Finding Frequent, Closed and Maximal Itemsets from Combined Items Database

Submitted By
Fakhr Ullah
389-FBAS-MSCS-F07

Faculty of Basic & Applied Sciences

Department of Computer Science

International Islamic University Islamabad.

A dissertation submitted to the Department of Computer Science, International Islamic University Islamabad, in partial fulfillment of the degree of MSCS 2012

1. Database design
2. Database management system

International Islamic University, Islamabad
Faculty of Basic & Applied Sciences
Department of Computer Science

Dated: *April 23*, 2012

FINAL APPROVAL

It is certified that we have read the thesis, entitled "**Finding Frequent, Closed and Maximal Itemsets from Combined Items Database**", submitted by Fakhr Ullah Reg. No. 389-FBAS/MSCS/F07 .It is our judgment that this thesis is of sufficient standard to warrant its acceptance by the International Islamic University Islamabad for MS Degree in Computer Science.

PROJECT EVALUATION COMMITTEE

External Examiner:

Dr. Hafiz Farooq
Associate Professor
NUST, Islamabad.

13/6/12

Internal Examiner:

Mr. M.Imran Saeed
Assistant Professor
Department of Computer Science,
International Islamic University,
Islamabad, Pakistan.

Supervisior

Mr. Muhammad Sarwar
Assistant Professor
ITM College Sargodha.

Sarwar

Co-Supervisor:

Mr. Shezad Ashraf Chaudhry
Lecturer
Department of Computer Science,
International Islamic University,
Islamabad, Pakistan.

Dedicated to

My parents and Teachers

Declaration

I hereby declare and affirm that this thesis neither as a whole nor as part thereof has been copied out from any source. It is further declared that I have completed this thesis entirely on the basis of my personal effort, made under the sincere guidance of our supervisor. If any part of this report is proven to be copied out or found to be a reproduction of some other, we shall stand by the consequences. No portion of the work presented in this report has been submitted in support of an application for other degree or qualification of this or any other University or Institute of learning.

Fakhr Ullah
389-FBAS/MSCS/F07

Acknowledgments

Nothing is possible to achieve without the help, guidance and prayers of others. If no one is helping you in your real work still many may be attached with you with their moral and spiritual supports. I was just blessed by God almighty with every type of support and guidance. Several individuals contributed to this work. I would like to express my sincere appreciation to my supervisor/Sir Mr. Muhammad Sarwar for his insight, guidance, and patience. He always has been a great source of aspiration in looking into the minute details, understanding the arguments and keeping focus in achieving the desired objectives of the research. It was just not possible to complete this work without him. I am thankful to my supervisor Sir Shezad Ashraf and especially to Dr. Ayyaz for his cooperation.

I would like to thank Bushra and Asia for their tremendous help at numerous times. Few of my other students named as Sana, Sidra and Munib also supported me in completing this work. Lastly, I am grateful to all those individuals who prayed for me and motivated me time by time. Many of my friends, family members and students are included in this list. If I want to name them all it will require another chapter.

Abstract

Data miming can be defined as a science of extracting useful information or facts from huge amount of data. Clustering classification, outlier analysis and association rule mining are different techniques used for data mining. Association rule mining is the process of analyzing data to find interesting correlation between different entities or items. The process of mining association rules can be summarized as: (i) decide what type of knowledge is required; (ii) use an algorithm to mine rules; and (iii) use some interestingness measures to separate interesting and uninteresting rules. These three steps are combined to form association rule mining.

Association rule mining is usually applied to large amount of data; therefore algorithm efficiency is very important aspect for association rule mining. A lot of research has been carried out to improve the efficiency of algorithm. Different database layouts and different algorithms for finding frequent, closed and maximal itemsets has been presented in order to achieve efficiency. Although different database layouts caused improvements in the efficiency of algorithms with respect to time and space but there is no special layout has been presented keeping in mind the way itemsets are constructed. We propose a database layout called CID (combined items database) that is specially designed for finding different types of itemsets efficiently. After presenting the CID layout we propose an algorithm called FFIUCID (Finding Frequent itemsets using combined items database) used to find out frequent itmesets from our proposed database layout. Experiments on various data sets show that CID layout and FFIUCID algorithm are more efficient with respect to time and space as compared to other layouts and algorithms.

Table of Contents

List of Tables

List of Figures:

Chapter 1: Introduction

Machines are playing an incredible role in human life and these have made human's life so easier and comfortable. In fact man is very much dependent on machines for performing various tasks in an efficient manner. Computers are such machines that have facilitated people in multiple tasks therefore with the invention of computers, computer scientists knew that this machine will make many human tasks much easier and cheap; this was the big reason that caused a rapid progress in this field and nowadays we are going through an age of computer technology.

Today, almost all of us use computer directly or indirectly due to its benefits and secondly due to rapidly decreased cost of hardware and software in the last decade. As the uses of computers are increasing day by day, data stored on computers is also increasing in volume as one can store all type of his useful data on computers. Permanent storage devices are very cheap therefore large amount of data can be stored on it.

It is human nature to learn from past experiences, incidents and events. Computer users wanted that their computers should also adopt this nature. As the growth of stored data on computer system is increasing day by day, the demands of the users of the computer system from the data they are storing is increasing as well. People want their computers to be more and more useful for them. People want that systems should learn, think and make decision for them. For centuries, manual procedures have been carried out for extracting patterns from data, old methods are not that much productive because the volume of data has been increased with the passage of time, therefore automated approaches are carried out. Early methods of identifying patterns in data include Bayes' theorem (1700s) and Regression analysis (1800s). Data Mining is the concept that is trying to fulfill the requirements of users in a more automatic way.

The concept of databases was introduced for storing data only. Analysis of data was done by a non-automatic ways. Users have to specify queries using a query language e.g. SQL for analysis purpose. Queries are not sufficient for high level analysis. The fast growing, abundant amount of data requires powerful data analysis tools. Intelligent decision making can be carried out if the hidden information can be extracted from the databases. Data mining is a process that is useful for analyzing large amounts of data.

This chapter is organized as follows. Section 1.1 presents various definitions of data mining. In section 1.2 we present various techniques used for data mining. Section 1.3 explains concept of association rule mining with examples.

1.1 Data Mining Definitions

Data mining can be defined as: (i) *The process of employing one or more computer learning techniques to automatically analyze and extract knowledge from large amount of data* (Roiger and Geatz, 2002), or (ii) *Data mining is the analysis of (often large) observational data sets, to find unsuspected relationships and to summarize the data in novel ways, that are both understandable and useful to the data owner* (Hand et al., 2001), or (iii) *It is the processes of discovering interesting knowledge from large amount of data stored in databases, data warehouses, or other information repositories* (Han and Kamber), or (iv) *Data mining is the use of automated data analysis techniques to uncover previously undetected relationships among data items*, or (v) *Data mining often involves the analysis of data stored in a data warehouse (Mike Chapple),* or (vi) Data mining is the transformation of large amounts of data in to meaningful patterns and rules, further it can be broken down in to two types directed and undirected. In directed data mining you are trying to predict a particular data point-the sales price of a house given information about other houses for sale in the neighbored hood. In Undirected data mining you are trying to create groups of data or find patterns in existing data-creating the "Sccer Mom" demographic group, for example in every U.S. censes is data mining as the government looks to gather data about everyone in the country and turn it into useful information. Association rule mining plays vital part in knowledge mining.

In modern age data mining is considered an important tool which transforms data into information. It is widely use in profiling practices, e.g. fraud detection, marketing, scientific discovery and surveillance. The term data mining was introduced in the 1990s. There are many terms that can be used to represent data mining concepts in different prospects such as knowledge extraction, knowledge mining, knowledge discovery, data dredging, pattern analysis, data archaeology and data analysis.

Data mining can be applied on different repositories e.g. relational databases, transactional databases, advanced database systems flat files, data warehouses, data streams. It

can be applied to any kind of data, to extract useful knowledge from it, which in turn will be useful in effective decision making for the organizations. Artificial intelligence, machine

learning and statistics concepts are also useful in data mining. The basic purpose of using data mining is to collect observations from behaviors. Those observations can further be used for the purpose of decision makings and predictions.

There are various applications of data mining but information industry has attracted it by a great deal in recent years. This is due to the wide availability of huge amount of data and the imminent need for turning such data into useful information and knowledge. Because this huge amount of data contains very much important knowledge for the organizations and that knowledge needs to be extracted from large data repositories. The knowledge and information obtained through data mining can be used in many fields such as market analysis, production control, engineering design, business management and science exploration.

1.2 Techniques Used for Data Mining

Concept of data mining was introduced in mid of 1990, since then; there are many techniques and procedures for data mining have been introduced. Experts spend 10 to 30 years in this field. It is true that data mining is not as simple as to run a function in word or excel etc. Data mining plays a great role to attract attention in many perspectives. It can be applied to any kind of data, to extract useful knowledge. Various techniques are used for data mining e.g. regression, clustering, classification, outlier analysis and association rule mining. In the next section we will briefly explain all these data mining techniques.

1.2.1 Regression

Regression is a statistical technique used for data mining. It uses a numerical data set and generates a mathematical formula according to the data. It is used for predicting future behavior. When a mathematical formula is computed from the given data then we can predict about any new instance by putting the data of the new instance in the computed formula. It has a limitation that it can apply only on continuous quantitative data such as age, weight and speed. Regression becomes insufficient if you have categorical data where order is not significant such as gender, name and color.

1.2.2. Clustering

The process of grouping a set of physical or abstract objects into classes of similar objects is called clustering (Han and Kamber). Clustering is a process in which we partition data elements or objects into meaningful groups called Clusters. In other words, it is a data mining technique in which we place data instances into related clusters having no advance knowledge about the properties of a cluster. The objects or items of one cluster are identical with the objects or items of that cluster and are different with the objects or items of all other clusters. So we can say that data elements or objects of same cluster have strong association with each other but data elements or objects of one cluster have weak association with data elements or objects of another cluster. Different clusters are linked with each other by either a single linkage or a complete linkage or an average linkage. Data segmentation is another name of clustering in some applications. Clustering can be a very much useful technique of data mining in various perspectives. Clustering is an example of unsupervised learning. Unsupervised Learning does not rely on predefined classes and does not learn through examples. In childhood we learn to distinguish between different objects through clustering, therefore it is an important human activity for learning. Clustering is used to understand natural grouping or structure in a data set.

1.2.2.1 Types of Clustering

Hierarchal, Non hierarchal, Partitioned, Density base and Subspace are the different types of clustering. These clustering are established by using different algorithms. We can get successive clusters using hierarchal algorithms on the behalf of previously established clusters. Hierarchal algorithms have two versions; (i) Agglomerative algorithms and (ii) Divisive algorithms. In agglomerative algorithms a successive larger cluster is made up by setting each element as separate elements and then merges these separate elements. In Divisive algorithm we get smaller cluster from the whole set. In hierarchal clustering, clusters are not defined by users predetermining number of clusters rather than clusters are defined by data. In Divisive algorithm we move clusters up and down in hierarchy to increase or decrease clusters.

Non hierarchal clustering is of two types; (i) Single pass methods and (ii) Reallocation methods. In Single pass method, as the name shows it passes through database for only one time to create clusters. In this way each record is only read from the database once. In Reallocation method, as the name shows there is movement of records from one cluster to another to create a

new cluster. It is true that reallocation method generates new clusters using many passes through databases but it is still relatively faster as compared to hierarchal techniques. In non-hierarchal algorithms end-users have to bind the number of clusters that are defined already. Predefining number of clusters rather than driven from data is not a good approach. For example if the total number of predefined cluster is 10 but from data we can define 12 clusters then clearly non hierarchal approach become inefficient.

Partitioned algorithms find all clusters at once. Arbitrary shaped clusters are defined by density base clustering algorithms. In density base technique a cluster can be seen as a region in which density of data elements exceeds a threshold for example DBSCAN and Optics.

Subspace clustering methods look for those clusters which are only related to each other due to some specific projection. Subspace clustering ignores irrelevant attributes. The general problem of this type is correlation clustering while the special case of axis parallel subspaces is known as Two-way clustering, Co-clustering or bi-clustering. In these types of clustering we can also cluster the feature of data elements as well as data elements. For example if a matrix (data element) is a cluster then its rows and columns (features of data element) are also cluster. However they usually do not work with arbitrary feature combinations as in general subspace methods. There are many additional techniques for clustering. Cluster analysis is also a hot area for research.

1.2.3 Classification

Classification is a data mining technique in which we predict group membership for data elements. We arrange the data into predefined groups. In other words classification is a process which divides dataset into mutually exclusive groups such that one group has those data instances which are as close as possible to one another. In addition dataset which belong to a particular group are as different as possible from other groups. In classification groups are made with respect to a specific variable i.e. there is database of many companies which are required to divide in groups, that are as homogeneous as possible with respect to credit worthiness variable with values good and bad. Classification has been used in various fields such as retail target marketing, fraud detection, and medical diagnosis. It is also used to predict future data trends. For example an email program can be classified into sent, inbox, spam, trash etc. mails. Another

example is we want to classify data element into group such as either a day (data element) is rainy, sunny or cloudy. Many classification methods are proposed in machine learning, statistics and pattern recognition. Well known algorithms for clustering are nearest neighbor, Decision Tree Learning, naive Bayesian classification and neural network. Rule-based methods, case–based reasoning, memory based reasoning, genetic algorithms, support vector machines, k-nearest-neighbor, fuzzy logic and rough sets are also approaches for classification.

1.2.4 Comparison of Clustering and Classification

Classification is similar to clustering in this way that classification also segments customer records into distinct segments called classes. But in classification end-user/analyst knows in advance that how classes are defined. For example to represent possibility that customer is default on loan can be defined by classes. In this case every record of customer must have a value for attribute used to define classes here that attribute on which classification will performed is loan. As end-user/analyst decides classes on attribute so that classification is less exploratory than clustering. In short, classification objective is not to explore data for new classes or segment, but it is used to classify new records.

1.3 Association Rule Mining

Association rule mining is used to uncover relationship between seemingly irrelevant data present in information repository or relational databases. Association rule mining uses the if/then statement for revealing relationship between data elements or object. An example of association rule is, if a customer buys a cigarette, then there is 70% chance that he will also buy match box. Basically, association rules have two parts. An antecedent is 'if' part of the rule and a consequent that is 'then' part of the rule. An antecedent is an item found in data and a consequent is an item found in data with respect to antecedent. Our daily life consists of many things which are attached with different other things. Finding attachment of few things with other few is sometime very useful for different purposes. For example, attachment of eid day with 'mithai' is useful for all sweats shops. Finding attachment or relation of an entity with another is named as association rule mining. Decision making becomes effective if we make accurate and early estimations. For effective decision making accurate and early estimation are essential. Association rule mining became a popular technique for data mining after the

published article of Agrawal in 1993. Association rule mining searches for relationships between variables. It is the process of analyzing data to find interesting correlations between different entities or items. *Association rule mining aims to extract interesting correlations, frequent patterns, associations or causal structures among sets of items in the transaction databases or other data repositories* (Zhao and Bhowmick, 2003). For example, if a customer buys beef and potatoes together, he or she can buy onions. It can be written as a rule in a formal way showing relationship among these items, beef $\wedge$ potatoes $\Rightarrow$ onion, these types of information can be used as the basis for decisions about marketing activities. Another example of association among different entities is if there are clouds and breeze is cold then there is 80% chance of rain. It can formally be written as clouds $\wedge$ cold-breeze $\Rightarrow$ rain. Market basket is also a well-known example of association rule mining in which we try to analyze that how customers buy different items which are associated with each other and place them in shopping basket.

1.3.1 Uses

The basic purpose of Association rule mining is to analyze or predict customer's behavior about purchasing different products. In market basket analysis example, we can plan marketing by making groups of those items that have probability to purchase on a single trip from a store by a customer. Advertising strategies can be planed. The discovery of useful correlations can help in many business decision-making processes, such as catalog design, customer shopping and cross marketing behavior analysis. Knowledge of associations between different entities can be applied for a variety of tasks. For example if we know that certain items are purchased together, we can put them in close proximity in our store to boost their sale. If we know that certain diseases are often found together, we can plan a combined approach to prevent them. Association rules are used to build program those have machine learning ability. Machine learning is a branch of artificial intelligence used to build programs those are efficient and have ability to learn without explicit programming. Association rule mining has successfully been applied to support other data mining tasks. Like classification, clustering, filling missing values and to analyze data in various fields e.g. credit card transactions, telecommunication purchase behavior, banking services, insurance claims, medical patient history, genome data, genetic engineering and text documents.

1.3.2 Formal Example of Association

Association rule mining is typically applied to transactional databases. Given below is a database that represents records of customers at a bakery. Each row of the table consists of a transaction ID and items purchased.

Table 1.1: Records of Customers

TID	Items Purchased
1	Bread, Eggs
2	Bread, Milk, Eggs
3	Coke, Biscuits
4	Bread, Eggs, Coke
5	Bread, Diaper

By analyzing the above data, we can say that the customer who purchased Bread is likely to purchase Eggs. Alternatively, we can write Bread $\Rightarrow$ Eggs; this is called an association rule.

In above example, bread and milk may have also a relation, now it is time to distinguish that which rule or relation will be useful and which will not. Not all mined rules are useful. We have to set criteria that will tell whether a rule will be useful or not. The most well-known criteria's that helps us in this regard are support and confidence. A rule is useful if and only if it has a specified minimum support and confidence.

1. Support is used to find out all frequent itemsets from a database.

2. Confidence is used to form rules.

Finding all possible itemsets is difficult because set of possible itemsets is power set over I and has size 2n-1excluding empty set. The power set will increase if input is increased so that downward clause property (anti-monotonicity) of support guarantees that all those items are frequent which are subset of frequent item set and all super set of an infrequent item set will be infrequent. Let A $\Rightarrow$ B be an association rule, where A and B can be a single item or a set of items. *Support* of $A \Rightarrow B$ is the probability that A and B appear together in the database. *Confidence* of $A \Rightarrow B$ is the probability of occurrence of B given that the transaction contains A.

A rule is said to be interesting if its support is greater than a predefined minimum support as well as its confidence is greater than a predefined minimum confidence. Detailed description of support and confidence will be given later in this chapter.

1.3.3 Formal Definition of Association Rules

Let $I = \{a_1, a_2, ..., a_m\}$ be a set of items, and $DB = \{T_1, T_2, ... T_n\}$ be a transaction database, where $T_i (i \in \{1, 2, ..., n\})$ is a transaction which contains a set of items in I.

K-itemset: If $X \subseteq I$ and $|X| = k$ then X is called a k-itemset.

Support of an itemset: If X is an itemset, then the percentage of transactions containing X is called support of X. Support can also be defined as the probability of occurrence of X, denoted as $P(X)$.

Frequent itemset: If X is an itemset and support of X is greater than a predefined minimum support threshold (*minsup*) then X is called a frequent itemset.

Association rule: If $X \subseteq I$, $Y \subseteq I$ & $X \cap Y = \phi$, then an implication of the form $X \Rightarrow Y$ is an association rule. X is called antecedent and Y is called consequent.

Support of $X \Rightarrow Y$ is the number of transactions containing both X & Y. Support of an association rule can also be defined as the probability of occurrence of X and Y together, denoted as $P(X, Y)$

Confidence of $X \Rightarrow Y$ is the percentage of transactions containing X that also contains Y. Confidence of $X \Rightarrow Y$ can also be defined as the probability of occurrence of Y given X.

$$Support(X \Rightarrow Y) = P(X, Y)$$

$$Confidence(X \Rightarrow Y) = \frac{P(X, Y)}{P(X)}$$

If support is greater than a predefined minimum support (threshold), and its confidence is greater than a predefined minimum confidence (threshold) then association rule is said to strong or *interesting or useful.* Support and confidence are called *interestingness measures.*

Let the item set is: I = {U, V, W, X, Y, Z}, let minimum support and confidence are 50%. Database is given in table 1.2, where each row consists of a transaction and each transaction consists of items from set I. Each transaction has a unique identifier named as TID.

Table 1.2: Transactional Database

TID	Items
1	U,V,Z
2	U,V,X,Z
3	W,X,Y,Z
4	U,V,X
5	U,V,W,X,Z
6	V,X,Z
7	U,V,Y,Z
8	W,X,Z
9	U,V,Z
10	V,X,Z

Now we will calculate the support of each itemset and will construct frequent one itemsets from table 1.2. As the occurrence of W and Y is 3 and 2 respectively therefore their support is less than 50% and both will not be included in the table of frequent one itemsets. We will show the results of frequent one itemsets in table 1.3

Table 1.3: Frequent one itemsets.

Itemset	Support
U	60%
V	80%
X	70%
Z	90%

Now according to given support we will construct frequent two itemsets. Frequent two itemsets will be made using frequent one itemsets only. Combined support of U and X is less than minimum support; therefore this combination will not be included in two itemsets. Results of frequent two itemsts are shown in table 1.4.

Table 1.4: Frequent two item sets.

Itemset	Support
U,V	60%
U,Z	50%
V,X	50%
V,Z	70%
X,Z	60%

Frequent three itemsets will be constructed with the help of results shown in table 1.4. Table of frequent three itemsets will have only a single row because it has only one combination that has support greater or equal to minimum support. Results of frequent three itemsets are shown in table 1.5.

Table 1.5: Frequent three itemset.

Itemset	Support
U,V,Z	50%

Frequent four itemsets will be made using the results of table 1.5 but none of the 4-itemset is frequent because there is only one row in table 1.5. We can manually check that it is true by looking at table 1.2. None of the 4-itemset has support greater than the minimum support in the given example.

1.4 Problem Statement

A lot of work has been done by different researchers in the field of association rule mining. There are some issues that must be considered while proposing a new algorithm for association rule mining because these can degrade the performance of the algorithms. All the existing algorithms work on the databases stored in the form of horizontal, vertical and diffset layouts. There is no special format of the database that is made up for finding frequent itemsets.

Main problem identified is that all the existing methods for storing databases does not support finding frequent, closed and maximal itemsets efficiently therefore there should be a special method for storing database that should help in finding frequent itemsets efficiently.

Chapter 2: Database Layouts and proposed database Layout

This chapter is organized as follows. Section 2.1 describes Horizontal layout with example. Section 2.2 presents Vertical layout with example. In section 2.3 we present Diffset layout with example. In the last section 2.4 we will discuss our proposed layout with description.

In this chapter, we will look for different ways to organize data in databases. Performance of data mining algorithms heavily depends upon the organization of data in database. Different layouts are introduced just for the sack of efficiency of algorithms. There are both advantages and disadvantages to these methods and we will distinguish between them in this chapter. Depending on the business requirements and development needs, various ways of storing date can be implemented.

In mining, calculating support is a vital part for almost all algorithms of finding frequent itemsets. A better layout for mining algorithms is the one which efficiently calculates support for itemsets. In the following sections we will discuss different layouts for organizing data in database.

2.1 Horizontal Layout

This is the most simple and well known method for storing databases. Horizontal layout used in almost all the general purpose databases which are commonly used in our markets for storing records. This layout traditionally stores data into tables. In this method a new row is inserted for each new record. Each transaction is considered as a record and it is given a unique transaction identifier (TID). All items of transactions are written in front of respective transaction identifier. Table 2.1 shows the horizontal data layout for a shop that contains the items related to sports. Database contains total five items and eight transactions. Transaction ID's are represented by digit values.

Table 2.1: Horizontal Layout

TID	Items
1	Bat, Ball, Racket, Shirt
2	Bat, Ball, Racket, Shuttle
3	Racket, Shirt
4	Bat, Ball, Racket, Shuttle, Shirt
5	Bat, Ball, Racket, Shuttle
6	Bat, Shirt
7	Bat, Ball, Racket
8	Bat, Ball, Racket, Shirt

Horizontal layout is one of the most common and basic structure for storing data into databases, therefore a number of algorithms for finding frequent itemsets have been presented that work on this layout. The most well-known algorithm for finding frequent itemsets named as "Apriori" also works on horizontal database layout. Many other algorithms that work on this layout will be introduced later.

Traditional horizontal layout, with columns and predetermined structure, has a number of advantages as well as disadvantages. For example, if a user designs dynamic form screen and there is no restriction on number of fields and field names. In this situation, it is easy for the user to create a new field, give it a name and assign a value whenever the user requires a new field.

On the other hand, in modern age due to progress in E-commerce, horizontal layout becomes insufficient as in above example of form screen just the solution in commerce is to add a new field with name and new product is inserted in this field. As in horizontal layout for each new product we have to create a new field and value is inserted in this field. So, maximum column count can exceed. Maximum column exceed becomes cause of 90% null saturation in a database table. Performance issues are also encountered when database has a large number of columns and when query returns only a few columns.

For association rule mining horizontal layout is not too much efficient. Almost all algorithms calculate support for items. Calculating support for items in horizontal layout is relatively inefficient process. Let's consider database in table 2.1, for example if we want to

calculate the support for bat, we have to check the first row of table and if first row contains bat we have to add one in support of bat, similarly then we will have to move to next row and repeat the same process. We have to repeat the same process until all the rows of the table will be examined. In our example support for item bat is 7 because bat is present in 7 transactions and we have calculated this support after examining all the 8 transactions. So we can say that if there are total *n* transactions we have to examine *n* transactions for the sack of calculating support for any item.

2.2 Vertical Layout

Horizontal layout is not effective to find frequent item sets as described in above section. Therefore, researchers proposed a new method for storing database that was a better method for association rule mining algorithms.

Vertical layout is an opposite way as compared to horizontal layout. As the name suggests it keeps record of all transactions that has a particular item. Vertical Layout has its own advantages as well as disadvantages. Table 2.2 shows the vertical data layout. All the transactions that contain particular items are given in the list of that item.

Table 2.2: Vertical database layout.

Bat	Ball	Racket	Shuttle	Shirt
1	1	1	2	1
2	2	2	4	3
4	4	3	5	4
5	5	4		6
6	7	5		8
7	8	7		
8		8		

Finding support is an efficient process in vertical database layout. Support can easily be computed by intersection operation; therefore, it increases efficiency as compared to horizontal layout. For example, support of bat is clearly 7 because there are 7 transactions Id's in the bat's

list. Similarly, support of combined two items can also be calculated very efficiently by an intersection operation only. For example, combined support of bat and ball can be calculated as: if we consider bat and ball two sets than Bat = {1,2,4,5,6,7,8} and Ball = {1,2,4,5,7,8} then Bat ∩ Ball = {1,2,4,5,7,8}, here is the combined support of bat and ball. Transaction 1, 2, 4, 5, 7 and 8 contains both bat and ball, therefore combined support of these two items will be 6. From here we can conclude that finding support is a faster process in vertical layout as compared to horizontal layout. Compressed Binary Mine (CBMine), Bodon's Apriori and MAFIA are few algorithms that works on the basis of vertical database layouts.

On the other hand there are few shortcomings of vertical layout. Vertical table lost strong data typing. In vertical layout each value of a column must match with only one type. For example if a column data type is varchar then all values must be converted into varchar, to store under this column, as well as, this data type must be consistent during safe and retrieve operation.

Data consistency is a major drawback of vertical layout. Column names are entries in key column which assist uses and application to store identical data by using different key columns.

For example one user can create a new column and save it as "organization" with value of "oracle" but n the other hand, another user create a new column and save it as "company" with value of "oracle". Writing queries in vertical layout is relatively harder than horizontal layout.

We can conclude that although vertical layout has few drawbacks. But for association rule mining algorithms vertical layout is better as compared to horizontal layout. It leads to better performance and increases efficiency of algorithms.

2.3 Diffset Layout

Mining frequent patterns on the vertical data structures usually shows improvements of performance over the classical horizontal structure. This is because the vertical data structure supports fast frequency counting via intersection operations on transaction identifiers. Diffset is a vertical data representation that only keeps track of the difference in the transaction identifiers.

To illustrate the concept of Diffset structure, let's assume a database with five transactions with their transaction identifiers from 1 to 5 and set of items is denoted by alphabets from A to E. Formally DB = {1, 2, 3, 4, 5} and I = {A, B, C, D, E} be a set of five different items in the database. Table 2.3 depicts a common data format that has been used often in mining associations. In this traditional horizontal approach, each transaction has a TID along with the itemsets comprising the transaction. In contrast, the vertical format maintains for each item its tidset, the set of all TID's where the item occurs.

The Diffset structure is based on the concept of vertical database format. It avoids storing the entire tidset of each item by keeping track of only the differences in the TID's between the tidset of each item and the whole tidset (the set of all TID's). These differences in TID's are stored in what has been called the diffset. Pseducode for diffset is as given below:

2.3.1 Pseducode for Calculating Diffset

Diffset(dataset)
1. *(for each) item X*
2. *(for each transaction Y*
3. *(SIf X is not present in Y*
4. *Add transaction Identifier (TID) of Y to diffset of X*

In the following section we will construct diffset layout from a simple transactional database. Table 2.3 show a simple transactional database from which we are going to construct diffset layout.

Table 2.3: Transactional Database

TID	Item Bought
1	A,B,D,E
2	A,C,E
3	A,B,C,D,E
4	C,D,E
5	B,E

Now when we consider item A, we will check all the transactions turn by turn. When we checked transaction 1 we came to know that item A is present in this transaction, therefore we will not add TID of this transaction in the diffset for item A. After checking all the transactions regarding to item A we came to know that only fourth and fifth transaction does not contain item A, so we will add TID's of fourth and fifth in diffset of item A. This procedure is repeated for all items. Final result after checking all the transactions regarding each item is show in table 2.4.

Table 2.4: Diffset Layout

A	B	C	D	E
4	2	1	2	
5	4	5	5	

Clearly, diffset drastically cut down the size of memory required. From above table we can note that diffset is utilizing less memory as compared to horizontal as well as vertical layout. Different algorithms are proposed for finding frequent, closed and maximal itemsets from diffset. For example Novel Vertical Mining on Diffset Structure for finding frequent itemsets, Genmax for finding maximal itemsets and CHARM for finding closed itemsets. In the next section we will explain our proposed layout of database with example.

2.4 Proposed Layout

The most commonly used layouts are horizontal, vertical and diffset. However, these methods are not too much helpful for finding frequent itemsets. We propose a new technique called CID (Combined Items Database) for storing databases. We have focused that proposed method should helpful in finding frequent, closed and maximal itemsets efficiently.

The concept of diffset is already explained above. Our proposed method works on the basis of diffset. Diffset becomes the input for our proposed method therefore we will calculate diffset before proceeding towards our proposed method. Our proposed layout made different groups from diffset layout. Identical groups are made regarding each item. Each group has different members. We combine items to form members depending upon their existence with each other. These groups are helpful in finding frequent, closed and maximal itemsets efficiently. We can say that our proposed method is specially made for finding frequent itemsets. Our

method will be lossless because frequent, closed and maximal itemsets can be constructed from our proposed database structure. We will show that our proposed database structure will use less space as compared to many existing methods for storing databases. We will show that our proposed algorithms for finding frequent itemsets is also efficient as compared to many existing algorithms for finding frequent. In the previous section we have explained the method for constructing diffset layout. Pseducode for CID as given below:

2.4.1 Pseducode for Combined items database

```
CID(Diffset)
1.  for each X in diffset
2.      for each Y in diffset
3.          Z=Y-X
4.          If ((total transaction-(/z/+/x/))>=minsup )
5.              If (Any existing member has same Tidlist as of Z)
6.                  Add Y in that member
7.              Else
8.                  Make a new member XY and add Z in Tidlist of XY
```

Now we are going to explain the working of our algorithm. Our objective is to combine the items in different sets on the basis of their occurrences with each other. First and second lines of the algorithm are nested loops. These lines are included in the algorithm because we have to check every item in the diffset with respect to every other item in the diffset therefore nested loops are used to implement this concept. Total number of groups will depend upon the total number of items in the diffset. We will combine those items with each other that have any chance of making frequent itemsets in the future.

Our procedure is very simple we use the set difference operation in order to make the decision about combing the items. Every item is considered as a set and TID's (transaction identifiers) written in front of each item is considered as an elements for each set. Line number three of the algorithm performs set difference operation. Set Z contains those TID's in which X and Y are not jointly present excluding those in which X is not present. We calculate the total number of transactions in which X and Y are not present in line number four. TID's list of item X is representing those transactions where item X is not present. TID's list in Z contains those

transactions in which item Y is not present excluding those transactions in which X was not present. Length of X and Z is representing the total number of transactions in which X and Y are not jointly present. If the total number of transactions where X and Y are jointly absent is greater than minimum support than we will not join item X and Y.

We make different members in a single group. If X and Y are joined in one member, the decision to put any other item in that member depends upon the TID's set difference of that particular item with the item w.r.t. we are making members. If the set difference of that particular item is matched with any member TID's list then that item is joined with that particular member. Otherwise, a new member is made. Last four lines of the algorithm are implementing this idea. In order to understand this concept clearly we will explain it using an example.

Example

In this example we will explain that how our proposed database layout is made. We will start from a simple database in a horizontal format. First of all, we will make diffset layout from the horizontal view. After calculating diffset we will show how our proposed method will be constructed from diffset. We start from a simple database; table 2.5 is a simple database having 6 transactions.

Table 2.5: Transactional Database

1	A	B	C	D	F	G	
2	C	D	E	F	H		
3	A	B	C	D	F	G	
4	A	C	E	F			
5	A	B	C	D	E	F	G
6	B	D	E	G	H		

First of all we will calculate diffset from the database in table 2.5. As the method for calculating diffset is already been explained therefore, here we will not explain it. We will simply show the results after calculating diffset structure. Table 2.6 shows the diffset layout of database.

Table 2.6: Diffset Layout

A	B	C	D	E	F	G	H
2	2	6	4	6	2	1	1
6	4				4	3	3
							4
							5

First step after calculating the diffset is to remove those items from the diffset layout that has support less than a minimum predefined support. Let the minimum support threshold is 3, then item H should be removed because H is the only item having less support than the minimum support. Reason for removing item H is very simple. Item H is not frequent therefore any item combined with H will not be frequent too. Table 2.7 shows the diffset layout after removing item H from the list.

Table 2.7: After removing item H

A	B	C	D	E	F	G
2	2	6	4	6	2	1
6	4				4	3

From diffset layout we will split our database into different groups. For example for item A, we will combine any other item with A if it has a specified number of occurrences with A and we will add TIDs (transaction identifier) in tidlist of that member where combined item is not present and A is present. Now ABDG are in 3 transactions and TID 4 is the only transaction where BDG is missing. We are talking about item A therefore no need to concern about transaction 2 and 6 because both do not contain item A. Given below is detailed layout of our proposed method on table 2.7.

Item A: Group	ABDG 4, ACF Null
Item B: Group	BCF 6, BDG Null
Item C: Group	CD 4, CF Null, CG 2 4, CE 1 3
Item D: Group	DF 6, DG 2, DE 1 3
Item F: Group	FG 2 4, FE 1 3

This method is specially made for finding frequent items because the structure of the database is almost telling about the frequent itemsets. Finding frequent itemsets will be an efficient task from this structure. As it is mentioned earlier that we will propose methods for finding frequent, closed and maximal itemsets, therefore we can says that it is a lossless method for storing database.

Chapter 3: Interestingness Measures and Types of Itemsets

Association rule mining is implemented with the help of interesting measures. In this chapter we will discuss interesting measures and their types. Section 3.1 will explain different interesting measures used in finding associations among the items. Different types of itemsts are used in association rule mining. Section 3.2 describes these itemsets.

Association rule mining can generate large quantity of rules, most of which are not interesting to the user, they may have redundant information and thus all of them cannot be used directly for an application. Pruning is necessary to get very important rules or knowledge. The difficult task is discovering knowledge or useful rules from the large number of generated rules. There should be some measures for filtering rules.

To mark a rule as interesting or uninteresting we need some criteria. The criterion for selection of interesting rules is known as an interestingness/usefulness measure. The most commonly used interestingness measures are support and confidence. (Agrawal et al., 1993)

Interestingness measures are used to find the truly interesting rules. As it is mentioned earlier that there may be many rules but we have to select only the useful ones for association rule mining. Support and confidence are most well-known interesting measures. Support is used in almost all algorithms of association rule mining and it is the base for all other measures. Beside support and confidence, various other measures have been used by researchers.

Interesting measures can be divided into two categories: objective measures based on the statistical strengths or properties of the discovered rules, and subjective measures which are derived from the user's beliefs or expectations of their particular problem domain. Interestingness measures used in association rule mining are usually borrowed from other fields such as statistics (e.g. Correlation coefficient) and information theory (e.g. Gain). In the following sections, we introduce some well-known objective interestingness measures.

3.1 Interesting Measures

There are many perspectives according to which we can construct different type of itemsets in association rule mining. Brief introduction of these measures are given below.

3.1.1 Support

Support is defined on itemsets and gives the proportion of transactions which contains item. It is used as a measure of significance (importance) of an itemset. Since it basically uses the count of transactions it is often called a *frequency constraint.* The support measure defined as Support $(X \Rightarrow Y) = P(X,Y)$ was introduced by Agrawal et al. (1993). It is a symmetric measure and lies in the range [0, 1]. An itemset with a support greater than equal to a set of minimum support threshold is called a *frequent or large itemset.*

It is defined as the probability of occurrence of an itemset in the database. If the database contains n transactions and an itemset X appears is r transactions then support of X will be r/n. Thus, support is the actual or calculated probability of occurrence of X in the database. Support provides level of significance of rule i.e. a rule having higher support is said to be more significant than a rule having less support. An itemset whose support is greater than a user specified minimum support threshold is called a *frequent itemset.*

One advantageous property of frequent itemsets is *downward closure* property i.e. subsets of a frequent itemset are also frequent, or super sets of infrequent itemsets cannot be frequent. Thus if we know that a certain itemset is infrequent we can ignore all its super sets. This results in a reduced search space for searching interesting rules. Downward closure property is exploited by an association rule mining algorithm called Apriori (Agrawal and Sirikant, 1994) to speed up the task of mining. There is also a drawback of using support as an interestingness measure: itemsets with low support are pruned even if they satisfy other measures. In some situations, low support itemsets may also be important e.g. in document clustering and information retrieval, a high frequency term is not as useful in calculating relevancy between documents as compared to a low frequency term. High frequency terms often represent stop words or other commonly used terms, whereas a low frequency term often represents some terminology or special word. In general, a high support itemset may represent common sense knowledge, whereas a low support itemset may represent unusual information. In general we can summarize support as one of most important interesting measure from all other interesting measures for association rule mining.

3.1.2 Confidence

Confidence is defined as the probability of seeing the rule's consequent under the condition that the transactions also contain the antecedent. Confidence is directed and gives different values for the rules **X => Y** and **Y => X**.

Confidence is not down-ward closed. The confidence measure defined as Confidence $(X \Rightarrow Y) = \frac{P(X,Y)}{P(X)}$, was introduced by Agrawal et al. (1993). It is an anti-symmetric measure and lies in the range [0, 1]. Support is first used to find frequent (significant) itemsets exploiting its down-ward closure property to prune the search space. Then confidence is used in a second step to produce rules from the frequent itemsets that exceed a minimum confidence threshold.

A problem with confidence is that it is sensitive to the frequency of the consequent (**Y**) in the database. Caused by the way confidence is calculated, consequents with higher support will automatically produce higher confidence values even if there exists no association between the items.

Confidence is the ratio of calculated probability of occurrence of X and Y together and the calculated probability of occurrence of X. Thus, it is the expected probability of occurrence of Y in the transaction, given that the transaction contains X.

Support and confidence alone do not ensure rules with a real interest (Lallich et al., 2006). Confidence is never the preferred method to compare association rules since it does not account for the baseline frequency of the consequent (Brijs et al., 2003). To illustrate consider the data given in table 3.1, which represents the sale of computers and printers, in the form of a contingency table.

Table 3.1: Sale of printers and computers.

	Computer	~Computer	Total
Printer	2000	1000	3000
~Printer	1750	250	2000
	3750	1250	5000

Consider the rule Printer⇒ Computer.

Support (Printer) = 3000/5000=3/5=60%

Support (Computer) = 3750/5000 = 3/4 = 75%

Support (Printer⇒ Computer) = 2000/5000 = 2/5 = 40%

Confidence (Printer⇒ Computer) = (2/5)/ (3/5) = 66%.

Since Confidence (Printer⇒ Computer) = 66%, a reasonably high value, one may get an impression that sale of printer influences the sale of computer. Let us analyze the situation. Overall 75% of customers are buying computer. Confidence (Printer⇒ Computer) = 66% which is less than the overall support of computer (75%). Thus, purchase of printer actually reduces the probability of buying computer from 75% to 66%. This means that printer and computer are actually negatively correlated. However, confidence is unable to capture this negative correlation and is providing a misleading result. From the above example, we can say that if support of consequent is high, then value of confidence will usually be high, even though antecedent and consequent are negatively correlated. Thus, we can say that support and confidence may be misleading in some situations.

3.1.3 Lift

The lift measure defined as Lift $(X \Rightarrow Y) = \frac{P(X,Y)}{P(X)P(Y)}$, was introduced by (Brin et al. 1997). It is a symmetric measure and lies in the range [0,∞]. It was originally called interest.

Lift measures how many times more often X and Y occurs together than expected if they were statistically independent. Lift is not down-ward closed and does not suffer from the rare item problem. Also lift is susceptible to noise in small databases. Rare itemsets with low counts (low probability) which per chance occur a few times (or only once) together can produce enormous lift values.

Since $P(X,Y)$ is the calculated probability of occurrence of X and Y together. And $P(X)P(Y)$ is the expected probability of occurrence of X and Y together, therefore lift is a ratio of actual and expected probabilities of occurrence of X and Y together. If this ratio is less than one, then actual probability is less than the expected probability, and therefore X and Y are

negatively correlated. If the ratio is greater than one, then actual probability is greater than the expected probability. Therefore, X and Y are positively correlated. If value is equal to one then X and Y are independent.

3.1.4 Conviction

Conviction was developed as an alternative to confidence which was found to not capture direction of associations adequately. Conviction compares the probability that X appears without Y if they were dependent with the actual frequency of the appearance of X without Y. In that respect it is similar to lift, however, it contrast to lift it is a directed measure since it also uses the information of the absence of the consequent. An interesting fact is that conviction is monotone in confidence and lift. The conviction measure defined as $\text{Conviction}\ (X \Rightarrow Y) = \frac{P(X)P(\overline{Y})}{P(X,\overline{Y})}$, was introduced by (Brin et al 1997). It is an anti-symmetric measure and lies in the range [0,∞].

3.1.5 Leverage

Leverage measures the difference of X and Y appearing together in the data set and what would be expected if X and Y where statistically dependent. The leverage measure defined as $\text{Leverage}\ (X \Rightarrow Y) = P(X,Y) - P(X)P(Y)$, was introduced by Shapiro (1991). It is a symmetric measure and lies in the range [−∞, 1]. The rational in a sales setting is to find out how many more units (items X and Y together) are sold than expected from the independent sells. Using minimum leverage thresholds at the same time incorporates an implicit frequency constraint. E.g., for setting a minimum leverage thresholds to 0.01% (corresponds to 10 occurrence in a data set with 100,000 transactions) one first can use an algorithm to find all itemsets with min. support of 0.01% and then filter the found item sets using the leverage constraint. Because of this property leverage also can suffer from the rare item problem.

Leverage is also known as Piatetsky Shapiro's measure. Leverage is the difference between calculated and expected probabilities of occurrence of X and Y together. A positive value indicates that calculated probability is greater than expected, so antecedent and consequent are positively correlated. A negative value indicates that actual probability is less than expected.

So antecedent and consequent are negatively correlated. A drawback of leverage is that it cannot be used to find relative importance of itemsets. Consider the following database:

Table 3.2: Transaction database for leverage.

TID	Items
1	X, Y
2	X, Y
3	X, Y
4	X, Y
5	X, Y
6	X, Y
7	X, Y
8	A, B, Y
9	A, B
10	B

$P(X)$ =0.7, $P(Y)$ =0.8, $P(X, Y)$ =0.7 and Leverage ($X \Rightarrow Y$) = 0.14

$P(A)$ =0.2, $P(B)$ =0.3, $P(A, B)$ =0.2 and Leverage ($A \Rightarrow B$) = 0.14

From the above calculations we see that leverage has given equal weight to both $X \Rightarrow Y$ and $A \Rightarrow B$. However, just looking on the above database it is clear that X and Y are more highly correlated than A and B.

3.1.6 Coverage

Coverage is called antecedent support. It measures how often a rule X > Y is applicable in a database. The coverage measure is defined as Coverage ($X \Rightarrow Y$) = $P(X)$. It is an anti-symmetric measure and lies in the range [0, 1].

3.1.7 Correlation

The correlation measure is defined as Correlation $(X \Rightarrow Y) = \frac{P(X,Y) - P(X)P(Y)}{\sqrt{P(X)P(Y)(1-P(X))(1-P(Y))}}$. Borrowed from statistics, it is a symmetric measure and lies in the range [−1, +1].

It is a measure of linear relationship between X and Y. If value is near −1 then X and Y are negatively correlated. If value is near +1 then X and Y are positively correlated. A value near zero indicates that X and Y are independent.

3.1.8 Odds Ratio

The odds ratio measure is defined as Odds Ratio $(X \Rightarrow Y) = \frac{P(X,Y)P(\overline{X},\overline{Y})}{P(X,\overline{Y})P(\overline{X},Y)}$. Borrowed from statistics, it is a symmetric measure and lies in the range [0,∞].

It considers the presence and absence of both antecedent and consequent. A value near zero means X and Y are independent. If value is ∞ (or some very large value) then X and Y are strongly correlated.

3.1.9 Cosine

The cosine measure is defined as Cosine $(X \Rightarrow Y) = \frac{P(X,Y)}{\sqrt{P(X)P(Y)}}$. Borrowed from statistics, it is a symmetric measure and lies in the range [0, 1].

Cosine is very similar to lift, except for the presence of square root in the denominator. This makes cosine independent of the total number of transactions in the database. It depends only upon the number of transactions containing both or any of X and Y. Thus adding new transactions without X, Y will not affect the value of cosine. Due to this property, cosine can easily be used in situations where new transactions are incrementally added to the database. On each update, we need only to recalculate the value of cosine for those itemsets that occur in the newly added transactions.

3.2 Types of Itemsets

We are heading toward our main task which is related to finding frequent itemsets. The concept frequent itemsets is very important to understand in order to understand the next chapters. Different types of itemsets have been introduced for different reasons. The aim of this study is to extract useful knowledge from stored data to apply performance optimization techniques, and more importantly for indexing techniques it means we will design a tool which extracts the frequent itemsets from stored data and by using indexing configuration it helps for optimization data access time. Given below is the detailed explanation of different types of itemsets.

3.2.1 Frequent Itemsets

The concept of frequent itemset is very simple to understand. As the word frequent itself explaining that occurrence of any thing again and again. Frequent patterns or itemsets are such that appears in a data frequently (but there is question that how much frequent is enough for a frequent item set? 10 occurrences? 50 occurrences? 100 occurrences? No at all, there is parameter which decides that is any itemset is a frequent itemset? This parameter is "Support of that particular item set". Actually support is required percentage of that item to be occurred in data store to become a frequent itemset. To find frequent itemset Support is first defined and all those item sets will be frequent which have support as previously set. For example if we are interested in item sets which have 40% minimum support. In our example means, that itemsets which have been purchased by at least 4 customers out of 10 are only frequent item for this particular example. In short, frequent item is one that occurs in stored data for at least a user-defined percentage (support).

Finding all the frequent itemsets is a complex problem. A property of frequent itemsets makes this problem less complex. Every subset of frequent item is also frequent. This concept is also known as Apriori Property or Downward Closure Property. According to this Property if all subsets of an itemset are not frequent then that item set will also be not frequent and similarly support of an itemset can never exceeds the support for its subsets.

3.2.2 Closed Itemsets

A frequent itemset X is called closed if there is no proper superset Y of X such that support(X) = support(Y). The set of closed itemsets is a subset of set of frequent itemsets. However, using set of closed itemsets we can generate the set of all frequent itemsets and their support. Hence, we can say that a set of closed itemsets is a condensed and lossless representation of frequent itemsets. Often the number of frequent itemsets is very large and it is difficult to represent the results of mining session to the user in a comprehensive way. Closed itemsets are very useful to represent frequent itemsets in a condensed form.

Mining closed itemsets is a hot research topic. A naïve approach to find closed itemsets can be to first find frequent itemsets and then decide which of them are closed. However, researchers have proposed efficient algorithms to find closed itemsets without first enumerating frequent itemsets. Some of these algorithms include (Zaki and Hsiao, 2002), (Pasquier et al., 1999), (Bastide et al., 2000), (Stumme et al., 2002), (Grahne and Zhu, 2003) and (El-Hajj and Zaiane, 2005). Wang and Han (2005) have proposed an algorithm for efficiently mining only top k frequent closed itemsets without enumerating all closed itemsets. A good survey of algorithms for mining closed itemsets can be found in (Yahia et al., 2006).

3.2.3 Maximal Itemsets

If a frequent itemset X is not a subset of any other frequent itemset, then X is called a maximal frequent itemset. Because X is frequent, any subset of X will also be frequent. Hence, we can say that set of maximal itemsets is a subset of the set of frequent itemsets. Like set of closed itemsets, set of maximal itemsets can also be used as a condensed representation of frequent itemsets. However, this representation is not lossless i.e. using set of maximal itemsets we can generate set of frequent itemsets but not their support.

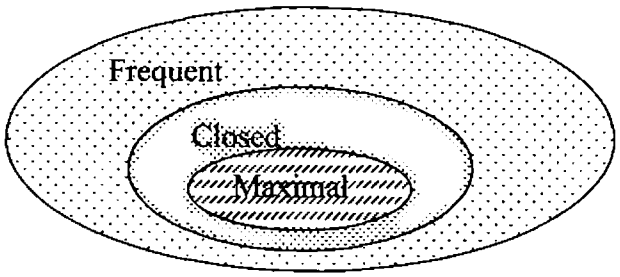


Figure 3.1 Relationship between the sets of Frequent, Closed and Maximal itemsets.

To find maximal itemsets, a simple approach can be to first find frequent itemsets and then to find set of maximal itemsets in a post processing step. However, researchers have proposed many algorithms for directly finding maximal itemsets. A popular algorithm is MAFIA (Maximal Frequent Itemsets Algorithm) proposed in (Burdick et al., 2001). A performance study of algorithms for maximal itemsets conducted by Burdick et al., 2003) showed that MAFIA performed better than other existing algorithms at that time. Later a more efficient algorithm GenMax was proposed by Gouda and Zaki (2001) which performs better than MAFIA. An algorithm proposed in (Lian et al., 2007) can handle database updates efficiently i.e. it can maintain the set of maximal itemsets without scanning the old database again.

Chapter 4: Algorithms for Finding Frequent Itemsets

The concept of association rule mining as well as the first algorithm for association rule mining was introduced by Agrawal et al. (1993). Since then a number of association rule mining algorithms have been proposed. In this chapter, we present a review of some well-known algorithms for association rule mining.

This chapter is organized as follows. Section 4.1 presents a brief introduction, and categorizes the types of association rule mining algorithms. In Section 4.2, we present some results about cost complexity of mining association rules. Section 4.3 explains the working of the most well-known association rule mining algorithm called Apriori. Section 4.4 explains the working of Frequent Pattern Tree (FP-Tree) algorithm. Section 4.5 gives a review of enhancements made so far to the basic Apriori algorithm.

4.1 Categories of Algorithms

Since the introduction of association rule mining as a data mining technique in early 1990s, many algorithms have been proposed. Some of the most popular algorithms are Agrawal, Imielinski and Swami (AIS) (Agrawal et al., 1993), Set Oriented Mining (SETM) (Houtsma and Swami, 1993), Apriori (Agrawal and Sirikant, 1994), Apriori TID (Agrawal and Sirikant, 1994), Apriori Hybrid (Agrawal and Sirikant, 1994), Direct Hashing and Pruning (DHP) (Park et al., 1995), Partition (Savasere et al., 1995), Frequent Pattern Tree (FP-Tree) (Han et al., 2000), Using Inverted Database and List Intersection (Aly et al., 2001), Using Bitmap and Granular Computing (Lin et al., 2003), Coarsen Frequent Pattern Tree (COFI-Tree) (El-Hajj and Zaiane, 2003), MATRIX (Yuan and Huang, 2005), Transaction Mapping Algorithm (TM) (Song and Rajasekaran, 2006), Cubic Structure Based Algorithm (Ivancsy and Vajk, 2005), Finding Associations by using Encoded Database (Jamali et al., 2005) and Finding Associations by Trimming the database (FAST) (Hwang and Kim, 2006).

Association rule mining is an expensive computational task. The general problem of finding association rules from transactional databases is NP-Complete (Angiulli et al., 2001). Researchers have used various techniques to speed up the task of association rule mining, introducing a variety of algorithms. Some use more memory, some take more time, some are

general purpose, some can only be used under certain constraints and some can mine only a subset of possible interesting association rules.

Algorithms for association rule mining can be divided into two basic categories:

(i) Memory based.

(ii) Disk based.

4.1.1 Memory based

The database is scanned one or two times. The whole database is stored in some in-memory compressed data structure. Two types of in-memory data structures are usually used: tree and bitmap matrix. All the remaining steps are performed on these in-memory data structures and no further database scans are performed. Since the database is scanned relatively few times as compared to disk based, therefore speed of mining is significantly better. However if database is too large, it is sometimes impossible to store the whole database in main memory.

Memory based algorithms are difficult to generalize for variants of association rule mining. Generally, it is difficult to handle database updates, since the process of constructing an in-memory data structure needs to be repeated if database is updated. Most well-known algorithm in this category is FP-Tree (Han et al., 2000).

4.1.2 Disk based

Mining is performed in a step by step manner. In step number k the database is scanned and frequent k-itemsets are discovered. In next step, the same process is repeated for frequent k+1 itemsets. Hence, multiple scans of database need to be performed. Since scanning the database multiple times requires more disk accesses, therefore these algorithms are generally slower than memory based algorithms. However, these algorithms can be easily adapted for variants of association rule mining. Since memory consumption is low, therefore these algorithms can be used for large databases. Most well-known algorithm in this category is Apriori (Agrawal and Sirikant, 1994).

Almost all algorithms for association rule mining are variants of Apriori algorithm and Tree Projection methods. Algorithms for association rule mining can be evaluated according to the following properties.

(i) Time complexity.
(ii) Space complexity.
(iii) Ability to handle large databases.
(iv) Ability to efficiently handle database updates.
(v) Ability to generalize for different variants of association rules.

All algorithms used for association rule mining work in two phases. In the first phase, support of frequent itemsets is calculated. In the second phase, support values for frequent itemsets are used to generate strong association rules. Overall execution time required for mining depends upon first phase. First phase is generally an expensive computational task, because of two main reasons: large size of databases involved in association rule mining and large number of possible itemsets.

4.2 Cost of Mining

Number of Possible Itemsets

Let the total number of items = m Then:

Total number of itemsets $= C(m, 0) + C(m, 1) + C(m, 2) + C(m, 3) + \ldots + C(m, m) = 2^m$

Number of Possible Rules

An association rule has two parts, antecedent and consequent, where antecedent and consequent are disjoint combinations of up to $m-1$ items.

Number of antecedents of size $i = C(m, i)$

Total number of antecedents $= \sum_{i=1}^{m-1} C(m, i)$

Since antecedent and consequent are required to be disjoint therefore:

Number of consequents for each antecedent of size $i = \sum_{j=1}^{m-i} C(m-i, j)$

Total number of possible rules can now be calculated as:

$$Total\ rules = \sum_{i=1}^{m-1}\left[C(m,i) \times \sum_{j=1}^{m-i} C(m-i,j) \right] = 3^m - 2^{m+1} + 1$$

Cost of counting Support

Let the number of transactions = n

To count support of itemsets we need to perform database scan and match each itemset with every transaction, to check whether the itemset is contained in the transaction or not. Let the cost of comparing a transaction with one itemset is t. Then:

$Cost\ of\ counting\ support = O(2^m nt)$

This is an exponential value, so counting support for all possible itemsets is intractable. We need some algorithm that only considers a reasonable number of itemsets and still possesses completeness. In the later section we will discuss few vary famous algorithms for finding frequent itemsets.

4.3. Apriori Algorithm

Apriori is a disk based algorithm. A key feature of Apriori algorithm is that it does not need to consider every possible itemset in order to generate strong association rules. To reduce the potential number of itemsets to consider it uses the *apriori* property. According to apriori property, *an itemset cannot be frequent if any of its subsets is not frequent.* Using this property, we can reduce the potential number of itemsets to consider. Once we know that a certain itemset is not frequent, we can ignore all its supersets because according to apriori property, we know that they cannot be frequent.

Apriori first finds frequent 1-itemsets by scanning the database, then frequent 2-itemsets by again scanning the database and continues until no further frequent itemsets are found. Hence, we can say that maximum number of database scans will depend on size of largest frequent itemset.

Initially all 1-itemsets are considered as candidates (potentially frequent) and their support is calculated by scanning the database. Those 1-itemsets are removed whose support is less than *minsup* and we are left with frequent 1-itemsets. Then a join step is performed on

frequent 1-itemsets to produce candidate 2-itemsets. Then support for these candidate 2-itemsets is calculated by scanning the database and candidates having support less than *minsup* are removed, and we are left with frequent 2-itemsets. Then frequent 2-itemsets are joined to produce candidate 3-itemsets, support for candidate 3-itemsets is calculated by scanning the database, candidates having support less than *minsup* are removed and we are left with frequent 3-itemsets. This process continues, until no further frequent itemsets are found.

Two sets are joined if they have same elements except the last. Thus joining any two sets of size k–1 produces a set of size k. Before counting support for candidates, some candidates are removed using apriori property. That is, the candidates having any infrequent subset are removed before the process of counting support.

Generally speaking, when frequent k–1 itemsets are found, a join step is performed to produce candidate k itemsets. Any two frequent k–1 itemsets are joined if they have same elements except the last. If any candidate k-itemset has an infrequent subset then this itemset is removed and support for remaining candidates is calculated by scanning the database.

Let the set of frequent itemsets of size k–1 be denoted by F_{k-1} and the set of candidate itemsets of size k be denoted by C_k. The steps performed by Apriori can be summarized as: candidate generation (joining elements of F_{k-1}), candidate elimination (using apriori property), database scan (to calculate support of candidates) and candidate pruning (candidates having support less than *minsup*). These steps involve set operations like joining, comparison of two sets for equality and generating subsets of a given set.

By using F_{k-1} to produce C_k and then eliminating itemsets using apriori property, many of the unnecessary itemsets are eliminated, i.e. those itemsets about which we know that they cannot be frequent are removed from further consideration. In this way, we do not need to consider all possible itemsets. This is not the only way to generate candidates for example (Mannila et al., 1994) uses the following two-step process to produce candidates

$$C_k' = \{X \cup X' \mid X, X' \in F_{k-1} \text{ and } \mid X \cap X' \mid = k-2\}$$

$$C_k = \{X \in C_k' \mid X \text{ contains } k \text{ members of } F_{k-1}\}$$

However, as proved in (Agrawal and Sirikant, 1994) the above method produces some unnecessary candidates. The method used in (Agrawal et al., 1993) produces candidates on the fly i.e. when a transaction is read; candidate k-itemsets are produced by extending frequent $k-1$ itemsets with the items present in the transaction. This method also produces a large number of candidates that are often very difficult to manage.

The pseudo code for first phase (finding all frequent itemsets) of the Apriori algorithm as given in (Agrawal and Sirikant, 1994) is presented in Figure 4.1 after minor modifications. The modifications are made only for simplicity; they do not alter the algorithm in any way.

Apriori

1. $C_1 \leftarrow \{candidate\ 1-itemsets\}$ *// all 1 – itemsets are considered as candidates*
2. $F_1 \leftarrow \{c \in C_1 \mid c.count \geq \min\sup\}$ *// remove infrequent candidates*
3. *for* $(k \leftarrow 2; F_{k-1} \neq \phi; k++)$ *// repeat until no more frequent itemsets are found*
4. $\quad C_k \leftarrow AprioriGenerate(F_{k-1})$
5. $\quad$ *for each transection* $t \in D$
6. $\quad\quad C_t \leftarrow Subsets(C_k, t)$ *// get subsets that are candidates*
7. $\quad\quad$ *for all candidates* $c \in C_t$
8. $\quad\quad\quad c.count++$
9. $\quad$ *end*
10. $\quad F_k \leftarrow \{c \in C_k \mid c.count \geq \min\sup\}$ *// remove infrequent candidates*
11. *end*
12. *return* $F \leftarrow \bigcup_k F_k$

AprioriGenerate(F_{k-1})

1. *for* $i \leftarrow 1$ *to* $|F_{k-1}|$
2. $\quad$ *for* $j \leftarrow 1$ *to* $|F_{k-1}|$
3. $\quad\quad$ *if* $JoinAble(F_{k-1}[i], F_{k-1}[j])$
4. $\quad\quad\quad C \leftarrow F_{k-1}[i] \cup F_{k-1}[j]$
5. $\quad\quad\quad$ *if* $HasInfrequentSubset(C)$
6. $\quad\quad\quad\quad delete(C)$
7. $\quad\quad\quad else\ C_k.add(C)$
8. *return* C_k

```
JoinAble(A, B)
1.  for i ← 1 to k − 2
2.      if A[i] ≠ B[i]
3.          return false
4.  if A[k − 1] ≠ B[k − 1]
5.      return true
6.  else return false
HasInfrequentSubsets(C)
1. for each k − 1 subsets s of C
2.   if s ∉ F_{k−1}
3.       return true
4. return false
```

Figure 4. 1: Pseudo code for Apriori algorithm.

Candidate itemsets are stored in a hash table. When a transaction is read, the hash table is accessed and counts of those itemsets that are present in the transaction are incremented. Since candidates are stored in a hash table, therefore counting support takes relatively less time. Database scan time is not dependent on the algorithm. Generation of C_k from F_{k-1} takes comparatively longer time. Therefore when analyzing the running time of Apriori we consider only candidate generation step. Another reason for considering candidate generation step is that we have improved its running time in our algorithm and we compare both in subsequent sections.

After finding frequent itemsets, association rules can be generated through the following procedure.

(i) For each frequent itemset l, generate all nonempty subsets of l.

(ii) For each nonempty subset s of l, output the rule $s \Rightarrow (l-s)$ if $\frac{support\_count(l)}{support\_count(s)} \geq min\_conf$

Generation of strong association rules from support values of frequent itemsets is not a time consuming process, because it does not involve database scan. The overall efficiency of algorithms thus depends upon the first phase.

4.3.1 Analysis of Apriori Algorithm

Let us consider the cost of a single set operation as the unit cost. Let the candidate k-itemset be denoted by C_k and let the set of frequent k–1 itemsets be denoted by F_{k-1}. Suppose $|F_{k-1}| = n$.

HasInfrequentSubsets: The size of set C is k, therefore it will have k subsets of size k–1. Line number 1: will execute k times.

Line number 2: is basically a linear search, searching for s in F_{k-1}. Therefore, cost of line number 2 is n. So the procedure HasInfrequentSubsets will have a cost of $k*n$.

JoinAble: It is a single set operation so its cost is constant.

Subsets: This procedure produces k subsets therefore its cost is k.

AprioriGenerate: The cost can be calculated as follows:

Line 1: n times.

Line 2: n times.

Line 3, 4: Constant time.

Line 5: $n*k$ times.

Line 6, 7: Constant time (Any one of them will execute).

Line 8: Constant time.

Total cost of procedure can be calculated as:

$$Cost = \sum_{i=1}^{n}\sum_{j=1}^{n}(1+1+nk+1)+1$$

$$= 3\sum_{i=1}^{n}\sum_{j=1}^{n}1+\sum_{i=1}^{n}\sum_{j=1}^{n}nk+1$$

$$= 3n^2+n^3k+1 = O(n^3k)$$

We illustrate the working of the Apriori algorithm using the following example. Suppose minimum support count is 2 and consider the database given in Table 4.1.

Table 4.1: Sample Transaction database.

TID	List of items
1	U V Y
2	V X
3	V W
4	U V X
5	U W
6	V W
7	U W
8	U V W Y
9	U V W

Counting Support

Initially all one itemsets are considered as candidates. Database is scanned and counts for one itemsets are calculated.

Table 4.2: Support of candidate one itemsets

Itemset	Support
U	6
V	7
W	6
X	2
Y	2

Candidates having support count less than 2 are removed and we are left with frequent one itemsets. Since support count of all one-itemsets is greater than 2, therefore all of them are frequent.

Table 4.3: **Frequent one itemsets for database**

Itemset	Support
U	6
V	7
W	6
X	2
Y	2

Frequent one itemsets are used to create candidate two itemsets. Each frequent one itemset is joined with every other frequent one itemset. The database is again scanned and support of candidate two itemsets is calculated.

Table 4.4: Support of candidate two itemsets.

Itemset	Support
UV	4
UW	4
UX	1
UY	2
VW	4
VX	2
VY	2
WX	0
WY	1
XY	0

Those two itemsets that have support count less than 2 are removed and we are left with frequent two itemsets.

Table 4.5: **Frequent two itemsets**

Itemset	Support
UV	4
UW	4
UY	2
VW	4
VX	2
VY	2

Now frequent two itemsets are used to find candidate three itemsets. The process is explained below:

(i) UV is joined with UW giving UVW. Subsets of UVW are UV, UW and VW. All subsets of UVW are frequent, therefore UVW is considered as candidate.

(ii) UV is joined with UY giving UVY. Subsets of UVY are UV, UY and VY. All subsets of UVY are frequent, therefore UVY is considered as candidate.

(iii) UV is not joinable with VW.

(iv) UV is not joinable with VX.

(v) UV is not joinable with VY.

(vi) UW is joined with UY giving UWY. Subsets of UWY are UW, UY and WY. WY is not frequent, hence UWY is not considered as candidate.

(vii) UW is not joinable with VW

(viii) UW is not joinable with VX

(ix) UW is not joinable with VY

(x) UY is not joinable with VW

(xi) UY is not joinable with VX

(xii) UY is not joinable with VY

(xiii) VW is joined with VX giving VWX. Subsets of VWX are VW, VX and WX. WX is not frequent, hence VWX is not considered as candidate

(xiv) VW is joined with VY giving VWY. Subsets of VWY are VW, VY and WY. WY is not frequent, hence VWY is not considered as candidate

(xv) VX is joined with VY giving VXY. Subsets of VXY are VX, VY and XY. XY is not frequent, hence VXY is not considered as candidate

Database is scanned once again and counts of candidate three itemsets are calculated.

Table 4.6: Support of candidate three itemsets

Itemset	Support
UVW	2
UVY	2

Since all candidates have support count greater than equal to 2, all of them are frequent.

Table 4.7: Frequent three itemsets.

Itemset	Support
UVW	2
UVY	2

Frequent three itemsets are now used to find candidate four itemsets.

(i) UVW is joined with UVY giving UVWY. Subsets of UVWY are UVW, UVY, UWY and VWY. UWY and VWY are not frequent, hence UVWY is not considered as candidate.

Since no four itemset becomes a candidate, therefore the process terminates.

Generation of association rules from frequent itemsets

(ii) UV

U⇒V confidence=4/6=67%

V⇒U confidence=4/7=57%

(iii) UW

U⇒W confidence=4/6=67%

W⇒U confidence=4/6=67%

(iv) UY

U⇒Y confidence=2/6=33%

Y⇒U confidence=2/2=100%

(v) VW

V⇒W confidence=4/7=57%

W⇒V confidence=4/6=67%

(vi) VX

V⇒X confidence=2/7=29%

X⇒V confidence=2/2=100%

(vii) VY

V⇒Y confidence=2/7=29%

Y⇒V confidence=2/2=100%

(viii) UVW

U⇒VW confidence=2/6=33%

V⇒UW confidence=2/7=29%

W⇒UV confidence=2/6=33%

UV⇒W confidence=2/4=50%

UW⇒V confidence=2/4=50%

VW⇒U confidence=2/4=50%

(ix) UVY

U⇒VY confidence=2/6=33%

V⇒UY confidence=2/7=29%

Y⇒UV confidence=2/2=100%

UV⇒Y confidence=2/4=50%

UY⇒V confidence=2/2=100%

VY⇒U confidence=2/2=100%

4.4 Frequent Pattern Tree Algorithm

Frequent Pattern Tree (FP-Tree) was proposed by Han et al. (2000). The Apriori heuristic achieves good performance gain by (possibly significantly) reducing the size of candidate sets. However, in situations with long patterns, or low minimum support thresholds, an Apriori like algorithm may still suffer from the following two nontrivial costs (Han et al., 2000):

(i) It is costly to handle a huge number of candidate sets. For example, if there are 10^4 frequent 1-itemsets, the Apriori algorithm will need to generate more than 10^7 length-2 candidates and test their occurrence frequencies. Moreover, to discover a frequent pattern of size 100, it must generate more than $2^{100}=10^{30}$ candidates in total. This is the inherent cost of candidate generation, no matter what implementation technique is applied.

(ii) It is tedious to repeatedly scan the database and check a large set of candidates by pattern matching, which is especially true for mining long patterns.

FP-Tree adopts a different approach. It first converts the database to a compact in-memory data structure, known as Frequent Pattern Tree. The rest of mining task is performed on this in-memory tree, and no further database scan is performed. Construction of FP-Tree takes two database scans. During first scan, support of one itemsets is calculated. One itemsets are then sorted in non-decreasing order according to support and infrequent one itemsets are removed. During second database scan, the transactions are read, sorted in decreasing order according of support of the items they contain and stored in a tree structure called Frequent Pattern Tree (FP-Tree).

The reason for sorting transactions in descending order according to support of the items they contain is that in this way there is more chance of sharing nodes and hence a smaller tree. However, this is only a heuristic, the size of FP-Tree can still be exponential. If the number of frequent one itemsets is n, the number of nodes in FP-Tree in worst case will be 2^n.

FP-Tree is a main memory based algorithm. It first projects the database to a tree and then uses this tree to find association rules. The tree is kept in main memory so the process becomes fast. Pseudo code of FP-Tree algorithm as presented in (Han et al., 2000) is given Figure 4.2.

Build Tree

1) Scan the transaction database and collect the set of frequent items F.
2) Sort F in support descending order as L, the list of frequent items.
3) Create the root of an FP-tree, T, and label it as "null".
4) For each Transaction in database do the following.
5) Select and sort the frequent items in Transaction, according to the order of L. Let the sorted frequent item list in Transaction be [p|P], where p is the first element and P is the remaining list. Call Insert tree ([p|P], T).

Insert tree ([pP]; T)

If T has a child N such that N.itemName = p.itemName, then increment N's count

Else create a new node N, and let its count be 1, its parent link be linked to T, and its node-link be linked to the nodes with the same item name via the node-link structure

If P is nonempty, call Insert tree (P, N) recursively.

FP-growth (FP-tree ; null).

Procedure FP-growth (Tree, α)

1) If Tree contains a single path P

   Then for each combination denoted as β of the nodes in the path P, generate pattern αUβ with support = minimum support of nodes in β.

2) Else for each a_i in the header of Tree
   i. Generate pattern $\beta = a_i \cup \alpha$ with support = support of a_i
   ii. Construct β's conditional pattern base and then β's conditional FP-tree Treeβ
   iii. If Treeβ≠null then call FP-growth (Treeβ, β)

Figure 4.2: Pseudo code for FP-Tree algorithm.

The node link structure is only used for easy traversal. We now explain the working of FP-Tree algorithm with the help of an example. Let the minimum support be equal to 3 and consider the transaction database given in Table 4.8.

Table 4.8: Database for FP-Tree example.

TID	Itemsets
1	Y U X V
2	X U W Y V
3	W U V Y
4	V U X
5	X
6	X V
7	U X Y
8	V W

Table 4.9: Support of one itemsets.

Item	U	V	W	X	Y
Support	5	6	3	6	4

Table 4.10: Sorted list of frequent one itemsets.

Item	V	X	U	Y	W
Support	6	6	5	4	3

Table 4.11: Transactions of database sorted in support descending order.

TID	Itemsets
1	V X U Y
2	V X U Y W
3	V U Y W
4	V X U
5	X
6	V X
7	X U Y
8	V W

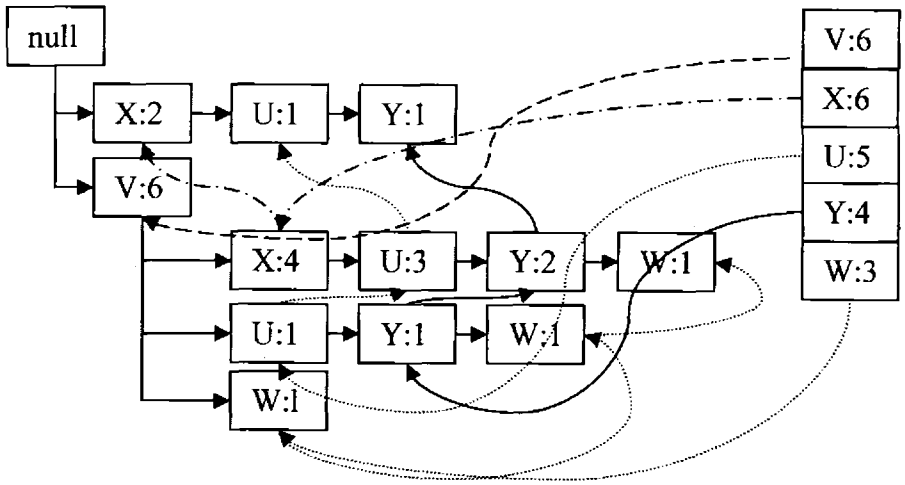


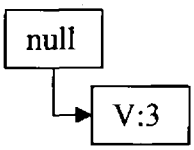
Figure 4.3 FP-Tree for database in Table 4.8.

F-List for original database = V: 6, X: 6, U: 5, Y: 4, W: 3

• Conditional pattern base for W: 3 = VXUY: 1, VUY: 1, V: 1.

F-List for W: 3 = V: 3

Condition FP-Tree for W: 3

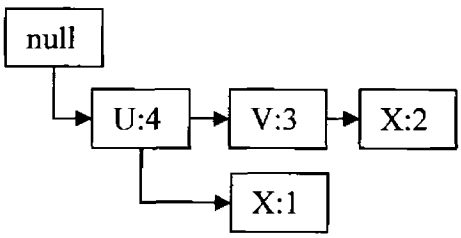


Frequent patterns generated =W: 3, VW: 3

• Conditional pattern base for Y: 4 = VXU: 2, VU: 1, XU: 1

F-List for Y: 4 = U: 4, V: 3, X: 3

Condition FP-Tree for Y: 4

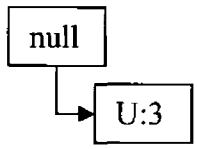


Recursion

Condition pattern base XY: 3 = U: 1, UV: 2

F-List for XY: 3 = U: 3

Conditional FP-Tee for XY: 3

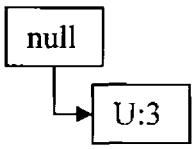


Frequent patterns generated = XY: 3, UXY: 3

Conditional pattern base for VY: 3 = U: 3

F-List for VY: 3 = U: 3

Conditional FP-Tree for VY: 3



Frequent patterns generated = VY: 3, UVY: 3

Condition pattern base for UY: 4 = null

F-List for UY: 4 = null

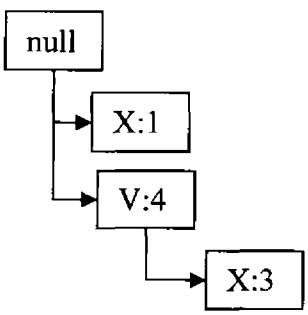
Conditional FP-Tree for UY: 4 = null

Frequent patterns generated = UY: 4, Y: 4

- Condition pattern base for U: 5 = VX: 3, V: 1, X: 1

F-List for U: 5 = V: 4, X: 4

Conditional FP-Tree for U: 5

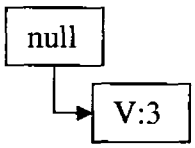


Recursion

Conditional pattern base for UX: 4 = V: 3

F-List for UX: 4 = V: 3

Conditional FP-Tree for UX: 4



Frequent patterns generated = UX: 4, UVX: 3

Conditional pattern base for UV: 4 = null

F-List for UV: 4 = null

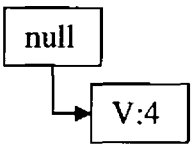
FP-Tree for UV: 4 = null

Frequent patterns generated = UV: 4, U: 5

- Conditional pattern base for X: 6 = V: 4

F-List for X: 6 = V: 4

Conditional FP-Tree for X: 6



Frequent patterns generated = X: 6, VX: 4

- Conditional pattern base for V: 6 = null

F-List for V: 6 = null

Conditional FP-Tree for V: 6 = null

Frequent patterns generated = V: 6

4.5 Improvements in Basic Apriori algorithm

Apriori was a ground breaking algorithm and became the basis for many algorithms, each of which tried to improve the basic algorithm. The strategies followed by these algorithms can be divided into following categories:

(i) Reducing the number of database scans required.

(ii) Using specialized data structures to speed up the mining task.

(iii) Exploiting parallelism.

(iv) Modifying the structure of database to help speed improvement.

(v) Sampling the database.

(vi) Adding extra constraints on the structure of patterns.

In the following sections, we provide a brief description of some of the algorithms.

4.5.1 Use of inverted database

Proposed by Aly et al. (2001), this algorithm modifies the structure of the database. The modified database will have a smaller size and is therefore more suitable for counting support of itemsets.

Transaction database used for association rule mining is initially in horizontal format. In horizontal format, each row represents a transaction and contains items contained in the transaction. A typical record in such a database can be written as {TID, Itemset}. The algorithm in its first phase converts the database to vertical format. In vertical format, each row represents an item and contains a list of transaction IDs in which this item appears. A typical record of transformed database can be written as {Item; TransactionSet}, where TransactionSet is a list of transactions in which the item appears. Thus for each item we have a corresponding row that gives the TIDs of transactions that contain it. A one itemset is frequent if length of its corresponding record is greater than the minimum frequency count. The size of the transformed database is roughly equal to the size of the original database, but more suitable for support counting.

During execution of algorithm, a record in the k^{th} iteration will have the form {Itemset; TransactionSet}, where TransactionSet is the set of transactions in which all the members of the Itemset appear. In higher order iterations (starting from two), to verify if a candidate k-itemset is frequent or not, simply perform the intersection operation between TransactionSets of any two rows that correspond to $(k-1)$-subsets of the candidate. If the cardinality of the resulting TransactionSet is higher than the required minimum support, the itemset is frequent, otherwise it is not. To optimize the process of intersection one may select the rows with minimum length. The working of algorithm is further explained in the following example. Suppose minimum support count is 2 and consider the database given in Table 4.1.

Table 4.12: Database of Table 4.1 in inverted format, obtained after first iteration.

One Itemset	TransactionSet	Length
U	1, 4, 5, 7, 8, 9	6
V	1, 2, 3,4 6, 8, 9	7
W	3, 5, 6, 7, 8, 9	6
X	2, 4	2
Y	1, 8	2

All one itemsets are frequent, therefore all two itemsets will be considered as candidates.

Table 4.13: Intersection of rows of **Table 4.12** obtained after second iteration.

Itemset	TransactionSet	Length
UV	1, 4, 8, 9	4
UW	5, 7, 8, 9	4
UX	4	1
UY	1, 8	2
VW	3, 6, 8, 9	4
VX	2, 4	2
VY	1, 8	2
WX		0
WY	8	1
XY		0

Itemsets having support count less than 2 are removed. The same process is repeated for higher order transactions. Since the length of list of transactions decreases as the size of itemset increases, higher order iterations will take less time.

Another variant of the above method, proposed in (Song and Rajasekaran, 2006) uses a compressed representation for storing the list of transactions. Rather than storing the IDs for all transactions, an interval based strategy is followed. For example, if list of transactions for some itemset consists of {1, 2, 3, 4, 6, 8, 10, 11, 12, 13, 18} we can store it as {1-6, 8, 10-13, 18}. In this way, the process of intersection takes less time and storage requirement is also reduced.

4.5.2 DHP Algorithm

DHP (Direct Hashing and Pruning) was proposed by Park et al. (1995). To efficiently find frequent itemsets, this algorithm uses a hash table (specialized data structure). By using hash table, the algorithm is able to reduce the number of candidate itemsets (especially during initial iterations). Initial iterations of Apriori generally take much more time as compared to the higher order iterations. The reason behind this is the large number of candidate itemsets in initial iterations.

In first step, when the database is scanned to calculate counts of one itemsets, the algorithm also calculates the counts of two itemsets in a hash table. The size of hash table is much smaller than the total number of two itemsets. More than one different itemsets will hash to the same location in the hash table. A counter is maintained for each location of hash table. The counter represents the number of itemsets hashed to this particular location. Therefore, the counter represents the combined support of the itemsets that have been hashed to that particular location. If this combined support is less than the minimum support then any itemset that has been hashed to this location cannot be frequent. Therefore, before counting actual support of two itemsets, some of the two itemsets are already known to be infrequent and hence the size of candidate two itemsets is reduced. The same process can be repeated for higher iterations.

The process is explained in the following example. Suppose minimum support count is 6 and consider the database given in Table 4.1. The database is scanned to calculate counts of one itemsets. When a transaction is read, all two itemsets contained in the transaction are generated and hashed into the corresponding bucket of a hash table. When an itemset is hashed to some bucket, the count of that bucket is incremented. Itemsets in bucket having count less than minsup cannot be frequent and therefore can be removed from C_2. In this way the size of C_2 is reduced, increasing the efficiency of next iteration. The hash table for two itemsets is given in Table 4.14. Suppose the hash function is ((order of first item) $\times 10$ + (order of second item)) mod 5.

Table 4.14: Hash table. $h(x, y) = ((order\ of\ x) \times 10 + (order\ of\ y)) \bmod 5$.

Bucket number	0	1	2	3	4		
Bucket count	5	0	4	8	3		
Bucket contents	UY VY WY		UV	UW VW	UX VX		

From the above table we know that combined support of UY, VY and WY is 5 and hence UY, VY and WY cannot be frequent. Support of UV is 4 so it cannot be frequent. Similarly, combined support of UX and VX is 3 therefore UX and VX cannot be frequent. Note that using Apriori algorithm the candidate two itemsets will consist of UV, UW and VW. Now from the above we know that UV cannot be frequent and hence it is removed from candidate two itemset.

The hash function used in the above example works for two itemsets only. By using appropriate hash function, the above technique can also be used for higher order iterations. However, because expected size of candidate itemsets in lower order iterations is usually much larger than the size of candidate itemsets in higher order iterations, hash based technique is especially useful for lower order iterations.

4.5.3 Encoding Databases Layout

Proposed by Jamali et al. (2005), this algorithm modifies the structure of database. The modified database is smaller and more efficient for support counting. Apriori scans the database multiple times. If the size of database is reduced without any loss of information, algorithm efficiency can be improved. Horizontal and vertical representations have almost same size. This algorithm first converts the database to an encoded format and uses this encoded format. The process is as follows:

Each item is assigned a unique prime number. The encoded database consists of two columns. The first column consists of TIDs as usual and second column contains a number obtained by multiplying the prime numbers assigned to the items contained in the transaction. A typical transaction in transformed database can now be represented as {TID, Value} where Value is the integer obtained by multiplying the corresponding prime numbers of the items contained in

the transaction. Thus, each transaction is represented by just an integer value. Similarly, each itemset is represented by an integer value, obtained by multiplying the prime numbers assigned to the items in the itemset. The resulting database will have much smaller size than the original database. Further, to check whether a particular itemset is contained in some transaction or not, simply divide the integer associated with the transaction by the integer associated with the itemset. If the remainder is zero, the itemset is contained in the transaction otherwise it is not. To illustrate the process of encoding, consider the database given in Table 4.1.

Table 4.15: Prime number assigned to items.

Item	Prime #
U	2
V	3
W	5
X	7
Y	11

Table 4.16: Database in encoded format.

TID	Value
1	66
2	21
3	15
4	42
5	10
6	15
7	10
8	330
9	30

Although the algorithm reduces the size of database and decreases the processing time, it involves multiplication and division of large integers, especially if number of items is large. The size of integers can be much larger than the word size of the underlying machine. One must use special algorithms to deal with this problem.

4.5.4 Partitioning the database

This technique was proposed by Savasere et al. (1995). Almost all Apriori like algorithms scan the database multiple times. However, partition algorithm scans the database only twice. Since the number of possible candidate itemsets is exponentially large, counting support for them in one scan of the database is almost impossible. However, if number of candidates is not very large, say a few thousand, then their support can be calculated in one scan of the database and the actual frequent itemsets can be discovered. This approach works only if the given candidate set contains all frequent itemsets along with some infrequent itemsets.

Partition algorithm works in two phases. In the first phase, the database is partitioned into non-overlapping partitions. The size of the partition is kept as large as possible but small enough to fit in the main memory. Partitions are read one by one and candidate itemsets (local candidates) from each partition are generated. At the end of the first phase, all local candidates are merged to generate a list of global candidates. This set is a superset of all frequent itemsets, i.e. it may contain false positives, but no false negatives. In second phase actual support for these candidate itemsets is calculated by scanning the database second time.

Although the algorithm works efficiently in many cases, it has some limitations:

1) The size of the database must be known in advance to calculate size of partition.
2) The algorithm needs modifications if underlying machine or size of the memory is changed.
3) For very large databases, the size of global candidate set may become large and may not fit in the main memory.

4.5.5 Bitmap and Granular Technique

This technique was proposed by Lin et al. (2003), further enhances the inverted database techniques. The database is converted to bit representation. Bitmaps are stored on disk in the so called "data pages". A fast disk access strategy is adapted, to further increase the speed of mining. Since the database is in the form of bitmaps the process of intersection becomes a simple AND operation.

The algorithm needs special routines for disk access, a kind of specialized disk driver to speed up the database scan. In most modern operating systems, use of such drivers is against the security policy and hence the algorithm cannot be used on such platforms.

4.5.6 Cubic algorithm

This technique was proposed by Ivancsy and Vajk (2005), it uses a matrix to store the support count of 1, 2, 3 and 4 itemsets. It discovers up to 4-itemsets in only two database scans. In first scan, it counts the support of one and two itemsets. To store the counters, an upper triangular matrix is used. Let n be the total number of items, then the algorithm will need an upper/lower triangular matrix having a size equal to $n(n+1)/2$. The diagonal elements of the matrix contain support of one itemsets, non diagonal elements contains the support of two itemsets. The matrix is accessed by direct indexing. Consider the database given in Table 4.1. The matrix used to store support for one and two itemsets in given in Table 4.17.

Table 4.17: Upper triangular matrix to store support of one and two itemsets.

U	V	W	X	Y	
6	4	4	1	2	U
	7	4	2	2	V
		6	0	1	W
			2	0	X
				2	Y

Three and four frequent itemsets are found by scanning the database once again. Counters for candidate three and four itemsets are stored in an in-memory cubic structure based index table. A cube is created for each frequent one itemset i.e. only for those items whose corresponding diagonal value is greater than minimum support. It means that one cube is created for those candidates of size three and four that begin with the same item. The process of indexing the cubic structure is as follows: the first item of candidate itemset selects the appropriate cube, the subsequent items select the cells in the cube. The matrix is processed row

wise. The i^{th} row is processed only if the value of i^{th} diagonal element is greater than the minimum support.

4.5.7 Sampling the Database

Finding Associations by Trimming (FAST) was proposed by Chen et al. (2002) and IFAST (Improved FAST) was proposed by Hwang and Kim (2006). The working of these algorithms is as follows: Take a sample of database, small enough to be accommodated in memory. Find frequent itemsets from the sample. Consider the frequent itemsets found from the sample as candidates. Scan the database again and calculate actual support for candidates. Remove candidates having low support.

Sampling techniques do not find all frequent itemsets. Although no false positives are included in result, some true positives can be missed. The only difference between FAST and IFAST is the method of taking sample. The method of sampling used by IFAST is better than FAST in that is misses fewer true positives.

4.5.8 Use of Support Vectors and Deleting Unnecessary Transactions

Algorithm LIG (the author does not provide any hint or explanation for the name LIG, one possible guess may be that L and I are from the name of first author and G is the last char of the name of second author) was proposed by Li and Zang (2003). It reduces the size of database in each scan and number of candidates is also reduced. The algorithm works as follows: when support of candidate k–1 itemsets is calculated by scanning the database, all transactions of length k–1 are deleted, because these transactions cannot contain an itemset of size k or larger. Besides calculating support for k–1 itemsets, the algorithm also constructs a list called *support vector* for each candidate itemset. Support vector has a size equal to the size of largest transaction. The i^{th} entry in the support vector contains the count of transactions of length i that contain this itemset. After finding frequent k–1 itemsets, candidate k-itemsets are generated. Support vectors for frequent k–1 itemsets are intersected to find support vectors for candidate k-itemsets. If the sum of all entries in the resulting support vector is less than minimum support then this candidate is removed. The process is explained through the following example.

Suppose minimum support count is 2 and consider the database given in Table 4.1. Support counts and support vectors of one itemsets is given in Table 4.18.

Table 4.18: Support and support vectors of one itemsets

Itemset	Support	Support Vector
U	6	(0,2,3,1)
V	7	(0,3,3,1)
W	6	(0,4,1,1)
X	2	(0,1,1,0)
Y	2	(0,0,1,1)

Since all one itemsets are frequent, all two itemsets will be included in candidate two itemsets. However if we intersect the support vectors of X (0, 1, 1, 0) and support vector of Y (0, 0, 1, 1) we get support vector of XY, which is (0, 0, 1, 0). The sum of all entries in this support vector is one, which is less than minimum support. Hence, XY can be removed from set of candidate. The same method can be used for higher order iterations.

4.5.9 Association Rule Mining Using Parallel Hardware

Mining association rules from large databases is time consuming. As proved by Angiulli et al. (2001) finding all association rules is NP-Complete (under most settings). Exploiting parallelism can be the ultimate choice for such a time consuming activity. Parallel algorithms for association rule mining can be divided into two categories:

1. Data parallelism based.
2. Task parallelism based.

The Count Distribution (CD) algorithm proposed by Agrawal and Shafer (1996) is based on data parallelism. It partitions the database into n disjoint partitions, where n is the number of processors. Each partition is assigned to exactly one processor. Each processor scans its local partition and calculates local support of candidates. The local supports are then summed up to find global support.

Data Distribution (DD) algorithm proposed by Agrawal and Shafer (1996) is based on task parallelism. Set of candidates is partitioned in to n parts, where n is the number of processors. Each processor scans the whole database and calculates the support of candidates assigned to it.

The Common Candidate Partitioned Database (CCPD) algorithm proposed by Agrawal and Shafer (1996) is based on data parallel approach. The algorithm was designed for shared memory architecture. The algorithm partitions the database logically into n equal sized partitions. Each processor generates a disjoint candidate subset and counts their support independently.

Beside all the above algorithms, there is another class of algorithms that try to improve the efficiency of Apriori by using more than one of the above techniques. These algorithms can be characterized as hybrid algorithms.

Chapter 5: Algorithm for Finding Frequent Itemsets from Proposed Database Layout

Association rule mining is a time consuming activity for large databases, thus algorithm efficiency is a key concern. In this chapter we will propose algorithm for finding frequent itemsets from the proposed database layout. Our layout is specially made up for finding frequent itemsets therefore it will find out frequent itemsets more efficiently as compared to any other algorithm.

This chapter follows the sequence as: In section 5.1 we will explain propose algorithm for finding two frequent itemsets. Section 5.2 shows implementation of two frequent itemsets. In section 5.3 we will describe algorithm for finding higher order itemsets and its implementation.

Our algorithm works in two phases for finding frequent itemsets. First phase includes the finding of two frequent itemsets. Second phase includes the finding of higher order frequent itemsets. Our proposed algorithm finds out all the higher order frequent itemsets from the two frequent itemsets, therefore, first of all we will write the algorithm for finding two frequent itemsets.

5.1 Algorithm for finding two frequent itemsets

On the basis of groups made in proposed layout we are going to find out two frequent itemsets in the first phase. The algorithm for finding two frequent itemsets will be as follows:

Frequent2itemsets(groups,diffset)

1. *for each member 'x' in group G*
2. *for each item 'v' in[x-leader[G]]*
3. *NewItemset=leader[G] + v*
4. *parentOf[NewItemset]←x*
5. *SupportOf[NewItemset]=[Total_Transaction-(length of group leader diffset + length of parent's Tidlist)]*
6. *If(SupportOf[NewItemset] > minSupport)*
7. *Add[list of two Item set]←NewItemset*

Now we will explain how this algorithm will find out the two frequent itemsets from the proposed database layout. We will explain the above algorithm line by line. The first line of the algorithm 'For each member 'x' in group G' is an outer loop which will be executed for number of members in each particular Group. For example, in case of first group which has the following members: ABDG 4 and ACF Null, this outer loop will be executed for two times. Similarly in another group which has the following three members: CD 4, CF Null and CG 2 4, then the outer loop will be executed three times. In short, if there are n members in a group then the outer loop will be executed for n times.

Now we have a control on the total number of members in each group through outer loop next we have to find out frequent two itmesets from each member. In order to achieve this goal we will have to join each item of the member with its group leader. Second line of the algorithm helps us in doing so by stating 'For each item 'v' in[x-leader [G]]'. This is an inner loop and this will be executed for each item present in particular member. For example if we talk about the first member ABDG 4 of the first group, then the loop will be executed for this particular member three times. Variable 'v' will corresponds to item 'B' then item 'D' and last but not the least item 'G.' Note that the leading item of a member is excluded. Leading item of the member is considered to be the group leader and we will have to join all the items with that leader, therefore we will exclude the leader. If a member has n items then inner loop will be executed for n-1 times.

As discussed earlier, leader of the Group is excluded. Leader of the group will be joined with each other item in that particular member in order to make two itemsets. Line number three of the algorithm 'NewItemset=leader[G]+v' is joining each item of the member with its group leader. This line shows that a new Itemset is being created with the combination of leader of the Group and those items which are present in that particular member. For example in the case of member ABDG 4, new itemsets will be the combination of A with B, A with D and A with G which can be written as AB, AD and AG in successive iteration of inner loop.

In order to calculate the support of each itemset later on, we will have to remember how an itemset was made, therefore we will have to save the member for each itemset from which it was made. Line number four of the algorithm 'parentOf [NewItemset] ←x' does it for us.

According to algorithm parent of each newItemset will be the member from whom that particular newItemset was made. For example ABDG 4 will be the parent for all newItemsets AB, AD and AG because all newItemsets were produced due to ABDG 4.

Finding support for itemsets in association rule mining is one of the most important and critical task. Our algorithm finds the support for each itemset in an interesting manner. Line number five of the algorithm 'SupportOf [NewItemset]=Number of lines-length of group leader diffset-length of parent's Tidlist" finds out support for each itemset. We will have to revise the meaning of ABDG 4 in order to understand the calculation of support. ABDG 4 means that these items are not collectively present in all those transaction where item 'A' is not present and there is only one additional transaction named as transaction number 4 where these all are not collectively present. To calculate the support for each itemset the above stated formula will be used. It consists of three ingredients. First of all is the total number of transactions in the database that are denoted by the number of lines. In our example total transactions are six therefore the number of lines will contain six. This value will remain constant for finding support for each itemset. Second variable in the formula is length of group leader's diffset. Its value depends upon the non-existence of the group leader from the transactions and it will be changed with respect to each group leader. For example, if 'A' is the group leader then its value will be two because 'A' is not present in two transactions, if 'B' is the group leader value of this variable will again be two because item 'B' is also absent in two transactions, in the case of item 'C' as a group leader this variable will contain the value one because item 'C' is absent only in one transaction. Third variable in the formula for counting support is the length of parent's tidlist. We stored tidlist for each member in Group. Now in order to find the support of newItemset AB, its parent's is ABDG 4, clearly the length of parent's tidlist is one because it contains only one transaction named as transaction number 4.

We have discussed all the variables in the formula for counting support for each new itemset. Now we will findout the support of few new itemsets for the sake of understanding. First of all we will calculate the support of the first new itemset named as 'AB'. According to formula, values involved in the formula are number of lines, which is six, length of group leader diffset, that is two and length of parent tidlist, which is one in our case. In order to calculate the

support we will have to put these values in the formula, so the support of AB will be: Support of [AB] = 6 – 2 – 1=3.

When the support for each itemset has been calculated, next task is to check that whether that itemset is frequent or not. Line number six of the algorithm 'If(SupportOf[NewItemset]>=minSupport)' is used to check whether the itemset is frequent or not. When we calculate support of an itemset, it is not necessary that it will be a frequent itemset. To determine whether an itemset is frequent or not, depends upon the value of minimum support. If support of new itemset is greater than or equal to minimum support then it is frequent itemset otherwise it is not a frequent itemset. Minimum support is also called threshold. In our example the value of the threshold is 3. According to this threshold itemset 'AB' is frequent because the support of 'AB' is equal to the minimum support value.

When we are able to find any frequent itemset, it is necessary to save that frequent itemset therefore further higher order frequent itemsets can be obtained from that itemset. Last line of the algorithm is used to perform the task of saving two frequent itemsets. Last line of the algorithm is stated as: Add[list of two frequent Itemset]←NewItemset. All those items that will be frequent will be added to the list of two frequent itemsets.

5.2 Implementation of algorithm on our proposed database Layout

Now we will calculate two itemsets by applying the above stated algorithm. First group contains only two members: ABCDG 4 and ACF null. Remember that ABDG4 means that ABDG items are not collectively present in transaction number 4 and where item 'A' is not present. ACF null means that these items are collectively present in all transaction except those transactions where item 'A' itself is not present. For the sake of simplification we give a numeric identity to each member of the group. By this simplification, we will just write down the numeric number of the member in front of each new itemset from which that itemset was made. Parent of each itemset will be denoted by that numeric value. Given below is the detail of two itemsets.

There is only one group according to item 'A' that has two members. ABDG 4 will be represented as 1 and ACF null will be represented as 2. Now we will make two itemsets

from the above members and will write 1 and 2 instead of ABDG 4 and ACF null. Two itemsets will be AB 1, AD 1, AG 1, AC 2 and AF 2.

There is only one group according to item 'B' that also has two members. BCF 6 will be represented as 1 and BDG null will be represented by 2. Here BCF 6 means that these items are not collectively present in one transaction named as six and where item 'B' itself is not present. BDG null means these itemsets are not collectively present only in those transactions in which B is not present otherwise all transaction have these all items because it's TID list is null. Now we will make two itemset from the above members and will write 1 and 2 instead of BDG null and BCF 6. Two itemsets will be BC 1, BF 1, BD 2 and BG 2.

There are two groups according to item 'C'. First group have three members CD 4 that will be represented by 1, CF null that will be represented by 2 and CG 2 4 that will be represented as 3. CD 4 means these itemsets are not collectively present in one transaction named as 4 and those transactions in which item 'C' itself is not present. CF null means these itemsets are not present in only those transactions where item 'C' itself is not present. CG 2 4 means these itemsets are not collectively present in two transactions named as 2 and 4 and those transactions in which item 'C' itself is not present. Now we will make two itemset from the above members and will write 1 and 2 and 3 instead of CD 4, CF null and CG 2 4 respectively. Two itemsets will be CD 1, CF 2 and CG 3. Second group contains three members CF null that will be represented by 1 and CE 1 3that will be represented by 2. Meaning of CF null is given above whereas CE 1 3 means that these items are not collectively present in two transactions named as 1 and 3 and in those transactions where 'C' is not present. By applying algorithm two itemset will be CF 1 and CE 2.

Now we will consider the groups according to item 'D'. There are two groups with respect to item 'D'. First group have two members DF 6 that will be represented by 1 and DG 2 that will be represented as 2. Here DF 6 means that these items are not collectively present in one transaction named as 6 and in those transactions in which D is not present. DG 2 means that these items are not collectively present in one transaction named as 2 and in those transactions where item 'D' itself is not present. By applying algorithm on these members we will get the following two itemsets: DF 1 and DG 2. Second group have only one member DE 1

3 that will be represented by 1. Here DE 1 3 means that these items are not collectively present in two transaction named as 1 and 3 and in those transactions in which D is not present. By applying algorithm on this member we will get only one two itemset DE 1.

There are two groups according to item 'F'. First group have only one member FG 2 4 that will be represented by 1. FG 2 4 means that these itemsets are not collectively present in two transactions named as 2 and 4 and those transactions in which item 'F' itself is not present. Now we will make two itemset from the above member and will write 1 instead of FG 2 4. Two itemset will be FG 1. Second group also have only one member FE 1 3 that will be represented by 1. FE 1 3 means that these itemsets are not collectively present in two transactions named as 1 and 3 and those transactions in which item 'F' itself is not present. Now we will make two itemset from the above member and will write 1 instead of FE 1 3. Two itemset will be FE 1.

We have constructed all the two itemsets , now we will calculate the support and find out only those items that are frequent. Itemsets whose support will be greater or equal to the minimum support will be considered as frequent itemsets. Minimum support for our example is 50%. We will find out the support of each itemset according to its group leader and from which member that item has been created.

First of all we will consider all items that are constructed using item 'A' as group leader. Length of diffset of A is 2 because item 'A' is not present in two transactions. If item is made of 1 then the length of TID list will be 1 and if it is made up of 2 then length TID list will be 0. Given below is the support of all items that are constructed by using item 'A' as group leader.

Support of [AB1]=6-2-1=3,frequent item set because support is equal to 3.

Support of [AD1]=6-2-1=3 ,frequent item set because support is equal to 3.

Support of [AG1]=6-2-1=3, frequent item set because support is equal to 3.

Support of [AC2]=6-2-0=4, frequent item set because support is greater than 3.

Support of [AF2]=6-2-0=4, frequent item set because support is greater than 3.

Now we will consider all items that are constructed using item 'B' as group leader. Length of diffset of item 'B' is also 2 because item 'B' is not present in two transactions. If item is made up of 1 then the length of TID list will be 1 and if it is made up of 2 then the length of TID list will be 0. Given below is the support of all items that are constructed by using item 'B' as group leader.

Support of[BC1]=6-2-1=3 frequent item set because support is equal to 3.

Support of[BF1]=6-2-1=3 frequent item set because support is equal to 3.

Support of[BD2]=6-2-0=4 frequent item set because support is greater than 3.

Support of[BG2]=6-2-0=4 frequent item set because support is greater than 3.

Next we will consider all items that are constructed using item 'C' as group leader. Length of diffset of item 'C' is 1 because item 'C' is not present in only one transaction. If item is made up of 1 then the length of TID list will be 1, if it is made up of 2 then the length of TID list will be 0 and if it is made up of 3 then the length of TID list will be 2. Given below is the support of all items that are constructed by using item 'C' as group leader.

Support of[CD1]=6-1-1=2 it is not frequent item because support is less than 3.

Support of[CF2]=6-1-0=5 frequent item set because support is greater than 3.

Support of[CG]=6-1-2=3 frequent item set because support is equal to 3.

Item 'C' has two groups therefore we will have to find out the support of all those itemsets that are made up from the members of group 2. Length of diffset of item 'C' will remain same means 1. If item is made up of 1 then the length of TID list will be 0 and if it is made up of 2 then the length of TID list will be 2. Given below is the support of all items that are constructed by using item 'C' as group leader with second group.

Support of[CF1]=6-1-0=5 frequent item set because support is greater than 3.

Support of[CE2]=6-1-2=3 frequent item set because support is equal to 3.

Next we will consider all items that are constructed using item 'D' as group leader. Length of diffset of item 'D' is 1 because item 'D' is not present in only one transaction. If item is made up of 1 or 2 then the length of TID list will remain 1. Given below is the support of all items that are constructed by using item 'D' as group leader.

Support of[DF1]=6-1-1=4 frequent item set because support is greater than 3.

Support of[DG2]=6-1-1=4 frequent item set because support is greater than 3.

Item 'D' also has two groups therefore we will have to find out the support of all those itemsets that are made up from the members of group 2. Length of diffset of item 'D' will remain same means 1. If item is made up of 1 then the length of TID list will be 2. Given below is the support of all items that are constructed by using item 'D' as group leader with second group.

Support of[DE1]=6-1-2=3 frequent item set because support is equal to 3.

Next we will consider all items that are constructed using item 'F' as group leader. Length of diffset of item 'F' is 1 because item 'F' is not present in only one transaction. If item is made up of 1 then the length of TID list will be 2. Given below is the support of all items that are constructed by using item 'F' as group leader.

Support of[FG1]=6-1-2=3 frequent item set because support is equal to 3.

Item 'F' also has two groups therefore we will have to find out the support of all those itemsets that are made up from the members of group 2. Length of diffset of item 'F' will remain same means 1. If item is made up of 1 then the length of TID list will be 2. Given below is the support of all items that are constructed by using item 'F' as group leader with second group.

Support of[FE1]=6-1-2=3 frequent item set because support is equal to 3

Note that all those items will be added to list of frequent itemsets set which are frequent under support calculation above.

5.3. Algorithm for finding higher order frequent itemsets

In the above section we have discussed the algorithm for finding two frequent itemsets. Now we will discuss the procedure for finding higher order itemsets from two frequent itemsets. Our proposed algorithm finds higher order frequent itemsets from two frequent itemsets, therefore, two frequent itemsets will be the input for our algorithm. The algorithm for finding higher order frequent itemsets will be as below:

```
Construct_Higher_Order_itemSet (twofrequentitemsets , groups)
1.  A←0
2.  for i←1 to |S|
3.      for j←i+1 to |S|
4.              Z=S[i] join S[j]
5.              if Z is frequent
6.                      A←A+Z
7.                      Output A
8.      Construct_Higher_Order_itemSet(A,group)
```

In order to understand the working of this algorithm we will construct higher order itemsets. For the formation of three order frequent itemsets we will take two order frequent itemsets as an input, similarly for the formation of four order frequent itemsets this algorithm will take three order frequent as an input and so on. In general for formation of n order frequent itemsets this algorithm will take n-1 order frequent itemsets. The procedure will continue until all the frequent itemsets will be generated. Now we will discuss this algorithm line by line.

We pass two parameters to this algorithm. In the initial case first parameter will be a set of two order frequent itemsets and the second parameter will be groups. In our example, two order frequent itemsets with group leader 'A' are AB, AD, AG, AC and AF and Groups have two members ABDG 4 and ACF null.

In the first line 'A←0' we have initialized an empty array. This array will contain three order frequent itemsets.

Second line of the algorithm 'for i←1 to |S|' is an outer loop. This loop will be executed for length of two order frequent itemsets. In our example the length of two order frequent itemset is five; therefore, body of the loop will be executed five times.

Third line of the algorithm is 'for j← i+1 to |S|' is again a loop. This is an inner loop and this loop will always be executed one less times as compared to outer loop. In our example this loop will executes its body four times when the value of 'i' will be one. It will executes its body three times when the value of 'i' will be two and so on.

Now we will have to make three order itemsets from 2 itemset by joining two order itemset. Line number four 'Z=S[i] join S[j]' is used to join the itmesets.

JOIN

Description of join operation is as follows. If we want to construct a three order frequent itemset it will be constructed with the help of two order frequent itemsets. Now we can join only those two order itemsets whose first item is same in order to get three order itemsets. Similarly when we want to construct four itemsets we will join only those three itemsets whose first two items will be same. In general if we want to construct n itemset from n-1 itemsets, we will only join those n-1 itemsets whose first n-2 itemsets are same.

According to our example if 'i' points to itemset AB then 'j' points to AD, AG, AC and AF respectively. After joining the produced three order frequent itemsets will be ABD, ABG, ABC and ABF. When 'i' points to two order frequent itemset AD then 'j' will points to AG, AC and AF respectively. After joining the produced three order itemsets will be ADG, ADC and ADF respectively. When 'i' points to two order frequent itemset AC then 'j' will points to AF. After joining the produced three order itemsets will be AGC and AGF respectively.

All the three order itemsets are checked whether they are frequent or not. Every three order itemset is temporary stored in Z. Line number five of the algorithm " if Z is frequent" checks whether Z is frequent or not.

All the frequent three order itemsets are added in A because we need all of them in the construction of four order frequent itemsets. Last line of the algorithm is a recursive call. All the

three order frequent itemsets are constructed now; it is time to construct four order frequent itemsets and so on.

This is how our algorithm works for finding frequent itemsets from our proposed database layout. This algorithm is far more efficient as compared to many other algorithms for finding frequent itemsets. In next chapter we will discuss the reasons for efficiency of our algorithm over the others.

5.4 Running Time Comparison of FFIUCID with Diffet

Since our work is based on the diffset layout therefore we will only compare it with algorithm based on the diffset layout.

Empirical Analysis

One type of empirical comparison is being presented that is execution time. We have used two different datasets. The details of data sets are given in 5.1. These data sets were obtained from UCI datasets repository (http://mlearn.ics.uci.edu/MLRepository.html). The results are shown in Figure 5.1 and 5.2 for different support thresholds. Reducing the support increases the time complexity. Therefore, for lower support the difference in efficiency between the two algorithms is more in comparison to higher support.

Table 5.1: Details of Datasets.

Data Set	Name	Size in KB	No. of transactions	No of items
1	Chess	335	3196	75
2	Mushroom	558	8124	119

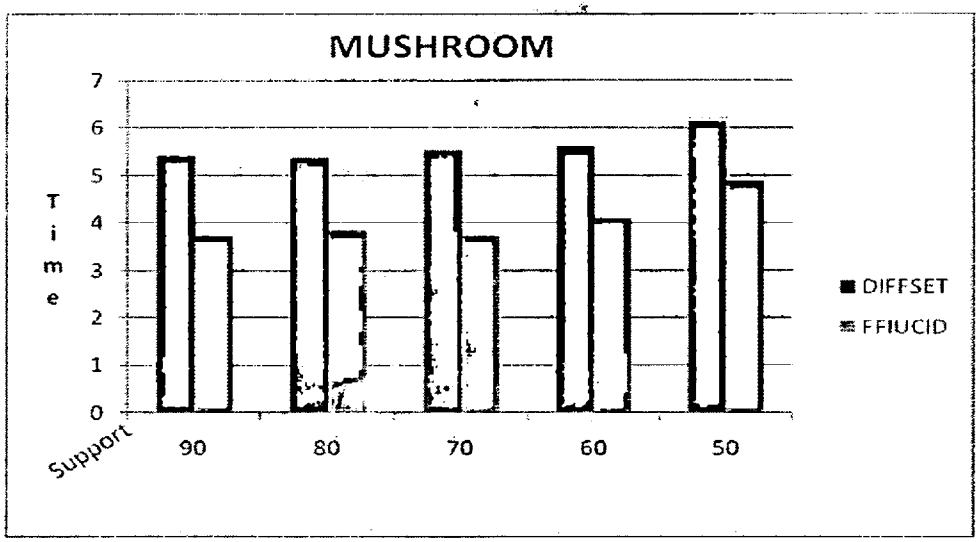


Figure 5.1: Comparison on Mushroom Dataset

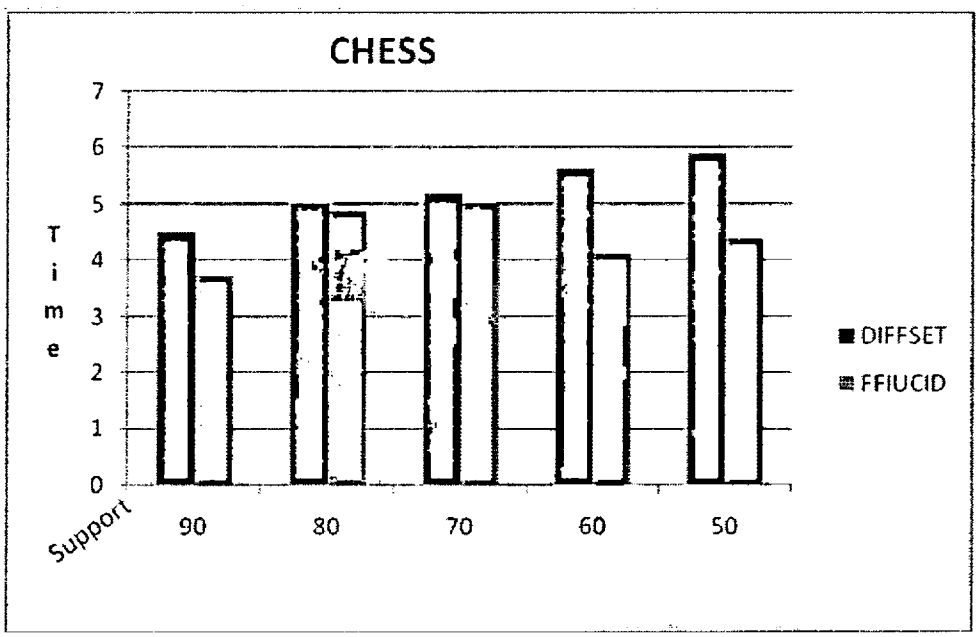


Figure 5.2: Comparison on Chess Dataset.

Chapter 6: Conclusion

6.1 Conclusion

Association rule mining is a well-researched area with a lot of applications. Association rule mining is applicable in other techniques of data mining such as classification and clustering, which increases its importance. A.R.M is time consuming activity particularly when the size of database is very large. Algorithm efficiency is a key concern for A.R.M.

In this dissertation, we have reviewed the application areas of A.R.M. We have reviewed different database layouts and proposed a new database layout (Chapter 2). We have reviewed different interesting measures used in A.R.M algorithm and different types of itemsets (Chapter 3). Different existing algorithms for A.R.M. were also reviewed (Chapter 4). We have proposed an algorithm for finding frequent itemsets (Chapter 5).

6.2 Summary of Research Contributions

Our objective was to design such a database layout that supports us in finding frequent, closed & maximal itemsets. Our proposed CID layout is specially made for finding different itemsets. Give below are the few advantages of our proposed CID layout over other existing layouts.

We have constricted members of groups in such a way that all the items are combined in only one member if they have a predefined occurrence with each other. Whenever we construct two itemsets all the constructed two itemsets will be frequent, because we are joining itemsets of only one member therefore resulting two itemsets will be frequent with same support. The benefit is that we will not have to calculate the support of each two itemset. Similarly we will not have to check that whether itemset is frequent or not, because all the constructed two itemsets from a member are frequent with same support.

Another advantage that increases the efficiency of algorithm for finding frequent itemsets from the proposed layout is that all subsets of a member according to its group leader are frequent with the same support. For example a group has a member ABDG 9, then AB,AD,AG,

ABD, ABG, and ADG all are frequent with same support therefore we do not need to calculate the support of each subset.

Finding closed & maximal itemsets from CID will be efficient as compared to any other database layout. All the members in the groups are already strong candidates for closed itemsets.

Whenever an itemset of a member is joined with another itemset, second itemset belongs to a member that has null in its TID (Transaction Identifier), new itemset will be frequent with same support. For example one member of a group is ABDG 4 and the second member is ACF Null. Suppose the first itemset is AB that belongs to member ABDG 4 and second itemset is AC that belongs to ACF Null. When AB and AC are joined we get a new itemset ABC. ABC will be frequent with the same support as of AB.

6.3 Future Work

In our research work we have proposed a new database layout and then we have proposed an algorithm for finding frequent itemsets from that new layout. Our layout has some characteristics that make the algorithm efficient. Closed and maximal itemsets are also an important aspect of association rule mining. We have yet not worked on finding closed and maximal itemsets from the proposed layout. In future, algorithms for finding closed and maximal itemsets from the proposed layouts can be explored. It has a high probability that these algorithms may have less time complexity because every member of a group is already a candidate for closed itemsets.

References:

1. Roiger R. J and Geatz M. (2002). *"Data Mining: A Tutorial Based Primer"*. Published by Addison-Wesley.
2. Agrawal R. and Sirikant R. (1994). *Fast Algorithms for Mining Association Rules.* In Proceedings of the Twentieth Conference on Very Large Databases, pp 487-499.
3. Agrawal and Shafer J. C. (1996). *Parallel Mining of Association Rules.* IEEE Transactions on Knowledge and Data Engineering, Volume 8, No. 6, pp 962-969.
4. Agrawal R., Imielinski T. and Swami A. (1993). *Mining Association Rules Between Sets of Items in Large Databases.* In Proceedings of the ACM SIGMOD Conference on Management of Data, pp 207-216.
5. Aly H. H., Amar A. A. and Taha Y. (2001). *Fast Mining Association Rules in Large Scale Problems.* In Proceedings of Sixth IEEE International Symposium on Computers and Communication, pp 107-113.
6. Angiulli F., Ianni G. and Palopoli L. (2001). *On The Complexity of Mining Association Rules.* In Proceedings of Ninth Italian Symposium on Advanced Database Systems, pp 177-184.
7. Brijs T., Vanhoof K. and Wets G. (2003). *Defining Interestingness for Association Rules.* International Journal on Information Theories and Applications, Volume10, No. 4, pp 370-375.
8. Brin S., Motwani R., Ullman J. D. and Tsur S. (1997). *Dynamic Itemset Counting and Implication Rules for Market Basket Data.* In Proceedings of ACM SIGMOD International Conference on Management of Data, pp 255-264.
9. Bastide Y., Pasquier N., Taouil R., Stumme G. and Lakhal L. (2000). *Mining Minimal Non Redundant Association Rules Using Frequent Closed Itemsets.* In Proceedings of the first International Conference on Computational Logic, Volume 1861, pp 972-986.
10. Burdick D., Calimlim M., Flannick J., Gehrke J. and Yiu T. (2003). *MAFIA: A Performance Study of Mining Maximal Frequent Itemsets.* In Proceedings of IEEE

International Conference on Data Mining, Workshop on Frequent Itemset Mining Implementation.

11. Chen B., Haas P. and Scheuermann P. (2002). *A New Two-Phase Sampling Based Algorithm For Discovering Association Rules.* In Proceedings of Eighth ACMSIGKDD International Conference on Knowledge Discovery and Data Mining, pp.462-468.

12. El-Hajj M. and Zaiane O. R. (2003). *Non Recursive Generation of Frequent k-itemsets From Frequent Pattern Tree Representation.* In Proceedings of Fourth International Conference on Data Warehousing and Knowledge Discovery, pp 371-340.

13. El-Hajj M. and Zaiane O. R. (2005). *Finding All Frequent Patterns Starting from the Closure.* In Proceedings of International Conference on Advanced Data Mining and Applications, pp 67-74.

14. Grahne G. and Zhu J. (2003). *Efficiently Using Prefix Trees in Mining Frequent Itemsets.* In Bart Goethals and Muhammad J. Zaki, editors, Proceedings of the IEEE International Conference on Data Mining, Workshop on Frequent Itemset Mining Implementations, Volume 90 of CEUR Workshop Proceedings.

15. Gouda K. and Zaki M. J. (2001). *Efficiently Mining Maximal Frequent Itemsets.* In Proceedings of International Conference on Data Mining, pp 163-170.

16. Hand D., Mannila H. and Smyth P. (2001). "*Principles of Data Mining*". MIT Press, Cambridge.

17. Han J. and Kamber M. (2000). "*Data Mining Concepts and Techniques*". Morgan Kaufmann Publishers.

18. Han J., Pei J. and Yin Y. (2000). *Mining Frequent Patterns Without Candidate Generation.* In Proceedings of SIGMOD International Conference, pp 1-12.

19. Hwang W. and Kim D. (2006). *Improved Association Rule Mining by Modified Trimming.* In Proceedings of Sixth IEEE International Conference on Computer and Information Technology, pp 24-28.

20. Ivancsy R. and Vajk I. (2005). *Fast Discovery of Frequent Itemsets: a Cubic Structure Based Approach.* Informatica, pp 71-78.

21. Jamali M., Yareh F. T. and Rahgozar M. (2005). *Fast Algorithm for Generating Association Rules with Encoding Database Layout.* In Proceedings of World Academy of Science, Engineering and Technology, Volume 4.

22. Lian W., Cheung D. W and Yiu S. M (2007). *Maintenance of Maximal Frequent Itemsets in Large Databases.* In Proceedings of ACM Symposium on Applied Computing pp 388-392

23. Lallich S., Teytaud O. and Prudhomme E. (2006). "*Quality Measures in Data Mining*" Series: Studies in Computational Intelligence, Volume 43, pp. 251-275.

24. Lin. T. Y., Hu X. and Louie E. (2003). *A Fast Association Rule Algorithm Based On Bitmap and Granular Computing.* In Proceedings of the IEEE International Conference on Fuzzy Systems, pp 678-683.

25. Li X and Zang X (2003). *A Fast Algorithm on Discovery Large Itemsets.* In Proceedings of the Second International Conference on Machine Learning and Cybernetics.

26. Pasquier N., Bastide Y., Touil R. and Lakhal L. (1999). *Discovering Frequent Closed Itemsets for Association Rules.* In Proceedings of Seventh International Conference on Database Theory, Volume 1540, pp 398-416.

27. Park J. S., Chen M. S. and Yu. P. S. (1995). *An Effective Hash Based Algorithm for Mining Association Rules.* In Proceedings of International ACM SIGMOD Conference on Management of Data, pp 175-186.

28. Shapiro P. (1991). *Discovery Analysis and Presentation of Strong Rules.* In Proceedings of International Conference on Knowledge Discovery in Databases, pp 229-248.

29. Savasere A., Omiecinski E. and *Navathe* S. (1995). *An Efficient Algorithm for Mining Association Rules in Large Databases.* In Proceedings of Twenty-First International Conference on Very Large Databases, pp 434-444.

30. Song M. and Rajasekaran S. (2006). *A Transaction Mapping Algorithm for Frequent Itemset Mining.* IEEE Transactions on Knowledge and Data Engineering, Volume 18, No. 4, pp 472-481.

31. Wang J. and Han J. (2005). *TFP: An Efficient Algorithm for Mining Top-K Frequent Closed Itemsets.* IEEE Transactions on Knowledge and Data Engineering, Volume 17, No. 5, pp 652-664.

32. Yahia S. B., Hamrouni T. and Nguifo E. M. (2006). *Frequent Closed Itemset Based Algorithms: A Thorough Structural and Analytical Survey.* SIGKDD Explorations, Volume 8, pp 93-104.

33. Zhao Q. and Bhow'mick S. S. (2003). *Association Rule Mining: A Survey.* Technical Report, CAIS, Nanyang Technological University, Singapore, No. 2003116.

34. Zaki M. J. and Hsiao C. J. (2002). *CHARM: An Efficient Algorithm for Closed Itemset Mining.* In Proceedings of Second SIAM International Conference on Data Mining, pp 457-473.