Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 3 (2011) 513–523

Procedia Computer Science

www.elsevier.com/locate/procedia

WCIT-2010

Cluster based bit vector mining algorithm for finding frequent itemsets in temporal databases

M. Krishnamurthy[a] *, A. Kannan[b], R. Baskaran [c], M. Kavitha[d]

[a]*Assistant Professor,Department of Computer Applications,Sri Venkateswara College of Engineering,Sriperumbudur,602105,India*
[b]*Professor,Department of Information Science and Technology,AnnaUniversity,Chennai,600025,India*
[c]*Assistant Professor,Department of Computer Science and Engineering,AnnaUniversity,Chennai,600025,India*
[d]*Final Year MCA Student,Sri Venkateswara College of Engineering,Sriperumbudur,602105,India*

Abstract

In this paper, we introduce an efficient algorithm using a new technique to find frequent itemsets from a huge set of itemsets called Cluster based Bit Vectors for Association Rule Mining (CBVAR). In this work, all the items in a transaction are converted into bits (0 or 1). A cluster is created by scanning the database only once. Then frequent 1-itemsets are extracted directly from the cluster table. Moreover, frequent k-itemsets, where k≥2 are obtained by using Logical AND between the items in a cluster table. This approach reduces main memory requirement since it considers only a small cluster at a time and as scalable for any large size of database. The overall performance of this method is significantly better than that of the previously developed algorithms for effective decision making.

Selection and/or peer-review under responsibility of the Guest Editor.

Keywords : Frequent Itemsets;Association Rule; Clustering; Cluster Table; Bit Vector; Temporal Database

1. Introduction

1.1 Data Mining

Data mining is the key step in Knowledge Discovery (KDD) process. It is increasingly becoming important tool in extracting interesting knowledge from large databases. Moreover, many data mining problems involve temporal aspects, with examples ranging from engineering to scientific research, finance and medicine. Temporal data mining is an extension of data mining which deals with temporal data. In this paper, we consider temporal database for finding out frequent items.

1.2 Frequent Itemsets

Finding out frequent itemsets is an important issue in many data management systems. The discovery of association rules has been discussed in the past using two steps namely finding the frequent itemsets and generating association rules [1]. Though Apriori is the basis for all rule mining algorithms. The database in Apriori has to be repeatedly scanned and large number of candidates has to be generated which is the major limitation of this Apriori algorithm. Therefore it is necessary to propose much faster algorithms to address the above issue.

* M.Krishnamurthy. Tel.: +919444349150; fax: +91-44-27162462
E-mail address: mkrish@svce.ac.in

1877-0509
doi:10.1016/j.procs.2010.12.086

1.3 Clustering

In a given set of data items, Clustering is used to partition a data set into a set of classes such that items with similar characteristics are grouped together. Clustering helps to make faster decisions and to explore data efficiently. Moreover, the process of creating clusters is iterative, which is time consuming and requires more space. Hence clustering along with bit vector can help to solve the above issues which are prominent in iterative methods.

1.4 Proposed Work

In order to reduce the temporal and space constraint as well as to address scalability, we introduce a new algorithm called Cluster based Bit Vector Association Rule Mining algorithm (CBVAR) to perform temporal mining in temporal database which is basically different from all the previous algorithms.

The main advantage of this algorithm is that it is scalable with all types of databases regardless of their sizes. Moreover, it is easy to implement as it uses simple cluster and bit vector concepts. However, it is efficient since it requires less memory and time to generate frequent itemsets. This proposed algorithm occupies less space because it uses Bit Vectors instead of full data.

1.5 Organization of Paper

The remainder of this paper is organized as follows: Section 2 explain the Temporal Mining and related works, Section 3, explains the Fast Updating Frequent Itemsets Algorithm (FUFIA) and Clustering and Graph based Association Rule (CGAR). Section 4, discusses our algorithm called Clustering and Bit Vector based Association Rule Mining (CBVAR). Section 5, provides the performance of these algorithms by showing the experimental results graphically and using tables. Section 6 gives the conclusion on this work and suggests some possible future works.

2. Temporal Mining

Temporal data mining is an extension of data mining which deals with temporal data. Mining temporal data poses more challenges than mining static data.

2.1 Related Works

There are many works in the literature that discuss about Association rules, Temporal Mining and Frequent Itemsets. The Association Rule mining raised by R.Agarwal [1] is an important research in data mining field. His Apriori algorithm can discover meaningful itemsets and build association rules. However, a large number of candidate sets are generated and the database needs repeated scanning.

In order to reduce the database scanning various studies were undergone. Further studies in data mining have presented many efficient algorithms for discovering association rules. In improved Apriori algorithm [4], the mining efficiency is very unsatisfactory when memory for database is considered.

In the past, the omission of time dimension in association rule was very clearly mentioned by Banu Ozden et al[11]. Different strategies were proposed after Apriori as in FP-growth [2], which outperforms all candidate generations but still have problems in the case of no common prefixes within the data items. Temporal FP tree uses divide and conquer technique for construction and traversing of tree which is used to decompose the mining task into a set of smaller task which reduces the search space. Howver, Temporal FP Tree technique is better only when the data is dense.. Paper [12] is useful for the retailer to create its own strategy as per the requirement of time. But, the performance is very less.

In paper [7], mining frequent itemsets using Matrix reduces scanning cost and execution times, but the algorithm works only for nine transactions. The partition algorithm [8], to further improve the efficiency, it does so by reducing the number of database scans, however, considerable time is still wasted in scanning infrequent candidate itemsets. In [14], the proposed Boolean algorithm mines association rules in two steps. In the first step logical OR and AND operations are used to compute frequent itemsets. In the second step, logical AND and XOR operations are applied to derive all interesting association rules based on the computed frequent itemsets. But the computational time is more as well as it occupies more memory.

Finally, Clustering and Graph based Association Rule [9] was proposed in which a cluster table is created by scanning the database and then the transactions are further clustered into clusters based on their length. Even though, the algorithm is scalable much time is wasted in constructing graphs for each cluster, which reduces the performance.

2.2 Limitation of Existing Works

In the existing works [10][11][13], though many algorithms were proposed to reduce space and efficiency a considerable disadvantage was addressed. Most of the algorithms occupied more space or generate many candidate itemsets. Comparing with the previous work our algorithm CBVAR reduces these problems and the Performance is considerably increased. Hence, the scanning cost is also reduced by using our new proposed algorithm.

2.3 Three Dimensional Itemset Matrix Experiment: Bit Vector:

Let BVi = (bv_1, bv_2,….bv_N) be Bit Vector, where N is the number of transactions in database D. If an item i is present in the transaction represent it as 1 else 0. The number of 1's shows the presence of an item in the transaction.

2.3.1 Matrix Formation:

The values of M [I_i, I_j, I_k] are determined by concurrence number of (I_i, I_j) and it is stored. The following algorithm explains to find the matrix possibilities.

Algorithm:

- For each transaction in S find its length L.
- For each value of x find the substring P=substring (S, x) where x = 1,2,....L.
- For each value of y find the substring Q=substring (S, y) where y = x+1…L.
- Increase the count of M [P, Q, 1].
- For each value of z find the substring R=substring (S, z) where z = y+1…L.
- Increase the count of M [P, Q, R].

2.3.2 Finding Frequent Itemsets:

Algorithm:

1. For every value of M [I_i, I_j, I_1], if M [I_i, I_j, I_1] ≥ Min_num then store v_1v_2 in L[a], where i=1 to N, j=i+1 to N and a is the length of the array. If count of M [I_i, I_j, I_k] ≥Min_num then append I_k, where k=j+1 else if I_i, I_j is in some one of L[a] then divide L[a] into two parts. Thus step1 finds the temporal frequent itemsets which has 2, 3, 4, etc itemsets values.

2. For every temporal item in L[] the length of the array should be greater than 3, now perform AND operation on BV_{I0}, BV_{I1}, BV_{I2} where I0, I1, I2 is the index of the first three characters of [i]. BV_{I0}, BV_{I1}, BV_{I2} are the bit vector corresponding to the three chars. The result of this operation shows the occurrence of item I0, I1, I2 in the respective transaction. The same operation should be performed with the remaining items j, where j≤N. If the number of 1's in the result is less than Min_num then divide the itemset into two.

3. Fast Updating Frequent Itemsets Algorithm (FUFIA)

In order to gain the frequent itemsets when the database or support threshold is changed, which is the updating strategy of TIMV, we know the process of gaining itemset matrix is dynamic, so we should only modify the primary itemsets matrix when the database is changed: If new transaction data are added, we should consider it as a part of primary data. So the primary itemsets matrix should be modified. The length of bit vector should be increased and the corresponding bits should be set. If we want to delete the value of M [I_i,I_j,I_k] decrease rather than increase, and lessen the length of bit vector.

3.1 Limitation:

The above algorithm occupies more space and the algorithm is having some complexity to proceed. This can be used for handling only 9 transactions.

3.2 Clustering and Graph-based Association Rule (CGAR)

Although, the cluster based Association Rule (CBAR) algorithm [10] outperforms Apriori as it scans the database only once, CGAR has been proposed using graph data structure to provide efficiency which simplifies the process of generating frequent k-itemsets, where k≥2.

CGAR scans the database only once to build a cluster table as a two-dimensional array where the columns represent items and the rows represent transaction ID's. The contents of the table are 0 and 1's where 1 is to indicate the presence of an item and 0 is to indicate the absence of an item.

Frequent 1-itemsets are determined by counting the number of 1's in the cluster table. If its threshold isn't less than the minimum threshold then it is considered further for building the graph otherwise discarded. Frequent 1-itemsets are recorded by providing a sequential number to each item in order to construct the graph. The graph is constructed by doing logical AND operation between each pair of consecutive frequent 1-itemsets $<item_i, item_j>$ | i<j, if the number of 1's is greater than or equal to minimum support threshold, a directed edge is drawn from $item_i$ to $item_j$ which is repeated for all frequent 1-itemsets. Frequent 2-itemsets are generated from the graph and it will direct to find the frequent k-itemsets such as k≥3.

3.3 Limitation

This algorithm will work for Boolean Association rules only. The graph has to be constructed for each and every cluster table which is tedious process.

4. Cluster based Bit Vector Association rule Mining (CBVAR) Algorithm

The above mentioned issues are reduced by using the combination of Clustering and Bit Vector concepts. This new algorithm is named Cluster based Bit Vector Association Rule Mining.

CBVAR scans the database of transactions only once to build the clustering table as a two dimensional array where the columns represent items and the rows represent Transaction ID's (TID). The table consists of bits (0 or 1) to indicate the presence or absence of an item. 1 indicates the presence of an item and 0 indicates the absence of an item. The cluster table now consists of bit vectors for all individual items. The number of 1's indicates the presence of an item in a transaction. The number of 1's multiplied by total number of items gives the support threshold of each item.

If the support threshold is greater than the minimum support threshold then the item is considered frequent 1-itemset. The cluster table is updated with only frequent items.

Frequent 2-itemset is determined by doing logical AND between each pair of consecutive frequent 1-itemset.

Frequent 3-itemset is determined by doing logical AND between each pair of consecutive frequent 2-itemset. Now, the cluster table is updated (i.e.) all the transactions with 2-itemsets are removed and the cluster table will consist of transactions with k≥3, where k is the total number of items. Thus after applying logical AND, the cluster table is updated till N-1 transactions where N is the total number of transactions.

We provide an example to understand the algorithm. Let the minimum threshold be 45%. There are 18 transactions with 5 different items in the transaction database shown in Table 1. All the items are first converted into bit vectors and stored as shown in table 2.

In our example, the maximum transaction length is 4 where, length indicates the number of items and hence there will be four clusters in a cluster table. Since, there is no transaction with length 1; the total number of clusters will be 3.

4.1 Proposed Algorithm

Input: Temporal Database, TD

Output: Frequent Itemsets

begin

Form a cluster table from given transaction database
Convert the given transaction database into bit vectors
Get the minimum threshold, min_thres
Determine the frequent 1-itemsets

```
begin
    Calculate the support count for each bit vector
     begin
         Implement i loop from 1 to NI times
         Implement j loop from 1 to NT times
           If itembit = 1
                    Increment the count of item i
          end
          Sup_count = count*NI
          if sup_count>min_thresh
                Print i as frequent 1-itemset
                Return L1
          end
          else
                Delete i
          end
     end
end
Determine frequent k-itemsets
Begin
      Lk: frequent itemset of size k
      Ck: candidate itemset of size k
      Generate candidate itemset, ck by joining lk-1with itself where k=2
      Set the minimum threshold to new threshold, new_thresh if needed
      Make logical AND (^) between each pair of lk-1 itemsets
     Calculate the support count, sup_count for each frequent-1itemset
    if
        Sup_count >min_thresh
        Return Lx
        Delete the cluster with k-1 items
    end
end
end
```

4.2 Implementation of CBVAR: Consider the following medicine table as transaction database.

Table 1: Transaction Database

TID	ITEMS	DATE	TID	ITEMS	DATE
T1	Benoquin, Dialyte ,Ibuprofen	<02,04,09>	T10	Benoquin, Nutradrops	<07,04,09>
T2	Dialyte, Ibuprofen	<02,04,09>	T11	Benoquin, Dialyte, Nutradrops	<07,04,09>
T3	Ibuprofen, Veetids	<03,04,09>	T12	Ibuprofen, Veetids	<09,04,09>
T4	Benoquin ,Ibuprofen, Nutradrops,Veetids	<03,04,09>	T13	Benoquin, Dialyte, Ibuprofen,Veetids	<11,04,09>
T5	Benoquin, Ibuprofen	<03,04,09>	T14	Ibuprofen, Nutradrops	<13,04,09>
T6	Benoquin, Ibuprofen, Veetids	<04,04,09>	T15	Dialyte, Ibuprofen, Nutradrops	<17,04,09>
T7	Ibuprofen, Veetids	<05,04,09>	T16	Benoquin, Nutradrops,Veetids	<19,04,09>
T8	Dialyte, Ibuprofen, Veetids	<06,04,09>	T17	Dialyte, Nutradrops,Veetids	<20,04,09>
T9	Benoquin, Dialyte, Ibuprofen, Nutradrops	<06,04,09>	T18	Benoquin, Ibuprofen, Nutradrops	<26,04,09>

The bit vectors for the items Benoquin, Dialyte, Ibuprofen, Nutradrops and Veetids are as follows (based on table 2):

$BV_{Benoquin}$ = 011010011010101111 $BV_{Dialyte}$ = 100000010111010011

$BV_{Ibuprofen}$ = 101101111101001111 $BV_{Nutradrops}$ = 000010100011111110

$BV_{Veetids}$ = 010101001100110101

Support threshold for an item = (Number of 1's) * (Total Number of items)

Support threshold for items Benoquin, Dialyte, Ibuprofen, Nutradrops and Veetids are:

$BV_{Benoquin}$ = 11 * 5 = 55% $BV_{Dialyte}$ = 8 * 5 = 40%

$BV_{Ibuprofen}$ = 13 * 5 = 65% $BV_{Nutradrops}$ = 9 * 5 = 45%

$BV_{Veetids}$ = 9 * 5 = 45%

For our convenience, Let us replace these real time items Benoquin, Dialyte, Ibuprofen, Nutradrops and Veetids with A,B,C,D and E respectively in tables.

The support threshold of item B(**Dialyte)** is less than 45% and hence removed from the database. Since, the support thresholds of A(**Benoquin), C(Ibuprofen), D(Nutradrops)** and **E(Veetids)** are greater than are equal to 45% the frequent 1-itemsets are A(**Benoquin), C(Ibuprofen), D(Nutradrops)** and **E(Veetids)** and is shown in Table 3.

Table:2 C1uster table for the data base in Table 1

Item / Transaction	**A**	**B**	**C**	**D**	**E**	**Date**
T2	0	1	1	0	0	<02,04,09>
T3	1	0	0	0	1	<03,04,09>
T5	1	0	1	0	0	<03,04,09>
T7	0	0	1	0	1	<05,04,09>
T10	1	0	0	1	0	<07,04,09>
T12	0	0	1	0	1	<09,04,09>
T14	0	0	1	1	0	<13,04,09>
T1	1	1	1	0	0	<02,04,09>
T6	1	0	1	0	1	<04,04,09>
T8	0	1	1	0	1	<06,04,09>
T11	1	1	0	1	0	<07,04,09>
T15	0	1	1	1	0	<17,04,09>
T16	1	0	0	1	1	<19,04,09>
T17	0	1	0	1	1	<20,04,09>
T18	1	0	1	1	0	<26,04,09>
T4	1	0	1	1	1	<03,04,09>
T9	1	1	1	1	0	<06,04,09>
T13	1	1	1	0	1	<11,04,09>

Table 3: Frequent 1-itemsets

Table 4: Table with all 2-itemsets

Item / Transaction	A	C	D	E	Date
T2	0	1	0	0	<02,04,09>
T3	1	0	0	1	<03,04,09>
T5	1	1	0	0	<03,04,09>
T7	0	1	0	1	<05,04,09>
T10	1	0	1	0	<07,04,09>
T12	0	1	0	1	<09,04,09>
T14	0	1	1	0	<13,04,09>
T1	1	1	0	0	<02,04,09>
T6	1	1	0	1	<04,04,09>
T8	0	1	0	1	<06,04,09>
T11	1	0	1	0	<07,04,09>
T15	0	1	1	0	<17,04,09>
T16	1	0	1	1	<19,04,09>
T17	0	0	1	1	<20,04,09>
T18	1	1	1	0	<26,04,09>
T4	1	1	1	1	<03,04,09>
T9	1	1	1	0	<06,04,09>
T13	1	1	0	1	<11,04,09>

Item /Transaction	{A, C}	{A, D}	{A,E}	{C, D}	{C,E}	{D,E}	Date
T2	0	0	0	0	0	0	<02,04,09>
T3	0	0	1	0	0	0	<03,04,09>
T5	1	0	0	0	0	0	<03,04,09>
T7	0	0	0	0	1	0	<05,04,09>
T10	0	1	0	0	0	0	<07,04,09>
T12	0	0	0	0	1	0	<09,04,09>
T14	0	0	0	1	0	0	<13,04,09>
T1	1	0	0	0	0	0	<02,04,09>
T6	1	0	1	0	1	0	<04,04,09>
T8	0	0	0	0	1	0	<06,04,09>
T11	0	1	0	0	0	0	<07,04,09>
T15	0	0	0	1	0	0	<17,04,09>
T16	0	1	1	0	0	1	<19,04,09>
T17	0	0	0	0	0	1	<20,04,09>
T18	1	1	0	1	0	0	<26,04,09>
T4	1	1	1	1	1	1	<03,04,09>
T9	1	1	0	1	0	0	<06,04,09>
T13	1	0	1	0	1	0	<11,04,09>

Table 5: Frequent 2-itemsets

Item /Transaction	{A, C}	{A, D}	{C, E}	Date

T2	0	0	0	<02,04,09>
T3	0	0	0	<03,04,09>
T5	1	0	0	<03,04,09>
T7	0	0	1	<05,04,09>
T10	0	1	0	<07,04,09>
T12	0	0	1	<09,04,09>
T14	0	0	0	<13,04,09>
T1	1	0	0	<02,04,09>
T6	1	0	1	<04,04,09>
T8	0	0	1	<06,04,09>
T11	0	1	0	<07,04,09>
T15	0	0	0	<17,04,09>
T16	0	1	0	<19,04,09>
T17	0	0	0	<20,04,09>
T18	1	1	0	<26,04,09>
T4	1	1	1	<03,04,09>
T9	1	1	0	<06,04,09>
T13	1	0	1	<11,04,09>

All the candidates with 2-itemsets are shown in Table 4. Frequent 2-itemsets are determined by doing Logical AND between each pair of frequent 1- itemsets. Let the threshold support be 30%. For example, let us find the Logical AND between A and C. In Transaction T2 the Logical AND of 0 and 1 is 0, in Transaction T3 the Logical AND of 1 and 0 is 0, in Transaction T5 the Logical AND of 1 and 1 is 1, and so on.
Thus, the frequent 2-itemsets will be {**Benoquin, Ibuprofen**}, {**Benoquin, Nutradrops**} and {**Ibuprofen, Veetids**} as shown in Table 5. Since, there are no transactions with 1 item on updating the table will yield the same result.

Frequent 3-itemsets are obtained by doing Logical AND between each pair of frequent 2-itemsets. Logical AND is done similarly as shown in finding 2-itemsets. Let the threshold be 15%. The frequent 3-itemsets obtained are {**Benoquin, Ibuprofen, Nutradrops**} and {**Benoquin, Ibuprofen, Veetids**}. Now the table is updated by deleting all the 2-itemsets as shown in table 6.

Table 6: Frequent 3-Itemsets

Item/ Transaction	**{A, C, D}**	**{A, C, E}**	**Date**
T1	0	0	<02,04,09>
T6	0	1	<04,04,09>
T8	0	0	<06,04,09>
T11	0	0	<07,04,09>
T15	0	0	<17,04,09>
T16	0	0	<19,04,09>
T17	0	0	<20,04,09>
T18	1	0	<26,04,09>
T4	1	1	<03,04,09>
T9	1	0	<06,04,09>
T13	0	1	<11,04,09>

Frequent 4 itemset is {**Benoquin, Ibuprofen, Nutradrops, Veetids**} whose support threshold is 5% which is less than the minimum threshold. Hence, the algorithm terminates with frequent 3-itemsets. In real time, thousands of data will be there. Hence, this algorithm reduces space and works for N transactions.

5. Experimental Results:

Figure1 shows a comparison of results of Apriori, CGAR and CBVAR algorithms for various values of minimum thresholds. From the diagram it can be seen that the time taken for CBVAR is considerably reduced. Moreover, the Space occupied by CBVAR is also very less when compared with Apriori algorithm and CGAR algorithm.

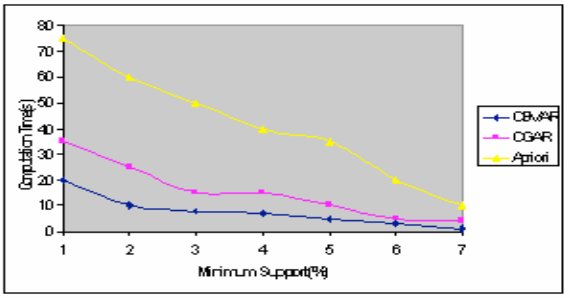

Figure 1: Comparison of Execution Times of Apriori, CGAR and CBVAR

Time complexity

Apriori algorithm

The apriori algorithm visits the lattice of itemsets in a level-wise fashion, as shown in Figure 2 and Algorithm: Apriori. Thus it is a *breadth first-search* or BFS procedure. At each level the data base is scanned to determine the support of items in the *candidate itemset Ck.*

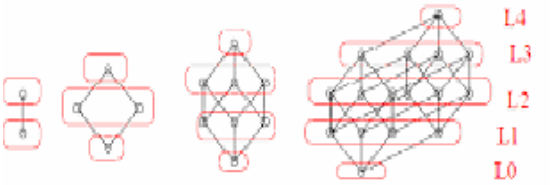

Fig .2. Apriori Algoritham

Apriori Algorithm:

$C1 = A(X)$ is the set of all one-itemsets, $k = 1$

while $Ck \neq 0$; do

scan database to determine support of all a_y with $y \in C_k$

extract frequent itemsets from C_k into L_k

generate C_{k+1}

$k := k + 1$.

end while

The major determining parameter for the complexity of the algorithm is $C = \sum_k m_k k$ where $m_k = |Ck|$.

It is often pointed out that much of the time is spent in dealing with pairs of items. We know that $m_1 = d$ as one needs to consider all single items. Furthermore, one would not have any items which alone are not frequent and so one has $m_2 = d(d\text{-}1)/2$. Thus we get the lower bound for C: $C \leq m1 + 2m_2 = d^2$.

As one sees in practice that this is a large portion of the total computations one has a good approximation $C \approx d^2$. Including the dependence on the data size we get for the time complexity of apriori: $\boldsymbol{T = O(d^2 n)}$.

Thus we have scalability in the data size but quadratic dependence on the dimension or number of attributes.
Consider the first (row-wise) storage where $T \approx d^2 n\tau$. If we have $d = 10,000$ items and $n = 1{,}000{,}000$ data records and the speed of the computations is such that $\tau = 1$ns the apriori algorithm would require 10^5 seconds which is around 30 hours, more than one day.

CBVAR

The complexity of the algorithm is $C = \sum_k m_k k$ where $m_k = \left| Ck \right|$.

We know that $m_1 = d$ as one needs to consider all single items. Since, it requires only one database scan, and also the database is updated after finding the frequent itemsets, $m_2 = d-1$. Thus we get the lower bound for C:

$C \leq m1 + 2m_2 = d.$

So, the time complexity of CBVAR is less than that of apriori algorithm which is $\boldsymbol{T = O(dn)}$.

If we have $d = 10,000$ items and $n = 1{,}000{,}000$ data records and the speed of the computations is such that $\tau = 1$ns the apriori algorithm would require 10 seconds. Thus the time spent for the algorithm is clearly considerable.

6. Conclusions and Future Works

The existing CGAR uses more space and consumes time. However in some applications it is necessary to handle large volume of data. In such situations our new algorithms provides better performance in terms of time and space complexity when it is used with temporal database for mining frequent itemises. Since our CBVAR uses only single scan, the number of database scans are reduced and hence the computation time taken is also very less. By taking synthetic data, the efficiency of the algorithm is explained theoretically and experimentally. Future work in this direction could be the use of association rules in past data to predict the future.

References

[1] R.Agarwal, T.Imielinski, and A.Swami, "Mining Association Rules Between Sets of Items in Large Databases", In Proceedings of the ACM SIGMOD Conference on Management of Data, Washington, DC, 1993, pp 207-216.
[2] Han J, Pei J, Yin Y "Mining Frequent Patterns Without Candidate Generation" Proceedings of the ACM SIGMOD International Conference on Management of Data. New York, ACM press 2000, pp. 1-12.
[3] R.Agarwal, and R.Srikant, "Fast Algorithms for Mining Association Rules", In the Proceedings of VLDB Conference 1994, pp.487-499.
[4] Li Chao, Yu Zhao-ping, "Improved Method of Apriori Algorithm based on Matrix[j]", In the Proceedings of Computer Engineering of China", 2006, Vol. 23, pp.68-69.
[5] Niu Xiao-fei, Shi Bing. "A High Efficiency Algorithm Based on Vectors and Matrix for Mining Association Rules". Chinese Journal of Computer Engineering and Application, 2004, Vol. 12, pp.170-173.
[6] F.Berzal, J.C.Cubero, N.Marin, J.M.Serrano, TBAR "An Efficient Method for Association Rule Mining in Relational Databases", In Elsevier, Data and Knowledge, Engineering Vol 37, 2001 pp. 47-64.
[7] 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.

[8] Ashok Savasere, Edward Omiecinski, and Shamkanth Navathe, "An Efficient Algorithm for Mining Association Rules in Large Databases" In VLDB 1994, Zurich, Switzerland, pp.432-443.
[9] Wael A.Alzoubi, Azuraliza Abu Bakar, Khairuddin Omar, "Scalable and Efficient Method for Mining Association Rules" 2009 International Conference on Electrical Engineering and Informatics 5-7 August 2009.
[10] Yuh-Jiuan Tsay, Jiunn-Yann Chiang, "CBAR: An Efficient Method for Mining Association Rules", Knowledge-Based Systems Vol. 18, 2005, pp.99-105.

[11] Banu Ozden, Sridhar Ramaswamy, Avi Siberschatz R: "Cyclic Association Rule", In Proceedings of Fourteenth International Conference on Data Engineering 1998, pp 412-425.
[12] Keshri Verma, O.P. Vyas, "Efficient Calendar Based Temporal Association Rule", SIGMOD Record, Vol.34, No.3, Sept.2005, pp.63-70.
[13] Muthukumar, Nadarajan, "Efficient and Scalable Partition Based Algorithm for Mining Association Rules". Academic Open Internet Journal(ISSN 1311-4360), 2006, Vol. 19.
[14] Suh-Ying Wur and Yungho Leu, "An Efficient Boolean Algorithm for Mining Association Rules in Large Databases". 6th International Conference on Database Systems for Advanced Applications, 1999, pp: 179.-186.