Abstract—Discovery of association rules from large volumes of data is an important Data Mining problem. In this paper a novel method for discovering most accurate set of frequent itemsets using partition algorithm is implemented. This method uses the concept of Reduced Pattern Count Tree, Ladder merging and Reduced Minimum Support to discover most accurate set of frequent itemsets in a single scan of database. This algorithm is a modified version of the existing Partition Algorithm, but leads to the significant reduction in time and disk input/output, and has lower memory requirements as compared to some of the Existing algorithms.

Index Terms—Frequent itemset, Path, Reduced minimum Support, Reduced Pattern Count Tree.

I. INTRODUCTION

The discovery of association rules is the most well studied problem and is an important problem in data mining. Let $I = \{i_1, i_2, i_3, \ldots, i_m\}$ be a set of items. Let $D$ be a set of transactions, where each transaction $T$ contains a set of items. A transaction $t$ is said to support an item $i_j$, if $i_j$ is present in $t$. $t$ is said to support a subset of items $X$ contained in $I$, if $t$ supports each item in $X$. An itemset $X$ contained in $I$ has a support $s$ in $D$ if $s\%$ of transactions in $D$ supports $X$. An itemset with at least the user defined minimum support is called a frequent itemset. A frequent set is said to be a maximal frequent set if it is frequent and no superset of this is a frequent. Every subset of maximal frequent itemset is frequent by downward closure property [1] of frequent. An association rule [2] is an implication of the form $X \Rightarrow Y$, where $X \subseteq I$, $Y \subseteq I$ and $X \cap Y = \phi$. The association rule $X \Rightarrow Y$ [2] holds in the database $D$ with support $s$ if $s\%$ of transactions in $D$ contains $X \cup Y$. The association rule $X \Rightarrow Y$ [2] holds in the database $D$ with confidence $c$ if $c\%$ of transactions in $D$ that contain $X$ also contain $Y$. Mining of association rules is to find all association rules that have support and confidence greater than or equal to the user-specified minimum support and minimum confidence respectively [1]. The first step in the discovery of association rules is to find all frequent itemsets with respect to the user specified minimum support. The second step in forming the association rules from the frequent itemsets is straightforward as described in[1]. There are many interesting algorithms for finding frequent itemsets. One of the key features of all algorithms is that each of these methods assumes that the underlying database size is enormous, and involves either candidate generation process or non-candidate generation process. The algorithms with candidate generation process require multiple passes over the database and are not storage efficient. The famous algorithms Apriori[2] and Partition[2] use candidate generation and pruning function at every iteration. The number of database passes in the case of Apriori depends on the largest size of the frequent itemsets. When any one of the frequent itemsets becomes longer, the algorithm has to go through much iteration and as a result the performance decreases. The partition algorithm[2], even though it involves candidate generation procedure, is considered to be better than Apriori algorithm since the size
of the global candidate set is considerably smaller than the set of all possible itemsets and the number of database scans require is equal to two.

The partition algorithm works in two phases as given in Figure 1.

A partition \( P \) of the database refers to any subset of the transactions contained in the database. A local support for an itemset is the fraction of the transaction containing that particular itemset in a partition. An itemset is said to be a local frequent itemset if its local support in a partition is at least the user defined minimum support.

In Phase I, the partition algorithm divides the whole database into a number of non-overlapping partitions, each having equal number of transactions. The partitions are considered one at a time and all local frequent itemsets of that partition are discovered and stored separately in the memory. At the end of Phase I, these frequent itemsets are merged to generate a global set of all potential frequent itemsets. Hence, only at the end of this step, local frequent itemsets of all partitions are merged and the resulting set does not contain any repeated local frequent itemset.

In Phase II, the actual support for these itemsets is generated and the frequent itemsets are identified.

### II PROPOSED ALGORITHM

The proposed algorithm uses the concept of Reduced Pattern Count tree, reduced minimum support and ladder merging. The Pattern Count tree (PC-tree)[5] is the contribution of V.S.Ananthanarayana et al, which is a complete and compact representation of the database. They discovered frequent itemsets by converting PC-tree to LOFP tree[5] in a single database scan. PC-tree[5] is a data structure, which is used to store all the patterns occurring in the tuples of a transaction database, where a count field is associated with each item in every pattern, which is responsible for a compact realization of the database. The completeness property of the PC-tree motivated us to discover all frequent itemsets using Reduced Pattern Count-tree. Reduced Pattern Count Tree is a complete representation of the frequent itemsets for a given database with respect to minimum threshold \( s \).

This method divides the whole database into a number of non-overlapping partitions, each having equal number of transactions. The partitions are read into memory one at a time. As and when a every transaction of partition is read into the memory a branch of PC-tree is constructed. At the end of every partition we have in memory a PC-tree representing all the transactions of that read partition. Based on the local support, PC-tree is reduced to contain only local frequent itemsets called Reduced Pattern Count Tree. Further, by scanning the Reduced Pattern count Tree, all local frequent itemsets of that partition are discovered along with their frequencies of occurrence. The local frequent itemsets of each partition are progressively merged with local frequent itemsets of the previous partition before another partition is processed. This step eliminates the superfluous storing of itemsets. This technique is called Ladder Merging [3]. When merging the local frequent itemsets with those of the previous partition, their counts are incremented by the number of times they occur in the current partition. In case of new frequent itemsets, their counts are simply stored along with them. After all the partitions have been processed, we have the merged set of local frequent itemsets from all partitions along with their total counts. Since this approach stores the count of each local frequent itemsets in order to save the second scan of the database and our aim is to discover frequent itemsets from the merged set using global support, we may miss some of the global frequent itemsets. To lessen this possibility, we use reduced minimum supports in the small neighborhood of global support to find most accurate set of frequent itemsets. In this regard, it is required to carry out some iteration for discovering frequent itemsets at different minimum supports starting with Global support. This process is continued until we get most accurate set of frequent itemsets.

In Figure 2, LFI = Local Frequent Itemsets. \( P_1, P_2, P_3, P_4 \) are partitions of the database \( D \) i.e. \( D = P_1 \cup P_2 \cup P_3 \cup P_4 \) and \( P_1 \cap P_2 \cap P_3 \cap P_4 = \phi \).
Algorithm
The following is the algorithm to discover frequent itemsets, which uses the concept of Reduced Pattern Count Tree, Ladder Merging and Reduced minimum support.

Input:
\[ D = \text{given database} \]
\[ D = P_1 \cup P_2 \cup P_3 \ldots \cup P_n \]
where \( P_i \), \( i = 1, 2, \ldots, n \) are partitions of \( D \) and \( P_1 \cap P_2 \cap P_3 \cap \ldots \cap P_n = \emptyset \)
\[ N = \text{total number of transactions in } D \]
\[ n = \text{number of partitions} \]
\[ L = \text{holds all local frequent itemsets of every partition} \]
\[ G = \text{holds all merged local large itemsets} \]
\[ LG = \text{holds all frequent itemsets of } D \]
\[ s(c) = \text{actual support of an itemset } c. \]
\[ \text{local sup} = \text{user defined support for an itemset to be frequent in the partition}. \]
\[ \text{Global} \_\text{support} = \left( \frac{\text{Globalsupport in terms of percentage} - 0.5\% \times N}{100} \right) \]

Output: Most accurate set of frequent itemsets

Step 1: /*Partition wise Scan of the database*/
for \( i = 1 \) to \( n \) do begin
Read_in_partition \( (P_i) \).
Construct Reduced Pattern Count Tree.
Find all frequent itemsets of \( P_i \) along with their support count \( s(c) \) by scanning Reduced Pattern Count Tree and store it in \( L \).
\( G = G \cup L \). i.e. Add only those itemsets of \( L \) to \( G \), which are not in \( G \). For those itemsets present in \( G \), update their support counts.
end

Step 2: /*Find global frequent itemsets*/
freq_is = no. of frequent itemsets in \( G \)
value = \( \left\lceil (\text{Globalsupport in terms of percentage} - 0.5\%) \times N \right\rceil + 100 \)

while(Globalsupport=value) do
for \( i = 1 \) to \( \text{freq_is} \) do begin
\( L^G = \{ c \in G | s(c) = \text{Globalsupport} \} \)
end
Globalsupport = \( \left\lceil (\text{Globalsupport in terms of percentage} - 0.1\%) \times N \right\rceil + 100 \)
Answer = \( L^G \).

Construct Reduced Pattern Count Tree.
store closed frequent itemsets; fcount[k] represents the frequency of reduced path r_k; N the number of reduced paths in r and s a user defined support.

for every item I in f do begin
  Do a preorder scan of the Reduced PC-tree to get all possible paths ending with item number I.
  for all transactions heads h in the reduced PC-tree do begin
    if (h >= I) remove h from the reduced PC-tree;
    else {store count[I ] as first element, all other elements before I and element I, as next in path p } continue search for I in Reduced PC –tree until the end of the tree end.
  /*Reduce all the paths in p to contain only frequent itemset associated with I. */
  for all paths p do begin
    for all I in p do begin
      If I is not frequent, remove I from p
      else, add I to the set T.
    end
  end
  /* Finding the association amongst the elements in T. */
  for all I_j in T, j from 1 to m do begin
    for all reduced paths p do begin
      if I_j € p, put all elements after I_j and before I in r.
      count[j] = count[I_j]
    end
  end
  /*Reduce r*/
  for all paths r do begin
    for all I in r do begin
      if I is not frequent with I_j remove I from r
    end
    if r contains more than two frequent items then
      count[r ] = count[I ]
    end
  end
  /*Finding the frequency of reduced paths.*/
  for all reduced paths r_k, k from 1 to N do begin
    for all reduced paths r_l, l from 1 to N, k ≠ l do begin
      If (r_k = r_l)
      N=N-1(Delete r_l)
      count[k] = count[k] + count [l]
      fcount[k]=count[k]
      If (r_k is contained in r_l)
      count[k] = count[k] + count [l]
      fcount[k]=count[k]
    end
  end
  /* Discovery of Frequent itemsets*/
Apply downward closure property for paths resulted in the above step.

III. PERFORMANCE ANALYSIS

a) Synthetic data generator
For the performance analysis, the data is generated using synthetic data generator. Synthetic data generated is the simulation of buying patterns of customers in retail environment.

b) Parameters and data set Used

<table>
<thead>
<tr>
<th>Set</th>
<th>#D</th>
<th>#T</th>
<th>#II</th>
<th>#LI</th>
<th>#DPC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set_1</td>
<td>0.5K</td>
<td>10</td>
<td>4</td>
<td>1000</td>
<td>1000</td>
</tr>
<tr>
<td>Set_2</td>
<td>1K</td>
<td>10</td>
<td>4</td>
<td>1000</td>
<td>1000</td>
</tr>
<tr>
<td>Set_3</td>
<td>2K</td>
<td>10</td>
<td>4</td>
<td>1000</td>
<td>1000</td>
</tr>
</tbody>
</table>

Table 1: Parameters used

The above algorithm is verified for data sets containing 1/2K, 1K, 2K transactions by taking 100 transactions per partition against Global support 2% and the reduced minimum support taken are respectively 1.9%, 1.8%, 1.7%, and 1.6%.

The Table 3 shows the total number of frequent itemsets discovered by Existing Partition Algorithm, Single Scan Partition Algorithm using Reduced Minimum support (SSPARS)[4] and Reduced Pattern Count Tree Algorithm. Figure 3 Shows frequent itemsets graph for all the above three algorithms.
Table 4 gives the storage space needed for all the above three algorithms for the data sets set_1, set_2, set_3 with local support value = 2% and number of transactions taken per partition = 100. Figures 4 shows the storage space needed for all the above three algorithms. Figure 5 shows the execution time for the above algorithms.

Table 4: Storage Space Used

<table>
<thead>
<tr>
<th>Algorithms Used</th>
<th>Storage Space Used In 10^5 Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Size of the Data Sets</td>
</tr>
<tr>
<td></td>
<td>0.5K</td>
</tr>
<tr>
<td>SSPARS</td>
<td>0.58318</td>
</tr>
<tr>
<td>Existing Partition Algorithm</td>
<td>0.6751</td>
</tr>
<tr>
<td>RPCA</td>
<td>0.18348</td>
</tr>
</tbody>
</table>

The above experiment is carried out in the neighborhood of Global support = 2% and is observed from the results that this new approach is more time and space efficient than the existing Partition Algorithm and SSPARS and most accurate sets of frequent itemsets are obtained for minimum support greater than or equal to (Global support − 0.5 %).

Theoretical Comparisons with some Algorithms:

1. We know that for Apriori algorithm, the database has to be read at least \( k \) times if we have to find frequent itemsets of length \( k \). For our method, only one scan is required to discover most accurate set of frequent itemsets. This will save considerable execution time.
2. If \( N \) is the number of transactions in the given database, our method requires exactly one scan to read the \( N \) transactions and construct Reduced Pattern Count Tree. Therefore our algorithm is in
O(N), where as Apriori which depends on size of
large itemsets Lk, is in O(k*N) and Existing
Partition Algorithm is in O(2N) since it requires two
scans of the database to discover all frequent
itemsets.
3. Since our algorithm does not generate any candidate ,
it can handle very large databases also.
In view of all the above, our approach is extremely
effective in efficiently mining most frequent itemsets, and
is able to gracefully handle very low support values, even
in dense datasets.

IV CONCLUSION
We have described an algorithm, for discovering most
accurate set of frequent itemsets in databases, which is
more space and time efficient than existing SSPARS and
Partition algorithm. In addition to this our algorithm
discovers most accurate set of frequent itemsets in a single
scan of the database. Since our algorithm does not generate
any candidates, it can be used for discovering frequent
itemsets from large volumes of data.

REFERENCES
between sets of items in large databases. In Proceedings of the 1993
ACMSIGMOD International Conference on Management of Data,
Algorithm with Ladder Merging , 2nd National Conference On
Intelligent Systems and Networks (ISN-2005)
Partition Algorithm Using Reduced Minimum Support, Asian
conference on Intelligent systems and Networks, Haryana
Engineering College(AISN-2006).
“Scalable, distributed and dynamic mining of association rules using
association Rules. In proceedings of the 20th International Conference
on Very Large Databases, Santiago.
[7]Han, J., Pei, J., Yin, Y. Mining Frequent Patterns without Candidate