE Approach

[1]Biswaranjan Nayak, [2]Srinivas Prasad
[1]Dept. of CSE, Trident Academy of Technology, Bhubaneswar, Odisha, India
[2]Dept. of CSE, Gandhi Institute for Technological Advancement, Bhubaneswar, Odisha, India

Abstract
This paper deals with the effective utilization of association rule mining algorithms in large databases used for especially business organizations where the amount of transactions and items plays a crucial role for decision making. Frequent item-set generation and the creation of strong association rules from the frequent item-set patterns are the two basic steps in association rule mining. We have taken suitable illustration of market basket data for generating different item-set frequent patterns and association rule generation through this frequent pattern by the help of Apriori Algorithm and taken the same illustration for FP-Growth association rule mining and a FP-Growth Tree has been constructed for frequent item-set generation and from that strong association rules have been created. For performance study of Apriori and FP-Tree algorithms, experiments have been performed. The customer purchase behaviour i.e. seen in the food outlet environments is mimicked in these transactions. By using the synthetic data generation process, the observations has been plotted in the graphs by taking minimum support count with respect to execution time. From the graphs it has that as the minimum support values decrease, the execution times algorithms increase exponentially which is happened due to decrease in the minimum support threshold values make the number of item-sets in the output to be exponentially increased. It has been established from the graphs that the performance of FP-Growth is better than Apriori algorithm for all problem sizes with factor 2 for high minimum support values to very low level support magnitude.

Keywords
Association Rule Mining, Confidence, Support, Data Mining

I. Introduction

Basically, association rule mining is a data mining method which formulates correlation among a set of items with other set of items in a database which was first proposed by three researchers named R. Agrawal, T. Imielinski and A.N. Swami in 1993 [15]. The advantages of these rules are that they can be utilized for extracting exhilarating correlations, frequent association, patterns or clausal structures among sets of items that are present in the large transactional databases or any additional repositories [1]. Association rule mining covers wide application areas spanning from market-basket database used in big super market applications, telecommunication networks, risk management, financial investment, manufacturing and production, scientific application domains, health care to World Wide Web [7] where large volumes of data are stored either in centralized databases or in distributed databases.
Association rule mining try to find out the items in the database which are expected to be associated together with an interesting relationship so that it will pave the way for execution of effective decision making process of the organization while dealing huge set of transactional business records.

II. A Peep into Performance Measurement Factor

In [16], the researchers proposed techniques for efficiently mining frequent itemsets from large historical market basket databases. After that so many researchers proposed various techniques with designed algorithms as specified in literature, [4-6, 8-10, 12-14] and. In [6], a prefix tree structure for compactly representing and processing pattern information technique has proposed by researchers.
In literature [5], an algorithm called VIPER is proposed by researchers which can be utilized for organizing and processing the database on a vertical layout i.e. columnar basis in contrast to the traditional horizontal (row) layout.
In our study the data mining environment is based on single processor and the pattern lengths in the database are small enough such that all the frequent itemsets and intermediate results obtained by mining algorithms can fit in main memory.
We have emphasized on an association rule mining algorithm approach called "Oracle Algorithm" adopted in [6] which has the specialty to determine the identities of all frequent itemsets in the database in advance and actual supports of these itemsets are only required to perform the data mining tasks. As Oracle algorithm possesses good utilization in data structures and database organizations, we have implemented that and our experimental observation shows that the performance of Oracle is better than other existing mining algorithms.
We propose a new association rule mining algorithm approach called "ASRMOLE" which is made by modifying the Oracle structure and it follows a two pass scan over the database and its experimental observation shows that within a factor of two of the Oracle over real databases with the specified support specification and justifies its potentiality.
The notations specified in Table 1 are used in the proposed algorithm.

Table 1: Notations used in Algorithms

Notations	Definition
D	Database (Customer purchase transactions)
minsup	Minimum rule support (user defined)
F	Set of frequent itemsets in D
G	DAG structure used for storing candidates at the time of algorithm execution
P1,P2,P3..Pn	Set of n disjoint partitions of D
D	No of transactions in partitions scanned so far during algorithm execution except the current partition
d^+	No of transactions in partitions scanned so far during algorithm execution along with the current partition
O (X)	Cardinality of X

III. The Proposed Asrmole Algorithm

We propose a new algorithm called ASRMOLE (Association Rule Mining with Oracle) where we have emphasized on determining the minimal amount of change to the Oracle algorithm as it is required in online algorithm with new perspective approach as contrast to previous online algorithms.

Augmenting the same guideline form Oracle algorithm, in our ASRMOLE algorithm, database is also theoretically partitioned into n disjoint blocks P1, P2, …..Pn and two passes are made over the database. We have formed a set of candidate itemsets, G, which is certain to be a superset of the set of frequent itemsets in the first pass. In the first pass, the counts of candidates in G are determined over each partition in the similar manner followed in Oracle by maintaining the candidates in a DAG structure. As it is done in Oracle algorithm, the 1-itemsets and 2-itemsets are here also stored in lookup arrays with a difference that in contrast to Oracle, candidates are inserted and removed from G at the end of each partition. While computation of counts are done at the same time generation and removal of candidates is performed also. As the Method adopted in [9] along with each candidate X, we have also stored three integers which are as follows:

- X.Count : Used for the number of occurrences of X because X was last inserted in G.
- X.FirstPartition : Used as index of the partition at which X was inserted in G.
- X.MaxMissed : Used as upper bound on the number of occurrences of X before X was inserted in G. X.FirstPartition and X.MaxMissed are computed at the time X is getting inserted into G by using the approaches adopted in CARMA algorithm as specified in [9].

The pseudo-code of proposed ASRMOLE algorithm is specified in fig. 1.

A. Working Techniques of 1st & 2nd Pass

In this section, we explain about the working technicality of algorithm with respect to 1st pass and 2nd pass along with candidate generation and candidate removal process respectively.

```
ASRMOLE(𝒟, I, minsup)
Input: Database D, Set of Items I, Minimum Support minsup
Output: F ∪ N with supports
1. Np= Number of partition
//------1st Pass----------//
2. G=I
3. For i=1 to Np
4. ReadNextPartition(Pi,G)
5.      For each singleton X in G
6. X.count=X.count+ O(X.tidlist)
7.      Modify1(X,minsup)
8.    Next X
9.  Next i
//------2nd Pass----------//
10. Removesmall(G,minsup)
11. Outputfininshed(G,minsup)
12. For i=1 to Np
13.      If ( all candidates in G have been output)
14.            Exit
15.      Endif
16. ReadNextPartition(Pi,G)
17.      For each singleton X in G
18.         Modify2(X,minsup)
19.      Next X
20. Next i
21. End
```

Fig. 1: Pseudo-code of ASRMOLE Algorithm

1. 1st Pass and Candidate Generation Process

We divide the discussion into two parts as follows:

(i). Normal Activities in 1st Pass

Initially, the set of candidate itemsets G is initialized to the set of singleton itemsets in the 1st pass. The ReadNextPartition function reads tuples from the next partition and at the same time creates tid-lists of singleton itemsets in G. Once the entire partition is read, then the Modify1 function is applied on each singleton in G which increments the counts of existing candidates according to their corresponding counts in the current partition. Not only that, it also helps for generation and removal of candidates. After all these steps, the set of candidate itemsets G comprised with a superset of the set of frequent itemsets.

So when for a candidate in G which has been inserted at partition Pj, its count over the partitions P_j…….P_n will be obtainable automatically at that time.

(ii). Candidate Generation in 1st Pass

We have utilized the incremental mining algorithms techniques adopted in [2] & [11] for efficiently candidate generation purposes in ASRMOLE algorithm. Those incremental algorithms are designed to obtain the current mining output by utilizing previous mining results when a database is required to be updated by using an increment. Our algorithm treats the database scanned so far with d, as the original database and the current partition being processed as the "increment". Let's we denote d^+ as the portion of the database scanned so far including the current partition being processed. Assume that F^d and F^{d+} be the sets of frequent itemsets over d and d^+, respectively, and N^d and N^{d+} be taken as their corresponding negative borders. We have referred the theorem in [11] as follows:

Theorem 4: If X is an itemset that is not in F^d but is in F^{d+}, then there must be some subset x of X which was in N^d and is now in F^{d+}.

The itemsets which are moved from N^d to F^{d+} are called promoted borders.

As per the cited theorem, the only candidates that require to be generated are those which are actually the supersets of the promoted borders. To signify the process of generating the required supersets of P, the term called expanding a promoted border P is used.

We have used the Expand function presented in [2] which is specified in Figure 4, the inputs to which are P, the promoted border to be expanded and G, the current set of candidates. Actually the Expand is imitated from AprioriGen function the approach utilized in [14], because the siblings of P are accurately those itemsets in G which differ from P in the last item. The Expand function and its usage differ from AprioriGen function in three ways as specified follows:

- It is utilized in a dynamic fashion at that time when a candidate that was in the negative border becomes d-frequent.
- It is applied to individual itemsets in contrast to AprioriGen which utilizes the set of itemsets.
- A parent based pruning optimization approach is utilized here whereas in AprioriGen function, for pruning purpose, it enumerates all immediate subsets of a candidate.

```
Expand (P, G)
Input: Promoted Border P, DAG G
            for each sibling X of P in G
              if (X is d-frequent) then
                  S = P   X// new candidate
                  Insert S into G as a child of P
```

Fig. 2: Expand Function (Expanding a Promoted Border)

The Expand function is (refer to fig. 2) integrated into ASRMOLE algorithm by calling it from the Modify1 function which is invoked for each partition scanned all through the 1st pass. The Pseudo-code of Modify1 function is shown in fig. 3.

```
Modify1(M,minsup)
Input: DAG Node M, Minimum Support minsup
Output: M and its Descendents Updated
1. B=Transfer M.tidlist format into Tid.vector form
2. For each node X in M.childlist
3.       X.tidlist=Intersect(B,X,father.tidlist)
4.       X.count=X.count+X.tidlist
5. Next  X
6. For each node X in M.childlist
7.        If maxSupport(X)<=minsup  then
8.                If  O(X.childlist)>0  then
9.                Remove all supersets of X reachable from
X in DAG G
10.                 End if
11.          Else
12.                 If O(X.childlist)=0 then
13.                      Expand(X)
14.                  End if
15.          End if
16.  Next  X
17. For each node X in M.childlist
18.       Modify1(X)
19. Next X
20. End
```

Fig. 3: Pseudo-code of Mofify1 Function

Here in Modify1 function, the tidlist of a given node M is first converted to the tid-vector (TV) format. The tidlists of all children of M are computed are specified after which the same children are visited in a depth first search method. The candidate generation and removal are done here dynamically which is performed in one enumeration of all children of a given node. For each child X that is enumerated, if it has supersets but is not d-frequent, then all supersets of X that are reachable from X in the DAG are removed. But X itself is not removed in view of the fact that it could be part of the current negative border.

2. 2nd Pass & Candidate Removal Process

Here also we divide the discussion into two parts as follows:

(i). Normal Activities in 2nd Pass

In the 2nd pass, where candidates in G those are neither d-frequent nor part of the current negative border, they are required to be removed from G. At the 1st partition, those candidates that have been inserted in G their counts over the entire database will be obtainable easily and these itemsets with their counts are produced as output. The OutputFinished function outputs an itemset X and X does not have any supersets left in G and X is removed from G. The ReadNextPartition function in the 2nd pass reads tuples from the next partition and creates tid-lists of singleton itemsets in G. Once the whole partition is read, the Modify2 function is applied on each singleton in G. Before reading in the next partition begins, it is verified whether there are still any more candidates left out. If no candidates found, the mining process terminates at that point.

(ii). Candidate Removal in 2nd Pass

During 2nd pass, the candidate X is removed prerequisite to the following two conditions:

- When the count of X over the entire database is available, that becomes true when X.firstPartition is the next partition to be processed then the candidate X is removed.
- When X does not have supersets at that time X is removed.
- The Pseudo-code of Modify1 function is shown in Figure 4.

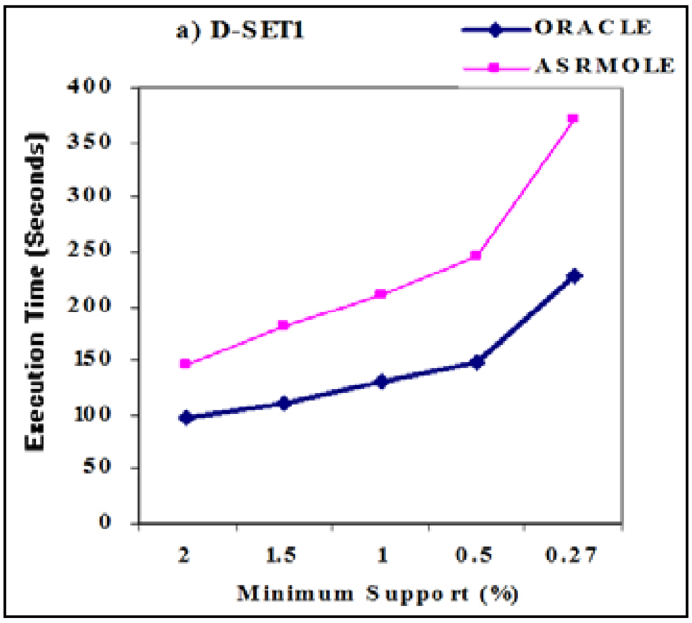
```
Modify2(M)
Input: DAG Node M
Output: M and all its descendents with updated counts
1.          B=Convert. Mn.tidlist to Tid.vector
2.          For each node X in Mn.childset
3.          X.tidlist= Inset( B,X,Father.tidlist)
4.          X.count=X.count+X.tidlist
5.          Next X
6.          For each node X in Mn.childset
7.              Modify(X)
8.          Next X
9.          End
```

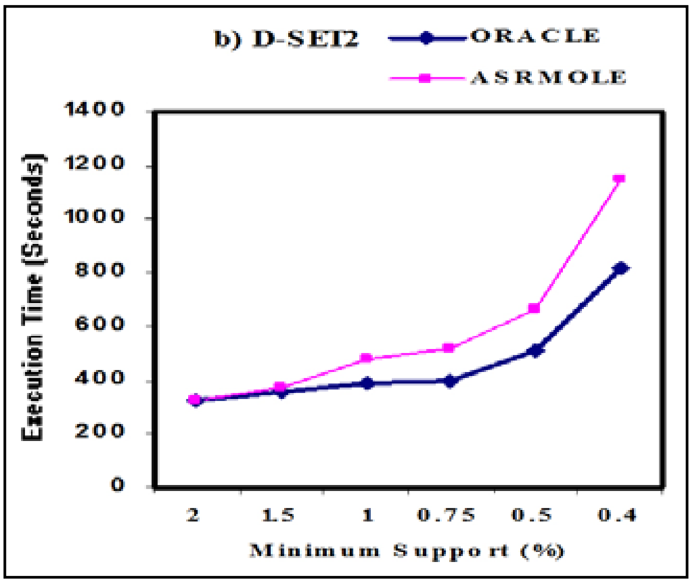
Fig. 4: Pseudo-code of Mofify2 Function

The Modify2 function, increment the counts of existing candidates by enumerating all its itemsets utilizing a depth first search which imitates update function of Oracle algorithm.When a node in the tree is visited, the tidlists of all its children are computed which ensures that when an itemset is visited, the tidlists of its mother and father have already been computed. At the end it also outputs candidates whose counts over the entire database are known. But if it outputs an itemset X and X does not possess any supersets left in G, then X is removed from G.

A. Experimental Observations of ASRMOLE

We have performed the experiments with synthetically datasets. We have used the synthetically generated datasets D-SET1, D-SET2 and D-SET3 where each of the dataset has the capacity of 10 millions transactional tuples in it. We have compared both the execution time of ORACLE algorithm, with respect to proposed ASRMOLE algorithm and projected the results in fig. 5 (a,b,c) respectively for three datasets.





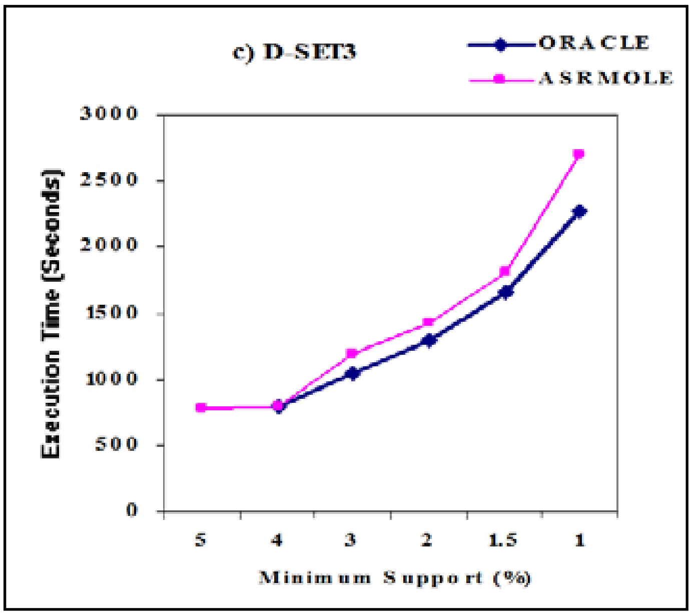


Fig. 5(a,b,c): Performance of ASRMOLE Using Synthetic Datasets

It is observed from the above graphs that for high supports ASRMOLE algorithm performs closer in performance to Oracle algorithm because at high supports the density of the frequent itemset distribution is sparse for which yield a few frequent itemsets with close minsup supports. So within most partitions, frequent itemsets are likely to be locally frequent. Although in some partitions if the itemsets are not locally frequent but they are still d-frequent over those partitions that's why their counters are updated even over these partitions. In the 2nd pass, it would produce the complete counts of most candidates for which the expectation lies in the fact that the performance of ASRMOLE will be nearer/close to Oracle algorithm. But in our experimental observations we found that at low supports our proposed algorithm decreased with respect to Oracle algorithm. But the important part of our part is that we have observed a consistence performance of ASRMOLE algorithm with a factor of 2(two) to Oracle algorithm as the performance ratios of Oracle and ASRMOLE algorithm by taking lowest support counts is depicted in the Table 2.

Table 2: Efficiency Ratio performance of ASRMOLE/Oracle

	Database		
	D-SET1	D-SET2	D-SET3
Minsup Threshold (%)	0.1	0.35	1.12
Time taken by Oracle (seconds)	226.99	814.01	2267.26
Time taken by ASRMOLE (Seconds)	371.44	1145.42	2699.64
Ratio of ASRMOLE/ORACLE	1.63	1.40	1.19

V. Conclusion

The experimental observations in our work has established the fact that there exists a gap between the performance of current mining algorithms and Oracle algorithms. The proposed online association rule mining ASRMOLE has been constructed with nominal changes to Oracle to obtain as an effective an online algorithm. ASRMOLE algorithm utilizes a new method of candidate generation which is not efficient but also dynamic and performs two scan over the database positively.

References

[1] Kotsiantis, S., Kanellopoulos, R.,"Association Rules Mining: A Recent Overview", GESTS International Transactions on Computer Science and Engineering, 32(1), pp. 71-82, 2006.
[2] V. Pudi, J. Haritsa,"How good are association-rule mining algorithms? In Proc. of Intl. Conf. on Data Engineering (ICDE), 2002.
[3] J. Han, J. Pei, Y. Yin,"Mining frequent patterns without candidate generation", In Proc. Of ACM SIGMOD Intl. Conf. on Management of Data, 2000.
[4] J. Han, J. Pei, Y. Yin.,"Mining frequent patterns without candidate generation", In Proc. Of ACM SIGMOD Intl. Conf. on Management of Data, pp. 1-12, 2000.
[5] P. Shenoy, J. Haritsa, S. Sudarshan, G. Bhalotia, M. Bawa, D. Shah,"Turbo-charging vertical mining of large databases", In Proc. Of ACM SIGMOD Intl. Conf. on Management of Data, pp. 143-160, 2000.
[6] V. Pudi, J. Haritsa,"Quantifying the utility of the past in mining large databases", Information Systems, 2000.
[7] S Lawrence, C Lee Giles,"Accessibility of Information on the Web [J]", 1999, pp. 403:107-109.

[8] R. C. Agarwal, C. C. Aggarwal, V. V. V. Prasad,"Depth First generation of long patterns", In Proc. of ACM SIGMOD Intl. Conf. on Management of Data, 1998.
[9] C. Hidber,"Online association rule mining", In Proc. Of ACM SIGMOD Intl. Conf. on Management of Data, pp. 85-93, 1999.
[10] E. Han, G. Karypis, V. Kumar, B. Mobasher,"Clustering based on association rule hypergraphs", In Workshop on Research Issues on Data Mining and Knowledge Discovery, 1997.
[11] S. Thomas et al,"An efficient algorithm for the incremental updation of association rules in large databases", In Proc. of Intl. Conf. on Knowledge Discovery and Data Mining (KDD), 1997.
[12] A. Savasere, E. Omiecinski, S. Navathe,"An eÆcient algorithm for mining association rules in large databases", In Proc. of Intl. Conf. on Very Large Databases (VLDB), 1995.
[13] J. S. Park, M. Chen, P. S. Yu,"An effective hash-based algorithm for mining ssociation rules", In Proc. of ACM SIGMOD Intl. Conf. on Management of Data, November 1995.
[14] R. Agrawal, R. Srikant,"Fast Algorithms for mining Association rules", In Proc. Of Intl. Conf. on Very Large Databases (VLDB), pp. 487-499, 1994.
[15] Agrawal, R., Imielinski, T., Swami, A. N.,"Mining association rules between sets of items in large databases", In Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, pp. 207-211, 1993.

Biswaranjan Nayak is working as Asst. Profesor in Dept. Of CSE at Trdent Academy of Technology, Bhubaneswar, Odisha, India. He has received his MCA and M.Tech degree in Computer Science from NIELIT, New Delhi. He has more than 20 years of teaching and software dévelopment experience.

Dr. Srinivas Prasad is working as Professor, Dept. Of CSE & Dean (Academics) at GITA, Bhubaneswar, Odisha, India. He has received his PhD. Degree in Comp. Sc. He has received his B.E in Computer Science from A.U, and M.Tech in Comp. Appl. from ISM Dhanbad, India and M.S. in System Software from. He has more than 22 years of Software development and teaching experience in USA and India.