



# Implementation of Apriori Algorithm in MATLAB using Attribute Affinity Matrix

**N. Badal**

*Department of Computer Science & Engineering  
Kamla Nehru Institute of Technology (KNIT)  
Sultanpur, U.P, INDIA.*

**S. Bagga**

*Department of Computer Science & Engineering,  
Dehradun Institute of Technology (DIT)  
Dehradun, U.K, INDIA.*

---

**Abstract:** *One of the most popular methods of data mining from large scale data warehouse is association rule mining with the help of Apriori algorithm. In this paper a new approach to implement Apriori algorithm using MATLAB is presented which efficiently mines the frequent data itemsets from a large database. Using this approach the Attribute Affinity Matrix-a new methodology for implementing Apriori Algorithm is presented. With the help of experimental results, it is being demonstrated that the Apriori algorithm when implemented in MATLAB performs better than to the existing Apriori algorithm.*

**Keywords:** *Association rule mining, frequent itemset, Apriori.*

---

## I. INTRODUCTION

Data mining plays an important role in lots of the applications. Data mining is a process that allows us to extract knowledge and valid and exploitable information from different data sources. In data mining, frequent item sets is used to identify the correlations between the various fields of database. Association rules are used to identify relationships among a set of items in database. These relationships are not based on inherent properties of the knowledge [2] themselves, but based on co-occurrence of the knowledge items. Frequent itemset and association rule mining [1] have become a widely researched area, and hence many algorithms have been introduced for frequent itemset and association rule mining. The Apriori algorithm is used for association rule mining. Apriori is the best-known basic algorithm for mining frequent item sets in a set of transactions. This paper describes the implementation of Apriori algorithm that in MATLAB to achieve maximum performance, with respect to execution time. In this paper Apriori algorithm is implemented using a new methodology that is Attribute Affinity Matrix. In the next section literature review is presented followed by Apriori algorithm and then methodology used is explained.

## II. BACKGROUND

A brief literature review of related work is presented in this section. The literature review is being partitioned in two subsections. The first subsection is concerned with the work related to the progress on mining of frequent data itemset followed by the limitations of existing work.

First subsection deals with the issue of scalability that are as follows.

### A. Tries:

S. Brin et. al. in [4] introduced the concept of storing candidates in a Trie. Trie is alternatively known as a prefix tree. It takes the advantage of redundancies in the key that are located in the tree.

Trie is a data structure developed by E. Fredkin in [5]. J. L. Bentley in [3] states that item of a candidate correspond to one node along that path. Then to retrieve a candidate from the structure, one reads the sequence of items encountered while traversing to the leaf. Bodon [10] described, this Trie idea is a ubiquitous approach and adopted in the state-of-the-art Apriori implementations. The concerned topic is also reviewed by many researchers in [11, 12, 7, 6].

If the data structure does not fit in the main memory then this approach has the potential to break down on large datasets. This approach also has the potential to break down on large datasets if the data structure no longer fits in main memory. The depth of the trie is equal to the length of those candidates. To fit all nodes into main memory requires those candidates to overlap quite substantially. When they do not, the effect of the Trie's heavily pointer-based makeup is very poor localisation and cache utilisation. Consequently, traversing it causes one to thrash on disk and the efficiency of the structure is quickly consumed by I/O costs.

### B. FP-Growth

In [13] J. Han et al. introduced a new algorithm to solve the problem of frequent data itemset mining. The FP-Growth Algorithm is used to find out frequent itemsets without using candidate generations, thus improving performance. This algorithm works as follows: first it compresses the input database creating an FP-tree instance to 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. Finally, each such database is mined separately.

FP-tree is a compact structure that stores quantitative information about frequent patterns in a database.

Han defines the FP-tree as the tree structure defined below:

- 1) One root labeled as "null" with a set of item-prefix subtrees as children, and a frequent-item-header table;
- 2) Each node in the item-prefix subtree consists of three fields:
  - a) *Item-name*: registers which item is represented by the node;
  - b) *Count*: the number of transactions represented by the portion of the path reaching the node;
  - c) *Node-link*: links to the next node in the FP-tree carrying the same item-name, or null if there is none.
- 3) Each entry in the frequent-item-header table consists of two fields:
  - a) *Item-name*: as the same to the node;
  - b) *Head of node-link*: a pointer to the first node in the FP-tree carrying the item-name.

### C. VS\_Apriori Algorithm

N. Badal et. al. in [14] introduced the concept of VS\_Apriori algorithm. In this method, the mining of frequent data itemset can be done using minimum support threshold. Data itemsets are both horizontally and vertically sorted, so all those itemsets that are nearly equal appears same.

#### Limitations of existing work:

The major problem with the classical Apriori algorithm is in increasing the transactions of the database. The Trie structure solves the problem occurred due to increase in transaction in database. In Trie the process of matching the candidate to the transaction at the same time completes the process of loading the counter because the counter is stored in the leaf of the Trie structure. This approach is also not fast and is presented by comparing 1.7 million transactions with 30 million candidates as done on Webdocs dataset which is given by C. Lucchese et. al. in [9]. The disadvantage of FP-Growth algorithm is that it works on conditional pattern bases and is used to build FP-tree recursively. It does not perform well for data sets having large patterns. Another problem with FP-Growth algorithm is that it lacks the behavior of incrementing counter which results in fault tolerance.

## III. THE APRIORI ALGORITHM

The first algorithm introduced for frequent itemset and Association rule mining was Apriori. Apriori algorithm is proposed by R. Agrawal and R Srikant in 1994. The name of algorithm is based on the fact that the algorithm uses prior knowledge of frequent item set properties. It uses a breadth-first search strategy for counting the support of itemsets. It also uses a "bottom up" approach, where frequent subsets are extended one item at a time (a step known as candidate generation) and groups of candidates are tested against the data.

It is a seminal algorithm, which uses an iterative approach known as a level-wise search, where k-itemsets are used to explore (k+1)-itemsets. It uses the Apriori property to reduce the search space: All nonempty subsets of a frequent itemset must also be frequent.

Apriori algorithm works on two concepts

- a) Self Join- two itemsets having one and only one different item can be joined.
- b) Pruning- Candidates containing non-frequent subsets are removed.

Steps of Apriori algorithm-

- i. First, the set of frequent 1- itemsets is found which is denoted by C1.
- ii. The next step is support calculation which means the occurrence of the item in the database. This requires scanning of complete database.
- iii. Then the pruning step is carried out on C1 in which the items are compared with the minimum support parameter. The items which satisfy the minimum support criteria are only taken into consideration for the next process which is denoted by L1.
- iv. Then the candidate set generation step is carried out in which 2-itemset are generated this is denoted by C2.
- v. Again the database is scanned to calculate the support of the 2-itemset. According to the minimum support the generated candidate sets are tested and only the item set which satisfies the minimum support criteria are further used for 3- item set candidate set generation.

This above step continues till there is no frequent item set or candidate set that can be generated.

User can easily understand the concepts used by the Apriori with the help of following example. Table I shows a transactional database having 4 transactions.

TID is a unique identification given to the each transaction.

TABLE I: SAMPLE DATABASE

TID	ITEMS
T001	I1,I3,I4
T002	I2,I3,I5
T003	I1,I2,I3,I5
T004	I2,I5

After the first step user will get the support of each item given in database which is shown in Table II.

TABLE II: SUPPORT OF EACH ITEM

ITEMSET	SUPPORT
{I1}	2
{I2}	3
{I3}	3
{I4}	1
{I5}	3

Assuming minimum support here as 2. User will get only those items which have support greater than or equal to 2. Table III shows the result of pruning.

TABLE III: PRUNING AT FIRST LEVEL

ITEMSET	SUPPORT
{I1}	2
{I2}	3
{I3}	3
{I5}	3

Table IV shows all the possible combination that can be made from Table itemset.

TABLE IV: COMBINATION OF ITEM FROM TABLE 3

ITEMSET	SUPPORT
{I1,I2}	1
{I1,I3}	2
{I1,I5}	1
{I2,I3}	2
{I2,I5}	3
{I3,I5}	2

From Table IV two itemsets will be removed. After pruning the user will get the following results.

TABLE V: PRUNING AT SECOND LEVEL

ITEMSET	SUPPORT
{I1,I3}	2
{I2,I3}	2
{I2,I5}	3
{I3,I5}	2

The same procedure gets continued till there are no frequent item sets or candidate set that can be generated. Step 6 is described in Table VI and Table VII.

TABLE VI: COMBINATION OF ITEM FROM TABLE 5

ITEMS ET	SUPPO RT
{I1,I2,I3 }	1
{I1,I2,I5 }	1
{I2,I3,I5 }	2

TABLE VII: FINAL RESULT

ITEMSET	SUPPORT
{I1,I2,I3}	1
{I1,I2,I5}	1
{I2,I3,I5}	2

Pseudo Code-

The pseudo code of Apriori algorithm is described by J. Han et. al. in [8] and is represented step wise.  
 In step 1 candidate itemset of k size is taken and is denoted as  $C_k$ .  
 In step 2  $L_k$  is taken which denotes frequent itemset of size k.  
 In step 3  $L_1$  contains the set of frequent itemset.  
 Step 4 includes the for loop, in which k is incremented in each step until  $L_k$  is equal to zero.  $C_{k+1}$  denotes the candidate set generated from  $L_k$ .

In step 5 the count of each candidate that is in  $C_{k+1}$  is incremented.

In step 6  $L_{k+1}$  represents the candidates in  $C_{k+1}$  with the minimum support count.

```

Step 1.  $C_k$ : Candidate itemset of size k
Step 2.  $L_k$ : frequent itemset of size k
Step 3.  $L_1 = \{\text{frequent items}\}$ ;
Step 4. for ( $k = 1$ ;  $L_k \neq \emptyset$ ;  $k++$ ) do begin
     $C_{k+1}$  = candidates generated from  $L_k$ ;
Step 5. for each transaction t in database do
    increment the count of all candidates in
     $C_{k+1}$  that are contained in t
Step 6.  $L_{k+1}$  = candidates in  $C_{k+1}$  with min_support
Step 7. End
Step 8. return  $\bigcup_k L_k$ ;
    
```

Figure 4.1: Pseudo Code of Apriori Algorithm

#### IV. PROPOSED METHODOLOGY

New methodology to implement Apriori algorithm is Attribute affinity matrix, which is generated from attribute usage matrix. Attribute usage matrix is a usage of attributes in important transactions. Each row in attribute usage matrix contains single transaction. If the value in the usage matrix is '1' then it indicates that the query uses the corresponding attribute. If the value is '0' then it means that the attribute is not used that query.

Attribute affinity can be calculated using the formula given in [15]-

$$\text{aff}(A_i, A_j) = \sum_k \text{ref}_1(q_k) \text{acc}_1(q_k) \quad (1)$$

for  $k$  such that  $use(q_k, A_i) = 1 \wedge use(q_k, A_j) = 1$  (2)

for  $\forall l$ ,  $\text{acc}_l(q_k)$  is the application access frequency (3)

$\text{ref}_l(q_k)$  is the number of accesses to attribute  $(A_i, A_j)$  for each execution of application  $q_k$  at site  $S_k$  (4)

Attribute affinity matrix can be easily understood with the help of following example-

Consider 4 queries  $q_1, q_2, q_3, q_4$  for any relation  $P$  where

$q_1 = \text{SELECT BUDGET FROM } P \text{ WHERE } PNO = \text{value}$

$q_2 = \text{SELECT } PNAME, BUDGET \text{ FROM } P$

$q_3 = \text{SELECT } PNAME \text{ FROM } P \text{ WHERE } LOC = \text{value}$

$q_4 = \text{SELECT SUM (BUDGET) FROM } P \text{ WHERE } LOC = \text{value}$

Let attribute  $A_1 = PNO$ ,

attribute  $A_2 = PNAME$ ,

attribute  $A_3 = BUDGET$ ,

attribute  $A_4 = LOC$

Figure 4(a) represents the attribute usage matrix.

Figure 4(b) represents the access frequencies.

Assume that the query access the attribute only once.

Now calculating the attribute affinity matrix using the formula described above

$$\text{aff}(A_1, A_3) = 15 * 1 + 20 * 1 + 10 * 1 = 45$$

Figure 4(c) shows the attribute affinity matrix calculated from the equation (i).

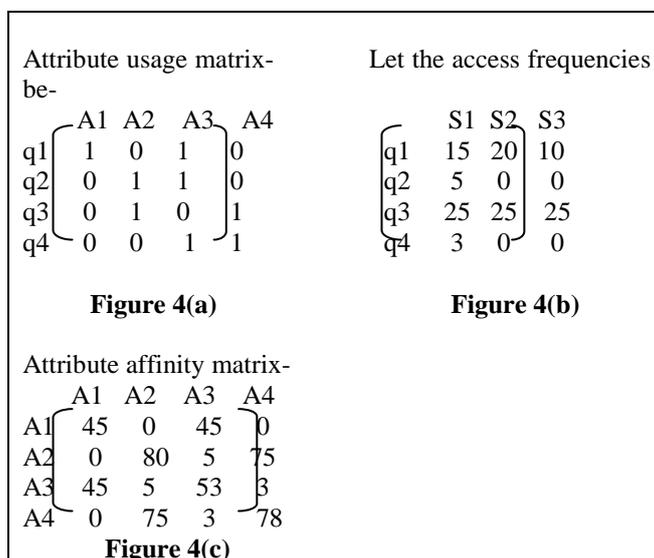


Figure 4: Matrices of Attribute Affinity

### V. APRIORI ALGORITHM IN MATLAB

In this new approach, Apriori algorithm is implemented in MATLAB. Implementing Apriori algorithm in MATLAB reduces execution time and scans the database only once. In this approach, a text file is created in which the transactions on which the Apriori algorithm is to be applied are stored. The text file created is stored in MATLAB in matrix form. Then the Apriori algorithm is applied on matrix.

User can easily understand this concept with the help of following example.

Pseudo Code –

The pseudo code of the proposed system is presented. Step1 is the candidate itemset of size k denoted by  $C_k$ . Step 2 represents  $L_k$ , the frequent itemset of size k. Step 3 denotes the matrix S that contains the text file. Text file contains database. Step 4 denotes the set of frequent itemset  $L_1$ . In step 5 the text file is loaded into the matrix S. In step 6  $L_k$  is incremented until it is equal to zero.  $C_{k+1}$  represents the candidate generated from  $L_k$ . In step 7 the count of each candidate in  $C_{k+1}$  is incremented. In step 8  $L_{k+1}$  denotes the candidates with minimum support count that are in  $C_{k+1}$ .

```

Step 1.  $C_k$ : Candidate itemset of size k
Step 2.  $L_k$  : frequent itemset of size k
Step 3. S : Matrix containing text file
Step 4.  $L_1 = \{ \text{frequent items} \};$ 
Step 5.  $S = \text{load}(\text{'text\_file.txt'});$ 
Step 6. for (k = 1;  $L_k \neq \emptyset$ ; k++) do begin
     $C_{k+1}$  = candidates generated from  $L_k$ ;
Step 7. for each transaction t in database do
    increment the count of all candidates in
     $C_{k+1}$  that are contained in t
Step 8.  $L_{k+1}$  = candidates in  $C_{k+1}$  with min_support
Step 9. End
Step 10. return  $\hat{U}_k L_k$ ;
```

Figure 4.2: Pseudo Code of Apriori Algorithm in MATLAB

Comparative analysis of pseudo code-

In this section pseudo code of Apriori algorithm when implemented in other language and when implemented in MATLAB is compared. The pseudo code of Apriori algorithm contains two new steps 3 and 5. In step 3 the matrix S is taken which contains the database in the form of text file. In step 5 load function is used to load the text file into the matrix S.

<u>Apriori algorithm</u>	<u>Apriori algorithm in MATLAB</u>
1. $C_k$ :Candidate itemset of size k	1. $C_k$ :Candidate itemset of size k
2. $L_k$ : frequent itemset of size k	2. $L_k$ :frequent itemset of size k
3.(NOT APPLICABLE)	3.S:Matrix containing text file
4. $L_1 = \{ \text{frequent items} \};$	4. $L_1 = \{ \text{frequent items} \};$
5. (NOT APPLICABLE)	5. $S = \text{load}(\text{'text\_file.txt'});$
6. <b>for</b> (k = 1; $L_k \neq \emptyset$ ; k++) <b>do begin</b>	<b>6.for</b> (k = 1; $L_k \neq \emptyset$ ; k++) <b>do begin</b>
7. $C_{k+1}$ =candidates generated from $L_k$ ;	7. $C_{k+1}$ =candidates generated from $L_k$ ;
8. <b>for each</b> transaction t in database do Increment the count of all candidates in $C_{k+1}$ that are contained in t	8. <b>for each</b> transaction t in database do increment the count of all candidates in $C_{k+1}$ that are contained in t
9. $L_{k+1}$ = candidates in $C_{k+1}$ with min_support	9. $L_{k+1}$ = candidates in $C_{k+1}$ with min_support
10. <b>end return</b> $\hat{U}_k L_k$ ;	10. <b>end return</b> $\hat{U}_k L_k$ ;

Figure 4.3: Comparison of Pseudo Code

## VI. EXPERIMENTAL RESULT

The computation time is calculated for the implementation of Apriori algorithm in MATLAB and is compared with the computation time of Apriori algorithm implemented in C. The same database is used for both the methods.

In figure 6.1, the comparison between the execution time and number of items is presented considering threshold value 2.

The execution time for C is 80% more than when implemented in MATLAB.

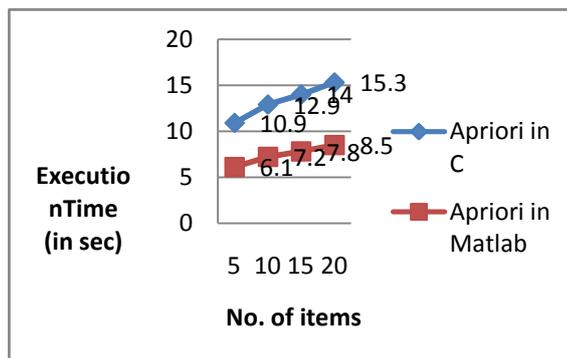


Figure 6.1: Comparison between implementation Of Apriori algorithm in c and Matlab (execution time vs. number of items)

In figure 6.2, the comparison between the different levels of Apriori algorithm in Matlab is presented. From the figure 6.2, it is clear that as the level increases the execution time decreases. When Apriori algorithm is implemented in C then the level wise execution time cannot be calculated.

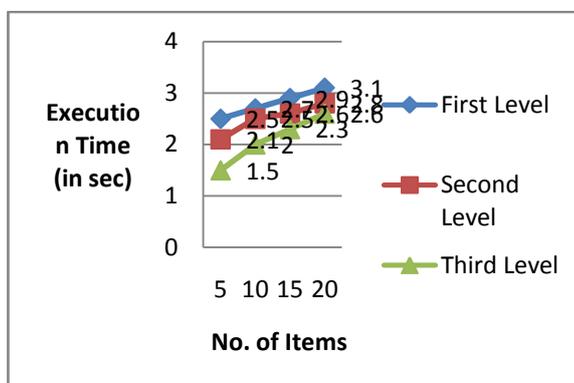


Figure 6.2: Comparison between different levels of Apriori algorithm in Matlab (execution time vs. number of items)

## VII. CONCLUSION AND FUTURE SCOPE

The new approach to implement Apriori algorithm in MATLAB using Attribute Affinity Matrix is presented in this paper to improve the efficiency of existing Apriori algorithm. This approach reduces the execution time and is more reliable than classical Apriori algorithm. In Apriori algorithm the execution time at different levels can be calculated which is not possible when implemented in C language. Therefore, it is proved that Apriori algorithm when implemented in MATLAB is 80% much better than existing Apriori algorithm. Further, the work may be extended to incorporate the dynamic nature in Apriori algorithm using MATLAB.

## REFERENCES

1. R. Agrawal, T. Imielinski, and A. Swami, "Mining Associations between sets of items in Massive Databases". In proc. Of the ACM-SIGMOD 1993 int'l conf. on Management of Data, Washington D.c USA, 1993A, pp. 207-216.
2. K. Vanitha and R. Santhi, "Using hash based Apriori algorithm to reduce the candidate 2- itemsets for mining association rule", Vol. 2, No. 5, April 2011.
3. J. L. Bentley, "Multidimensional binary search trees used for associative searching" in International journal of Communications of the ACM, Vol. 19, 1975, pp. 509-517.
4. S. Brin, R. Motwani, J. D. Ullman, and S. Tsur, "Dynamic data itemset counting and implication rules for market basket data," SIGMOD Rec., Vol. 26, No. 2, 1997, pp. 255-264.
5. E. Fredkin, "Trie memory". Commun. ACM, Vol. 3, No. 9, 1960, pp. 490-499.
6. P. Chan and S. Stolfo. "Experiments in Multistrategy Learning by Meta-Learning", In Proc. of the second International conference on Information And Knowledge Management, 1993, pp. 314-323.

7. P. Keleher, A. L. Cox, and W. Zwaenepoel, "Lazy Release Consistency for Software Distributed Shared Memory". In Proc. of the 19th Annual Int'l Symposium on Computer Architecture, 1992, pp. 13-21.
8. J. Han and M. Kamber, "Data mining concepts and techniques", Elsevier, 2nd Edition. Chapter 5.
9. C. Lucchese, S. Orlando, R. Perego, and F. Silvestri, "Webdocs: a real-life huge transactional dataset". In Jr. et. al.
10. Bodon, "A fast Apriori implementation," in Proc. of The IEEE ICDM Workshop on Frequent Data itemset Mining Implementations (FIMI'03), Melbourne, Florida, Vol. 90, 2003, pp 90-118.
11. Anderson and D. Shasha, "Persistent Linda: Linda + transactions + query processing". In J. P. Banatre and D. Le Metayer, editors, Research Directions in High-Level Parallel Programming Languages, Vol. 574, 1992, pp. 93-109.
12. E. Camahort and I. Chakravarty, "Integrating volume data analysis and rendering on distributed memory architectures" in Proc. of Parallel Rendering Symposium, ACM Press, 1993, pp. 89—96.
13. J. Han, J. Pei, and Y. Yin, "Mining frequent patterns without candidate generation," in SIGMOD Conference, 2000, pp. 1–12.
14. N. Badal and S. Tripathi, "Frequent Data Itemset Mining Using VS\_Apriori Algorithms", in International Journal on Computer Science and Engineering Vol. 02, No. 04, 2010, pp. 1111-1118.
15. M. Ozsu and P. Valduriez, "Principles of Distributed Database Systems," Prentice Hall, 1999, pp. 73-75.

### About the Authors

N. Badal is a Assistant Professor in the Department of Computer Science & Engineering at KNIT, Sultanpur (U.P.), INDIA. He received B.E. (1997) from Bundelkhand Institute of Technology (BIET), Jhansi in Computer Science & Engineering, M.E. (2001) in Communication, Control and Networking from Madhav Institute of Technology and Science (MITS), Gwalior and PhD (2009) in Computer Science & Engineering from Motilal Nehru National Institute of Technology (MNNIT), Allahabad. He is Chartered Engineer (CE) from Institution of Engineers (IE), India. He is a Life Member of IE, IETE, ISTE and CSI-India. He has published about 35 papers in International/National Journals, conferences and seminars. His research interests are evinced at Distributed System, Parallel Processing, GIS, Data Warehouse & Data mining, Software engineering and Networking. E-mail: n\_badal@hotmail.com

S. Bagga is an M.Tech student in the Department of Computer Science & Engineering at DIT, Dehradun (U.K.), INDIA. She received B.Tech (2011) from Lovely Professional University (LPU), Jalandhar in Information Technology. E-mail: srishti\_bagga@yahoo.com