Mining Frequent Patterns from High Dimensional Data

Jasneet Singh
Fakultät für Informatik
Otto von Guericke Universität
Magdeburg, Germany
jasneet1@st.ovgu.de

Jayanth Varma Dantuluri
Fakultät für Informatik
Otto von Guericke Universität
Magdeburg, Germany
jayanth.dantuluri@st.ovgu.de

Muralidhar Reddy Kuluru
Fakultät für Informatik
Otto von Guericke Universität
Magdeburg, Germany
muralidhar.kuluru@st.ovgu.de

Naveeth Reddy Chitti
Fakultät für Informatik
Otto von Guericke Universität
Magdeburg, Germany
naveeth.chitti@st.ovgu.de

Shiva Kumar Sirasala
Fakultät für Informatik
Otto von Guericke Universität
Magdeburg, Germany
shiva.sirasala@st.ovgu.de

Varun Kumar Reddy Kota
Fakultät für Informatik
Otto von Guericke Universität
Magdeburg, Germany
varun.kota@st.ovgu.de

***Abstract*—Data mining is a process that utilizes specific techniques and algorithms to help turn raw data into information that can be knowledgeable and useful, which has applications in fields like Biomedical, E-commerce and many more. These techniques can help pattern formation in raw data and help develop strategies and conclusions. With the increase in the amount of information available, the features describing an object in today's world have significantly increased to describe them better. High Dimensional data have many features, so it becomes difficult to mine patterns because the patterns grow exponentially in number with increased numbers of features and are computationally expensive. Hence it becomes very challenging to mine large patterns called colossal patterns. This paper uses a pruning technique that will eliminate the insignificant features and instances and uses a depth-first search to mine frequent closed colossal patterns and then later use a statistical method to evaluate the interestingness of the patterns mined.**

I. Introduction

There are many studies on Frequent pattern mining over the years, which aim to obtain frequent patterns from the data. Agarwal et al.(1993) pointed out the association rules between things bought at a supermarket, also called Market Basket Analysis, which set the basis for Frequent Pattern Mining. It has applications such as e-commerce, image classification, activity monitoring, analyzing customer reviews, malware detection. Depending on the application, various datasets are generally defined by three characteristics: Dimensionality, Sparsity and Resolution. In this paper, we specifically deal with High Dimensional Datasets.

So as the name suggests, High Dimensional datasets have a very high number of features. It is crucial to study High Dimensional Datasets in today's world and get useful information due to their wide range of applications such as the Biomedical field, Natural Language Processing and Analysis of web access patterns. Due to a large number of dimensions, it becomes difficult to analyze and visualize the data; and organize it in a low dimensional setting which is generally referred as the Curse of Dimensionality.

It is not easy to extract patterns from High Dimensional Data due to the curse of dimensionality and limited computational power available. Moreover, the patterns increase exponentially in number with the increase in dimensionality; say we have 'm' items in a dataset, the number of patterns generated is as high as $2^m - 1$. So, the existing algorithms struggled to deal with such datasets and are time and memory inefficient.

The algorithms proposed earlier can be broadly divided into three types [9]. The first type consists of apriori like methods which utilizes candidate generation and then look for frequent patterns. This type's drawback is that they have to generate all possible candidates and then scanned repeatedly to find frequent patterns in the database. To overcome this limitation of type one, FP-Growth [1] and its variations were proposed. FP-growth mines the patterns without candidate generation and does not require to perform multiple scans over the database. The third type of algorithm mines patterns representing the data in vertical format.

With the introduction of High Dimensional datasets, mining longer sequences in patterns are of more significance than the shorter sequences. These larger patterns/ sequences are called colossal patterns [8]. One extensive problem of frequent pattern mining is that the subset of a frequent pattern is also frequent. It implies that the subpatterns of a colossal pattern are also frequent. This property, therefore, leads to a burst of a high number of frequent patterns. The use of closed colossal frequent problems can somewhat address this issue. Although, using closed colossal pattern mining with a generic mining process, the computation and the memory consumption are high.

Although we develop an algorithm that mines patterns from High Dimensional datasets, the number of frequent patterns mined can be large and complicated to interpret and not suitable for the specific application. The interestingness of the mined pattern can be subjective; i.e., it can depend on the user or the application. However, based on the statistics of the data,

it can be objective. Using a statistical measure can help to eliminate the uninteresting rules in order to reduce the number of mined patterns.

Outline of our contributions to the paper as follows:

1) We analyzed the general characteristics of High Dimensional data, Frequent closed colossal pattern mining, and carefully studied algorithms with a horizontal representation of data. The essential idea proposed here is a modified version of the PCP Miner [6].
2) Pre-processing of data includes a pruning technique to reduce the number of insignificant features and instances. This pruning technique utilizes both minimum support and minimum cardinality thresholds.
3) Use of the Depth-first search approach to mine the frequent patterns—moreover, usage of the concept of subsuming to prune the tree further.
4) Our model's comparison with the existing PCP miner algorithm over two real-time High Dimensional Datasets in terms of the time elapsed.
5) Introduction of Statistical measures, namely Kulczynski measure and Imbalance ratio to check the reliability and the interestingness of the patterns mined.

II. Background

A. Frequent Pattern

For a given set of transactions in a database, an item X is a frequent pattern only if the number of sequences or transactions containing item X in the database is greater than or equal to *minSup* [8]. Where number of sequences containing X is given as support of X (*Sup(X)*) and *minSup* is specified by the user. For example, from the sample transaction database table I, consider features I = *A,B,C,E,F,H* and let item set X=*A,C* ,Y=*B,C* and *minSup* = 3 then item set X is frequent because *Sup(X)* = 3 which is equal to *minSup* whereas item set Y is not frequent as *Sup(Y)* = 2 which is less than *minSup*.

B. Colossal Pattern

The colossal patterns are long patterns that are robust in terms of support, i.e. even with minor deduction of item from the patterns, the resulting support of the pattern is similar [8].

C. Closed Pattern

A frequent pattern is closed frequent only if the pattern does not have any superset equally frequent or more frequent than itself [9]. Closed Colossal frequent patterns are long frequent patterns with no superset having support same or greater than *minSup*. For example, consider a set of features *x,y,z* with *minSup* = 2 (Minimum support threshold) and let item sets with supports *Sup(xy)* = 2 and *Sup(xyz)* = 1 then item set *(xy)* is a closed pattern as support of its superset is not frequent as itself. In another case let *Sup(xy)* = 2 and *Sup(xyz)* = 2 the item set *(xy)* is not closed pattern because its superset *(xyz)* is equally frequent.

D. Bit wise Representation

In this subsection, we give an outline of how the set of transactions from the database represented as bit vectors [9]. Consider a sample transaction table as shown in Table I with each row as a transaction consisting features(items) ranging from *A* to *H* with a unique transaction ID, TId. A bit matrix is a matrix in which each row corresponds to a transaction and each column corresponds to an item in lexicographic order. Each row of the transaction is converted into a bit string. If the transaction has the item, the corresponding bit string is set to *1* or else *0*, as shown in Table II. Based on the minimum support *minSup* and minimum cardinality *minCard*, this matrix is pruned and used further in the algorithm.

TId	Transactions
1	A,B,C,E
2	A,C,F,H
3	A,C,E,H
4	B,C,F,H

Table I
A SAMPLE TRANSACTION DATABASE

rid	A	B	C	E	F	H	rs
1	1	1	1	1	0	0	4
2	1	0	1	0	1	1	4
3	1	0	1	1	0	1	4
4	0	1	1	0	1	1	4
cs	3	2	4	2	2	3	

Table II
BIT MATRIX OF SAMPLE TRANSACTION DATABASE

E. Bottom-up Approach and Top-down Approach

In the bottom-up approach tree begins with root node which is a null set at the first level, in the next level has m items(rows in the database) represented with unique TId's [9]. In the next level, child nodes of 1 are generated by combining row 1 with the next corresponding row. Other nodes constructed similarly, as shown in Fig.1. On the other hand, in the top-down approach, the tree begins with the largest row set, and later levels form small row sets as shown in Fig.2 [9]. Based on their advantages and disadvantages, favourable approaches are used accordingly for different applications.

F. Depth-First search

This approach explores the tree starting from the root node and traverses each branch's possible extent before backtracking [2].

III. Related Work

Frequent pattern mining has received a lot of attention since its emergence and has seen a lot of growth. In this section, we discuss some key algorithms that paved way for mining frequent patterns.

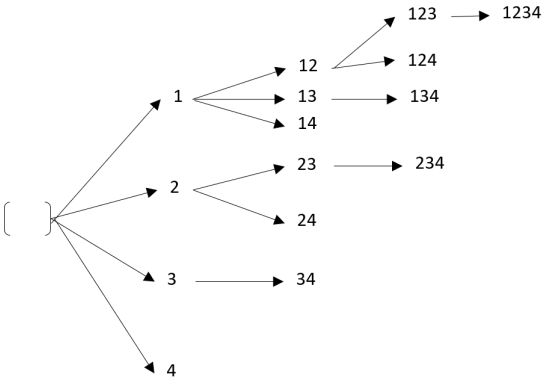

Figure 1. Bottom-Up approach Tree

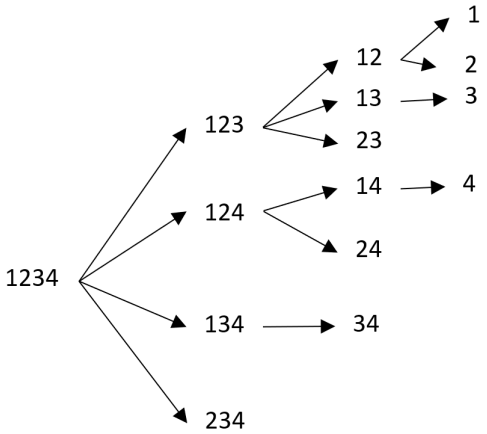

Figure 2. Top-Down approach Tree

A. Frequent Pattern Tree(FP Tree)

An extension of a normal prefix-tree structure was adopted for storing important information about frequent patterns by . Han et al. in [1]. To achieve high efficiency, the large database is compressed to avoid redundant scans. The mining task is divided into much smaller tasks to mine limited number of patterns in conditional databases and to decrease search space with a pattern growth method. This growth of the patterns is attained through the concatenation of suffix pattern with the generated frequent patterns. The smaller trees saves the cost of scanning entire database and the pattern growth evades the candidate generation algorithm which is also very costly. Performance studies states that this method has reduced search time vastly and this algorithm was extended into many other approaches. However, this method has certain limitations like incremental updates of the tree, generation of the tree for projected databases and materialization of the FP-tree.

B. CARPENTER

Pan, Feng, et al in [2] proposed this algorithm in 2003 to handle biological datasets. CARPENTER is a frequent itemset extraction algorithm primarily designed to handle datasets which have small number of transactions but a huge number of items per transaction. In the algorithm, firstly the matrix representing the dataset is transposed and a depth-first row-wise enumeration is performed. The algorithm adopts bottom-up traversal to search the enumeration space. This algorithm is not efficient with increasing number of dimensions as it impacts performance and was primarily developed for bioinformatics dataset making it unsuitable for other datasets which paved a way for further modifications.

C. Top-Down Row Enumeration Approach(TD Close)

Top-Down Row Enumeration Approach [3]employs top-down mining, blended with row-enumeration with a key idea to make use of the pruning power of minimum support threshold and reduce the size of the tree dramatically. In this algorithm, the main table is transposed and, split into sub-tables. These sub-tables are named as x-excluded transposed table where x is a rowset excluded from the table. The size of excluded table becomes smaller with the increase of excluded rid's due to *minSup* threshold. An efficient *closeness checking method* was also proposed to avoid the generation of all frequent items during mining process. A skip-row set is added to each excluded table to keep a track of the rid's which are removed from the same tuple of all its parent tables. This method is proven efficient as it reduces the search space but also has certain drawbacks like increase in the number of rows affects the run time and it is slower that the FP Growth algorithm for high values of minimum support.

D. Parallel Map-Reduce Algorithm(PaMPa-HD)

A MapReduce-based frequent closed itemset mining algorithm for high dimensional datasets and an efficient method has been proposed for parallelizing and speeding up the mining process by the authors of [4]. In order to parallelize the mining process, this algorithm splits the depth first search into a set of independent sub-processes which independently evaluate the sub-trees of the search space. Centralised version of CARPENTER algorithm is applied on each sub-processes to extract a subset of final closed itemsets which are later merged to calculate the whole itemset result. These independent sub-process make use of a distributed computing platform like Hadoop for parallel execution.

E. FI Miner using Zero Suppressed Binary Decision Diagrams(ZBDD)

The authors of the paper [5] have tried to find the interesting patterns from High Dimensional datasets by using acylic graph namely Zero Suppressed Binary Decision Diagrams(ZBDD). Binary decison diagrams are canonical directed acyclic graph with a single source code, mulitple internal nodes and sink nodes which are labelled as *1* and *0* which do not allow multiple identical nodes. The ZBDD is a extension of normal binary decision diagram with reduction rules like merging identical sub-trees and deleting nodes with whose 1-child is sink and replacing them with their 0-childs. The canonicity of the tree is used for compressing the intermediate structures. The support information is represented in the form of bitmaps

and based on the increasing frequency levels, the items are arranged in the decision diagram. This ZBDD is traversed in a top-down fashion to find the frequent itemsets. Database is induced with an empty set of frequent items with minimum support threshold. If the induced data is a sink node, then the case is terminated. An item is added into the prefix set if the support of the item in the set is not less than the minimum support, another item is added into the set and the patterns are mined.

F. Frequent Colossal Closed Itemset Mining(FCCIM)

As most of the algorithms proved to be inefficient to mine complete set of itemsets, authors in [6] have proposed a pre-processing technique and a bottom-up row enumeration method to discover significant frequent colossal itemsets. The pre-processing strategy employs bit wise representation, as it really fast and reduces the memory space. Data set is represented as bitTable where the columns(features) check for *minSup* and rows(items) check for *minCard*. If the instances and features are less than the specified minimum cardinality and minimum support respectively, then they are pruned reducing the number of insignificant instances and features. The itemsets are mined from the algorithm through depth-first traversal of enumerated space. When compared with other algorithms, this method proved to be efficient by producing a good number of itemsets but it is computationally expensive and fail at pruning the irrelevant rows and features.

G. Dynamic Switching Frequent Colossal Closed Itemset Mining(DSFCCIM)

DSFCCIM algorithm [7] was proposed recently to minimize the certain limitations of FCCI algorithms. This algorithm changes from row enumeration to column enumeration and from column enumeration to row enumeration during the process of mining based on certain traits of the data. This is the first dynamic switching algorithm to mine frequent colossal closed item sets from high dimensional datasets. Based on the bitset approach, a pre-processing technique was also proposed by making use of minimum support threshold for pruning the features that are not useful and, minimum cardinality threshold for pruning irrelevant rows resulting in fast computation. This technique is invoked in an iterative fashion until all the columns and rows in bitTable satisfy *minSup* and *minCard*. The condition to switch from one approach to another depends on the number of nodes to be traversed in the sub-tree. This algorithm chooses row enumeration if the number of different row enumerated combinations of distinct rows taken a certain number at a time are less than the number of different feature enumerated combinations. It switches to feature enumeration if this condition holds true other wise. The condition is checked at each and every enumerated node of tree. When compared with other algorithms on biological dataset, this algorithm proved to be significant.

H. Bit-wise Vertical Bottom Up Colossal Pattern Mining (BVBUC)

BVBUC algorithm [8] was proposed in 2012 for mining colossal patterns based on vertical bottom-up approach with bit wise matrix representation. This algorithm combines 1-Transactions to generate 2-transactions and so on until the the tree is expanded up to *minSup* level. Basically, the algorithm searches to minimum support level of the tree and generates the corresponding pattern of the nodes at this level and prunes their children. In this algorithm, the tree is constructed in vertical bottom-up fashion to the minimum support level and checks for the colossal patterns which is the main pruning strategy. The branches that do not reach to the minimum support level are pruned after calculating the maximum level of branches. However, the method has certain limitations (i) BVBUC generates lot of duplicates and take lot of time to check them.(ii)Uses the downward closure property to prune the items but does not remove transactions. (iii) Patterns of set transactions are computed many times making the algorithm inefficient and (iv) Finds patterns based on set of transactions by computing the intersections between them.

I. CP Miner and PCP Miner

To overcome the limitations of Bit-wise Vertical Bottom-Up Colossal Pattern Mining, CP miner and PCP miner algorithms [9] were developed in 2017 in which non-colossal patterns are pruned early before building the tree. This method uses dynamic bit-vectors, and they calculate the pattern of a set of transactions only once. These algorithms form the basis of our implementation. In this algorithm, a Colossal Pattern tree was designed to prune and compute the patterns effectively without any loss. For efficient mining of the colossal patterns, a sorting strategy was implemented to decrease the number of significant candidates and minimise the time needed to check subsets.The CP Miner algorithm [6] computes item's support and deletes items whose support value does not satisfy minimum support threshold. It also removes zero-transactions that do not contain any item after generating the bitwise matrix. The CP tree is constructed similar to the vertical bottom-up tree in the BVBUC algorithm until the *minSup* level and checked for colossal patterns in that level. In PCP Miner algorithm [6], to mine the colossal patterns efficiently, a sorting strategy is developed, which reduces the number of significant candidates and minimizes the time needed to check them. This algorithm also has limitations like (i) When the transactions are more and *minSup* is less, the PCP-Miner algorithm is computationally slow when compared with greater *minSup*. (ii) Only basic pruning is done before building the tree.

IV. Modified PCP Miner

The first step of the pre-processing is to convert the dataset into a bit matrix. The column's length in the bit matrix is equal to the total number of unique items in the dataset. Each transaction represents a unique Tid and *1s* and *0s* to represent the patterns in the bit matrix. If an item is in the transaction, then *1* is appended to the respective position, and in case if

the item is not present in the transaction, *0* is appended. For a detailed explanation, refer to background section D.

Hereon, we consider a sample transaction dataset mentioned in Table III to explain the working model of our algorithm.

Tid (rid)	Transactions
1	A, B, D, F, J
2	A, B, D, G, H
3	B, D, G, H
4	A, B, F, H, I, J
5	A, C, D, G, H, J
6	B, D, I
7	E, G
8	E, K

Table III
A TRANSACTION DATABASE (DATASET)

rid	A	B	C	D	E	F	G	H	I	J	K	rs
1	1	1	0	1	0	1	0	0	0	1	0	5
2	1	1	0	1	0	0	1	1	0	0	0	5
3	0	1	0	1	0	0	1	1	0	0	0	4
4	1	1	0	0	0	1	0	1	1	1	0	6
5	1	0	1	1	0	0	1	1	0	1	0	6
6	0	1	0	1	0	0	0	0	1	0	0	3
7	0	0	0	0	1	0	1	0	0	0	0	2
8	0	0	0	0	1	0	0	0	0	0	1	2
cs	4	5	1	5	2	2	4	4	2	3	1	

Table IV
BIT MATRIX REPRESENTATION OF THE DATASET

rid	A	B	D	E	F	G	H	I	J	rs
1	1	1	1	0	1	0	0	0	1	5
2	1	1	1	0	0	1	1	0	0	5
3	0	1	1	0	0	1	1	0	0	4
4	1	1	0	0	1	0	1	1	1	6
5	1	0	1	0	0	1	1	0	1	5
6	0	1	1	0	0	0	0	1	0	3
7	0	0	0	1	0	1	0	0	0	2
8	0	0	0	1	0	0	0	0	0	1
cs	4	5	5	2	2	4	4	2	3	

Table V
PRUNING STEP-1: MINSUP = 2

rid	A	B	D	E	F	G	H	I	J	rs
1	1	1	1	0	1	0	0	0	1	5
2	1	1	1	0	0	1	1	0	0	5
3	0	1	1	0	0	1	1	0	0	4
4	1	1	0	0	1	0	1	1	1	6
5	1	0	1	0	0	1	1	0	1	5
cs	4	4	4	0	2	3	4	1	3	

Table VI
PRUNING STEP-2: MINCARD = 4

rid	A	B	D	F	G	H	J	rs
1	1	1	1	1	0	0	1	5
2	1	1	1	0	1	1	0	5
3	0	1	1	0	1	1	0	4
4	1	1	0	1	0	1	1	6
5	1	0	1	0	1	1	1	5
cs	4	4	4	2	3	4	3	

Table VII
PRUNING STEP-3: MINSUP = 2

A. *Effective Pre-processing*

Unlike in the previous studies where only the minimum support was used to prune the unimportant features, we also consider minimum cardinality here. After the deletion of unimportant features, there can be many instances in the dataset with zero items or with a very few items which become irrelevant when mining colossal patterns. With the consideration of minimum support and minimum cardinality not only the unimportant features but also the unimportant instances are deleted. The extra row in the bit matrix represents the support information of each item in Itemset. The extra column represents the cardinality information of each transaction. Each item's support information is checked, if it is less than the minimum support, then the item is pruned from the bit matrix. Likewise, we prune every item whose support is less than minimum support. Once all the irrelevant items are pruned, then cardinality information is checked for each transaction. If the cardinality of a transaction is less than minimum cardinality, then that transaction is pruned.

From this compressed bit matrix again support information is calculated for each item. As some transaction were removed from the matrix, there is a possibility that the support of few items is reduced. So, the items that have low support value are pruned and then checked for the cardinality to remove any transactions. The same process is carried out iteratively until all the features and transactions satisfy their respective thresholds.

Consider the example shown in Table V, "*cs*" indicates support of the items and "*rs*" indicates cardinality of the transactions. Here we consider minimum support is *2* and minimum cardinality is *4* . In Table V, all the items satisfy minimum support but transaction *5, 6, 7, 8* have cardinality less than minimum cardinality, so they were pruned. Then, support of every item is recomputed as shown in Table VI. Now, items e and i have low support. So, these items are pruned. In Table VII we see that all the transactions have cardinality greater than or equal to the minimum cardinality, hence we stop the pruning process. The final table is given to the modified PCP miner algorithm.

B. *Algorithm*

The Modified PCP miner algorithm takes four arguments. The compressed bit matrix along with cardinality information, final list of colossal patterns, minimum support and minimum cardinality.

There are certain pruning rules used in this algorithm

1) In the same level of a tree if $pattern2 \subseteq pattern1$ then delete pattern2 and add it as a child of pattern1.
2) If a node of a tree cannot reach minimum support level of a tree, then prune that node.
3) If the cardinality of $pattern2 \cap pattern1$ is less than minimum cardinality, then the new pattern will not be processed further.

Algorithm 1 Modified PCP Miner Algoithm

procedure MODIFIEDPCP($Tree, CP, minSup, minCard$)
  Sort Patterns($Tree$)
  if $Tree.support == minSup$ **then**
    for Every pattern in $Tree$ **do**
      if Colossal Checking($pattern$) == $True$ **then**
        Add $pattern$ in CP
      end if
    end for
    return
  end if

  for all nodes $\mathbf{n}_A$ in $Tree$ **do**
    if $\mathbf{n}_A.support$ + length($Tree$) - A >= $minSup$ **then**
      $NewTree$ = []
      for all nodes $\mathbf{n}_B$ in $Tree$ with B>A **do**
        NewNode.$pattern$ = $\mathbf{n}_A.pattern \cap \mathbf{n}_B.pattern$
        NewNode.$support$ = $\mathbf{n}_A.support$ + 1
        if NewNode.$pattern \subseteq \mathbf{n}_B.pattern$ **then**
          Remove $\mathbf{n}_B$ from the $Tree$
          if cardinality (NewNode) >= $minCard$ **then**
            Add NewNode to $NewTree$
          end if
        end if
      end for
      MODIFIEDPCP($NewTree, CP, minSup, minCard$)

4) If a pattern is a subset of a colossal pattern then all its children can not be colossal pattern
5) If a colossal pattern is a subset of a pattern, then the colossal pattern is removed from the final colossal pattern list and this new pattern is added to the list.

The first step in the Modified PCP Miner algorithm is sorting the transactions based on their cardinality. By doing this most of the colossal patterns are found at the starting phases of the tree. Most of the patterns followed are a subset of the starting patterns, as a result, tree building will be faster. Then, the support value of the nodes of the tree is checked. If the support is at least equal to minimum support, then the colossal patterns are checked. Initially the support of each transaction is 1. For each node of tree it checks whether the node can reach till minimum support level or not. If yes, an intersection operation is performed between that node and every other node in the tree. The resulting pattern is compared with the pattern of the subsequent node. If it is a subset, then subsequent node is removed and make the new node as child of the current node. Next, increase the support of this new node by 1. Unlike the PCP miner algorithm[8], the cardinality information is also checked before adding the new pattern to the next level. By checking the cardinality of the new pattern we can eliminate the patterns that can not be colossal patterns. With this we reduce a lot of computations. With one forward pass all new nodes will be generated and stored as a tree. Then, a recursive function is called by passing this new tree as input. The recursive function will stop when the depth of the tree reaches the minimum support level. Then, the patterns are checked for colossal property. For each pattern in the tree it checks whether the pattern is a subset of the colossal pattern in the final colossal pattern list. If it is a subset then the pattern is not added to the final colossal pattern list. We also check for the other way round. If a colossal pattern is a subset of any new pattern then that colossal pattern is replaced with the new pattern. The process continues until all the nodes at the minimum support level of a tree were checked.

The algorithm takes Table III as input and a tree is built whose depth is equal to minimum support level. From Figure 3, we can observe that there are nine patterns which has reached till depth *2*. From the nine patterns we can obtain the following colossal patterns.

1) Node 1 is colossal because initially CP list is empty.
2) Node 2 is colossal as it is not a subset of Node 1.
3) Similarly Nodes 3, 4, 5, 6 are colossal patterns.
4) Node 7 will be pruned because it is a subset of Nodes 4 and 5.
5) Node 8 will be pruned as it is subset of Nodes 4 and 6.
6) Node 9 is a colossal pattern.

The final colossal patterns are Nodes 1, 2, 3, 4, 5, 6, 9. As the tree is sorted in every phase it is highly likely that most of the colossal patters be found at stating levels. And all the following patterns will most likely be subsets of the starting patterns.

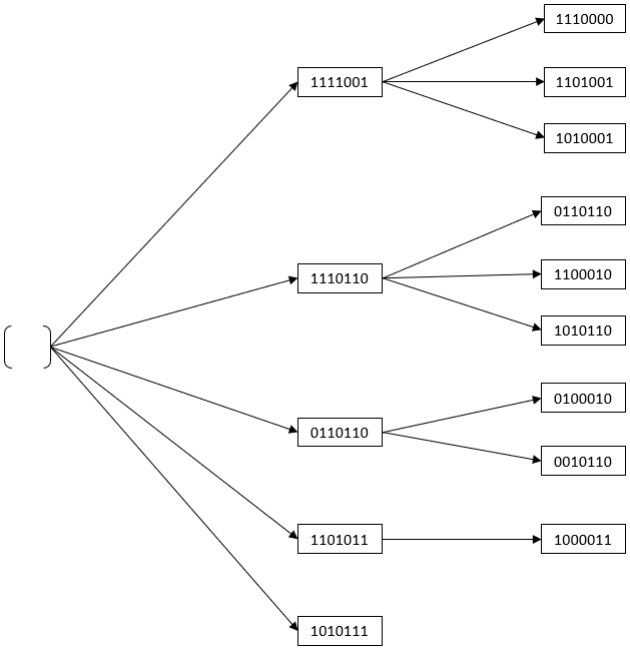

Figure 3. Modified PCP miner tree with level 2.

V. RESULTS AND DISCUSSION

In this section the Modified PCP miner's performance has been discussed. Colossal patterns generated after applying our

algorithm to the transaction dataset can be seen in Figure.3, they are [[1 1 1 0 0 0 0 0], [1 1 0 1 0 0 1], [1 0 1 0 0 0 1], [0 1 1 0 1 1 0], [1 1 0 0 0 1 0], [1 0 1 0 1 1 0], [0 1 0 0 0 1 0], [0 0 1 0 1 1 0], [1 0 0 0 0 1 1]].

Here we study the performance of our Modified PCP miner algorithm compared to the PCP miner algorithm. We have considered two real-time high dimensional datasets; Retail and USCensus. The Retail dataset comprises the data of customer transactions from an anonymous Belgian retail store which having records of *88,612* Transactions and *16,470* Items and USCensus dataset is transformed from 1990 US Census dataset with *1,000,000* Transactions and *396* Items. The transactions and item count are tabulated in Table VIII.

Datasest	Transactions	Items
Retail	88,612	16,470
USCensus	1,000,000	396

Table VIII
DATASET CHARACTERISTICS

Due to limited computational power, we have considered a specified number of transactions from both datasets. Thousand transactions considered from the Retail dataset, since it is a sparse dataset and one hundred and fifty transactions considered from the USCensus dataset, since it is a dense dataset.

The performance is analyzed based on the time taken by the algorithms with respect to different *minSup* thresholds. The *minSup* threshold is set at around one percent of the length of the dataset considered and is increased with certain step size.

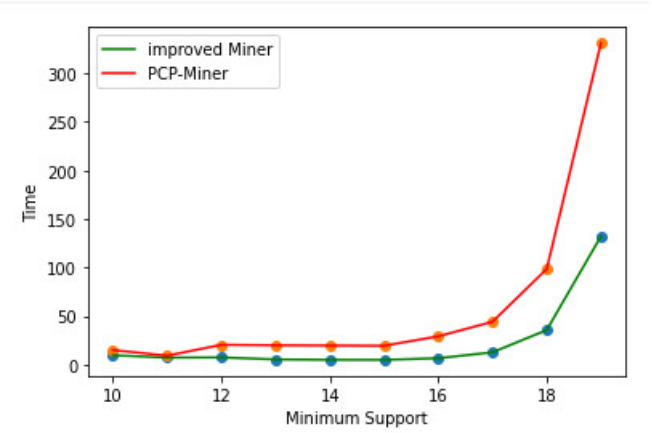

Figure 4. Retail Dataset.

Figure.4 and Figure.5 show the performances of Modified algorithm and the PCP miner algorithm on the two datasets. We can clearly observe that the modified algorithm performs better than PCP miner algorithm.

VI. EVALUATION

After the generation of the patterns, the next thing to do is check how reliable or interesting those patterns are. Usually, in many mining approaches, different association

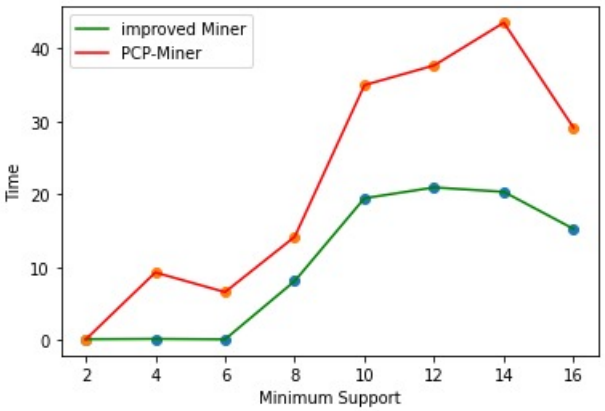

Figure 5. USCensus Dataset.

rules are generated from each pattern, and for each rule, computation of support and confidence values. This eliminates all the uninteresting rules. Though the Support-Confidence framework eliminates many rules, there could still be many unwanted rules. This problem arrives mainly when the goal is to find long patterns. So, employing the Support-Confidence framework here is not much useful. Further, in this section, an effective way of evaluating the patterns is presented in detail.

Evaluation checks the interestingness of the patterns found. Enumeration of all the patterns in the form of lists first and then transformed into the itemsets by replacing 1s in the patterns by the database's corresponding items. Each itemset generates a powerset which is a set of all the subsets possible from it. From this powerset, elimination of the empty set and the itemset itself. These subsets generate association rules, and finally, these rules checked for interestingness. Consideration of each subset of a powerset as *"P"* and the corresponding itemset - *P* considered as *"Q"*. Now, the possibility of the creation of an association rule between *P* and *Q*. Hence, many association rules can be generated by one powerset of an itemset considering one subset at a time. Elimination of all the redundant rules. i.e., suppose *P* in one association rule is the same as either *P* or *Q* of some other association rule. It is eliminated because the correlation measure value is the same if the association rule is from *P* to *Q* or the other way around. Kulczynski measure [11] is used to evaluate or check the interestingness of the patterns generated.

Kulczynski measure:

$$Kule\,(P,Q) = \frac{1}{2}\left(P\,(P/Q) + P\,(Q/P)\right)$$

Where,

$P\,(P/Q)$ is conditional probability of P given Q,
$P\,(Q/P)$ is conditional probability of Q given P

It has null-invariant property, i.e., it does not influence by the number of null-transactions in the data, which is quite common in large databases. If the kulc measure value for a rule is greater than or equal to the threshold value, then the rule can be reported as interesting. However, if the kulc measure value

is low, the rule is not eliminated directly because there are two reasons for this low score. One reason is a low correlation between the itemsets on either side of the rule, and the other reason could be due to irregularities in the data.

The pattern evaluation can sometimes be biased due to the imbalance in the data called a class imbalance. Hence, there can be interesting patterns but still not pass the evaluation because of the imbalance in the data. To tackle this problem, a measure called Imbalance Ratio (IR) [11] is calculated.

$$[IR\,(P,Q) = \frac{|Support\,(P) - Support\,(Q)\,|}{Support\,(P) + Support\,(Q) - Support\,(P \cup Q)'}]$$

If the bidirectional implication between P and Q are the same, then the IR value is zero. Otherwise, if the difference is high, the IR value is also high. If the kulc value is less than the threshold and the imbalance ratio is high, then the rule is preserved. Else if the IR value is low, then the rule is eliminated.

Studies state that the Kulczynski measure, combined with the Imbalance ratio, is the best way to check the association rules' interestingness. Example: Consider the patterns generated in section 4.

First, convert the patterns' bit-vectors into original itemsets as in the database. [['A', 'B', 'F', 'J'], ['A', 'B', 'D'], ['A', 'D', 'J'], ['A', 'D', 'G', 'H'], ['B', 'D', 'G', 'H'], ['A', 'B', 'H'], ['A', 'H', 'J']]

Generate powersets and thereby all possible association rules for each item-set or pattern. Eliminate redundant rules. Then compute the Kulczynski measure and Imbalance ratio for each rule generated.

For example consider pattern5 ['B', 'D', 'G', 'H']

Powerset= ['B'], ['D'], ['B', 'D'], ['G'], ['B', 'G'], ['D', 'G'], ['B', 'D', 'G'], ['H'], ['B', 'H'], ['D', 'H'], ['B', 'D', 'H'], ['G', 'H'], ['B', 'G', 'H'], ['D', 'G', 'H']

The threshold for Kulc measure is set as *0.75* and for Imbalance ratio the threshold is set as *0.5*

No.	P	Q	Kulc(P,Q)	IR(P,Q)	Result
1	['B']	['D', 'G', 'H']	0.533	0.333	Fail
2	['D']	['B', 'G', 'H']	0.700	0.600	Pass
3	['B', 'D']	['G', 'H']	0.583	0.200	Fail
4	['G']	['B', 'D', 'H']	0.750	0.500	Pass
5	['B', 'G']	['D', 'H']	0.833	0.333	Pass
6	['D', 'G']	['B', 'H']	0.666	0.000	Fail
7	['H']	['B', 'D', 'G']	0.750	0.500	Pass

Table IX
POSSIBLE ASSOCIATION RULES AND THEIR RESPECTIVE KULC AND IR VALUES

Similarly the evaluation is done on every pattern and interesting association rules are listed. Finally the patterns from which these interesting rules are generated considering the example dataset are ['A', 'B', 'F', 'J'] ['A', 'D', 'G', 'H'] ['B', 'D', 'G', 'H'] Hence, these patterns are frequent as well as interesting and are to be reported to the user.

VII. CONCLUSION

In this paper, we present an algorithm that finds frequent closed colossal patterns in High Dimensional Datasets. Previously developed algorithms faced difficulties in mining patterns concerning time and memory used due to high dimensionality. Our approach employs an effective pruning technique that considers both minimum support and minimum cardinality to eliminate the insignificant features and instances. Then a depth-first approach is implemented to mine the patterns. A closeness checking method is used to eliminate redundant patterns, and these closed patterns are further checked for the colossal property.

The experimental study consisted of comparing our model with a baseline model of PCP miner in terms of performance with variations in minimum support value. This experimental study conducted with two real-time High Dimensional datasets- Retail and USCensus showed that our algorithm outperformed PCP miner. Moreover, the mined patterns' interestingness is checked by Kulczynski measure along with the Imbalance Ratio and the interesting association rules are reported.

REFERENCES

[1] Han, Jiawei, Jian Pei, and Yiwen Yin. "Mining frequent patterns without candidate generation." ACM sigmod record 29.2 (2000): 1-12.

[2] Pan, Feng, et al. "Carpenter: Finding closed patterns in long biological datasets." Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining. 2003.

[3] Liu, Hongyan, et al. "Mining frequent patterns from very high dimensional data: A top-down row enumeration approach." Proceedings of the 2006 SIAM international conference on data mining. Society for Industrial and Applied Mathematics, 2006.

[4] Apiletti, Daniele, et al. "A parallel map reduce algorithm to efficiently support itemset mining on high dimensional data." Big Data Research 10 (2017): 53-69.

[5] Loekito, Elsa, and James Bailey. "Are zero-suppressed binary decision diagrams good for mining frequent patterns in high dimensional datasets ?." Proceedings of the sixth Australasian conference on data mining and analytics-Volume 70 . 2007.

[6] Vanahalli, Manjunath K., and Nagamma Patil. "Association analysis of significant frequent colossal itemsets mined from high dimensional datasets." 2016 IEEE Uttar Pradesh Section International Conference on Electrical, Computer and Electronics Engineering (UPCON). IEEE, 2016.

[7] Vanahalli, Manjunath K., and Nagamma Patil. "An efficient dynamic switching algorithm for mining colossal closed itemsets from high dimensional datasets." Data and Knowledge Engineering 123 (2019): 101721..

[8] Nguyen, Thanh-Long, Bay Vo, and Vaclav Snasel. "Efficient algorithms for mining colossal patterns in high dimensional databases." Knowledge-Based Systems 122 (2017): 75-89.

[9] Sohrabi, Mohammad Karim, and Ahmad Abdollahzadeh Barforoush. "Efficient colossal pattern mining in high dimensional datasets." Knowledge-Based Systems 33 (2012): 41-52.

[10] Philippe Fournier-Viger , Jerry Chun-Wei Lin†, Bay Vo‡§,Tin Truong Chi¶, Ji Zhangk, Hoai Bac LeA "Survey of Itemset Mining"

[11] Jiawei Han,Micheline Kamber,Jian Pei "Data Mining Concepts and Techniques(Third Edition)"

[12] Xiang Deng, Beizhan Wang, Haifang Wei and Minkui Chen "The Key Data mining models for high dimensional data".

[13] Dongmei Ai, Hongfei Pan, Xiaoxin Li, Yingxin Gao and Di He "Association rule mining algorithms on high dimensional datasets"

[14] Wei Wang and Jiong Yang "Mining High Dimensional Data"

[15] Jiawei Han, Hong Cheng, Dong Xin and Xifeng Yan "Frequent pattern mining: Current status and future directions"

[16] Tom Brijs, Koen Vanhoff "Defining interestingness of Association rule"