A BETTER APPROACH TO MINE FREQUENT ITEMSETS USING APRIORI AND FP-TREE APPROACH

Thesis submitted in partial fulfillment of the requirements for the award of degree of

Master of Engineering
in
Computer Science and Engineering

Submitted By
Bharat Gupta
(Roll No. 800932006)

Under the supervision of:

Mr. Karun Verma
Assistant Professor, CSED

Dr. Deepak Garg
Assistant Professor, CSED

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT
THAPAR UNIVERSITY
PATIALA – 147004

June 2011

CERTIFICATE

I hereby certify that the work which is being presented in the thesis entitled, "**A Better Approach to Mine Frequent Itemsets Using Apriori and FP-tree Approach"**, in partial fulfillment of the requirements for the award of degree of Master of Engineering in Computer Science and Engineering submitted in Computer Science and Engineering Department of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of Dr. Deepak Garg and Mr. Karun Verma refers other researcher's work which are duly listed in the reference section.

The matter presented in the thesis has not been submitted for award of any other degree of this or any other University.

(**Bharat Gupta**)

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.

(Mr. Karun Verma)
Computer Sc. & Engg. Department,
Thapar University
Patiala

(Dr. Deepak Garg)
Computer Sc. & Engg. Department,
Thapar University,
Patiala

Countersigned by

(Dr. Maninder Singh)
Head
Computer Sc. & Engg. Department,
Thapar University
Patiala

(Dr. S. K. Mohapatra)
Dean (Academic Affairs)
Thapar University
Patiala

ACKNOWLEDGEMENT

It is a great pleasure for me to acknowledge the guidance, assistance and help I have received from **Dr. Deepak Garg**. I am thankful for his continual support, encouragement, and invaluable suggestions. He not only provided me help whenever needed, but also the resources required to complete this thesis report on time.

I wish to express my gratitude to **Mr. Karun Verma**, Computer Science and Engineering Department, for introducing me to the problem and providing invaluable advice throughout the thesis.

I am also thankful to **Dr. Maninder Singh**, Head, Computer Science and Engineering Department for his kind help and cooperation.

I would also like to thank all the staff members of Computer Science and Engineering Department for providing me all the facilities required for the completion of my thesis work.

I would like to say thanks for support of my classmates. I want to express my appreciation to every person who contributed with either inspirational or actual work to this thesis.

I am highly grateful to my parents and brother for the inspiration and ever encouraging moral support, which enabled me to pursue my studies.

Bharat Gupta

(800932006)

ABSTRACT

As with the advancement of the IT technologies, the amount of accumulated data is also increasing. It has resulted in large amount of data stored in databases, warehouses and other repositories. Thus the Data mining comes into picture to explore and analyze the databases to extract the interesting and previously unknown patterns and rules known as association rule mining.

In data mining, Association rule mining becomes one of the important tasks of descriptive technique which can be defined as discovering meaningful patterns from large collection of data. Mining frequent itemset is very fundamental part of association rule mining.

Many algorithms have been proposed from last many decades including horizontal layout based techniques, vertical layout based techniques, and projected layout based techniques. But most of the techniques suffer from repeated database scan, Candidate generation (Apriori Algorithms), memory consumption problem (FP-tree Algorithms) and many more for mining frequent patterns.

As in retailer industry many transactional databases contain same set of transactions many times, to apply this thought, in this thesis present a new technique which is combination of present maximal Apriori (improved Apriori) and FP-tree techniques that guarantee the better performance than classical aprioi algorithm.

Another aim is to study and analyze the various existing techniques for mining frequent itemsets and evaluate the performance of new techniques and compare with the existing classical Apriori and FP- tree algorithm.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

1. Introduction

With the increase in Information Technology, the size of the databases created by the organizations due to the availability of low-cost storage and the evolution in the data capturing technologies is also increasing,. These organization sectors include retail, petroleum, telecommunications, utilities, manufacturing, transportation, credit cards, insurance, banking and many others, extracting the valuable data, it necessary to explore the databases completely and efficiently. Knowledge discovery in databases (KDD) helps to identifying precious information in such huge databases. This valuable information can help the decision maker to make accurate future decisions. KDD applications deliver measurable benefits, including reduced cost of doing business, enhanced profitability, and improved quality of service. Therefore Knowledge Discovery in Databases has become one of the most active and exciting research areas in the database community.

1.1. Data Mining

This is the important part of KDD. Data mining generally involves four classes of task; classification, clustering, regression, and association rule learning. Data mining refers to discover knowledge in huge amounts of data. It is a scientific discipline that is concerned with analyzing observational data sets with the objective of finding unsuspected relationships and produces a summary of the data in novel ways that the owner can understand and use. Data mining as a field of study involves the merging of ideas from many domains rather than a pure discipline the four main disciplines [25], which are contributing to data mining include:

- Statistics: it can provide tools for measuring significance of the given data, estimating probabilities and many other tasks (e. g. linear regression).
- Machine learning: it provides algorithms for inducing knowledge from given data (e. g. SVM).

- Data management and databases: since data mining deals with huge size of data, an efficient way of accessing and maintaining data is necessary.
- Artificial intelligence: it contributes to tasks involving knowledge encoding or search techniques (e. g. neural networks).

1.2. Data Mining Applications

Data mining has become an essential technology for businesses and researchers in many fields, the number and variety of applications has been growing gradually for several years and it is predicted that it will carry on to grow. A number of the business areas with an early embracing of DM into their processes are banking, insurance, retail and telecom. More lately it has been implemented in pharmaceutics, health, government and all sorts of e-businesses (Figure 1-1).

One describes a scheme to generate a whole set of trading strategies that take into account application constraints, for example timing, current position and pricing [24]. The authors highlight the importance of developing a suitable back testing environment that enables the gathering of sufficient evidence to convince the end users that the system can be used in practice. They use an evolutionary computation approach that favors trading models with higher stability, which is essential for success in this application domain.

Apriori algorithm is used as a recommendation engine in an E-commerce system. Based on each visitor‘s purchase history the system recommends related, potentially interesting, products. It is also used as basis for a CRM system as it allows the company itself to follow-up on customer‘s purchases and to recommend other products by e-mail [13].

A government application is proposed by [26]. The problem is connected to the management of the risk associated with social security clients in Australia. The problem is confirmed as a sequence mining task. The action ability of the model obtained is an essential concern of the authors. They concentrate on the difficult issue of performing an evaluation taking both technical and business interestingness into account.

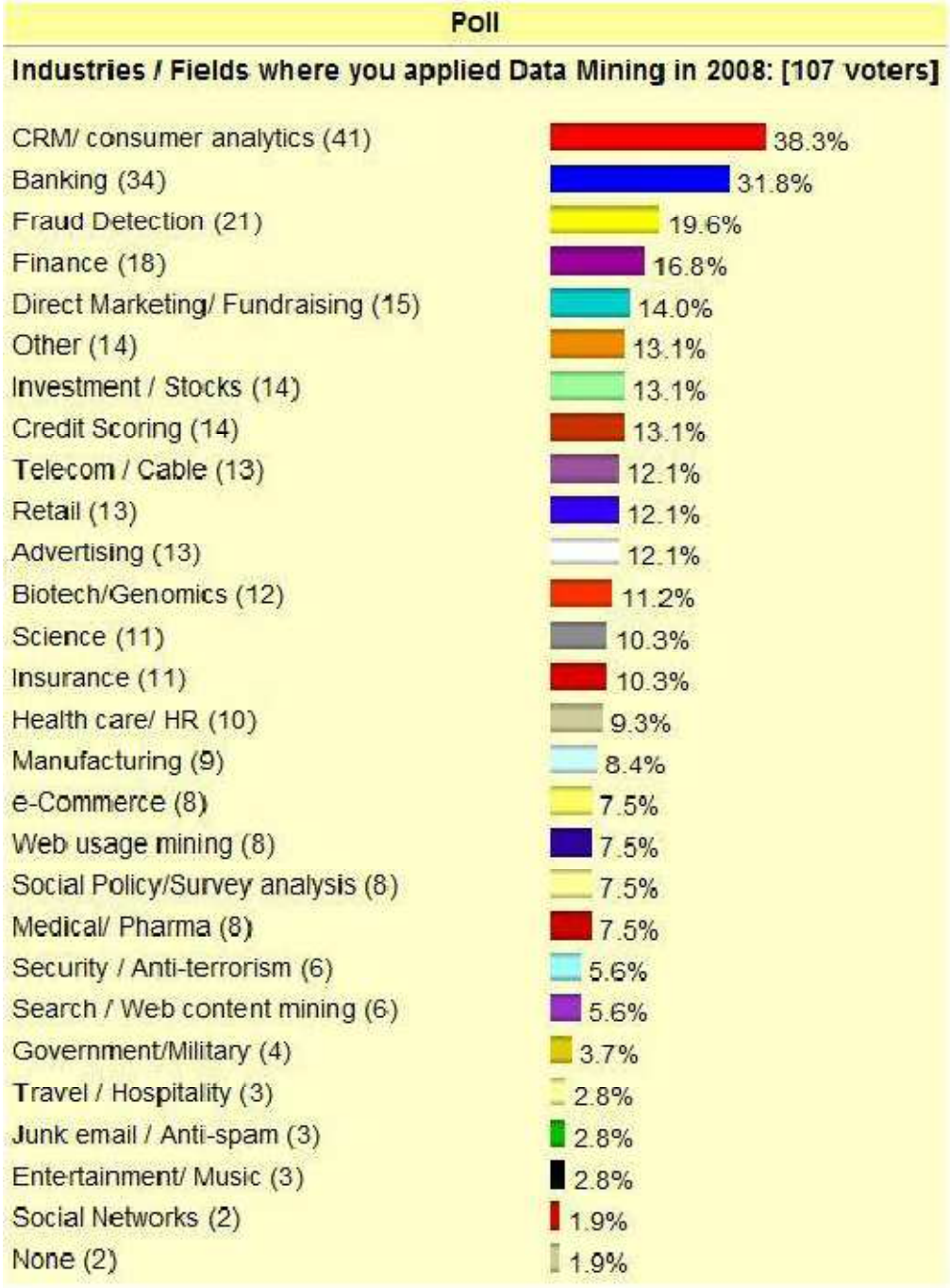


Figure 1-1: Data mining applications in 2008(http://www. kdnuggets. com).

1.3. The Primary Methods of Data Mining

Data mining addresses two basic tasks: verification and discovery. The verification task seeks to verify user's hypotheses. While the discovery task searches for unknown

knowledge hidden in the data. In general, discovery task can be further divided into two categories, which are descriptive data mining and predicative data mining.

Descriptive data mining describes the data set in a summery manner and presents interesting general properties of the data. Predictive data mining constructs one or more models to be later used for predicting the behavior of future data sets.

There are a number of algorithmic techniques available for each data mining tasks, with features that must be weighed against data characteristics and additional business requirements. Among all the techniques, in this research, we are focusing on the association rules mining technique, which is descriptive mining technique, with transactional database system. This technique was formulated by [2] and is often referred to as market-basket problem.

1.4. Introduction to Association Rule Mining

Association rules are one of the major techniques of data mining. Association rule mining finding frequent patterns, associations, correlations, or causal structures among sets of items or objects in transaction databases, relational databases, and other information repositories [13]. The volume of data is increasing dramatically as the data generated by day-to-day activities. Therefore, mining association rules from massive amount of data in the database is interested for many industries which can help in many business decision making processes, such as cross-marketing, Basket data analysis, and promotion assortment. The techniques for discovering association rules from the data have traditionally focused on identifying relationships between items telling some aspect of human behavior, usually buying behavior for determining items that customers buy together. All rules of this type describe a particular local pattern. The group of association rules can be easily interpreted and communicated.

A lot of studies have been done in the area of association rules mining. First introduced the association rules mining in [1, 2, 3]. Many studies have been conducted to address various conceptual, implementation, and application issues relating to the association rules mining task.

Researcher in application issues focuses on applying association rules to a variety of application domains. For example: Relational Databases, Data Warehouses, Transactional Databases, and Advanced Database Systems (Object-Relational, Spatial and Temporal, Time-Series, Multimedia, Text, Heterogeneous, Legacy, Distributed, and WWW) [26].

1.5. Basic Concepts

[2] Defined the problem of finding the association rules from the database. This section introduces the basic concepts of frequent pattern mining for discovery of interesting associations and correlations between itemsets in transactional and relational database. Association rule mining can be defined formally as follows:

I= {i1, i2, i3, …, in} is a set of items, such as products like (computer, CD, printer, papers, …and so on). Let DB be a set of database transactions where each transaction T is a set of items such that T⊆I. Each transaction is associated with unique identifier, transaction identifier (TID). Let X, Y be a set of items, an association rule has the form $X \rightarrow Y$, where $X \cap Y = \emptyset$. X *i*s called the antecedent and Y is called the consequent of the rule where, X, Y is a set of items is called as an **itemset** or a **pattern.** Let $freq(X)$ be the number of rows (transactions) containing X itemset in the given database. The **support** of an itemset X is defined as the fraction of all rows containing the itemset, i.e. $freq(X)/D$.

The **support** of an association rule is the support of the union of *X* and *Y, i.e.*

$$support(X \rightarrow Y) = (X \cup Y)/D$$

The **confidence** of an association rule is defined as the percentage of rows in *D* containing itemset X that also contain itemset *Y, i.e.*

$$confidence\ (X \rightarrow Y\ )\ =\ P\ (\ X\ /\ Y)\ = support(XUY)/support(X)$$

An itemset (or a pattern) is frequent if its support is equal to or more than a user specified minimum support (a statement of generality of the discovered association rules). Association rule mining is to identify all rules meeting user-specified constraints such as minimum support and minimum confidence (a statement of predictive ability of the

discovered rules). One key step of association mining is frequent itemset (pattern) mining, which is to mine all itemsets satisfying user specified minimum support. [10]

However a large number of these rules will be pruned after applying the support and confidence thresholds. Therefore the previous computations will be wasted. To avoid this problem and to improve the performance of the rule discovery algorithm, mining association rules may be decomposed into two phases:

1. Discover the large itemsets, i.e., the sets of items that have transaction support above a predetermined minimum threshold known as frequent Itemsets.
2. Use the large itemsets to generate the association rules for the database that have confidence above a predetermined minimum threshold.

The overall performance of mining association rules is determined primarily by the first step. The second step is easy. After the large itemsets are identified, the corresponding association rules can be derived in straightforward manner. Our main consideration of the thesis is First step i.e. to find the extraction of frequent itemsets.

1.6. Searching Frequent Itemsets

Frequent patterns, such as frequent itemsets, substructures, sequences term-sets, phrase-sets, and sub graphs, generally exist in real-world databases. Identifying frequent itemsets is one of the most important issues faced by the knowledge discovery and data mining community. Frequent itemset mining plays an important role in several data mining fields as association rules [1] warehousing [25], correlations, clustering of high-dimensional biological data, and classification [13]. Given a data set d that contains k items, the number of itemsets that could be generated is 2k - 1, excluding the empty set[1]. In order to searching the frequent itemsets, the support of each itemset must be computed by scanning each transaction in the dataset. A brute force approach for doing this will be computationally expensive due to the exponential number of itemsets whose support counts must be determined. There have been a lot of excellent algorithms developed for extracting frequent itemsets in very large databases. The efficiency of algorithm is linked to the size of the database which is amenable to be treated. There are two typical

strategies adopted by these algorithms: the first is an effective pruning strategy to reduce the combinatorial search space of candidate itemsets (Apriori techniques). The second strategy is to use a compressed data representation to facilitate in-core processing of the itemsets (FP-tree techniques).

1.7. Why Mining Frequent Itemsets for Association Rules

Database has been used in business management, government administration, scientific and engineering data management and many other important applications. The newly extracted information or knowledge may be applied to information management, query processing, process control, decision making and many other useful applications. With the explosive growth of data, mining information and knowledge from large databases has become one of the major challenges for data management and mining community.

The frequent itemset mining is motivated by problems such as market basket analysis [3]. A tuple in a market basket database is a set of items purchased by customer in a transaction. An association rule mined from market basket database states that if some items are purchased in transaction, then it is likely that some other items are purchased as well. Finding all such rules is valuable for guiding future sales promotions and store layout.

The problem of mining frequent itemsets are essentially, to discover all rules, from the given transactional database D that have support greater than or equal to the user specified minimum support.

1.8. Methodology

This thesis is conducted through: a review of the current status and the relevant work in the area of data mining in general and in the area of association rules in particular; analyze these works in the area of mining frequent itemsets; propose the new scheme for extracting the frequent itemsets based on hybrid approach of maximal Apriori and FP-tree algorithm that has high efficiency in term of the time and the space; validate its efficiency and seek avenues for further research.

CHAPTER 2

2. Literature Review

As frequent data itemsets mining are very important in mining the Association rules. Therefore there are various techniques are proposed for generating frequent itemsets so that association rules are mined efficiently. The approaches of generating frequent itemsets are divided into basic three techniques.

- Horizontal layout based data mining techniques
    - Apriori algorithm
    - DHP algorithm
    - Partition
    - Sample
    - A new improved Apriori algorithm
- Vertical layout based data mining techniques
    - Eclat algorithm
- Projected database based data mining techniques
    - FP-tree algorithm
    - H-mine algorithm

There are dozens of algorithms used to mine frequent itemsets. Some of them, very well known, started a whole new era in data mining. They made the concept of mining frequent itemsets and association rules possible. Others are variations that bring improvements mainly in terms of processing time. Some of the most important algorithms briefly explained in this report. The algorithms vary mainly in how the candidate itemsets are generated and how the supports for the candidate itemsets are counted.

This section will introduce some representative algorithms of mining association rules and frequent itemsets.

General steps:

1. In the first pass, the support of each individual item is counted, and the large ones are determined

2. In each subsequent pass, the large itemsets determined in the previous pass is used to generate new itemsets called candidate itemsets.

3. The support of each candidate itemset is counted, and the large ones are determined.

4. This process continues until no new large itemsets are found.

2.1. Algorithms for Mining from Horizontal Layout Database

Definition: In this each row of database represents a transaction which has a transaction identifier (TID), followed by a set of items. One example of horizontal layout dataset is shown in (Table 2-1)

Table 2-1: Horizontal Layout Based Database

TID	ITEMS
T1	I1, I2, I3, I4, I5, I6
T2	I1, I2, I4, I7
T3	I1, I2, I4, I5, I6
T4	I1, I2, I3
T5	I3, I5

2.1.1. Apriori Algorithm

The first algorithm for mining all frequent itemsets and strong association rules was the AIS algorithm by [3]. Shortly after that, the algorithm was improved and renamed Apriori. Apriori algorithm is, the most classical and important algorithm for mining frequent itemsets. Apriori is used to find all frequent itemsets in a given database DB. The key idea of Apriori algorithm is to make multiple passes over the database. It

employs an iterative approach known as a breadth-first search (level-wise search) through the search space, where k-itemsets are used to explore (k+1)-itemsets.

The working of Apriori algorithm is fairly depends upon the Apriori property which states that" All nonempty subsets of a frequent itemsets must be frequent" [2]. It also described the anti monotonic property which says if the system cannot pass the minimum support test, all its supersets will fail to pass the test [2, 3]. Therefore if the one set is infrequent then all its supersets are also frequent and vice versa. This property is used to prune the infrequent candidate elements. In the beginning, the set of frequent 1-itemsets is found. The set of that contains one item, which satisfy the support threshold, is denoted by L_1. In each subsequent pass, we begin with a seed set of itemsets found to be large in the previous pass. This seed set is used for generating new potentially large itemsets, called candidate itemsets, and count the actual support for these candidate itemsets during the pass over the data. At the end of the pass, we determine which of the candidate itemsets are actually large (frequent), and they become the seed for the next pass. Therefore, L_1 is used to find L_2, the set of frequent 2-itemsets, which is used to find L_1, and so on, until no more frequent k-itemsets can be found. The feature first invented by [2] in Apriori algorithm is used by the many algorithms for frequent pattern generation. The basic steps to mine the frequent elements are as follows [3]:

- Generate and test: In this first find the 1-itemset frequent elements L_1 by scanning the database and removing all those elements from C_1 which cannot satisfy the minimum support criteria.
- Join step: To attain the next level elements C_k join the previous frequent elements by self join i.e. $L_{k-1} * L_{k-1}$ known as Cartesian product of L_{k-1}. i.e. This step generates new candidate k-itemsets based on joining L_{k-1} with itself which is found in the previous iteration. Let C_k denote candidate k-itemset and L_k be the frequent k-itemset.
- Prune step: C_k is the superset of L_k so members of C_k may or may not be frequent but all $K-1$ frequent itemsets are included in C_k thus prunes the C_k to fnd K frequent itemsets with the help of Apriori property. i.e. This step eliminates some of the candidate k-itemsets using the Apriori property A scan of the database to

determine the count of each candidate in C_k would result in the determination of L_k (i.e., all candidates having a count no less than the minimum support count are frequent by definition, and therefore belong to L_k). C_k, however, can be huge, and so this could involve grave computation. To shrink the size of C_k, the Apriori property is used as follows. Any (k-1)-itemset that is not frequent cannot be a subset of a frequent k-itemset. Hence, if any (k-1)-subset of candidate k-itemset is not in L_{k-1} then the candidate cannot be frequent either and so can be removed from C_k. Step 2 and 3 is repeated until no new candidate set is generated.

To illustrate this, suppose n frequent 1-itemsets and minimum support is 1 then according to Apriori will generate $n^2 + (n\ 2)\text{candidate } 2 - \text{itemset } (n\ 3)\text{candidate } 3 - \text{itemset}$ and so on. The total number of candidates generated is greater than $\sum_{k=1}^{n}(n\ k)$Therefore suppose there are 1000 elements then 1499500 candidate are produced in 2 itemset frequent and 166167000 are produced in 3-itemset frequent [11].

It is no doubt that Apriori algorithm successfully finds the frequent elements from the database. But as the dimensionality of the database increase with the number of items then:

- More search space is needed and I/O cost will increase.
- Number of database scan is increased thus candidate generation will increase results in increase in computational cost.

Therefore many variations have been takes place in the Apriori algorithm to minimize the above limitations arises due to increase in size of database. These subsequently proposed algorithms adopt similar database scan level by level as in Apriori algorithm, while the methods of candidate generation and pruning, support counting and candidate representation may differ. The algorithms improve the Apriori algorithms by:

- Reduce passes of transaction database scans
- Shrink number of candidates
- Facilitate support counting of candidates

These algorithms are as follows:

2.1.2. Direct Hashing and Pruning (DHP):

It is absorbed that reducing the candidate items from the database is one of the important task for increasing the efficiency. Thus a DHP technique was proposed [7] to reduce the number of candidates in the early passes C_k for $k > 1$ and thus the size of database. In this method, support is counted by mapping the items from the candidate list into the buckets which is divided according to support known as Hash table structure. As the new itemset is encountered if item exist earlier then increase the bucket count else insert into new bucket. Thus in the end the bucket whose support count is less the minimum support is removed from the candidate set.

In this way it reduce the generation of candidate sets in the earlier stages but as the level increase the size of bucket also increase thus difficult to manage hash table as well candidate set.

2.1.3. Partitioning Algorithm:

Partitioning algorithm [1] is based to find the frequent elements on the basis partitioning of database in n parts. It overcomes the memory problem for large database which do not fit into main memory because small parts of database easily fit into main memory. This algorithm divides into two passes,

1 In the first pass whole database is divided into n number of parts.
2 Each partitioned database is loaded into main memory one by one and local frequent elements are found.
3 Combine the all locally frequent elements and make it globally candidate set.
4 Find the globally frequent elements from this candidate set.

It should be noted that if the minimum support for transactions in whole database is min_sup then the minimum support for partitioned transactions is min-sup number of transaction in that partition.

A local frequent itemset may or may not be frequent with respect to the entire database thus any itemset which is potentially frequent must include in any one of the frequent partition.

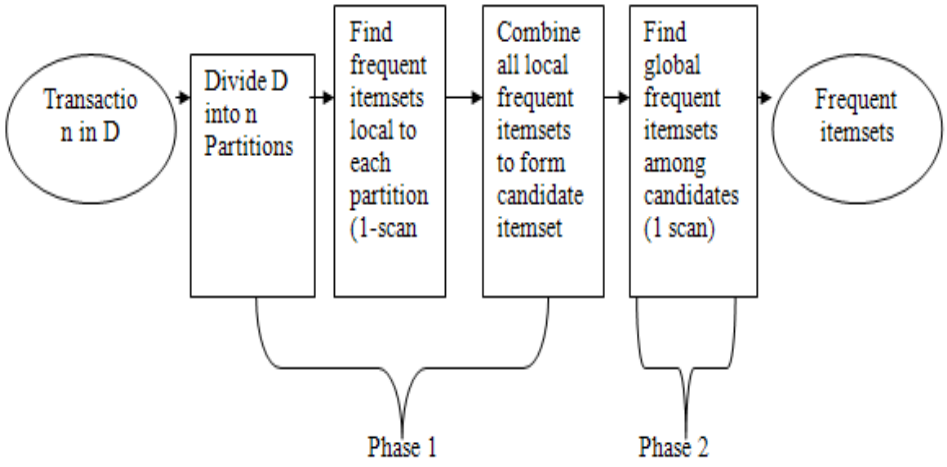


Figure 2-1: Mining Frequent itemsets using Partition algorithm [13]

As this algorithm able to reduce the database scan for generating frequent itemsets but in some cases, the time needed to compute the frequency of candidate generates in each partitions is greater than the database scan thus results in increased computational cost.

2.1.4. Sampling Algorithm:

This algorithm [10] is used to overcome the limitation of I/O overhead by not considering the whole database for checking the frequency. It is just based in the idea to pick a random sample of itemset R from the database instead of whole database D. The sample is picked in such a way that whole sample is accommodated in the main memory. In this way we try to find the frequent elements for the sample only and there is chance to miss the global frequent elements in that sample therefore lower threshold support is used instead of actual minimum support to find the frequent elements local to sample. In the best case only one pass is needed to find all frequent elements if all the elements included in sample and if elements missed in sample then second pass are needed to find the itemsets missed in first pass or in sample [13].

Thus this approach is beneficial if efficiency is more important than the accuracy because this approach gives the result in very less scan or time and overcome the limitation of memory consumption arises due to generation of large amount of datasets but results are not as much accurate.

2.1.5. Dynamic Itemset Counting (DIC):

This algorithm [4] also used to reduce the number of database scan. It is based upon the downward disclosure property in which adds the candidate itemsets at different point of time during the scan. In this dynamic blocks are formed from the database marked by start points and unlike the previous techniques of Apriori it dynamically changes the sets of candidates during the database scan. Unlike the Apriori it cannot start the next level scan at the end of first level scan, it start the scan by starting label attached to each dynamic partition of candidate sets.

In this way it reduce the database scan for finding the frequent itemsets by just adding the new candidate at any point of time during the run time. But it generates the large number of candidates and computing their frequencies are the bottleneck of performance while the database scans only take a small part of runtime.

Assumption [12, 13]: The performance of all the above algorithms relies on an implicit assumption that the database is homogenous and thus they will not generate too many extra candidates than Apriori algorithm does. For example, if all partitions in Partition algorithm are not homogenous and nearly completely different sets of local frequent itemsets are generated from them, the performance cannot be good.

2.1.6. Improved Apriori algorithm

It was absorbed in [15] [13] that **t**he improved algorithm is based on the combination of forward scan and reverse scan of a given database. If certain conditions are satisfied, the improved algorithm can greatly reduce the iteration, scanning times required for the discovery of candidate itemsets.

Suppose the itemset is frequent, all of its nonempty subsets are frequent, contradictory to the given condition that one nonempty subset is not frequent, the itemset is not frequent.

Based on this thought, proposes an improved method by combining forward and reverse thinking: find the maximum frequent itemsets from the maximum itemset firstly, then, get all the nonempty subsets of the frequent itemset. We know they are frequent on the

basis of Apriori's property. Secondly, scan the database once more from the lowest itemset and count the frequent. During this scanning, if one item is found out being excluded in the frequent set, it will be processed to judge whether the itemsets associated with it is frequent, if they are frequent, they will be added in the barrel-structure (include frequent itemsets).we get all the frequent itemsets. The key of this algorithm is to find the maximum frequent itemset fast.

Advantage:

- According to [15] The consumed time of Apriori and the improved algorithm is:

Table 2-2: Comparison of aprori and improved Apriori [16]

• Algorithm	• Time
• Apriori	• 23 min
• Improved Algorithm	• 10 min

- This algorithm gets the maximum frequent itemsets directly, then, get their subsets and compare them with the items in the database. Thus, it saves much time and the storing space.

Disadvantages:

- It will lose mean if the maximum frequent cannot be found fast.
- It cannot fit the situation that if there are still many items not included in the frequent set consisted of the maximum frequent itemsets and all of their nonempty subsets.

2.2 Algorithms for Mining from Vertical Layout Database

In vertical layout data set, each column corresponds to an item, followed by a TID list, which is the list of rows that the item appears. An example of vertical layout database set is as shown in diagram for the table 2-2.

Table 2-3: Vertical layout based database

ITEM	TID_list
I1	T1, T2, T3, T4
I2	T1, T2, T3, T4
I3	T1, T4, T5
I4	T1, T2, T3
I5	T1, T3, T5
I6	T1, T3
I7	T2

2.2.1 Eclat algorithm

It is a set intersection, depth first search algorithm [9], unlike the Apriori. It uses vertical layout database and each item use intersection based approach for finding the support. In this way, the support of an itemset P can be easily computed by simply intersecting of any two subsets $Q, R \subseteq P$, such that $P = Q \cup R$.

In this type of algorithm, for each frequent itemset i new database is created Di. This can be done by finding j which is frequent corresponding to i together as a set then j is also added to the created database i.e. each frequent item is added to the output set. It uses the join step like the Apriori only for generating the candidate sets but as the items are arranged in ascending order of their support thus less amount of intersection is needed between the sets. It generates the larger amount of candidates then Apriori because it uses only two sets at a time for intersection [9]. There is reordering step takes place at each recursion point for reducing the candidate itemsets.

In this way by using this algorithm there is no need to find the support of itemsets whose count is greater than 1because Tid-set for each item carry the complete information for the corresponding support. When the database is very large and the itemsets in the database corresponding also very large then it is feasible to handle the Tid list thus it produce good results but for small databases its performance is not up to mark.

2.3 Algorithms for Mining from Projected Layout Based Database

The concept of projected database was proposed and applied to mine the frequent itemsets efficiently because early approaches are able to mine the frequent itemsets but use large amount of memory. This type of database uses divide and conquer strategy to mine itemsets therefore it counts the support more efficiently then Apriori based algorithms. Tree projected layout based approaches use tree structure to store and mines the itemsets. The projected based layout contains the record id separated by column then record.

Tree projection is defined as the lexicographic tree with nodes contains the frequent itemsets [14]. The lexicographic trees usually follow the ascending order for saving the frequent itemsets according to the support for better mining.

Tree Projection algorithms based upon two kinds of ordering breadth-first and depth-first. For breath-first order, nodes are constructed level by level in the lexicographic tree for frequent itemsets [11]. In order to compute frequencies of nodes (corresponding frequent itemsets) at k level, tree projection algorithm maintained matrices at nodes of the k-2 level and one database scan was required for counting support [5]. Every transaction is projected by node sequentially. The projected set of transaction for reduced set is used to evaluate frequency.

For depth-first order, database is projected along the lexicographic tree and also requires fitting into main memory [13]. The advantage is that the projected database will become smaller along the branch of the lexicographic tree while the breadth-first needs to project the database from the scratch at each level.

The disadvantage of depth-first is obvious that it needs to load database and projected databases in memory. The breadth-first method will also meet the memory bottleneck when the number of frequent items is large and the matrix is too large to fit in memory [5].

2.3.1 FP-Growth Algorithm

FP-tree algorithm [5, 6] is based upon the recursively divide and conquers strategy; first the set of frequent 1-itemset and their counts is discovered. With start from each frequent pattern, construct the conditional pattern base, then its conditional FP-tree is constructed (which is a prefix tree.). Until the resulting FP-tree is empty, or contains only one single path. (Single path will generate all the combinations of its sub-paths, each of which is a frequent pattern). The items in each transaction are processed in L order. (i.e. items in the set were sorted based on their frequencies in the descending order to form a list).

The detail step is as follows: [6]

FP-Growth Method: Construction of FP-tree

1 Create root of the tree as a “null”.
2 After scanning the database D for finding the 1-itemset then process the each transaction in decreasing order of their frequency.
3 A new branch is created for each transaction with the corresponding support.
4 If same node is encountered in another transaction, just increment the support count by 1 of the common node.
5 Each item points to the occurrence in the tree using the chain of node-link by maintaining the header table.

After above process mining of the FP-tree will be done by Creating Conditional (sub) pattern bases:

1 Start from node constructs its conditional pattern base.
2 Then, Construct its conditional FP-tree & perform mining on such a tree.
3 Join the suffix patterns with a frequent pattern generated from a conditional GP-tree for achieving FP-growth.
4 The union of all frequent patterns found by above step gives the required frequent itemset.

In this way frequent patterns are mined from the database using FP-tree.

Definition: Conditional pattern base [6]

A “subdatabase” which consists of the set of prefix paths in the FP-tree co-occuring with suffix pattern.eg for an itemset X, the set of prefix paths of X forms the conditional pattern base of X which co-occurs with X.

Definition: Conditional FP-tree [6]

The FP-tree built for the conditional pattern base X is called conditional FP-tree.

Let sample database in table 2-3 and corresponding the support count in table 2-4 is:

Table 2-4 : Sample database

Tid	Items
T1	I1, I2, I3, I4, I5
T2	I5, I4, I6, I7, I3
T3	I4, I3, I7, I1, I8
T4	I4, I7, I9, I1, I10
T5	I1, I5, I10, I11, I12
T6	I1, I4, I13, I14, I2
T7	I1, I4, I6, I15, I2
T8	I16, I7, I9, I17, I5
T9	I1, I9, I8, I10, I11
T10	I4, I9, I12, I2, I14
T11	I1, I3, I5, I6, I15
T12	I3, I7, I5, I17, I16
T13	I8, I3, I4, I2, I11
T14	I4, I9, I13, I12, I18
T15	I5, I3, I7, I9, I15
T16	I18, I7, I5, I1, I3
T17	I1, I17, I7, I9, I4
T18	I4, I3, I16, I5, I1

Table 2-5: Frequency of Sample Database

Item	Support	Item	Support
I1	11	I10	3
I2	4	I11	3
I3	9	I12	3
I4	10	I13	2
I5	10	I14	2
I6	3	I15	3
I7	8	I16	3
I8	3	I17	3
I9	7	I18	3

Suppose minimum support is 5. Thus delete all infrequent items whose support is less them 5.After all the remaining transactions arranged in descending order of their frequency. Create a FP- tree. For Each Transaction create a node of an items whose support is greater than minimum support, as same node encounter just increment the support count by 1.

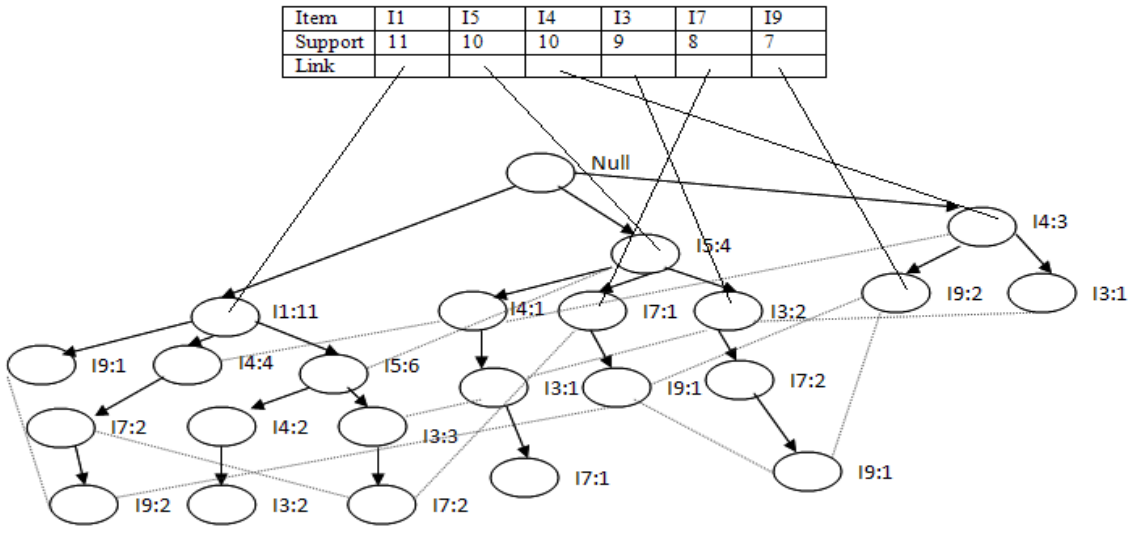


Figure 2-2: FP-Tree Constructed For Sample Database

In this way after constructing the FP-Tree one can easily mine the frequent itemsets by constructing the conditional pattern base

2.3.2 COFI-Tree Algorithm

COFI tree [19] generation is depends upon the FP-tree however the only difference is that in COFI tree the links in FP-tree is bidirectional that allow bottom up scanning as well [5,20]. The relatively small tree for each frequent item in the header table of FP-tree is built known as COFI trees [20]. Then after pruning mine the each small tree independently which minimise the candidacy generation and no need to build he conditional sub-trees recursively. At any time only one COFI tree is present in the main memory thus in this way it overcome the limitations of classic FP-tree which can not fit into main memory and has memory problem.

COFI tree is based upon the new anti-monotone property called global frequent/local non frequent property [20]. It states that all the nonempty subsets of frequent patterns with respect to the item X of an X-COFI tree must also be frequent with respect to item X. In this approach trying to find the entire frequent item set with respect to the one frequent item sets. If the itemset participate in making the COFI tree then it means that item set is globally frequent but this doesn't mean that item set is locally frequent with respect to the particular item.

Steps: Create a COFI-Tree

1 Take FP-tree as an input with bidirectional link and threshold value.
2 Consider the least frequent item from the header table let it be X.
3 Compute the frequency that share the path of item X and remove all non frequent items for the frequent list of item X.
4 Create COFI tree for X known as X-COFI tree with support-count and participation=0
5 If items on Y which is locally frequent with respect to X form a new prefix path of X-COFI tree

6 DO, Set support count= support of X and participation count =0 for all nodes in a path.

7 Else adjust the frequency count and pointers of header list until all the nodes are not visited.

8 Repeat step 2 goes on until all frequent items not found.

9 Mine the X-COFI tree.

Support count and the participation count are used to find candidate frequent pattern and stored in the temporary list, which will be more clear after example.

Let the above FP-tree in figure 2-3 is the input for making COFI tree. Consider the links between the nodes are bidirectional. Then according to algorithm the COFI tree forms are first consider the itemset I9 as is least frequent item set, when scan the FP-tree the first branch is (I9,I1) has frequency 1, therefore frequency of the branch is frequency of the test item, which is I9. Now count the frequency of each item in a path with respect to I9. It is found that (I7, I3, I4, I5, I1) occur 4, 2, 4, 2, 3 times respectively thus according to anti-monotone property I9 will never appear in any frequent pattern expect itself. In the same way find the global frequent items for (I7, I3, I4, I5) which are also locally frequent, like for I7 two items I3 and I5 globally frequent item are also found locally frequent with frequency 5and 6 respectively thus a branch is created for each such items with parent I7. If multiple items share same prefix they are merged into one branch and counter is adjusted. COFI trees for items are follows:

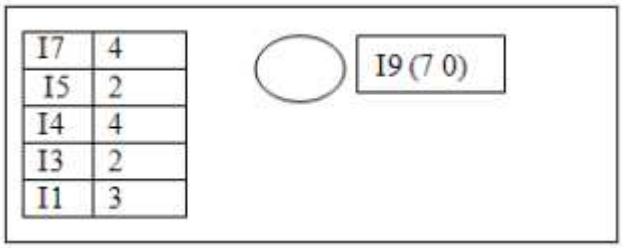


Figure 2-3: I9 COFI Tree

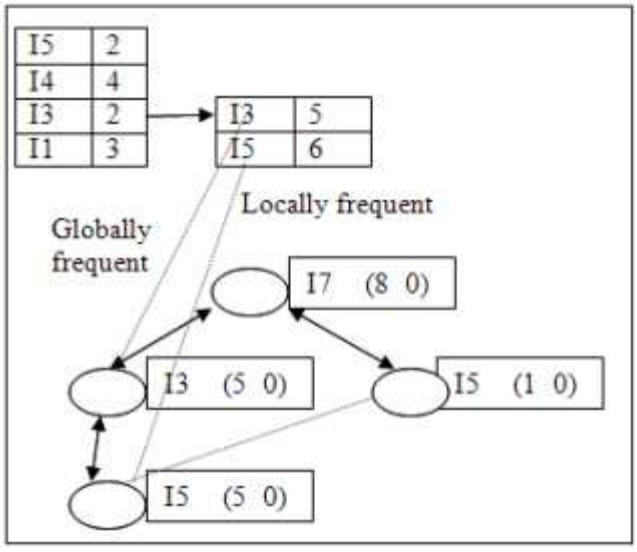


Figure 2-4: I7 COFI Tree

Similarly COFI tree is built for I3, I4, and I5. First after the globally frequent, find the itemsets which are also locally frequent , than find the support counter and make participation counter always equal to '0'. We are representing only for I7 in above example.

Once the COFI trees have built then we have to mine the frequent items from these COFI trees with the help of following procedure:

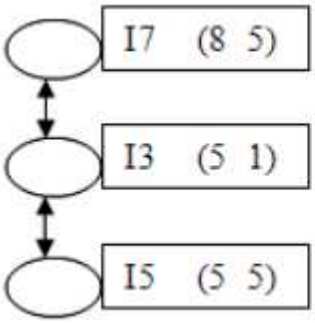
Steps MINE X-COFI Tree

10 For node X select the item from the most frequent to least frequent with its chain
11 Until there are no node left, select all nodes from node X to root save in D list and in list F save the frequency count and participation count.
12 Generate all non discarded patterns Z from items D.
13 Add the list with frequency =F whose patterns not exist in X candidate list, else increment the frequency by F.
14 Increment the participation value by F for all items in D.
15 Select the next node and repeat step 2 until there is no node left.
16 Remove all non frequent items from X-COFI tree.

The COFI trees of all frequent items are mined independently one by one [8], first tree is discarded before the next COFI tree is come into picture for mining. Based on the support count and participation count frequent patterns are identified and non frequent items are discarded in the end of processing.

Let's take an example for I7 COFI-Tree

As mining process start from the most local frequent item I5 and as from the figure 3, I5 exist in two branches I5, I3, I7 and I5, I7. Therefore

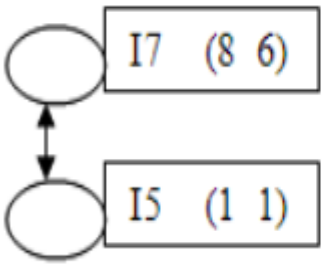


Pattem			P
I7	**I3**	**I5**	**5**
I7	I5		5
I7	I3		5

Figure 2-5: Mining E COFI tree for branch (I7, I3, I5)

As the frequency for each branch is equal to frequency of first item minus participation count of that node, Thus I5 has frequency 5 and participation count is 0 therefore first frequent set is found i.e. (I7, I3, I5:5).

Participation value is incremented by 1 for branch (I7, I3, I5) increment by 5 same as the frequency of I5. the pattern I7,I5 generates a pattern 6 and 1 which is already exist therefore increment the previous participation for I7, I5 by 1.therefore for I7 is become for I5 it become 1 for branch I7,I5.



Pattem			P
I7	I3	I5	5
I7	**I5**		**6**
I7	I3		5

Figure 2-6: Mining I7 COFI tree for branch (I7, I5)

Similarly mine for sub branch (I7, I3) and it is found that frequent patterns (I7, I3: 5), (I7, I5: 6), (I7, I3, I5: 5) for COFI tree I7. Similarly main the frequent patterns for I3, I4, and I5 itemsets.

In this way COFI tree mine the frequent item sets very easily then the FP-growth algorithm with the help of FP-tree [6]. It saves to memory space as well as time in comparison to the FP-growth algorithm. It mines the large transactional database with minimal usage of memory. It does not produce any conditional pattern base. Only simple traversal is needed in the mining process to find all the frequent item sets. It is based upon the locally and globally frequent item sets thus easily remove the frequent item sets in the early stages and don't allow any locally non frequent elements to takes part in next stage.

2.3.3 CT-PRO Algorithm

CT-PRO is also the variation of classic FP-tree algorithm [21]. It is based upon the compact tree structure [21, 22]. It traverses the tree in bottom up fashion. It is based upon the non-recursive based technique. Compress tree structure is also the prefix tree in which all the items are stored in the descending order of the frequency with the field index, frequency, pointer, item-id [22]. In this all the items if the databases after finding the frequency of items and items whose frequency is greater than minimum support are mapped into the index forms according to the occurrence of items in the transaction. Root of the tree is always at index '0' with maximum frequency elements. The CT-PRO uses the compact data structure known as CFP-tree i e. compact frequent pattern tree so that all the items of the transactions can be represented in the main memory [21, 22, 23].

The CT-PRO algorithm consists following basic steps [21]:

1 In the first step all the elements from the transaction are found whose frequency is greater than the minimum user defined support.
2 Mapping the elements according to the index value.
3 Construct the CFP-tree which is known as globally CFP-tree.
4 Mine the Global CFP-tree by making local CFP-tree for each particular index.

In this way by following the above steps can easily find the frequent item sets. The frequency of item sets greater then minimum support which defined as '5' for the sample database present in Table 1. After the mapping the Global Tree formed from the transactions are follows:

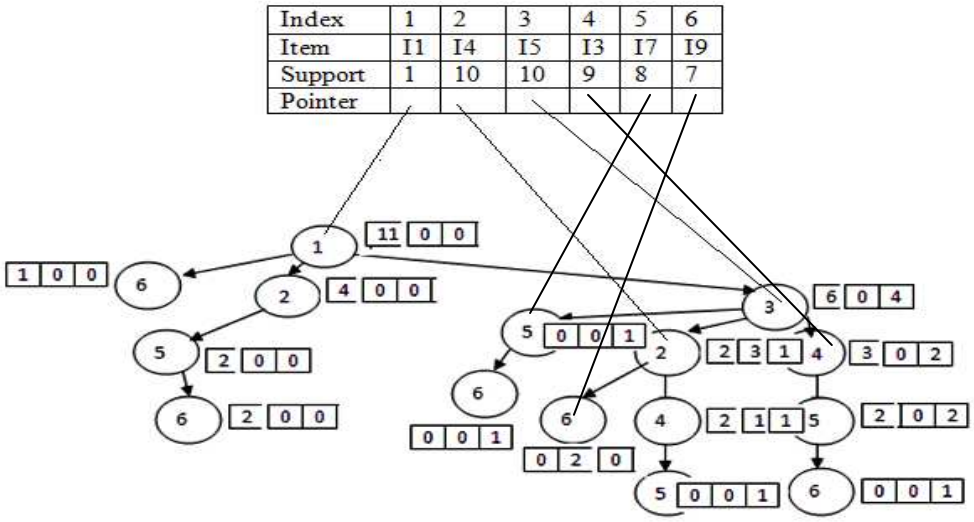


Figure 2-7: CFP-Tree for Table 2-3

Note: No transaction starts from Item I3, I7, and I9 therefore no pointer is there.

Steps for Making Global CFP-tree

1 Take Database and threshold value as input.

2 Find the frequent items from database and sort in descending order in new list.

3 Then map the frequent items of the transaction in the index form, and sort in ascending order of their transaction Id (Tid).

4 Make maximum frequency item is the root node of the tree and makes it for index 1; insert all sub children in the tree.

5 For new index starting items, adjust pointer and build sub trees and give incremented index value or level as in above figure 2-3.

6 CFP-Tree has been built. Mine the CFP-tree index wise as projections.

After the global CFP-tree, the local CFP-Tree is build for each index item separately. It starts from the least frequency element.

The local CFP-Tree for the index 5 i.e. for the item I7 is found by first counting the other indexes that occur with the index 5. The other indexes are: 1, 3, and 4 which occur 4, 6 and 5 time respectively. As minimum support is 5 thus index 1 is pruned from local tree. The corresponding items are I1, I5 and I3 Item 1 is not locally frequent thus eliminated. Now construct the local CFP-tree projection for item I7 is as follows:

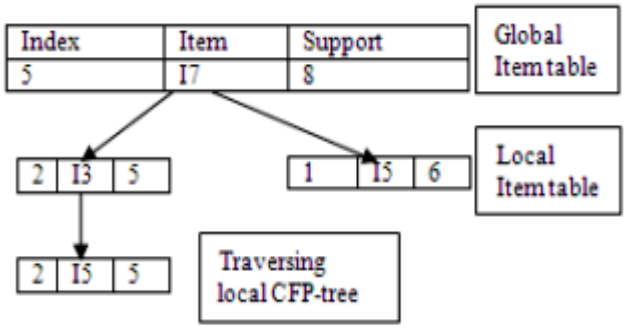


Figure 2-8: Frequent itemsets in Projection 5

The frequent items are that can be easily found by the above projection for index 5 is as follows:

(I7, I3, I5:5), (I7, I5: 6), (I7, I3: 5)

Similarly the frequent patterns are easily found for other indexes.

Algorithm: Mine Global CFP-Tree

1 Take Global CFP-tree as input.
2 Start from least frequent item index, check all the indexes come together in the global tree with the desired index.
3 Count support of all the indexes find on above step.
4 Prune all those indexes whose support is less then minimum support means those are not locally frequent.

5 Construct the local CFP-tree by those remaining indexes by again mapping along with the support.

6 Join the links between the items as same the linkage in the global CFP-tree in between only that item's index i.e. parent should be parent and child will be child of particular existing projection.

7 The new supports of nodes are the support of frequent items of that particular projection.

8 Repeat the process until no index is left.

In this way CFP-tree Provide facility to easily mine the frequent items with the help of projections which prune the not frequent items locally and utilize the memory space efficiently by mining projections one by one. For large database the items can also easily fit into main memory.

2.3.4 H-mine Algorithm

H-mine [8] algorithm is the improvement over FP-tree algorithm as in H-mine projected database is created using in-memory pointers. H-mine uses an H-struct new data structure for mining purpose known as hyperlinked structure. It is used upon the dynamic adjustment of pointers which helps to maintain the processed projected tree in main memory therefore H-mine proposed for frequent pattern data mining for data sets that can fit into main memory. It has polynomial space complexity therefore more space efficient then FP-growth and also designed for fast mining purpose. For the large databases, first in partition the database then mine each partition in main memory using H-struct then consolidating global frequent pattern [8]. If the database is dense then it integrates with FP-Growth dynamically by detecting the swapping condition and constructing the FP-tree.

This working ensures that it is scalable for both large and medium size databases and for both sparse and dense datasets [14]. The advantage of using in-memory pointers is that their projected database does not need any memory the memory required only for the set of in-memory pointers.

Chapter 3

3. Problem Formulation

The problem of mining frequent itemsets arises in the large transactional databases when there is need to find the association rules among the transactional data for the growth of business. Many different algorithms has been proposed and developed to increase the efficiency of mining frequent itemsets including (Horizontal layout based algorithms, Vertical Layout Based algorithms [1, 2, 4, 8, 9, 10], Projected layout based algorithms [23] and Hybrid algorithms [16, 18]. These different algorithms have strengths and weakness in different type of datasets. As a measure of performance mainly the average number of operations or the average execution times of these algorithms have been investigated and compared.

3.1 Motivation

Studies of Frequent Itemset (or pattern) Mining is acknowledged in the data mining field because of its broad applications in mining association rules, correlations, and graph pattern constraint based on frequent patterns, sequential patterns, and many other data mining tasks. Efficient algorithms for mining frequent itemsets are crucial for mining association rules as well as for many other data mining tasks. The major challenge found in frequent pattern mining is a large number of result patterns. As the minimum threshold becomes lower, an exponentially large number of itemsets are generated. Therefore, pruning unimportant patterns can be done effectively in mining process and that becomes one of the main topics in frequent pattern mining. Consequently, the main aim is to optimize the process of finding patterns which should be efficient, scalable and can detect the important patterns which can be used in various ways.

3.2 Gap Analysis

- All the algorithms produce frequent itemsets on the basis of minimum support.

- Apriori algorithm is quite successful for market based analysis in which transactions are large but frequent items generated is small in number.
- The Apriori variations (DHP, DIC, Partition, and Sample) algorithms among them DHP tries to reduce candidate itemsets and others try to reduce database scan.
- DHP works well at early stages and performance deteriorates in later stages and also results in I/O overhead.
- For DIC, Partition, sample algorithm performs worse where database scan required is less then generating candidates.
- Vertical Layout based algorithms claims to be faster than Apriori but require larger memory space then horizontal layout based because they needs to load candidate, database and TID list in main memory.
- For projected layout based algorithms include FP-Tree and H-mine, performs better then all discussed above because of no generation of candidate sets but the pointes needed to store in memory require large memory space.
- FP-Tree variations include COFI-Tree and CT-PRO performs better than classical FP-tree as COFI-tree performs better in dense datasets but with low support its performance degrades for sparse datasets and for CT-PRO algorithm performs better for sparse as well for dense data sets but difficult to manage the compress structure.

Therefore these algorithms are not sufficient for mining the frequent itemsets for large transactional database.

3.3 Problem statement

Let $I = \{i_1, i_2, i_3 \ldots, i_n\}$ be a set of items and n is considered the dimensionality of the problem. Let D be the task relevant database which consists of transactions where each transaction T is set of items such that $T \epsilon I$. A transaction T is said to contain itemset X,

which is called a pattern, i.e. $X \epsilon T \epsilon I$. A transaction is a pair which contains unique identifier Tid and set of items [3].

A transaction T is said to be maximal frequent if its pattern length is greater than or equal to all other existing transactional patterns and also count of occurrence (support) in database is greater than or equal to specified minimum support threshold [15].

An itemset X is said to be frequent if its support is greater than or equal to given minimum support threshold i.e. count(X)>minsup [2].

Transactional database D and minimum support threshold is given, therefore the problem is to find the complete set of frequent itemsets from Transactional type of databases to increase the business, so that relation between customers behaviour can be found between various items.

3.4 The Proposed Objectives

The problems or the limitations defined in the above section of this chapter are proposed to be solved by:

1. To observe the effect of various existing algorithms for mining frequent itemsets on various datasets.
2. To propose a new scheme for mining the frequent itemsets for retailer transactional database i.e. for the above problem.
3. To validate the new scheme on dataset.

3.5 Methodology Used

To implement the proposed solution of the problem that is being taken care of in this thesis work, the following methodology is used:

1. To analyze the various existing techniques and find their strengths and weakness by the literature survey.
2. To compare the existing techniques.

3. Build a program for our desired problem by using maximal Apriori (Improved Apriori technique) and FP-tree structure.
4. Validate the program by desired input.

3.6 Importance

Mining frequent itemsets is the very crucial task to find the association rules between the various items. In the market industry, everyone wants to enhance the business thus it is very important to find out the items which are more frequently sale or purchase. Once the selling or purchasing trend of the customer is known then one can easily provide the good services to customer with result in enhancing the business. As it is very common in retail selling database that two or more items sell or purchase together many times [17] therefore database contains same set of items many times, by using this concept aim to overcome the limitation of above existing approaches [2, 6] and propose a novel approach to mine frequent itemsets from a large transactional database without the candidate generation with the help of existing techniques, Because till now no single technique using this property to overcome the limitation is so much efficient.

Chapter 4

4. Implementation of Novel Approach

This chapter includes detail of implementation of the new approach to mine the frequent itemsets with example. The main objective of the research is to develop and propose a new scheme for mining the association rules out of transactional data set. The proposed scheme is based on two approaches: Improved Apriori approach and FP-Growth approach. The proposed scheme is more efficient than Apriori algorithm and FP-growth algorithm, as it is based on two of the most efficient approaches. To achieve the research objective successfully, a series of sequence progresses and analysis steps have been adopted. Figure 4-1 depicts the methodologies to extract frequent itemsets from the transactional data set using the new scheme [25].

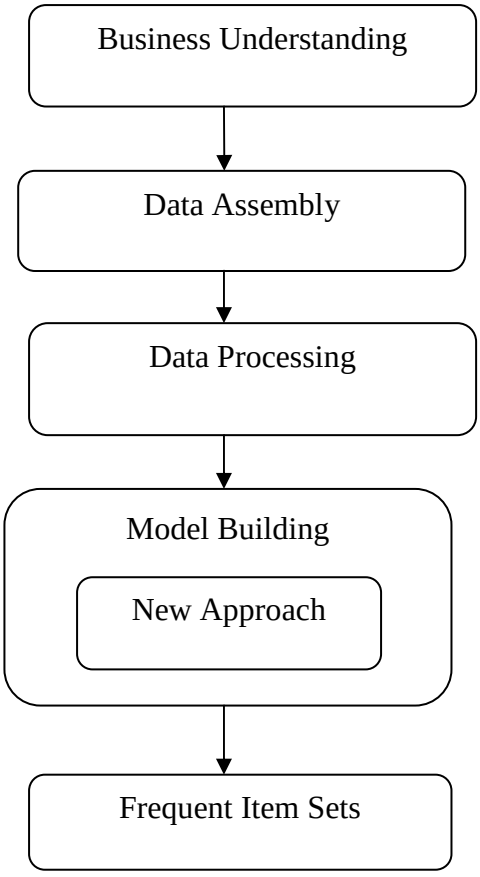


Figure 4-1: Methodology used to mine frequent itemsets

4.1 Business Understanding

The concern of this stage is to identify the problem area and describe the problem in general terms. In another word, the enterprise decision makers need to formulate goals that the data mining process is expected to achieve. Then the first step in the methodology is to clearly defined business problem:

4.1.1 Market Based analysis

The retail industry is a most important application area for data mining, since it collects enormous amounts of data on sales, customer shopping history, service, and goods transportation, consumption. Progress in bar code technology has made it possible for retail organizations to collect and store massive amounts of sales data, referred as the basket data. Such market basket databases consist of a large number of transaction records. Each record lists all items bought by a customer on a single purchase trip. Using market basket analysis is a key factor of success in the competition of supermarket retailers. Market basket analysis provides manager with knowledge of customers and their purchasing behavior which brings potentially huge added value for their business. Recent marketing research has suggested that in-store environmental stimuli, such as shelf-space allocation, and product display, have a great influence upon consumer buying behavior and may induce substantial demand.

4.1.2 Objective of Market Based Analysis

One possibility to do so is to make the store layout construction and the promotional campaign through the introduction of market basket analysis. Market basket analysis has the objective of individuating products, or groups of products, that tend to occur together (are associated) in buying transactions (baskets). The knowledge obtained from a market basket analysis can be very valuable, and it can be employed by a supermarket to redesign the layout of the store to increase the profit through placing interdependencies products near to each other and to satisfy customers through saving time and personalized the store layout. Another strategy, Items that are associated can be put near to each other;

it increases the sales of other items due to complementarily effects. If the customers see them, it has higher probability that they will purchase them together.

4.2 Data Assembling

Data set: Data Assembly also include collecting of data, in this thesis for testing purpose the data is collected from the Fimi website. The data is challenging due to the number of characteristics which are the number of the records, and the sparseness of the data (each records contains only small portion of items). In our experiments we chose different dataset with different properties, to prove the efficiency of the algorithms, Table 4-1 shows the datasets and the characteristics from the fimi website.

Table 4-1: The Datasets

Data set	#Items	Avg. Length	#Trans	Type	Size
T10I4D100K	1000	10	100, 000	Sparse	3. 93 MB
Mushroom	119	23	8, 124	Dense	557 KB

4.3 Existing Techniques Comparisons

There are several algorithms for mining the frequent itemsets. Those algorithms can be classified and Apriori-like algorithms (candidate generate-and-test strategy) and FP-growth-like algorithms (divide-and-conquer strategy). In this research we are focusing on FP-growth-like algorithms and improved Apriori algorithm to find the frequent itemsets. Many variations of the FP-growth algorithm have been proposed which focus on improving the efficiency of the original algorithm. Some algorithms are best suited for sparse dataset and some are best suited for dense datasets. Some shows the outstanding results on specific type of database. Apriori based algoritms performs well for dense datasets and FP- Tree based algorithms performs well for sparse datasets among them there are many variations therefore to find the best algorithm for database we perform some comparisons:

4.3.1 Comparison of Classical Algorithms:

Table 4-2: Comparison of classical algorithms

	Horizontal layout based algorithms					Vertical layout based algorithm	Projected layout based algorithms	
Algorithm / Parameter	Apripri Algorithm	DHP algorithm	Partition Algorithm	DIC algorithm	Sample Algorithm	Eclat algorithm	FP-tree Algorithm	H-mine Algorithm
Storage Structure	Array based	Array Based	Array based	Array based	Array based	Array based	Tree based	Tree based
Technique	Use Apriori property and join and prune method	Use hashing technique for finding frequent itemsets	Partition the database for finding local frequent item first	Based upon dynamic insertion of candidate items.	Pick any random sample for checking frequency of whole database at lower threshold support	Use intersection of Transaction ids list for generating candidate itemsets.	It constructs conditional frequent pattern tree and conditional pattern base from database which satisfy the minimum support.	It uses the hyperlink pointers to store the partitioned projected database in main memory.
Memory utilization	Due to large amount of candidate are produced so require large memory space	Require less space at earlier passes but more in later stages	Each partition is easily occupy in main memory	Require different amount of memory at different point of time	Very less amount of memory is needed	Require less amount of memory compare to apriori if itemsets are small in number	Due to compact structure and no candidates generation require less memory	Memory is utilized according to needs and partitions of projected database
Databases	Suitable for sparse datasets as well as dense datasets	Suitable for medium databases	Suitable for large databases	Suitable for medium and low databases	Suitable for any kind of dataset but mostly not give accurate results	Suitable for medium and dense datasets but not suitable for small datasets.	Suitable for large and medium datasets	Suitable for sparse and dense datasets.
Time	Execution time is more as time wasted in producing candidates at every time	Execution time is small for small databases.	Execution time is more because of finding locally frequent then globally frequent	Execution time is small because dynamic itemset are added according to situation.	Execution time is very much small.	Execution time is small then apriori algorithm	Execution time is large due to complex compact data structure	Execution time is large then FP-tree and others because of partition the database.

Thus from the above comparisons it is found that Maximal Apripri (Improved Apriori is the best suited for the database which contains the repeated transaction occurrences greater than the minimum support. So that many frequent items are found as in one time. But it leaves many frequent itemsets which are not included in the maximal itemset.

4.3.2 Comparison of FP-Tree variations:

As from reviewing the various techniques i.e. FP-Growth [6], COFI-Tree [20], CT-Pro [22] and many more [14, 16, 17, 18], we can differentiate them by the following considerations:

Table 4-3: The Comparison of FP-variations

Algorithm parameters	FP-Growth	COFI-Tree	CT-PRO
Structure	Simple Tree Based structure.	Uses Bidirectional FP-Tree structure.	Uses compressed FP-Tree data structure.
Approach	Recursive	Non- Recursive	Non- Recursive
Technique	It constructs the conditional frequent pattern tree and conditional pattern base from database which satisfy the minimum support.	It constructs the bidirectional FP-Tree and builds the COFI-Trees for each item then mines the COFI-Tree locally for each item.	It constructs the compact FP-Tree through mapping into index and then mine frequent itemsets according to projections index separately
Memory Utilization	Low as for large database complete Tree structure cannot fit into main	Better, Fit into main memory due to mining locally in parts for the complete tree, Thus	Best, as Compress FP-tree structure used and mine according to projections separately

	memory	every part represent in main memory	thus easily fit into main memory
Databases	Good for dense databases	Good for dense as well as Sparse databases. But with low support in sparse databases performance degrades.	Good for dense as well as for Sparse databases.
Complexity	It is less complex to manage. As it does not divide into parts.	It is difficult to manage due finding local frequent and global frequent.	It is difficult to manage due to large number of projections created.

Thus from the above comparison it is found that CT-PRO and COFI tree algorithms are better than the FP-Growth but due to their complex structure it is difficult to manage importantly when hybrid structure is used. Therefore FP-Growth is well suited to mine the frequent itemsets which is not mined by the maximal Apriori to increase the efficiency and performance.

4.4 Model Building

This stage is concerned with extraction of patterns for the data. The core of this research is mainly focused on model building. This phase concerns various view points and different aspects that should be given attention in order to yield sufficient results.

It starts with examine the existing techniques and found that maximal Apriori (improved Apriori) and the FP-tree performs better. Maximal Apriori performs better if the same transaction occurs in the database many times. But not finding the whole frequent itemsets. Therefore FP-Tree helps the maximal Apriori to find all remaining frequent itemsets.

First at the beginning, In the first scan same set of occurring transactions are found from the database and save in the two dimension array with the count of repetition. Then find the maximal frequent itemsets if contained in array i.e. find the maximal transactions whose occurrence is greater than the user specified limit (support). Take all its non empty subsets of maximal frequent itemsets as frequent [15]. In this way most of the frequent itemsets are found. For the remaining itemsets which are frequent but not include in maximal frequent itemsets will be mined by using tree structure. Therefore database is pruned by considering only those transactions which contain the 1-itemset frequent items but not include in maximal frequent itemsets. In this way now construct the FP-Tree only for pruned (reduced) database. Thus memory consumption for FP-tree is now less due to reduced database.

4.4.1 Implementation of New Mining Algorithm with Example

In this Section, a new algorithm based upon the improved Apriori and the FP-tree structure is present.

In a large transactional database like retailer database it is common that multiple items are selling or purchasing simultaneously therefore the database surly contains various transactions which contain same set of items. Thus by taking advantage of these transactions trying to find out the frequent itemsets and prune the database as early as possible without generating the candidate itemset and multiple database scan, results in efficiently usage of memory and improved computation.

This proposed algorithm is based upon the Apriori property [2] i.e. all non empty subsets of the frequent itemsets are frequent.

Algorithm has two procedures. In first procedure, find all those maximal transactions which are repeating in the database equal to or greater than min user defined support also known as maximal frequent itemset [15]. Then get all nonempty subsets of those maximal frequent itemset as frequent according to Apriori property. Scan the database to find 1-itemset frequent elements. There may be many items found which are 1-itemset frequent but not include in maximal frequent transactions. Therefore prune the database

by just considering only those transactions from the database which contain 1-itemset frequent elements, but not include in the maximal frequent itemsets. Now this pruned database is smaller than the actual database in the average cases and no item left in best case.

For the second procedure, pruned database is taken as input and scan the pruned database once find 1-itemset frequent and delete those items from transaction which are not 1-itemset frequent. Then construct the FP-tree [6] only for pruned transactions. In this way it reduces the memory problem for FP-tree because the database is reduced in most of cases. In best case no need to build FP-tree because all elements are found in first procedure. In the worst case if there is no maximal frequent transaction exist, then only second procedure run and also computational performance is same as FP-tree. The key of this idea to prune the database after finding the maximal frequent itemsets and formation of FP-tree for a pruned database thus reduce memory problem in FP-tree and make the mining process fast. The more detail step as follows:

Procedure1:

Input: Database D, minimum support

Step 1: Take a 2- dimensional array; Put the transaction into 2-dimmension array with their count of repetition.

Step 2: Arrange them in increasing order on the basis of the pattern length of each transaction.

Step 3: Find maximal transactions (k-itemset) from the array whose count is greater than or equal to the minimum support known as maximal frequent itemsets or transactions. If k-itemsets count is less than minimum support then look for k-itemsets and (k-1)-itemsets jointly for next (k-1) maximal itemsets and so on until no itemsets count found greater than minimum support. If no such transaction found then go to Procedure2.

Step 4: Once the maximal frequent transactions found, than according to Apriori property consider all its non empty subsets are frequent.

Step 5: There are itemsets remaining which are not included in maximal frequent itemset but they are frequent. Therefore find all frequent 1-itemset and prune the database just considering only those transactions which contain frequent 1-itemset element but not include in maximal frequent transaction.

Output: some or all frequent itemsets, Pruned database D1.

Procedure2:

Input: Pruned database D1, minimum support

Step 1: Find frequent 1-itemset from pruned database; delete all those items which are not 1-itemset frequent.

Step 2: Construct FP-tree for mine remaining frequent itemset by following the procedure of FP-tree algorithm [6] as discussed above in section 2B.

Output: Remaining frequent itemsets

Example:

Suppose table 1 is a database of retailer, D. There are 10 transactions. Suppose the minimum support is 2.

Table 4-4: Sample Retailer Transactional Database

Tid	**List of items**
T1	I1,I2,I5
T2	I2,I4
T3	I2,I3
T4	I1,I2,I4
T5	I1,I3
T6	I2,I3

T7	I1,I3
T8	I1,I2,I3,I5
T9	I1,I2,I3
T10	I1,I2,I3,I5

Procedure 1:

Input: Database D, Minimum support = 2

Step1: After scanning a database put items in 2-dimensional array with the count of repetition.

Table 4-5: Itemsets in array of Table 4-4

2- itemset	count	3-itemset	count	4-itemset	count
{I2,I3}	2	{I1,I2, I4}	1	{I1,I2,I3, I5}	2
{I1,I3}	2	{I1,I2, I3}	1		
{I2,I3}	2	{I1,I2, I4}	1		

Step 2: Find maximal itemset (4-itemset). Check weather its count is greater or equal to specified support, its count is 2 in our case which is equal to given support therefore this transaction is considered as maximal frequent. (If its count is less than support value then we scan k-1 and k-itemset in array for k-1 maximal itemset jointly and so on until finding all maximal frequent itemset from a array. i.e. 3-itemset and 4-itemset for checking 3-itemset maximal and so on).

Step 3: According to Apriori property subset of maximal frequent itemset is also considered as frequent .i.e. Maximal frequent itemset: {I1, I2, I3, I5}.All subsets are frequent (Apriori Property) i.e. {I1, I2, and I3},

{I1, I2, I5}, {I2, I3, I5}, {I2, I3}, {I2, I5}, {I1, I2}, {I1, I3}, {I1, I5}, {I3, I5}, {I1}, {I2}, {I3}.

Step 4: Scan the database for finding the above mined support.

Step 5: Find 1-itemset frequent from database, it is found that I4 which is frequent but not include in maximal frequent itemset. (There may be many items remain which are not include in maximal frequent itemsets, in our case only 1 item is there).

Prune the database by considering only transaction which contains I4 itemset.

Output: Some frequent itemsets ({I1, I2, I5}, {I2, I3, I5}, {I2, I3}, {I2, I5}, {I1, I2}, {I1, I3}, {I1, I5}, {I3, I5}, {I1}, {I2}, {I3}), Pruned database

Table 4-6: Pruned database of Table 4-4

TID	**List of items**
T2	I2,I4
T4	I1,I2,I4

Procedure2:

Input: Pruned database, minimum support = 2

Step 1: Find frequent 1-itemset from pruned database with support = 2, It is found I1 is not frequent therefore delete it. (In this case delete I1).

Table 4-7: frequency of itemsets of Table 4-4

Itemset	**frequency**
I2	2
I4	2
I1	1

Transactions become: T2: I2, I4 and T4: I2, I4.

Step 2: Construct FP-tree for remaining transaction in pruned database.

1. In this case I2 and I4 have same frequency, therefore no need to arrange in L order (descending order of their frequencies).

2. A new branch is created for each transaction. In this case single branch is created because of same set of transactions.

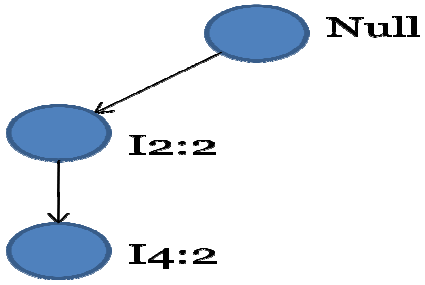


Figure 4-2 : FP-tree for transaction Table 4-4

3. Construct Conditional pattern base and FP-tree only for item I4.

Table 4-8: Mining the FP-tree by creating conditional (Sub-) pattern base of Table 4-4

Item	Pattern base	Conditional FP-tree	Frequent Item
I4	{I2: 2}	{I2:2}	{I4,I2:2}

Thus by this procedure we can easily find unmined frequent itemset i.e. {I4, I2} which some of previous algorithm [4] are not able to find. Now we get all the frequent itemset in a particular database.

Output: remaining itemsets ({I4, I2})

Thus the remaining frequent itemsets which are not mined by only maximal frequent itemsets are mined by the FP-Growth procedure without generation of candidate itemsets and also in efficiently usage of memory because after pruning whole database is easily fit into main memory.

CHAPTER 5

5. Testing and Result

This chapter demonstrates the experiments that we have performed to evaluate the new scheme. For the evaluation purpose we have conducted several experiments using the existing data set. Those experiments performed on computer with Core 2 Duo 2.00 GHZ CPU, 2.00 GB memory and hard disk 80 GB. All the algorithms were developed by C++ language and for the unit of measuring the time and the memory are second and megabyte respectively.

5.1 Comparison Analysis

5.1.1 Time Comparison

As a result of the experimental study, revealed the performance of our new technique with the Apriori and FP-Growth algorithm. The run time is the time to mine the frequent itemsets. The experimental result of time is shown in Figure 5-1 reveals that the proposed scheme outperforms the FP-growth and the Apriori approach.

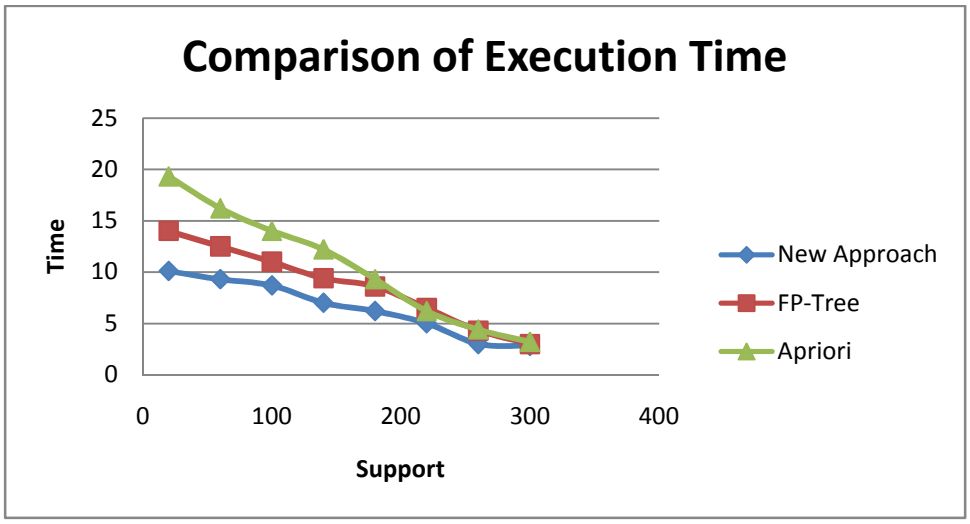


Figure 5-1: The Execution Time for Mushroom Dataset

As it is clear from the comparison new algorithm performs well for the low support value for the mushroom dataset which contains 8124 transactions and average length of items 23. But at the higher support its performance matches the FP-Tree and Apriori algorithms. Apriori performs with larger time. FP-tree produces the approximately same execution as of new approach in later stages.

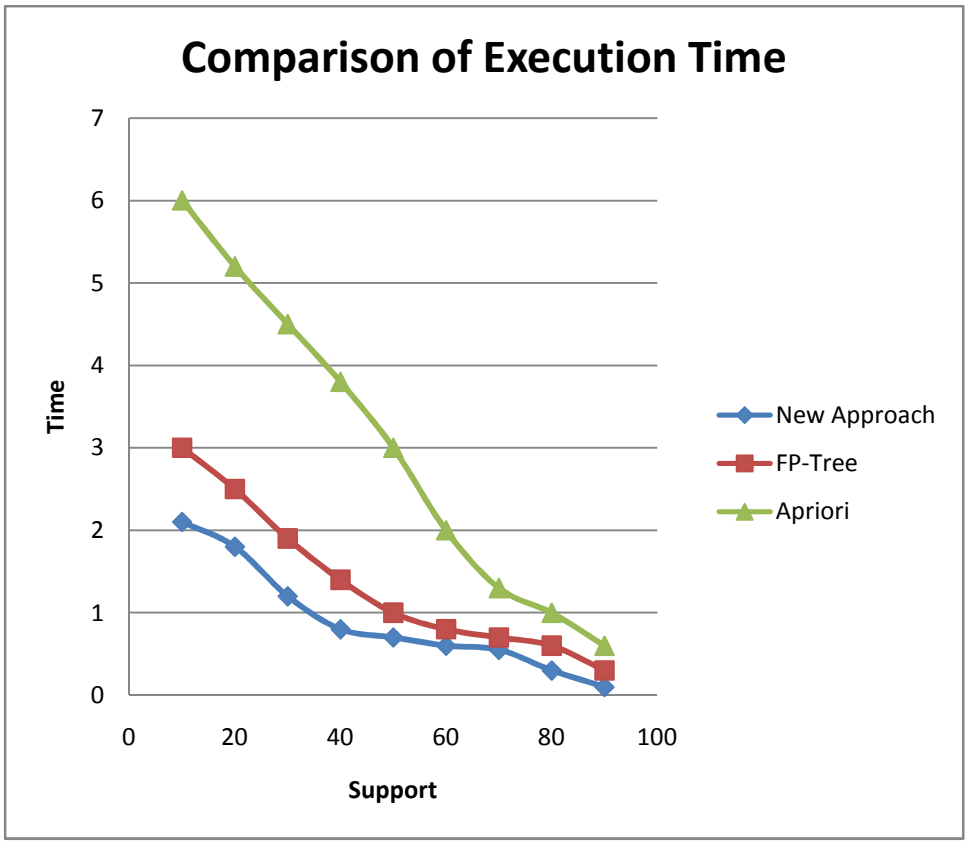


Figure 5-2: Execution Time for Artificial dataset

For the artificial dataset which contains the maximal frequent itemset in large amount shows better result with new approach as shown in figure 5-2 then FP-tree and Apriori algorithm. In the artificial dataset there are various transactions consider which occur repeatedly in the database and some transactions occur greater than the minimum support. The itemset remains for mining frequent itemset are mined with the help of second procedure whose complexity equals to the FP-Growth algorithm but due to procedure 1 the overall complexity reduce and become efficient.

5.1.2 Memory Comparison

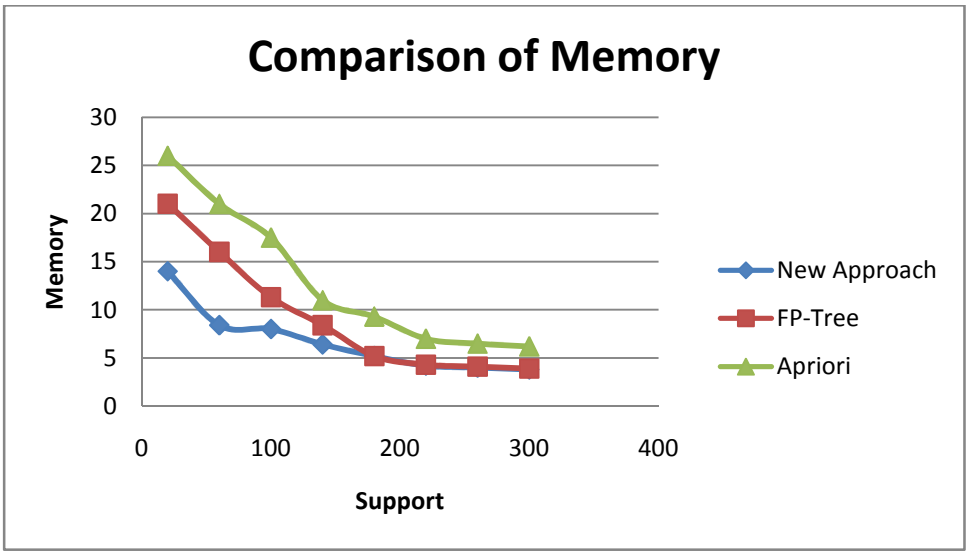


Figure 5-3: The memory usage at various support levels on Mushroom dataset

As it is clear from figure 5-3, the memory consumption for the Apriori algorithm is the highest at all level support because it produces candidate itemsets. The memory consumption for FP-tree at higher support levels is approximately same as the new approach because as the support increase the probability of finding the maximal itemset whose repetition is greater than the minimum support is less thus its working become same as the FP-Growth algorithm.

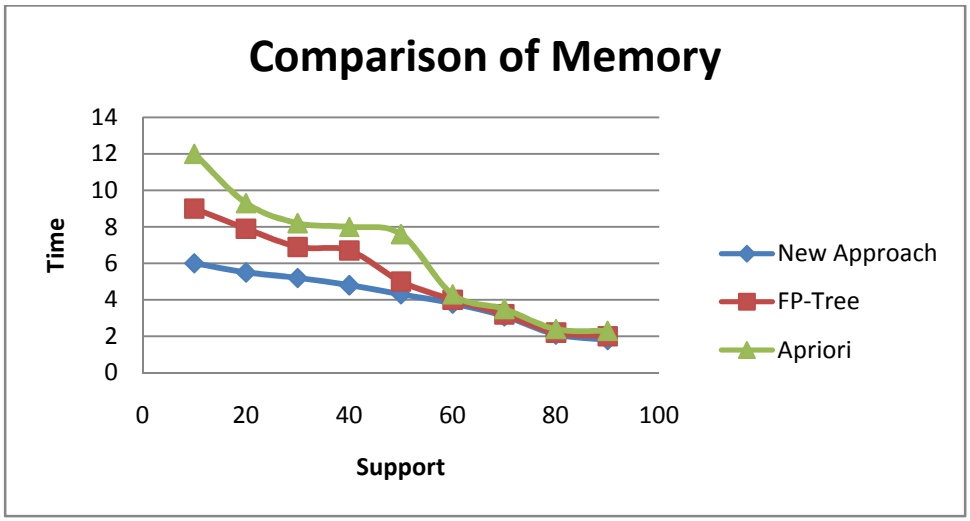


Figure 5-4: The memory usage at various support levels on artificial dataset

The above figure 5-4 shows the comparison takes place at sparse dataset, in sparse dataset data sets containing more enough zero entries (unmarked fields or items), in other words, the ratio number of fields / number of elements for the data database is smaller. Since the Apriori algorithm stores and processes only the non-zero entries, it takes the advantage of pruning most of the infrequent items during the first few passes. Therefore At high levels support the performances of our proposed scheme and Apriori are near. But at lower levels it shows that new approach performs well at all support level in consumption of memory. In this case also Apriori consume large amount of memory which is more than the FP-Tree and new approach due to its candidate generation problem. FP-tree approach performs better than the Apriori but not than the new approach.

We have performed several experiments to evaluate the performance of our scheme against FP-growth and Apriori, for generating the association rules. To perform the experiments different values of support were set because with different value of support the number of the frequent itemsets is different, and the running time and the memory consumptions are affected by the value of the support.

For both data sets the running time of our new scheme outperformed Apriori and FP-tree in the sparse data set as well as in dense dataset. The memory comparison for the new approach, Apriori and FP-Tree in the sparse dataset, at lower support the new approach performs well but at higher support Apriori and new approach performs approximately same as FP-Growth.

Therefore it is said from the above results the new approach which is combination of both improved Apriori and FP-Tree properties performs better than the individual Apriori and FP-Growth method.

CHAPTER 6

6. Conclusion and Future Research

Mining frequent itemsets for the association rule mining from the large transactional database is a very crucial task. There are many approaches that have been discussed; nearly all of the previous studies were using Apriori approach and FP-Tree approach for extracting the frequent itemsets, which have scope for improvement. Thus the goal of this research was to find a scheme for pulling the rules out of the transactional data sets considering the time and the memory consumption. This chapter summarizes the work done in this thesis and then the future scope is given.

6.1 Conclusion

In this thesis, we considered the following factors for creating our new scheme, which are the time and the memory consumption, these factors are affected by the approach for finding the frequent itemsets. Work has been done to develop an algorithm which is an improvement over Apriori and FP-tree with using an approach of improved Apriori and FP-Tree algorithm for a transactional database. According to our observations, the performances of the algorithms are strongly depends on the support levels and the features of the data sets (the nature and the size of the data sets). Therefore we employed it in our scheme to guarantee the time saving and the memory in the case of sparse and dense data sets. It is found that for a transactional database where many transaction items are repeated many times as a super set in that type of database maximal Apriori (improvement over classical Apriori) is best suited for mining frequent itemsets. The itemsets which are not included in maximal super set is treated by FP-tree for finding the remaining frequent itemsets. Thus this algorithm produces frequent itemsets completely. This approach doesn't produce candidate itemsets and building FP-tree only for pruned database that fit into main memory easily. Thus it saves much time and space and considered as an efficient method as proved from the results.

For both data sets the running time and memory consumption of our new scheme outperformed Apriori. Whereas the running time of our scheme performed well over the FP-growth on the collected data set at the lower support level where probability of finding maximal frequent itemsets is large and at higher lever running time is approximately same as the FP-Tree. The memory consumption is also approximately same as the FP-Tree at higher support and performed well at lower support.

The main contributions of this research:

We can summarize the main contribution of this research as follows:

- To study and analyze various existing approaches to mine frequent itemsets.
- To devised a new better scheme than classical Apriori and FP-tree alone using maximal Apriori and FP-tree as combined approach for mining frequent itemsets.

6.2 Future Trends

There are a number of future research directions based on the work presented in this thesis.

- Using constraints can further reduce the size of itemsets generated and improve mining efficiency.

- This scheme was applied in retailer industry application, trying other industry is an interesting field for future work.

- This scheme use Maximal Apriori and FP-Tree. We can use other combination to improve this approach.

Publications

- Bharat Gupta, Dr. Deepak Garg, Karun Verma " A Novel Approach to Mine Frequent Itemsets Using Maximal Apriori and FP-tree Method" International Journal of Advanced Computing (IJAC) ISSN: 0975-7686 pp.74-78 Vol 3, Issue 2 April, 2011.
- Bharat Gupta, Dr. Deepak Garg, Karun Verma " FP-Tree Based Algorithms Analysis: FP-Growth, COFI-Tree and CT-PRO" International Journal of Res Computeria, Scientia Publication ISSN 2230-9454 to be published in July.

References

[1] A. Savasere, E. Omiecinski, and S. Navathe. “An efficient algorithm for mining association rules in large databases”. In Proc. Int’l Conf. Very Large Data Bases (VLDB), Sept. 1995, pages 432–443.

[2] Aggrawal.R, Imielinski.t, Swami.A. “Mining Association Rules between Sets of Items in Large Databases”. In Proc. Int’l Conf. of the 1993 ACM SIGMOD Conference Washington DC, USA.

[3] Agrawal.R and Srikant.R. “Fast algorithms for mining association rules”. In Proc. Int’l Conf. Very Large Data Bases (VLDB), Sept. 1994, pages 487–499.

[4] Brin.S, Motwani. R, Ullman. J.D, and S. Tsur. “Dynamic itemset counting and implication rules for market basket analysis”. In Proc. ACM-SIGMOD Int’l Conf. Management of Data (SIGMOD), May 1997, pages 255–264.

[5] C. Borgelt. “An Implementation of the FP- growth Algorithm”. Proc. Workshop Open Software for Data Mining, 1–5.ACMPress, New York, NY, USA 2005.

[6] Han.J, Pei.J, and Yin. Y. “Mining frequent patterns without candidate generation”. In Proc. ACM-SIGMOD Int’l Conf. Management of Data (SIGMOD), 2000

[7] Park. J. S, M.S. Chen, P.S. Yu. “An effective hash-based algorithm for mining association rules”. In Proc. ACM-SIGMOD Int’l Conf. Management of Data (SIGMOD), San Jose, CA, May 1995, pages 175–186.

[8] Pei.J, Han.J, Lu.H, Nishio.S. Tang. S. and Yang. D. “H-mine: Hyper-structure mining of frequent patterns in large databases”. In Proc. Int’l Conf. Data Mining (ICDM), November 2001.

[9] C.Borgelt. “Efficient Implementations of Apriori and Eclat”. In Proc. 1st IEEE ICDM Workshop on Frequent Item Set Mining Implementations, CEUR Workshop Proceedings 90, Aachen, Germany 2003.

[10] Toivonen.H. “Sampling large databases for association rules”. In Proc. Int’l Conf. Very Large Data Bases (VLDB), Sept. 1996, Bombay, India, pages 134–145.

[11] Nizar R.Mabrouken, C.I.Ezeife. Taxonomy of Sequential Pattern Mining Algorithm”. In Proc. in ACM Computing Surveys, Vol 43, No 1, Article 3, November 2010.

[12] Yiwu Xie, Yutong Li, Chunli Wang, Mingyu Lu. "The Optimization and Improvement of the Apriori Algorithm". In Proc. Int'l Workshop on Education Technology and Training & International Workshop on Geoscience and Remote Sensing 2008.

[13] "Data mining Concepts and Techniques" by By Jiawei Han, Micheline Kamber, Morgan Kaufmann Publishers, 2006.

[14] S.P Latha, DR. N.Ramaraj. "Algorithm for Efficient Data Mining". In Proc. Int'l Conf. on IEEE International Computational Intelligence and Multimedia Applications, 2007, pp. 66-70.

[15] Dongme Sun, Shaohua Teng, Wei Zhang, Haibin Zhu. "An Algorithm to Improve the Effectiveness of Apriori". In Proc. Int'l Conf. on 6th IEEE Int. Conf. on Cognitive Informatics (ICCI'07), 2007.

[16] Q.Lan, D.Zhang, B.Wu. "A New Algorithm For Frequent Itemsets Mining Based On Apriori And FP-Tree". In Proc. Int'l Conf. on Global Congress on Intelligent System, 2009, pp.360-364.

[17] W.LIU, J.CHEn, S.Qu, W.Wan. "An Improved Apriori Algorithm. In Proc. IEEE International Conference, 2008, pp.221-224".

[18] S.P Latha, DR. N.Ramaraj. "Agorithm for Efficient Data Mining". In Proc. Int'l Conf. IEEE International Computational Intelligence and Multimedia Aplications, 2007, pp. 66-70.

[19] M. El-Hajj and O. R. Zaiane. "Inverted matrix: Efficient discovery of frequent items in large datasets in the context of interactive mining". In Proc. Int'l Conf. on Data Mining and Knowledge Discovery (ACM SIGKDD), August 2003.

[20] M. El-Hajj and O. R. Zaiane. "COFI-tree Mining:A New Approach to Pattern Growth with Reduced Candidacy Generation". Proceedings of the ICDM 2003 Workshop on Frequent Itemset Mining Implementations, Melbourne, Florida, USA, CEUR Workshop Proceedings, vol. 90, pp. 112-119, 2003.

[21] Y. G. Sucahyo and R. P. Gopalan. "CT-ITL: Efficient Frequent Item Set Mining Using a Compressed Prefix Tree with Pattern Growth". Proceedings of the 14th Australasian Database Conference, Adelaide, Australia, 2003.

[22] Y. G. Sucahyo and R. P. Gopalan. "CT-PRO: A Bottom Up Non Recursive Frequent Itemset Mining Algorithm Using Compressed FP-Tre Data Structure". In proc Paper presented at the IEEE ICDM Workshop on Frequent Itemset Mining Implementation (FIMI), Brighton UK, 2004.

[23] A.M.Said, P.P.Dominic, A.B. Abdullah. "A Comparative Study of FP-Growth Variations". In Proc. International Journal of Computer Science and Network Security, VOL.9 No.5 may 2009.

[24] Fayyad U. M., Piatetsky-Shapiro G. and Smyth, P. "Data mining to knowledge discovery in databases, AI Magazine". Vol. 17, No. 3, pp. 37-54, 1996.

[25] Tan P.-N., Steinbach M., and Kumar V. "Introduction to data mining, Addison Wesley Publishers". 2006

[26] Luo D., Cao L., Luo C., Zhang C., and Wang W. "Towards business interestingness in actionable knowledge discovery". IOS Press, Vol. 177, pp. 101–111, 2008.