International Journal of Engineering & Technology, 7 (1.5) (2018) 51-55

International Journal of Engineering & Technology

Website: www.sciencepubco.com/index.php/IJET

Research paper

Coalesce based binary table: an enhanced algorithm for mining frequent patterns

M. Sireesha [1], Srikanth Vemuru [2], S. N. TirumalaRao[3]

[1]Research Scholar, Asst. Prof, Department of CSE, Koneru Lakshmaiah Education Foundation, Vaddeswaram, Guntur, Andhra Pradesh, India 522502. Narasaraopet Engineering College
[2]Department of CSE, Koneru Lakshmaiah Education Foundation, Vaddeswaram, Guntur, Andhra Pradesh, India 522502
[3]Professor, Department of Computer Science and Engineering, Narasaraopeta Engineering College
**Corresponding author E-mail: sireeshamoturi@gmail.com*

Abstract

Frequent item set mining and association rule mining is the key tasks in knowledge discovery process. Various customized algorithms are being implemented in Association Rule Mining process to find the set of frequent patterns. Though we have many algorithms apriori is one of the standard algorithm for finding frequent itemsets, but this algorithm is inefficient because of several scans of database and more number of candidates to be generated. To overcome these limitations, in this paper a new algorithm called Coalesce based Binary Table is introduced. Through this algorithm the given database is scanned only once to generate Binary Table by which frequent-1 itemsets are found. To progress the process, infrequent-1 itemsets are identified and removed from the Binary Table to rearrange the items in support ascending order. To each frequent-1 itemset find Coalesce matrix and Index List to generate all frequent itemsets having the same support count as representative items and the remaining frequent itemsets are obtained in depth first manner. The significant benefits with the proposed method are the whole database is scanned only once, no need to generate and check each candidate to find the set of frequent items. On the other hand frequent items having the same support counts as representative items can be identified directly by joining the representative item with all the combinations of Coalesce matrix. So, it is proven that coalesce based Binary Table is panacea to cut short the time in identifying the frequent itemsets hence the efficiency is improved.

Keywords: *Frequent itemset; Association Rule; Coalesce matrix; Binary Table; Index list.*

1. Introduction

Nowadays, a vast amount of data are available in science, engg., business and many other areas due to miniature of devices, rapid advances in storage technology and digitization techniques. So, the most challenging task is to extracting useful information from vast amount of data. Data mining plays a crucial role in the knowledge discovery process by finding hidden patterns, associations, constructing analytical models, performing classification and prediction, performing clustering and presenting the mining results by using visualization tools and techniques. Data mining is also called as Knowledge Discovery in Databases (KDD) why because it integrates different techniques form different disciplines such as neural networks, statistics, machine learning, database technology and information retrieval, etc. [1]. Association Rule Mining is one of the major algorithms to find hidden knowledge in transactional database.

Association Rule Mining is the widely used technique for finding frequent itemsets. Discovering frequent itemsets is the first step in finding association rules. Then association rules can be directly generated from frequent itemsets. These rules must have the support and confidence greater than some predefined thresholds.

2. Related works

One of the first Algorithms for association rule mining was AgrawalImielinskiSwami (AIS) Algorithm [2]. In AIS Algorithm the candidate itemsets are generated and frequency is counted on-the-fly basis when the database is scanned. New candidate itemsets are generated by extending the large itemsets with other items in the transaction. The drawback of AIS is it makes multiple passes over the database that results in unnecessarily generating and counting too many candidates that turn out to be small.

One of best known Algorithm for finding the frequent itemsets is the Apriori Algorithm [3]. Apriori is a Breadth First Search Algorithm which uses level wise search to find out the frequent itemsets i.e frequent-k itemsets are used to generate frequent k+1 itemsets.

An Improved Apriori Algorithm (IAA) [4] based on the original Apriori Algorithm is introduced by Huan Wu et al. by using a new count based method in order to prune the candidate itemsets and uses new record generation method in order to reduce the size of the data scan.

To reduce the scanning time and to reduce the redundant generation of candidate itemsets an improved Apriori Algorithm called Transaction Reduction [5] is introduced by Jaishree Singh, Hari Ram, and Dr.J.S.Sodhi. In this method an attribute Size Of Reduction (SOT) is introduced. It reduces the scan time by cutting down the unnecessary transactions. But it has an overhead of creating new database after every generation of level wise relation.

A new Algorithm Transaction Reduction- Bit Array Matrix (TR-BAM) [6], [7] is developed by Vijayalakshmi et al. The entire database is scanned only once and then the data is represented in the form of a Bit Array Matrix. The transactions which are repeated in the database are represented by the Repetition Count (RC)

column and a new row sum is used to store the number of nonzero elements in the column. The time consumed between original Apriori and TR-BAM is greatly reduced when the value of support increases.

Two new Algorithms called CountTableFI (CTFI) and Binary CountTableFI (BCTFI) [8] are recommended by Marghny H. These Algorithms represent the transactional database in the form of a binary number and decimal number. In CTFI the original transactional data is transformed into smaller transactional data along with the information of frequent itemsets. CTFI Algorithm is based on the set and subset properties. . In BCTFI, first the original transactional database is transformed to binary data. By using this binary data the original data is transformed to decimal number.

An improved Apriori Algorithm named BE-Apriori [9] based on pruning optimization and transaction reduction is introduced by Zhuang Chen, et al. By using this improved Algorithm, the number of frequent itemsets becomes much less, therefore a significant reduction in running time by reducing the transaction length.

To overcome the problem found in many Apriori-like Algorithms i.e candidate set generation and test approach, a new Algorithm to find frequent patterns of length 'K' without candidate generation is Frequent Pattern growth (FP-growth) introduced by the Jiawei Han et al.[10] based on Depth First Search method. The FP-growth Algorithm, compresses the dataset using a compact data structure called as a Frequent Pattern tree (FP-tree) and directly extracts frequent patterns from an FP-tree by exploring the tree in a bottom-up fashion, which avoids costly repeated database scans. Only two database scans are enough for this Algorithm to find frequent patterns and no candidate generation is required.

There are many alternatives and extensions to the FP-growth and Apriori approaches. These two methods are finding frequent itemsets from the transactional database in horizontal format. Alternatively finding frequent itemsets can also be performed in vertical format.

A novel vertical data mining Algorithm called Diffset [11] using vertical data format was proposed, which only maintains the differences in the TId's of a candidate patterns from its generating frequent patterns i.e it avoids storing the entire TIDset for every member of a class. So, this method drastically reduces the size of memory required by orders of magnitude to store intermediate results that significantly increase the performance.

Mingjun Song, and Sanguthevar Rajasekaran recommended a novel Algorithm called Transaction Mapping Algorithm by exploring the vertical data representation for Frequent Itemsets Mining [12]. The transaction tree is used to represent all the transactions in the database. Each node in the tree has the name of the item and count that keeps track of number of transactions that contain this item.

Jie Dong, and Min Han presented an effective Algorithm named BitTableFI [13] to mine frequent itemsets, to compress the database and for quick candidate itemset support count generation a special data structure named BitTable is used horizontally and vertically. "However, the Bit Table is only focusing on candidate itemset generation and support counting issues, but it does not concentrate on any other techniques like to reduce the size of the candidate itemset and number of times scanning the database etc.

To reduce the cost of candidate generation and test [14] a new Algorithm Index-BitTableFI [15] is presented. It uses the BitTable horizontally and vertically. The Index-BitTableFI achieves good performance by computing the subsume index i.e frequent itemsets having the support count as representative items can be identified directly by its subsume index".

3. Apriori Algorithm for Finding Frequent Itemset

"Apriority Algorithm is the well-known classical Algorithm for association rule mining which is used to frequent itemset generation. To find frequent itemsets, Apriori makes multiple passes over the transactional database based on candidate set generation and test strategy. This Algorithm uses the frequent itemsets from the current level to construct candidate itemsets of next level. This Algorithm uses the frequent itemsets generated in the previous level with the length 'K' are used to construct candidate itemsets of present level with the length K+1 i.e where frequent K-itemsets from LK are used to construct candidate-K+1 itemsets of level LK+1 and these candidate itemsets support count is calculated from the transactional database. This process is repeated until no more candidate itemsets can be generated. In the first pass, it scans the entire transactional database to calculate the support count of each candidate. If the candidate support count is greater than or equal to the used defined support threshold then the candidate is placed in level wise relation L1.

To improve the efficiency of level wise relation and to reduce the search space or to reduce the number of candidates to be generated an important property called Apriori property or anti-monotone or downward closure property is used. Apriori property states that if an itemset is frequent then all nonempty subsets of a frequent itemset must also be frequent. Or if any itemset with the length K is not frequent all the supersets of that itemset with the length K+1 also cannot be frequent. So, this process deletes all the candidate-K itemsets whose subsets with the length K-1 are not frequent".

With the help of following example we can easily understand the concept of Apriori. Table I shows an instance of transactional database with nine transactions. Each transaction in a transactional database is uniquely identified with TId (Transaction Identifier). Let us consider minimum support threshold=2. Table II to Table VI shows level wise relations from L1 to L5.

TId	**ITEMS**
T1	ABCEF
T2	ACG
T3	E
T4	ACDEG
T5	ACEG
T6	E
T7	ABCEF
T8	ACD
T9	ACEG
T10	ACEG

TRANSACTIONAL DATABASE

C1		**L1**	
Itemsets	**Support**	**Itemsets**	**Support**
A	8	A	8
B	2	B	2
C	8	C	8
D	2	D	2
E	8	E	8
F	2	F	2
G	5	G	5

SET OF CANDIDATE – 1 ITEMSETS AND FREQUENT – 1 ITEMSETS IN L1

C2		**L2**	
Itemsets	**Support**	**Itemsets**	**Support**
AB	2	AB	2
AC	8	AC	8
AD	2	AD	2
AE	6	AE	6
AF	2	AF	2
AG	5	AG	5
BC	2	BC	2
BD	0	BE	2
BE	2	BF	2
BF	2	CD	2
BG	0	CE	6
CD	2	CF	2
CE	6	CG	5

CF	2	EF	2
CG	5	EG	4
DE	1		
DF	0		
DG	1		
EF	2		
EG	4		
FG	0		

SET OF CANDIDATE – 2 ITEMSETS AND FREQUENT – 2 ITEMSETS IN L2

C3		**L3**	
Itemsets	**Support**	**Itemsets**	**Support**
ABC	2	ABC	2
ABE	2	ABE	2
ABF	2	ABF	2
ACD	2	ACD	2
ACE	6	ACE	6
ACF	2	ACF	2
ACG	5	ACG	5
AEF	2	AEF	2
AEG	4	AEG	4
BCE	2	BCE	2
BCF	2	BCF	2
BCG	0	BEF	2
BEF	2	CEF	2
BEG	0	CEG	4
CEF	2		
CEG	4		

SET OF CANDIDATE – 3 ITEMSETS AND FREQUENT – 3 ITEMSETS IN L3

C4		**L4**	
Itemsets	**Support**	**Itemsets**	**Support**
ABCE	2	ABCE	2
ABCF	2	ABCF	2
ABEF	2	ABEF	2
ACEF	2	ACEF	2
ACEG	4	ACEG	4
BCEF	2	BCEF	2

SET OF CANDIDATE – 4 ITEMSETS AND FREQUENT – 4 ITEMSETS IN L4

C5		**L5**	
Itemsets	**Support**	**Itemsets**	**Support**
ABCEF	2	ABCEF	2
ABCEG	0		
ACEFG	0		

SET OF CANDIDATE – 5 ITEMSETS AND FREQUENT – 5 ITEMSETS IN L5

Drawbacks of Apriori

- ➔ When the size of the database is too large it takes more time to scan the database. So, this Algorithm is inefficient because of several scans of database.
- ➔ More number of candidate itemsets need to be generated which increases the space complexity and requires a lot of time to process.

4. Proposed Algorithm and Worked Example

To overcome these limitations, a new technique called Coalesce based Binary "Table for finding frequent patterns is introduced in this paper. The main advantage of this approach is that the number of database scans are greatly reduced, thus in turn reduces the amount of time required to find the frequent patterns. The main contributions in this paper are as follows. In this Algorithm the given transactional database is represented in the form of a Binary Table. The frequent patterns and the support count of each frequent pattern is obtained directly from the Binary Table. First the transactional database is represented in the form of 0's and 1's where 1 represents presence of an item and 0 represents absence of an item. The Binary Table contains collections of rows and columns. Each row represents one transaction in the given transactional database and each column represents one item in the given transactional database. The Binary Table is generated as shown in Table VII. Let TD be a given Transactional Database T={T1,T2,….Tp} be the set of transactions and I={I1,I2,…,Iq} be the set of items.

$$TD=(Tmn)pxq= \begin{cases} T_{mn}=1, \text{ if } I_n \in T_m \\ T_{mn}=0, \text{ if } I_n \notin T_m \end{cases}$$

Where m=1,2,3,…p and
n=1,2,….q

TId	**ITEMS**	**A**	**B**	**C**	**D**	**E**	**F**	**G**
T1	ABCEF	1	1	1	0	1	1	0
T2	ACG	1	0	1	0	0	0	1
T3	E	0	0	0	0	1	0	0
T4	ACDEG	1	0	1	1	1	0	1
T5	ACEG	1	0	1	0	1	0	1
T6	E	0	0	0	0	1	0	0
T7	ABCEF	1	1	1	0	1	1	0
T8	ACD	1	0	1	1	0	0	0
T9	ACEG	1	0	1	0	1	0	1
T10	ACEG	1	0	1	0	1	0	1

BINARY TABLE REPRESENTATION FOR THE GIVEN TRANSACTIONAL DATABASE

After representing the Binary Table the frequency or support count of every item is calculated. One row with the name Support Count (SC) is added to reflect this value to the Binary Table". Now the support count of one itemset is obtained by summing the number of nonzero elements in each column as shown in Table VIII.

$SC = \sum_{m=1}^{p} T_{mn}$ where n=1, 2, …, q and $T_{mn} > 0$

TID	**ITEMS**	**A**	**B**	**C**	**D**	**E**	**F**	**G**
T1	ABCEF	1	1	1	0	1	1	0
T2	ACG	1	0	1	0	0	0	1
T3	E	0	0	0	0	1	0	0
T4	ACDEG	1	0	1	1	1	0	1
T5	ACEG	1	0	1	0	1	0	1
T6	E	0	0	0	0	1	0	0
T7	ABCEF	1	1	1	0	1	1	0
T8	ACD	1	0	1	1	0	0	0
T9	ACEG	1	0	1	0	1	0	1
T10	ACEG	1	0	1	0	1	0	1
SC		**8**	**2**	**8**	**2**	**8**	**2**	**5**

BINARY TABLE WITH NEW ROW SUPPORT COUNT

If the transactional database contains infrequent-1 itemsets, then update the transactional database by eliminating those infrequent – 1 itemsets in order to reduce the search space and then sort the list of frequent-1 itemsets according to the support ascending order. If more than one item having the same support count, then they will be sorted according to lexicographical order. The sorted transactions are shown in Table IX.

TID	Items	Sorted Items
T1	ABCEF	BFACE
T2	ACG	GAC
T3	E	E
T4	ACDEG	DGACE
T5	ACEG	GACE
T6	E	E
T7	ABCEF	BFACE
T8	ACD	DAC
T9	ACEG	GACE
T10	ACEG	GACE

TRANSACTIONAL DATABASE ACCORDING TO SUPPORT ASCENDING ORDER OF ITEMS

To find out the frequency of each transaction add one new column as Transaction Frequency (TF) column to the Binary Table. If a transaction is repeated more than once then remove the duplicate entries from the Binary Table and represent the frequency of that transaction in the Transaction Frequency column. The Transaction Frequency column is used to maintain count of that particular transaction i.e how many times the transaction is repeating. If the transaction is not repeated then the Transaction Frequency column of that particular transaction is set to 1.

$$TF_m = \begin{cases} TF_m + 1, \text{ if } T_{sn} = T_{tn}\ (s \neq t) \\ 1 \text{ , otherwise} \end{cases}$$

Where n = 1, 2, .., q

m, s, t = 1, 2, …, p

Then the given transactional database with Transaction Frequency column is represented by Binary Table is shown in Table X.

B	D	F	G	A	C	E	TF
1	0	1	0	1	1	1	2
0	0	0	1	1	1	0	1
0	0	0	0	0	0	1	2
0	1	0	1	1	1	1	1
0	0	0	1	1	1	1	3
0	1	0	0	1	1	0	1

BINARY TABLE WITH NEW COLUMN TRANSACTION FREQUENCY

"Algorithm
Input: Transactional Database TD, min_sup value
Output: Frequent Itemsets
Step-1: Scan the Transactional Database TD once to generate set of frequent-1 itemsets from the Binary Table.
Step-2: Remove the infrequent-1 itemsets from TD.
Step-3: Sort the frequent-1 itemsets according to support ascending order as a1, a2, …, ap in TD.
Step-4: Represent the Transactional Database TD with Binary Table.
Step-5: Add Transaction Frequency column to the Binary Table to represent the number of similar transactions
Step-6: For every element index[j] of Index List do
Step-7: Index[j].item=aj
Step-8: For every element index[j] in Index List do
Step-9: Index[j].coalesce=ϕ
Step-10: candidate = ∩t∈g(index[j].item)t;
Step-11: for each i > j do
Step-12: If the value of the ith bit in candidate is set then
Step-13: index[j].coalesce← index[i].item;
Step 14: end if
Step 15: end for
Step 16: end for
Step 17: Write Index List;"
Algorithm 1: Evaluating Index List
In step-1 of Algorithm 1: first scan the Transactional Database TD to find frequent-1 itemsets. In step-2 infrequent-1 itemsets are removed form TD. In step-3 the items of TD are arranged in support ascending order. In step-4 the given TD is represented in the form of a Binary Table i.e the items corresponding to transaction t are set to 1 and the remaining items are set to 0. The Transaction Frequency column of step-5 represents the count of similar transactions. In steps 6 –7 the sorted frequent itemsets are assigned to Index List. Then the Index List is formed by steps 8-17. The Coalesce matrix is formed by intersecting the transactions whose contain index[j].item in steps 9-10. No need to calculate Coalesce matrix for last item why because we arrange the frequent-1 items in support ascending order. So, last item has the highest support count.
With the help of above Algorithm calculate coalesce matrix for every frequent-1 item one by one. For example if we take frequent item D as an example the 4th transaction is intersected with 8th transaction i.e 101111 ∩ 100110 then the result is 100110. This result contains 3 one itemsets which corresponds to fifth and sixth items forms coalesce of D, as DAC. Similarly we can form coalesces of all other items. After the completion of Algorithm-1 the representative items and those coalesces are as below i.e "finally the Index List contains (B, FACE), (D, AC), (F, ACE), (G, AC), (A, C), (C, Φ), (E, Φ)
Input: Index List, min_sup
Output: Frequent itemsets
Step-1: For each element index[j] of Index List do
Step-2: Write Out index[j].item and its support;
Step-3: if index[j].coalesce ==Φ then
Step-4: if (sup(index[j].item) > min sup) then
Step-5: Depth_First(index[j].item, t(index[j].item));
Step-6: end if
Step-7: else
Step-8: for each element s _ item ⊆index[j].coalesce do
Step-9: Write Out index[j].item ∪ s _ item and its support;
Step-10: end for
Step-11: if (sup(index[j].item) > min sup) then
Step-12: tail ← t(index[j].item)
Step-13: Depth_First(index[j].item, tail);
Step-14: for each element s _ item ⊆ index[j].coalesce do
Step-15: Depth_First(index[j].item∪s-item, tail);
Step-16: end for
Step-17: end if
Step-18: end else
Step-19: end if
Step-20: end for
Procedure Depth First (item set tail)
Step-21: if tail == Φ; then return;
Step-22: for each i ∈ tail do
Step-23: f - itemset ← itemset ∪ i;
Step-24: if sup(f – itemset) >= min sup then
Step-25: Write Out f-itemset and its support;
Step-26: tail ← tail \ i;
Step-27: Depth_First (f-itemset, tail);
Step-28: end if
Step-29: end for"

Algorithm 2: Coalesce based Binary Table

The pseudo code of Coalesce based Binary Table is shown in Algorithm2. The Step-2 of Algorithm2: writes the item and its corresponding support count. In Step-3 we find coalesce of that item is empty or not. Suppose if the coalesce of that item is empty then we check support of that item is >= minsup threshold with Step-4. If support count is greater than minsup threshold depth first extension is called in Step-5. "Suppose if the coalesce of that item is not empty and the support count is not greater than minimum support threshold then in order to form frequent itemsets the representative item is combined with every nonempty subset of its Coalesce matrix and the support of that frequent itemset is equal to the support of the representative item in Steps 8-10.
Suppose if we consider frequent item B it's support count is 2 which is equal to minimum support threshold. Coalesce matrix of B is FACE. Then the following frequent itemsets will be generated

by combining B with all the nonempty subsets of it's Coalesce matrix BF=2, BA=2, BC=2, BE=2, BFA=2, BFC=2, BFE=2, BAC=2, BAE=2, BCE=2, BFAC=2, BFAE=2, BFCE=2, BACE=2, BFACE=2. It will not be expanded any more why because the support count of B is equal to minimum support threshold. To generate these 15 frequent itemsets with the Apriori we need to generate all the candidates of the length less than or equal to 4. Then calculate the support count of these 15 frequent itemsets by scanning the entire database, so it requires more time. The support count of other representative items D and F is also equal to minimum support threshold, so the frequent itemsets generated for the representative items D and F are as below DA=2, DC=2, DAC=2, FA=2, FC=2, FE=2, FAC=2, FAE=2, FCE=2, FACE=2

Suppose if the coalesce of that item is not empty and support count of that item is greater than minimum support threshold then we call depth first search by removing the items in the Coalesce matrix. The resulting frequent itemsets will also be extended in depth first manner in Steps 11-12. The support count of any K-itemset is formed by intersecting tid's of it's (K-1) subsets in Steps 13-16.

Suppose the next representative item of Index List is G and its support count is five. Coalesce of G is AC. The frequent itemsets generated by combining G with all the nonempty subsets of it's Coalesce matrix are GA=5, GC=5, GAC=5. According to support ascending order, the set of items after G are ACE, but A and C are already coalesce of G, so the remaining item after G is E. The support count of G is greater than minimum support threshold, so it can be extended in depth first order, then GE=4 is generated. The frequent itemsets GA, GC, GAC can also be extended. By extending these the frequent itemsets to be generated are GAE=4, GCE=4, GACE=4. The remaining representative elements of Index List are A, C and E. These items can also be extended in the similar manner. Then the frequent items can be generated are as below.

D:2, DA:2, DC:2, DAC:2; F:2, FA:2, FC:2, FE:2, FAC:2, FAE:2,FCE:2, FACE:2".

5. Conclusion

In this paper, a new algorithm called Coalesce based Binary Table was proposed. The original apriori algorithm and its several variants were improved with the properties Coalesce matrix and Index List. Here the Binary Table is used horizontally and vertically to calculate the Index List and support count. With the help of Coalesce matrix and Index List can easily identify the support count of representative items and their combinations. It also achieves good performance though large number of frequent itemsets by reducing the total number of candidates to be generated. Eventually the total amount of time consumed to generate frequent itemsets lesser than the original apriori and its variants.

Reference

[1] Han.J, Kamber.M, "Data Mining: Concepts and Techniques", Morgan kaufmann Publishers, Book, 2000.

[2] R. Agrawal, T. Imielinski, A. Swami, "Mining associations between sets of items in large databases, Proceedings of the ACM SIGMOD 1993 Conference Washington DC, USA, May 1993.

[3] R. Agarwal and R. Srikant, " Fast Algorithm for mining association rules", Proceedings of the 20th international conference on very large databases , Margunkaufmann , PP. 487-499.

[4] Huan Wu, Zhigang Lu, Lin Pan, RongshengXu, "An Improved Apriori-based Algorithm for Association Rules Mining", Sixth International Conference on Fuzzy Systems and Knowledge Discovery, pp. 51-55, 2009.

[5] Jaishree Singh, Hari Ram, Dr. J.S.Sodhi, " Improving Efficiency of Apriori Algorithm Using Transaction Reduction", International Journal of Scientific and Research Publications, Vol.3, 2013.

[6] V. Vijayalakshmi, Dr. A Pethalakshmi, "Mining of Frequent Itemsets with an Enhanced Apriori Algorithm" International Journal of Computer Applications(0975-8887) Volume 81 – No. 4. November 2013.

[7] V. Vijayalakshmi, Dr. A Pethalakshmi, "An Efficient Count Based Transaction Reduction Approach For Mining Frequent Patterns", Procedia Computer Science, Vol.47, PP. 52-61, 2015.

[8] Marghny .H, Mohamed .M, and Darwieesh, " Efficient Mining Frequent Itemset Algorithms ", International Journal of Machine Learninbg and Cybernatics, Vol. 5, PP. 823-833, 2013.

[9] Zhuang Chen, Shiban Cai, Qiulin Song, and Chonglai Zhu, " An Improved Apriori Algorithm Based on Pruning Optimization and Transaction Reduction", Artificial Intelligence, Management Science and Electronic Commerce (AIMSEC), PP. 1908-19011, Aug-2011.

[10] Jiawei Han, Jianpei, and Yiwenyini, "Mininig Frequent Patterns without Candidate Generation", Proceedings of the ACM SIGMOD International Conference on Management of Data Pages , PP. 1-12, 2000.

[11] Mohammed J. Zaki and Karam Gouda, "Fast Vertical Mining using Diffsets" Proceedings of the ASM SIGKDD '03 Washiton, DC, USA, Aug-2003.

[12] Mingjun Song, and SanguthevarRajasekaran, "A transaction mapping Algorithm for frequent itemset mining ", in IEEE transactions on knowledge and Data Engg.

[13] Jie Dong, Min Han "BitTableFI: An efficient mining frequent itemsets Algorithm", Knowlwdge-Based Systems, vol.20, pp.329-335, 2007.

[14] Jaishree Singh, Hari Ram, Dr. J.S.Sodhi, " Improving Efficiency of Apriori Algorithm Using Transaction Reduction", International Journal of Scientific and Research Publications, Vol.3, 2013.

[15] Dr. Seetaiah Kilaru, Harikishore K, Sravani T, Anvesh Chowdary L, Balaji T "Review and Analysis of Promising Technologies with Respect to fifth Generation Networks", 2014 First International Conference on Networks & Soft Computing, pp.270-273,August2014.

[16] Rajesh, M., and J. M. Gnanasekar. "Congestion control in heterogeneous wireless ad hoc network using FRCC." Australian Journal of Basic and Applied Sciences 9.7 (2015): 698-702.

[17] S.V.Manikanthan and V.Rama"Optimal Performance Of Key Predistribution Protocol In Wireless Sensor Networks" International Innovative Research Journal of Engineering and Technology ,ISSN NO: 2456-1983,Vol-2,Issue –Special –March 2017.

[18] T. Padmapriya and V. Saminadan, "Inter-cell Load Balancing Technique for Multi- class Traffic in MIMO - LTE - A Networks", International Conference on Advanced Computer Science and Information Technology , Singapore, vol.3, no.8, July 2015.