IJCTA, 9(40), 2016, pp. 19-31

Frequent Itemset Generation using Reverse Mirroring Technique

R. Velmurugan* C. Jothi Venkateswaran and M. Krishnamurthy*****

Abstract : Data mining is an important tool in extracting interesting knowledge from large databases. Finding frequent itemsets and generating association rules from huge amount of database is an important task of it. The primary goal of this , is consumes less memory for storing database and minimize the execution time.

Methods: The frequent itemsets generation by the existing algorithms will have more number of scans of database and more number of candidate generations, it uses more number of intermediate tables and the duplicate transactions are not eliminated. In this paper a new technique is proposed by converting the transactions in the database into bit vectors by using counters and by the anti-mirroring concept.

Findings: Counters which are fixed to eliminate the duplicates and Anti-mirroring concept is used to reduce the storage space so that all the time, scanning and computing is reduced. The experimental results show the efficiency of the algorithm which outperforms than the existing algorithms.

Keywords : Frequent Itemsets; Association Rule; Counter; Bit vectors; Anti-Mirrors.

1. INTRODUCTION

APRIORI algorithm which is created by Agrawal and Srikant[1] to find frequent itemsets which uses breadth-first search technique for counting the support of the itemsets and generating candidate itemsets. However it scans database more number times and too many candidate itemsets generation.Partition algorithm presented by Ashok et al [2], which scan the database twice. In the first scan, a set of large itemsets are generated following divide the database into a number of non-overlapping partitions. In the second scan, counters for each of these itemsets accomplished and actual support is measured. It has drawback that consumes more time for scanning database twice.The Artificial immune systems Algorithm (AIS) was developed by Agrawal et al[3] which uses candidate generation for finding frequent itemses. It generates and counts too many candidate itemsets which is the disadvantages of this algorithm. Items in transactions represented in Matrix form to find frequent itemsets in paper[4]. This approach reduces scanning cost and execution times but it works only for nine transactions. In paper[5], The Divide and Conquer method is used to decompose the task into a set of smaller tasks for constructing and traversing of Frequent Pattern tree. It reduces the search space but it has limitations that when the data are dense. In[6], Logical operations OR, AND are used to find frequent itemsets, AND, XOR are performed to derive association rules. Though it uses Boolean ,it consumes more execution time and occupies memory. The Candidates Generation by SETM[7] algorithm based on the transaction read from the database. It uses equijoins for candidate generation and sorts them all and removes the ones that do not satisfied the minimum support. More number of candidate itemsets generation is the main drawback of this algorithm. In paper[8], there is a special data structure called Bit Table which is used to compress the database and the candidate itemsets generation. In this approach named as BitTableFi in which no pruning technique used

* Assistant Professor, Department of Computer Science, L.N. Govt. College, Ponneri, India, *vel_ram@yahoo.com*

** Dean & Head, Research Dept. of Computer Science,PresidencyCollege,Chennai, *jothivenkateswaran@yahoo.co.in*

*** Professor & Head, Department of Computer Science, K.C.G. College of Technology, Chennai, *mkrish@kcgcollege.com*

to reduce the size of candidate item sets so it takes more time to scan the database. In[9], Create Cluster table and constructing graph by scanning the database for determining the frequent itemsets and derive association rule is the primary task but it is expensive for constructing graphs. A new method is proposed in [10] to generate the Bit vector based on cluster in which read the database once and create table but it uses many tables for mining frequent itemsets. The structure of this paper organized as follows. Section 2 discusses about the background work of the paper.

2. BACKGROUND

When mining the frequent itemsets(FI) from the large set of databases, it is difficult to store the entire database in memory. The existing algorithms proposed by the various researchers to reduce space and time complexity are not appreciable. Hence, the utilization of memory space for discovering frequent itemset generation with cost effective is very much needed. In this paper, Reverse-mirroring for Association Rule mining is proposed to discover frequent itemsets efficiently. In this approach, In order to reduce space, the transactions in bitable is compressed for which the purchase pattern of transactions can be classified into two, the pattern of purchase transactions which are leading ones and zeors. In order to avoid storing pattern of purchase leading zeors, there are two variables count1,count2 introduced. The pattern of purchase leading ones only stored along with count1 which shows counts of the same transactions and count2 shows counts of complements of its transactions known as reverse mirroring.This helps to make faster scanning and to explore data efficiently and requires less memory space and consuming minimum time.

The method of frequent pattern mining is to find the complete set of frequent patterns for a given support threshold in a given transaction database. Here I = { $Item_1$, $Item_2$, $Item_3$,…, $Item_n$} be a set Items. Let each transaction T is a set items in a transactional database D. The transaction identifier TID is associated with each Transaction. Itemset has a set of items. K-itemset contains K items. The support count or frequency count means an itemsets occur number of times. If the itemsets support count satisfies given support threshold, then the Itemset I which is known as frequent itemset.

Table 1

Grocery Dataset

Transactions	*Items*
T1	Milk, Diaper, Beer
T2	Diaper, Beer
T3	Beer, Coke
T4	Milk, Beer, Breed, Coke
T5	Milk, Beer
T6	Milk, Beer, Coke
T7	Beer, Coke
T8	Diaper, Beer, Coke
T9	Milk, Diaper, Beer, Breed
T10	Milk, Breed
T11	Milk, Diaper, Breed
T12	Beer, Coke
T13	Milk, Diaper, Beer,Coke
T14	Beer, Breed
T15	Diaper, Beer, Breed
T16	Milk, Breed, Coke
T17	Diaper, Breed, Coke
T18	Milk, Beer, Breed

The table-1 is representing 18 transactions of Grocery database. Each transaction represents the different items which are purchased in the dataset.

3. CBVAR ALGORITHM

CBVAR is an algorithm for mining frequent itemsets from a large of amount of database.

Step 1: Create a table for a transaction data set

Step 2: Convert transaction data set in a table into bit vectors

Step 3: Get the minimum threshold, min_thres

Step 4: Calculate frequent item sets for 1..*m* items

(Perform logical AND operation to find frequent items of pairs of items and etc.,

Step 5: Calculate the sup_cnt for 1..*m* items in new tablesup_cnt = (Number of 1's) * (Total Number of items)

Step 6: If sup-cnt > min_sup threshold then print frequent item set

Else remove the item

Step 7: Create table for the retained items

Step 8: Repeat from step 4until there are no items in the table.

In this, First step,all the items are converted into zeros or ones. In second step, by reading the database once, a cluster table is created. Then one-frequent itemsets are extracted from it. K frequent itemsets where k is more one than one are obtained by Logical operation AND between the items in a cluster table. This method consumes less main memory requirement since it considers only a small cluster at a time and as scalable for any big database. Even though CBVAR outperforms the Apriori it fails to avoid the duplications while constructing cluster based table. The limitations of CBVARareusesof multiple tables and it generates redundant item-sets.

4. ICBVAR ALGORITHM

In this section the improved CBVAR algorithm is proposed. The cluster table is loaded to the main memory, from which the Improved Cluster Based Big Vector Association Rule (ICBVAR) algorithm can fetch the clusters. The algorithm's support count is performed on the cluster table and there is no need to scan all the transactions stored in the cluster table.

Step 1: Form a table for given transaction data set

Step 2: Convert the given transaction data set into bit vectors

$$X_{ij} = \begin{cases} 1 \text{ if } i^{th} \text{ transaction} \\ \text{list has } j^{th} \text{ item} \\ 0 \text{ otherwise} \end{cases}$$

Step 3: C_i = Count of i^{th} transaction list

For (1, 2,….*m*) item combination $1 \le m \le R$ where R be the number of itms.

$$SC[1, 2\ldots m] = \sum_{i=1}^{L}\left(\prod_{j=1}^{m} X_{ij}\right)C_i$$

Where SC-Support Count, L be number of list

$$ST[1, 2.. m] = SC[1, 2.. m]/ T.$$

Where T be the number of transactions

and ST stands for Support Threshold

Step 4: $ST = SC/N*100$

Step 5: Frequent item set are generated

Step 6: If ST > min_sup_thresholdthenprint frequent item set

Elseremove the item

Step 7: repeat from step 3 until there are no items in the table.

Limitations of ICBVAR

For example, when we have only 3-items, we have $2^3 = 8$ combinations of purchase. The following Table 1 illustrates the pattern of purchase.

Table 2

Bit Vector for 3-items

I1	*I2*	*I3*
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

The above table 2 contains 8 kinds of purchase. The first row contains 000 transactions and it is always not considered since no items are purchased.The ICBVAR method for storing the pattern of purchase will have more number of zeros. For example, when a customer purchases an item I3, the pattern of storing is 001. In order to store only one bit, we have to store 3-bits.*i.e*. The number of zeros is more in storage.

5. RMIRROR ALGORITHM

The work is to reduce the number of leading zeros stored in the memory and to reduce the scanning and the processing times.

Step 1: Form a table for given transaction dataset

Step 2: Convert the given transaction data set into bit vectors X_{ij} as follows:

$$X_{ij} = \begin{cases} 1 \text{ if } i^{th} \text{ transaction} \\ \text{list has } j^{th} \text{ item} \\ 0 \text{ otherwise} \end{cases}$$

Step 3: Checking the duplicates (that is if two or more rows are identical then replace them with single row and increment the counter with number identical rows)

Step 4: Get the minimum threshold

Introduce two variables *ct*1, *ct*2

Ct1 – counter for original transactions

Ct2 – counter for reverse mirror of original transactions(complement reflection)

Step 5: Check if each transaction has reverse-mirror

Let *t* be a transaction

If t has leading zeros then

Update *ct*2 of reverse mirror of transaction *t*.

Else update *ct*1 of original transaction of *t*.

Step 6: Calculating support count after reverse mirroring

(*i*) For each item

if (item1 = 1 AND *ct*1 > =1) then

*sct*1 = *sct*1 + *ct*1

else if (item1 = 0 AND *ct*2 > = 1) then

*sct*2 = *sct*2 + *ct*2

(*ii*) For each pair of items

if (item1 = item2 = 1 AND *ct*1 = 1) then

*sct*1 = *sct*1 + *ct*1

else if (item1 = item2 = 0 AND *ct*2 = 1) then

*sct*2 = *sct*2 + *ct*2

……

……

…….

if (item1= item2 =…. = item$_n$ = 1 AND *ct*1 = 1) then

*sct*1 = *sct*1 + *ct*1

elseif (item1 = item2 =…. = item$_n$ = 0 AND *ct*2 = 1) then

*sct*2 = *sct*2 + *ct*2

Total number of presence items = *sct*1 + *sct*2

$$\text{Sup\_cnt} = \frac{\text{Total number of presence transactions}}{\text{Total number of transaction}}$$

Step 7: If sup-cnt > MST then

Display frequent itemset

Else remove the item

Repeat from step 6 until there are no items in the table.

Step 8: Calculate for *k*-itemsets.

The Reverse-Mirroring (RMIRROR) mining algorithm is addressed, which is the enhancement of ICBVAR algorithm.In this section, it is generated a Bit table to improve the performance of discovering frequent itemsets. Bit table consists of integer in which each denotes an item. The table which is used for compressing the candidate itemsets and the database. To compress the candidate itemsets, if candidate itemset C having *i*, bit *i* element of the bit table's is denoted as one; otherwise as zero.

Each transaction as specified in the table-1 is converted into bit vectorformat. For example, the transaction T15 contains 3 items Diaper, Beer, Breed and it is represented as 01110 in bit vector format. The table-5 represents the bit table of Grocery database.

Implementation of Reverse-mirroring : Reverse-mirroring is the concept of eliminating number of zeros stored in memory and to reduce the scanning time and processing time respectively.

In the table 2 the above part contains more number of zeros. The number of ones are more in the below part. The below part can be chosen for computation, since the zeros are less in number.

The reflection of zeros in above part is reverse-mirrored as ones in below part and vice-versa. Hence the table 3 can be reduced further as given below.

Table 3

Reverse-mirrors of itemsets with counter

I1	*I2*	*I3*	*CNT*
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Total transactions = 8

Table 4

Simplified form of Table 3 with counter

I1	*I2*	*I3*	*CNT1*	*CNT2*
1	0	0	1	1
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

Total transactions = 4(CNT1) + 4(CNT2) = 8

Table 5

Bit vectors of Table 1

TID	*Milk*	*Diaper*	*Beer*	*Breed*	*Coke*	*CNT*
T3	0	0	1	0	1	1
T7	0	0	1	0	1	1
T12	0	0	1	0	1	1
T14	0	0	1	1	0	1
T17	0	1	0	1	1	1
T2	0	1	1	0	0	1
T8	0	1	1	0	1	1
T15	0	1	1	1	0	1
T10	1	0	0	1	0	1
T16	1	0	0	1	1	1
T5	1	0	1	0	0	1
T6	1	0	1	0	1	1
T18	1	0	1	1	0	1
T4	1	0	1	1	1	1
T11	1	1	0	1	0	1
T1	1	1	1	0	0	1
T13	1	1	1	0	1	1
T9	1	1	1	1	0	1

The CNT 1 is the counter for the existing bit vectors and CNT 2 is the counter for the reverse-mirrors respectively. For example, Consider the transaction 100, It shows that I1 is purchased and is existing in table 2 for 1 time. Therefore the CNT 1 value is 1. When I1 is purchased for the second time, the CNT 1 is incremented by 1 and is not be stored as a separate transaction row. Here, the duplicate transactions are eliminated. For example, when I2 and I3 are purchased, the transaction can be given as 011 as bit vectors. This transaction 011 is the reverse-mirror of 100, since the zeros are reflected as ones and vice-versa in these transactions. In such cases, the CNT2 will be used, in order to represent 011. The interesting concept over here is that, the transaction 011 is not stored as a separate row, but CNT2 is incremented by 1. If I2 and I3 are purchased for the second time, the CNT 2 is incremented by 1.

Consider Table1 has the Grocery database which contains 18-transactions. The above transactions are converted into bit vector. For example, consider T15 it contains 3 items Diaper, Beer, Breed. The items which are present are marked as 1 and the items which are absent are marked as 0.Hence, transaction T15 contains 01110 as bit vectors.For our convenience, Let us replace these real time items Milk, Diaper, Beer, Breed and Coke with A (Milk), B(Diaper), C(Beer) D(Breed), and E(Coke) respectively

The Transactions T3, T7, T12 in table 5are having same pattern of purchase, hence these three transactions are considered as single transaction and the CNT cell is updated as three instead of one in this way the duplicate transactions are removed.

Table 6

Elimination of Duplicate Transactions

TID	*A*	*B*	*C*	*D*	*E*	*CNT*
T3, T7, T12	0	0	1	0	1	3
T14	0	0	1	1	0	1
T17	0	1	0	1	1	1
T2	0	1	1	0	0	1
T8	0	1	1	0	1	1
T15	0	1	1	1	0	1
T10	1	0	0	1	0	1
T16	1	0	0	1	1	1
T5	1	0	1	0	0	1
T6	1	0	1	0	1	1
T18	1	0	1	1	0	1
T4	1	0	1	1	1	1
T11	1	1	0	1	0	1
T1	1	1	1	0	0	1
T13	1	1	1	0	1	1
T9	1	1	1	1	0	1

Now from the above table 6 we have to check for mirrors.T3, T7, T12 have T11 asanti-mirror.for example, consider transaction T17 it contains 01011 its mirror image is 10100 . Further transactions are listed in table 7.

Now in new table 8 create two counters CNT1 and CNT2 and replace the mirror transactions in the place of original transaction and update CNT1 and CNT2. T14 and T15 have no reverse-mirrors. Create reverse- mirrors for those transactions since they start with zeros and replace the created transaction in table 8 and increment only CNT2. The transactions leading T6, T18, T4, T1, T13, T9 transactions start with 1.Therefore, these transactions need not be represented as anti-mirrors.

Table 7

Mirrors and their concerned Transactions

Transactions	*Reverse Transactions*
T3,T7,T12	T11
T17	T5
T2	T16
T8	T10

Table 8

Simplified table after reverse mirroring

TID	*A*	*B*	*C*	*D*	*E*	*CNT1*	*CNT2*
T15	1	0	0	0	1	0	1
T10	1	0	0	1	0	1	1
T16	1	0	0	1	1	1	1
T5	1	0	1	0	0	1	1
T6	1	0	1	0	1	1	0
T18	1	0	1	1	0	1	0
T4	1	0	1	1	1	1	0
T14	1	1	0	0	1	0	1
T11	1	1	0	1	0	1	3
T1	1	1	1	0	0	1	0
T13	1	1	1	0	1	1	0
T9	1	1	1	1	0	1	0

Total transaction CNT1 + CNT2 = 10 + 8 = 18

In above table for example take transaction T11, since T11 is the mirror image of T3, T7 and T12 and these transactions start with zero, T11 is replaced in table 8 CNT 1 is updated with 1 and CNT2 is updated with 3.

Illustration of finding frequent itemset

In this work, reverse mirroring concept applied to finding frequent itemset which has two variable CNT1 and CNT2. CNT1 denotes counts of occurrence of original transactions and CNT2 denotes counts of reverse transitions.

$$\text{Support count for an item (SC)} = \frac{\sum \text{CNT1 if Presence of '1' of the Item} +}{\sum \text{CNT2 if Presence of '0' of the Item}}$$

Support threshold for an item = (SC) / Total number of transactions

The MST for 1-frequent itemset is 45%

Support threshold for items A(Milk), B(Diaper),C(Beer),D(Breed) and E(Coke) are:

Frequency of A (Milk) : Take CNT1 if A contains 1 and take CNT2 if A contains 0 and add them and divide by total number of transactions.

i.e. 10 + 0 = 10/18 = 55%

Calculate for remaining items BV_A = 10/18 = 55%

BV_B = 8/18 = 44%

BV_C = 14/18 = 77%

BV_D = 9/18 = 50%

BV_E = 9/18 = 50%

The support threshold of item B(Diaper) is less than 45% and hence removed from the database.

Since, the support thresholds of A(Milk), C(Beer), D(Breed) and E(Coke) are greater than are equal to 45% the frequent 1-itemsets are A(Milk), C(Beer), D(Breed) and E(Coke).

Table 9

Elimination of Duplicate Transactions

TID	*A*	*C*	*D*	*E*	*CNT1*	*CNT2*
T15	1	0	0	1	0	1
T10	1	0	1	0	1	1
T16	1	0	1	1	1	1
T5	1	1	0	0	1	1
T6	1	1	0	1	1	0
T18	1	1	1	0	1	0
T4	1	1	1	1	1	0
T14	1	0	0	1	0	1
T11	1	0	0	1	0	1
T1	1	0	0	1	0	1
T13	1	0	0	1	0	1
T9	1	0	0	1	0	1

The above table eliminates the duplicate transactions. Since T14, T11, T1, T13, T9 and T15 are similar make it as unique transaction and enter in table 10. Both counters CNT1 and CNT2 are updated accordingly. In table 9 consider transaction T10, it is similar to transaction T11. In table 10 transaction T10 is entered and CNT1 is updated as 1 + 1 = 2 and CNT2 is updated as 1 + 3 = 4.

Table 10

Simplified after reverse-mirroring

TID	*A*	*C*	*D*	*E*	*CNT1*	*CNT2*
T15	1	0	0	1	*uu*0	2
T10	1	0	1	0	2	4
T16	1	0	1	1	1	1
T5	1	1	0	0	2	1
T6	1	1	0	1	2	0
T18	1	1	1	0	2	0
T4	1	1	1	1	1	0

From the table 10 calculate the 2-frequent itemset, 3-itemset and 4-frequent itemset.

Let the support threshold be 30% for 2-itemset and it is 15% for the 3-itemset and 10% for 4-itemset. Calculate the support count for the generated sets and subsets.

Frequent 2-itemset $BV_{\{C, E\}}$ = 7/18 = 38%

$BV_{\{A, C\}}$ = 7 /18 = 38%

$$BV_{\{A, D\}} = 6/18 = 33\%$$
$$BV_{\{C, D\}} = 5/18 = 27\%$$
$$BV_{\{D, E\}} = 3/18 = 16\%$$
$$BV_{\{A, E\}} = 4/18 = 22\%$$

Items BV{C, D}, BV {D, E}, BV {A, E} does not satisfy the MST=30%, so they are eliminated.

Therefore frequent 2-itemsets will be {Milk, Beer}, {Milk, Breed} and {Beer, Coke}

Frequent 3-itemsets
$$BV_{\{A, C, E} = 3/18 = 16\%$$
$$BV_{\{A, D, E\}} = 2/18 = 11\%$$
$$BV_{\{A, C, D\}} = 3/18 = 16\%$$

BV {A, D, E} does not satisfy the MST = 15%, so it is eliminated. The frequent 3-itemsets obtained are {Milk, Beer, Breed} and {Milk, Beer, Coke}.

Table 11

Frequent Itemsets

1.	{A}, {C}, {D}, {E}
2.	{A,D}, {A,C}, {C,E}
3.	{A,C,D}, {A,C,E}

Where,
A – Milk,
C – Beer,
D – Coke
E – Breed

6. COMPARATIVE ANALYSIS-CBVAR, ICBVAR, RMIRROR

Table 12

Time complexity of CBVAR and ICBVAR, RMIRROR algorithms

S.No	*Transactions*	*Computing time (Time/sec)*		
		CBVAR	*ICBVAR*	*RMIRROR*
1.	1K	16	14.9	0.4854
2.	2K	16.32	15.198	0.68279
3.	3K	16.97	15.8059	1.466957
4.	4K	17.99	16.7543	2.251125
5.	5K	19.43	18.0946	3.035292
6.	6K	21.57	19.9041	6.291522
7.	7K	24.59	22.2926	10.453044
8.	8K	29.01	25.6365	12.31457
9.	9K	35.4	30.251	15.47609
10.	10K	44.95	36.9062	18.3429

The experiments are performed inIntel *i*5 Core processor with 4 GB RAM, with grocery dataset which consists of 10000 transactions of customers' buying behaviorand the results are reported in Table 4. Python, a high level programming language, is used for the implementation of the proposed RMIRROR, ICBVAR and CBVAR algorithm. The Groceries dataset contains 10000 records with each record representing single transaction. In each transaction, the products that are bought during that transaction is listed by comma

separated values, it means that during one transaction an anonymous customerbuys these products. Similarly, 10000 of those transactions are recorded and form the dataset. The Table-4 presents the run time performances of these algorithms.

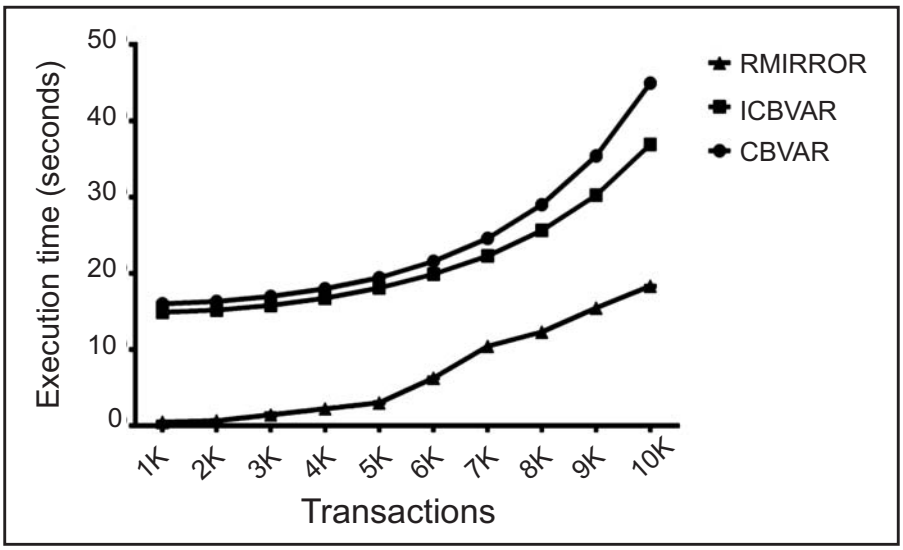

Figure 1: Time Complexity of Algorithms

Table 13

Space complexity of CBVAR and ICBVAR, RMIRRORalgorithms

S.No	*Transactions*	*Memory (KB)*		
		CBVAR	*ICBVAR*	*RMIRROR*
1.	1K	15.08	11.6	9.05
2.	2K	17.68	13.6	10.61
3.	3K	19.00	14.4	11.37
4.	4K	22.00	16.2	13.18
5.	5K	27.30	21.0	16.38
6.	6K	34.00	25.3	20.42
7.	7K	43.00	32.1	25.76
8.	8K	55.00	40.4	33.86
9.	9K	65.00	49.6	39.12
10.	10K	76.70	59.0	46.02

The Figure-1 shows the performance of execution time for different size of transactions(1000..10000) per iteration for executing the proposed RMIRROR, ICBVAR and CBVAR algorithm. As the support decreases, the size and the number of frequent itemsets increases. CBVAR has to make single passes over the database, and uses multiple tables. The time taken for executing transactions 1000..5000 by CBVAR and ICBVAR has minimum differences. The major time differences can be absorbed when the transactions size increases. The ICBVAR and RMIRROR has time differences for 7000..10000 transactions. The RMIRROR experimented in order to avoid redundancy and reduce memory space for storing. The resultant graph it can be seen that time taken for RMIRROR is considerably reduced from 44.95 to 36.9062 to 18.34290 secs for 10000 transactions with respect to CBVAR, ICBVAR and RMIRROR algorithms respectively.

The Figure-2 shows, the space occupied for storing different size of transactions(1000..10000) per iteration for executing the proposed RMIRROR, CBVAR and traditional Apriorialgorithm.It shows more differences between Apriori and CBVAR from transactions 1000..10000. The Apriori algorithm takes more memory space in order to store items as it is given but CBVAR has items which are converted into 0 ‘s and 1’s for storing.

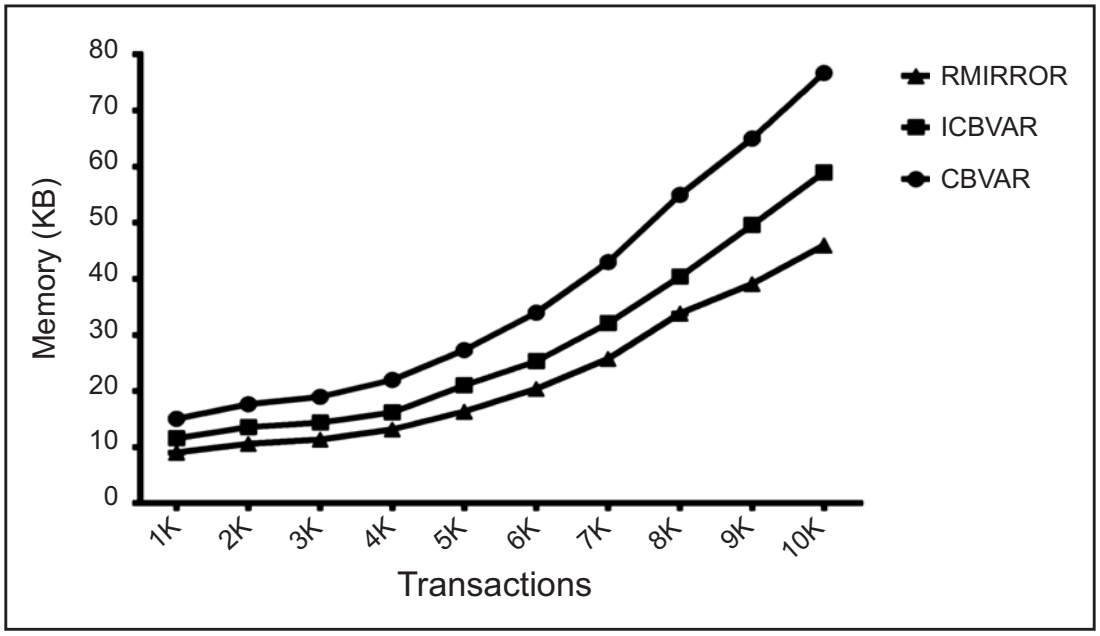

Figure 2: Space Complexity of Algorithms

RMIRROR experimented in order to avoid redundancy takes less space than ICBVAR . The major differences can be seen from transactions 7000..10000.because huge number of redundancy can occur in large transactions.

7. CONCLUSION

In this paper, new data mining algorithm namely, RMIRROR has been proposed and Implemented for perform frequent itemset mining and analyzed with CBVAR and ICBVAR. A new algorithm RMIRROR is developed which is the improvement of ICBVAR algorithm. It consumes less memory space for storing since it use reverse mirroring concept for compressing the database and reduce the number of scanning the database and redundancy.The underlying principle of Mirroring played a vital role in reduction of memory consumed. The experimental results shows Frequentitemsets can be mined efficiently, both in terms of time and space over the previous algorithms. The future work in the directions would be the use of association rules in past data to predict the future.

8. REFERENCES

1. Rakesh Agrawal, Tomasz Imielinski, Arun Swami. Miningassociation rules between sets of items in large databases,*Proceedings of the ACM SIGMOD Conference onManagement of Data*, Washington, D.C., 1993.
2. Ashok Savasere, Edward Omiecinski, ShamkantNavathe.An efficient algorithm for mining association rules in large databases. *The 21st VLDB Conference*, 1995.
3. Ramesh Agrawal, CharuAggarwal, and V.V.V. Prasad.Depth first generation of long patterns. *The 7th International Conference on Knowledge Discovery and Data Mining*, August 2000.
4. Chaohui Liu, Jiancheng an, The Software Engineering School, China “Fast Mining and Updating Frequent Itemsets”, 2008 ISECS International Colloquium on Computing, Communication, Control and Management, Vol.1, pp. 365-368.
5. Chaohui Liu, Jiancheng, and The Software Engineering School, China, “Fast Mining and Updating Frequent Itemsets”, ISECS International Colloquium on Computing, Communication, Control and Management, Vol.1, pp. 365-368, 2008.
6. Suh-Ying Wur and YunghoLeu, “An Efficient Boolean Algorithm for Mining Association Rules in Large Databases”. 6th International Conference on Database Systems for Advanced Applications, 1999, pp: 179.-186.
7. Mingju Song and SanguthevarRajasekaran. A transaction mapping for frequent itemsets mining.*IEEE transactions on Knowledge and Data Engineering*, vol.8(4) pp.472-480, 2006.

8. Jie Dong, Min Han. (2007), “BitTableFI: An Efficient Mining Frequent Itemsets Algorithm”, In Journal of Elsevier on Knowledge-Based Systems, Vol. 20, pp.329-335.

9. WaelA.Alzoubi, Azuraliza Abu Bakar, andKhairuddin Omar, “Scalable and Efficient Method for Mining Association Rules” International Conference on Electrical Engineering and Informatics,pp. 5-7 August 2009.

10 Krishnamurthy.M, Kannan.A, Baskaran.R, andKavitha.M, “Cluster based Bit Vector Mining Algorithm for Finding Frequent Itemsets in Temporal Databases”, In Journal of Elsevier on Procedia Computer Science, Vol. 3, pp: 513-523, 2001.