Centralized Association Rules Mining (C-ARMing)

Developed by

Zakia Jalil
Saleha Jamshaid

Supervised by

Dr. Malik Sikander Hayat Khiyal

Department of Computer Science
Faculty of Applied Sciences
International Islamic University, Islamabad.
(2006)

MS
006.3
[illegible] AC

M·[illegible]
9/12/10

Mathematical Statistics
Operation Research
Artificial intelligence

[illegible]

بسم الله الرحمن الرحيم

In The Name of

ALLAH ALMIGHTY

The Most Merciful, The Most Beneficent

Department of Computer Sciences

International Islamic University Islamabad

Date: 31-8-2006

Final Approval

This is to certify that we have read the thesis submitted by **Zakia Jalil** 214-CS/MS/04 and **Saleha Jamshaid** 219-CS/MS/04. It is our judgment that this thesis is of sufficient standard to warrant its acceptance by International Islamic University, Islamabad for the degree of MS Computer Science.

Committee:

External Examiner

Dr. Qasim Rind
Professor,
Preston University, Islamabad.

[illegible]

Internal Examiner

Mr. Asim Munir
Assistant Professor,
Faculty of Applied Sciences,
International Islamic University,
Islamabad.

[illegible]

Supervisor

Dr. M. Sikander Hayat Khiyal
Head, Department of Computer Science,
Faculty of Applied Sciences,
International Islamic University,
Islamabad.

[illegible]

A dissertation submitted to the
Department of Computer Science,
Faculty of Applied Sciences,
International Islamic University, Islamabad, Pakistan,
as a partial fulfillment of the requirements for the award of the degree of

MS in Computer Science

To
The Holiest Man Ever Born,
Prophet Muhammad (صلى الله عليه وسلم)

To
Our Parents and Families
We are most indebted to our parents and families, whose affection has always been the source of encouragement for us, and whose prayers have always been a key to our success.

To
Those Holy Seekers
Who give away their lives to make the stream of life flow
Smoothly and with Justice.

And
To
Our Honorable Teachers
Who have been a beacon of knowledge and a constant source of inspiration, for our whole life span.

Declaration

We hereby declare and affirm that this software neither as a whole nor as a part thereof has been copied out from any source. It is further declared that we have developed this software and accompanied report entirely on the basis of our personal efforts, made under the sincere guidance of our teachers. If any part of this project 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.

Zakia Jalil
214-CS/MSCS/F04
Saleha Jamshaid
219-CS/MSCS/F04

Acknowledgement

We bestow all praises to, acclamation and appreciation to Almighty Allah, The Most Merciful and Compassionate, The Most Gracious and Beneficent, Whose bounteous blessings enabled us to pursue and perceive higher ideals of life, Who bestowed us good health, courage, and knowledge to carry out and complete our work. Special thanks to our Holy Prophet Muhammad (SAW) who enabled us to recognize our Lord and Creator and brought us the real source of knowledge from Allah (SWT), the Qur'ān, and who is the role model for us in every aspect of life.

We consider it a proud privileged to express our deepest gratitude and deep sense obligation to our reverend supervisor **Dr. Malik Sikander Hayat Khiyal** who kept our morale high by his suggestions and appreciation. His motivation led us to this success. Without his sincere and cooperative nature and precious guidance; we could never have been able to complete this task.

It will not be out of place to express our profound admiration and gratitude for our teachers **Mr. Muhammad Imran Saeed** and **Mr. Atif P. Malik** for their dedication, inspiring attitude, untiring help and kind behavior throughout the project efforts and presentation of this manuscript. They did a lot of efforts for our success.

Finally we must mention that it was mainly due to our parent's moral support and financial help during our entire academic career that enabled us to complete our work dedicatedly. We owe all our achievements to our most loving parents, who mean most to us, for their prayers are more precious before any treasure on the earth. We are also thankful to **Miss Sadia Arshid** and **Miss Maham Javed** for their assistance during the project, and to our loving brothers, sisters, friends, and class fellows who mean the most to us, and whose prayers have always been a source of determination for us.

Zakia Jalil

214-CS/MSCS/F04

Saleha Jamshaid

219-CS/MSCS/F04

PROJECT IN BRIEF

Project Title:	**Centralized Association Rules Mining (C-ARMing)**
Organization	International Islamic University, Islamabad
Under Taken By:	**Zakia Jalil** **Reg. No# 214/CS/MSCS/F04** **Saleha Jamshaid** **Reg. No# 219/CS/MSCS/F04**
Supervised By:	**Dr.Malik Sikander Hayat Khiyal.** **Head of Department of Computer Science** **International Islamic University, Islamabad.**
Starting Date:	**February, 2006.**
End Date:	**July, 2006.**
Tools used:	**Matlab 7, VB.Net**
System Used	Pentium IV

ABSTRACT

With the explosive growth in information technology, the demands of the users of the computer systems from the data they are storing, is increasing day by day. The awareness about the importance of the data for the businesses has made them more demanding of their computer systems. Now they want their computers to think for them and assist them in decision making. So different Decision Support Systems are created for this purpose. These DSS take data of over decades of the years as an input and give output in the form of some smart business decision. This extraction of information from huge data warehouses is known as Data Mining. Association Rule Mining is one of the branches of Data Mining. It refers to the mining of interesting associations between different data items which can't be found out with traditional data models. Association Rule Algorithm finds these associations between the frequently sold items, so that the user could put such associated items with in close proximity.

Our devised algorithm, CMA, is efficient than the existing algorithms by a factor of 50 percent in terms of database scans. In previous algorithms, the supports of the itemsets were calculated by scanning the database for those particular itemsets. The multiple database scans are avoided, in our devised algorithm, by using a formula of **s*D** for calculating the supports of the itemsets. So the database is only scanned once for candidate set creation alone.

The experiments are performed over the synthetic database, created exclusively for this project. The synthetic database is used for performance efficiency.

TABLE OF CONTENTS

List of Tables

List of Figures

Chapter 1

Introduction

1. Introduction

Businessmen are always been in the competition with their opponents. It is observed that there is always an atmosphere of competition, whenever there are two stores in vicinity. Both the shopkeeper will be in the state of contention for grabbing the attention of the customers. There is, in fact, no harm in healthy business competition. It gives energy to the business, so not only the businessman will enjoy the full fruit of his struggle, but also the customers will be provided with best products.

The growth of the business depends upon a number of factors, i.e., businessman's attitude towards his customers, the environment of the store, the quality of the goods available, their prices, the packing, the brands, and above all, the way the goods are placed in the shelves. It has always been observed that it is not just the in-time availability of different products in the store to help increasing its sales, but also their proper placement on the shelves. Businessmen always try to do the best to increase the sales of their stores. In past, their efforts were supported with their observation alone. The emergence of Information Technology has changed the whole scenario of the business decisions. The users now analyze their data to facilitate their business decisions. For example, previously a store keeper was required to do a lot of observation in order to come to know how to place different products on the shelves? Whether to make different categories of different products and then assign them to different shelves or what? And what should be the criteria for categorizing different product items together? What shelf formation is going to be well accommodated in the store and will result in increasing sales? Entertaining such queries is not an easy task to handle. It requires extraordinary observatory skills, and a mind to make good analysis. Whereas now a days, with the help of data, and using a computer, a user can make any shelf planner and analyze on the basis of each planner, and finalize the best suitable one for his store. His data also tells him about the most frequently sold items, and what is the appropriate time to place an order for the new stock? What stock? How much? And with what frequency? Keeping in view all these issues and benefits of data, the users are not ready to loose any data. The day-to-day data kept in Operational System can not reside permanently on disk. It remains on disk for some specific time, and then dumped into archives. The user is ready to take any measures to preserve all its data, without loosing any amount of it. In return, with growing amount of data in databases, the user is also expecting more from it. A marketing manager is no longer satisfied with a simple listing of marketing contacts, but wants detailed information about customer's past purchases as well as predictions of future purchases [3], without realizing the capabilities of the Data Models in use. In order to satisfy analytical queries, Data Warehouses are the best forum.

1.1 Data Warehousing

A data Warehouse is a repository of information collected from multiple sources, stored under a unified schema, and which usually resides at a single site. Data Warehouses are constructed via a process of data cleaning, data transformation, data integration, data loading, and periodic data refreshing [2].

According to Inmon, Data Warehouse is "A subject-oriented, integrated, time-variant, and non-volatile collection of data in support of management's decision making process" [4]. With Data Warehousing, corporate-wide data (current and historical) are merged into a single repository. It contains *informational data*, which are used to support other functions such as planning and forecasting [3]. The data has been gone through a number of phases to become informational data from operational data. There are many routines applied on it, commonly known as Extract-Transform-Load (ETL) routines, through which the data from different systems, files, mediums, data models and paradigms, is Extracted, and then Transformed into some unified and summarized format after cleansing, and then Loaded onto the Data Warehouse. The basic motivation for this shift to the strategic use of data is to increase business profitability. Traditional data processing supports the day-to-day clerical and administrative decisions, while Data Warehousing supports long-term strategic decisions. A 1996 report by International Data Corporation (IDC) stated that an average ***return on*** *investment* (ROI) in Data Warehousing reached 401% [3].

Data Warehouse is used for Ad hoc queries. The data remains static in the Data Warehouse as no changes or modifications could be done on it after it's once been loaded into Data

Warehouse. The data schema used for it could either be the **Star Schema** (in which the tables or dimensions are kept de-normalized), or the **Snow-Flake schema** (which normalizes the tables or the dimensions).

A **Data Mart** contains a subset of corporate-wide data that is of value to a specific group of users. The scope is confined to specific selected subject [2]. The example of Data Mart could be easily taken from a University System, where we have different departments, i.e., Accounts Department, Examination Department, and Students Affairs Department. If we collect the historical data of a particular department from all sources, and clean them, transform into a uniform format, and then load it in our computer, in order to be able to get analytical queries answered, then it means we are building separate Data Marts for each department of the University System. All Data Marts of an organization collectively form a Data Warehouse. The data loaded on the computer system is supposed to be summarized data, like if two attributes of STUDENT Entity class are in the format "current_date" and "date_of_birth", after summarization, it would become a single attribute "student_age".

Depending upon the source of data, Data Marts can be categorized as **Independent** or **Dependent**. Independent Data Marts are sourced from data captured from one or more Operational Systems, or external information providers, or from data generated locally within a particular department or geographic area. Dependent Data Marts are sourced directly from enterprise Data Warehouses [2]. The Independent Data Marts are created using the **Bottom-Up** approach of creating the Data Marts, in which developers start implementing the Data Marts first, which eventually form a corporate-wide Data Warehouse. Whereas the Dependent Data Marts are created by dividing the corporate-wide Data Warehouse into subsets, according to the needs of the different departments. The approach followed for creating the Dependent Data Marts is the **Top-Down** approach.

Data Marts may be stored and accessed separately. The level is at a departmental, regional or functional level. These separate Data Marts are much smaller, and they more efficiently support analytical types of applications [3].

1.2 Data Mining

With limited amount of data, and information to be retrieved, simple SQL queries are used to retrieve and present the demanded information to the user. But with the huge amount of data and user's increasing demand for sophisticated information retrieval out of this data, the SQL queries can no longer fulfill the demands. However, the use of SQL is not always adequate to meet the end user requirements of specialized and sophisticated information from an unorganized large data bank. This necessitates looking for certain alternative technique to retrieve information from large and unorganized source of data [6]. The SQL is used for information retrieval from Operational Systems, which access the Database via queries that are well-defined in the language like SQL. It accesses the data we have in our database, a subset of the database. Whereas in Data Mining, the entire technique is different from that of the traditional database queries. The query with which the data set is accessed might not necessarily be well-defined or precisely stated [3]. It is mostly a fully fledged algorithm used to query the data set in Data Mining. The data accessed is usually a different version from that of the original Operational Database. The data have been cleansed and modified to better support the mining process. The output of the Data Mining query is not a subset of the database. Instead it is the output of some analysis of the contents of the Database [3]. So, Data Mining is the technique of extracting meaningful information from large and mostly unorganized data banks. It is the process of performing automated extraction and generating predictive information from large data banks [6]. It can be defined as finding hidden information in a database. Different Data Mining algorithms are used to fulfill the data mining tasks. To do so, the algorithm first analyzes the data and then determines for that data a closer model, according to its characteristics. There are two major types of the models, Predictive Model and Descriptive Model, as shown in fig.1-1.

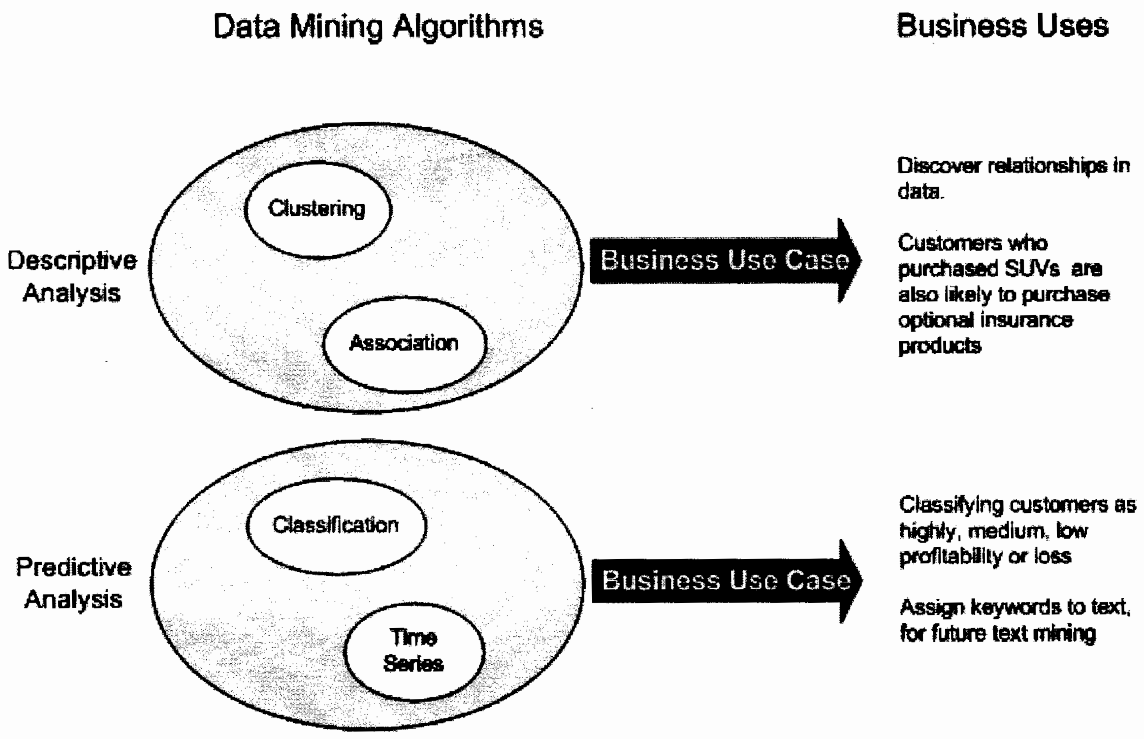

Data Mining Algorithms
Business Uses
Descriptive Analysis
Clustering
Association
Business Use Case
Discover relationships in data.
Customers who purchased SUVs are also likely to purchase optional insurance products
Predictive Analysis
Classification
Time Series
Business Use Case
Classifying customers as highly, medium, low profitability or loss
Assign keywords to text, for future text mining

Figure 1-1: Different Data mining models

1.2.1 Predictive Model

This model uses known results found from different data and then makes the prediction. A predictive Model enables to predict the values of data by making use of known results from a different set of sample data [6]. This model is sub divided into three tasks:

1.2.1.1 Classification

The data is mapped into predefined groups, called the **Classes**. Classification enables to classify data in a large data bank into predefined set of classes that are defined before studying or examining data in the data bank. That is why it is also referred to as *supervised learning*. Classification tasks not only enable to study and examine the existing sample data but also enable to predict the future behavior of that sample data.

1.2.1.2 Regression

The use of regression task enables to forecast future data values based on the present and past data values. Regression task examines the values of data and develops a mathematical formula. The result produced on using this mathematical formula enables to predict future behavior of existing data [6]. For example, a college professor wishes to a reach certain level of savings before her retirement. Periodically, she predicts what her retirement savings will be based on its current values and several past values. She uses a simple linear regression formula to predict this value by fitting past behavior to a linear function and then using this function to predict the values at points in the future. Based on these values, she then alters her investment portfolio [3].

1.2.1.3 Time Series Analysis

The use of time series analysis task enables to predict future values if the current set of values are time dependent. Time series analysis makes use of current and past sample data to predict the future values. The values to be used for time series analysis are evenly distributed as hourly, daily, weekly, monthly, yearly, and so on. A time series plot can be drawn to visualize the amount of change in data for specific changes in time [6]. There are three basic functions performed in time series analysis. In case one, distance measures are used to determine the similarity between different time series. In the second case, the structure of the line is examined to determine (and perhaps classify) its behavior. A third application would be to use the historical time series plot to predict future values [3].

1.2.2 Descriptive Model

A descriptive model identifies patterns or relationships in data. Unlike the predictive model, a descriptive model serves as a way to explore the properties of the data examined, not to predict new properties [3]. So a descriptive model enables to determine the patterns and relationships in a sample data [6]. This model is used to examine the data and explore its properties. That's how it is different from predictive model. This model further has four tasks:

1.2.2.1 Clustering

The use of clustering enables to create new groups and classes based on the study of patterns and relationship between values of data in a data bank [6]. Clustering is similar to classification except that the groups are not predefined, but rather defined by the data alone. Special type of clustering is called **Segmentation** (*unsupervised learning*). With segmentation a database is partitioned into disjointed groupings of similar tuples called segments [3].

1.2.2.2 Summarization

Summarization maps the data into subsets with associated simple descriptions. It extracts or derives representative information about the database. This may be accomplished by actually retrieving portions of the data. Alternatively, summary type information can be derived from the data [3]. So the summarization enables to summarize a large chunk of data containing in a web page or a document. The story of this summarized data enables to get the gist of the entire web page or the document. Thus, summarization is also known as **Characterization** or **Generalization** [6].

1.2.2.3 Sequence Discovery

Sequential Analysis or sequence discovery is used to determine sequential patterns in data. These patterns are based on a time sequence of actions [3]. So the use of sequence discovery enables to determine the sequential patterns that might exist in a large and unorganized data bank. The sequence in data bank is discovered using the time factor, i.e., by associating the data item by the time at which it was generated and the likes [6].

1.2.2.4 Association Rules

Link Analysis alternatively referred to as **Affinity Analysis** or **Association,** refers to the data mining task of uncovering relationships among data. The best example of this type of application is to determine the association rules. An association rule is a model that identifies specific types of data associations. These associations are often used in the retail sales community to identify items that are frequently purchased together [3]. It establishes association and relationships between large and unclassified data items based on certain attributes and characteristics. Association rules define certain rules of associativity between data items and then use those rules to establish relationship [6].

1.3 Association Rules Mining

As previously stated, it is not just the availability of the products that results in the enhancements of the sales of a particular store, but there are other factors as well. One of them is the proper placement of the items on the shelves. Shelf placement should always be done keeping in view the hidden associations between different products. Placing together such associated goods will result in increased sales of the store. To have better understanding of the concept, let's go through the following case study:

Mr. Junaid opened a general store in a crowded residential area. To his comfort, there was only one store in the vicinity, the Sheikh General store. So he was optimist for grabbing the attention of the customers very soon. But for his amazement, people still preferred sheikh general store for their buying. He started following the footsteps of the Sheikh's by taking the stock from the same wholesaler as the Sheikh's do, and treating the customers with politeness and honor. But still he was far away from meeting his sales goals, which was quite strange for him. He started examining Sheikh's quite critically. Every thing he did was compatible to those of Sheikh's. Then what's the difference? So he decided to change the shelf placement of his store. At the time, his shelves were arranged according to the item categories like all the biscuits, pretzels, and bread were placed on one shelf, drinks were on the other, snacks and sandwiches on the next, eggs were placed near butter, jam, honey and sandwich spread on separate shelf. Junaid was under impression that his shelf placement was more presentable easy, as compared to the Sheikh's, where different items were placed together on shelves, without any proper arrangement, according to Junaid's views. But surprised with his continuous failure while Sheikh's success, he adopted Sheikh's shelf placement as it is. Surprisingly, his sales also increased and soon his store became equally popular as that of Sheikh's. He then analyzed the keys of his success behind that change. He came across some very interesting results. He realized that shelf placement is actually how good you are in tempting a customer to buy something he hasn't planned to buy, and not just to help him buy the list of items he came from his home to buy. The trick is to target out the proper shelf placement technique. To target that out, he started analyzing the buying habits of the of customers, like if a customer is purchasing the milk, he might also be purchasing bread and eggs (which are associated to the breakfast), so by placing them together within close proximity may further encourage the sale of these items together within single visit to the store. For the customers purchasing snacks at tea time, he placed together snacks, biscuits, pretzels, sandwiches, pizzas, pastries and patties together, and for the teen-agers that are used to take fast food as lunch, he placed soft drinks closed with the burgers. And by doing this, he was now tempting a customer entered his store only to buy a burger, also to buy the Pepsi, and the girl purchasing nail polish, also a nail polish remover, and hence enjoying full fruit of his efforts.

The increase in Junaid's sales was due to the fact that Mr. Junaid discovered the hidden link, association, between different items. These associations are not casual relationships, so can not be targeted out plainly by any Entity Relationship Diagram, or any other Data Model, because they do not represent any relationship inherent in actual data, which is true in case of functional dependencies. There are certain measures of rule-interestingness. One of them is **Support**. The Support (s) for an association rule $X \Rightarrow Y$ is the percentage of transactions in the database that contain $X \cup Y$ [3]. The number of times the bread is purchased shows the support for bread in the database. The other measure of rule interestingness is the **Confidence**. The confidence or strength (α) for an association rule $X \Rightarrow Y$ is the ratio of the number of transactions that contain $X \cup Y$ to the number of transactions that contain X. confidence measure the strength of the rule, whereas support measures how often it should and the total number of transactions in D. The number of transactions required for the itemset satisfying minimum support is therefore referred to as the **minimum support count**. If an itemset satisfies minimum support, then it is a **frequent** itemset.

Association rule mining is a two-step process:

1. **Find all frequent itemsets:** by definition, each of those itemsets will occur at least as frequently as a pre-determined minimum support count.
2. **Generate strong association rules from the frequent itemsets:** By definition, these rules must satisfy minimum support and minimum confidence [2].

   The association rules are generated simply using the following formula:

   If

$$\frac{(\text{support}(\{Y, X\}))}{\text{support}(\{X\})} \geq \text{min\_conf}$$

   Then $X \Rightarrow Y$ is a valid rule.

Here X is called the **antecedent** of the rule, whereas Y makes the **consequent** of the rule.

1.4 Association Rule Mining Environments

The problem of association rule mining falls in two broad categories: The *Centralized Environment*, and the *Distributed Environment*.

In Centralized Environment, there is one, huge centralized database, from which the task is to identify the most frequently occurring together items. So to generate association rules in such an environment, not only the data to be examined is important, but also the size and amount of data. It requires large memories, for scanning the data, candidate set generation and support count calculation. Also, the efficiency of algorithm to be designed for centralized environment is very crucial.

In Distributed Environment, the data is either horizontally or vertically distributed across different nodes of a network, so the problem of one single huge database is solved to some extends, but it gives rise to another problem, i.e., the itemset found to be large at one node, need not be so in the entire network. Such problems are faced because of data skewness occur in the database [3]. Support and confidence respectively reflect the usefulness and certainty of discovered rules. A support of 2% for association rule means that 2% of all transactions under analysis show that bread and butter are purchased together. A confidence of 60% means that 60% of the customer who purchased bread, also bought the butter. Typically, association rules are considered interesting if they satisfy both a minimum support threshold and a minimum confidence threshold. Such thresholds can be set by users or domain experts.

1.3.1 Basic Concept

Let $I = \{i_1,i_2,\ldots,i_m\}$ be the set of items. Let D, the task-relevant data, be a set of database transactions where each transaction T is a set of items such that $T \subseteq I$. each transaction is associated with an identifier TID. Let A be a set of items. A transaction T is said to contain A if and only if $A \subseteq T$. An association rule is an implication of the form $A \Rightarrow B$, where $A \subset I$, $B \subset I$, and $A \cap B = \varphi$. The rule $A \Rightarrow B$ holds in the transaction set D with support s, where s is the percentage of transactions in D that contain A U B (i.e., both A and B). This is taken to be the probability, P(AUB). The rule $A \Rightarrow B$ has confidence c in the transaction set D if c is a percentage of transactions in D containing A that also contain B. This is taken to be the conditional probability, P(B|A). That is,

Support $(A \Rightarrow B) = P(AUB)$.

Confidence $(A \Rightarrow B) = P(B|A)$.

Rules that satisfy both a minimum support threshold (min_sup) and a minimum confidence threshold (min_conf) are called **strong**. By convention, we write support and confidence value so as to occur between 0% and 100%, rather than 0 to 1.0.

A set of items is referred to as an **itemset**. An itemset that contains k items is a k-itemset. The set {bread, butter} is a 2-itemset. **The occurrence frequency of an itemset** is the number of transactions that contain the itemset. This is also known, simply, as the **frequency**, **support count**, or **count** of the itemset. An itemset satisfies **minimum support** if the occurrence frequency of the itemset is greater than or equal to the product of *min_sup* property. So in order to determine whether the locally large itemset is also globally large or not, all the nodes broadcast its large itemsets across the network. But it results in increased message passing, so this is also a problem to be solved.

1.5 Existing Techniques

There are a number of techniques adapted to handle the problem of association rule mining. Each of these techniques strive to minimize the database scans, while generating smaller candidate sets to avoid extra efforts of counting support and maintaining the information about those false-positive candidates. The goal before each technique of association rule mining is to generate more and more true-positives. The techniques are as follows:

AIS algorithm is presented in paper [7], which is the earliest work for the mining of association rules of items in large databases. Its functionality is presented in detail in section 2.1.1. It takes a separate database scan for almost every step, like candidate itemset creation, large 1-itemset generation, support count, etc. The problem with this algorithm is that, it is confined to only the single consequent rule generation. Secondly the larger candidate set generated is its major drawback.

An algorithm **SETM** is presented in [8]. It works in the same way as **AIS**, but SQL is used to compute large itemsets. The detailed functionality is described in section 2.1.2. The technique discussed in **SETM** is also a single consequent rule generation technique. Its disadvantage is due to the larger size of candidate set generated. For each candidate itemset, the candidate generated has many entries as the number of transactions in which the candidates are present. Also to count the support for candidate itemsets, the candidate set is in wrong order and needs to be sorted on itemsets. After counting and pruning out small candidate itemsets that don not have minimum support, the resulting set of large items needs another sort on TID before it can be used for generating candidates in the subsequent passes.

The pioneer work is presented in [9] in which notorious **Apriori** algorithm is presented. All other subsequent algorithms are the adaptation of Apriori to some extents. Details of this paper are presented in section 2.1.3. The author discovered some shortcomings in his Apriori algorithm himself, so presented another version of Apriori called **AprioriTid**. It works in the same way as Apriori, but does not use the database for counting support after the first scan, instead it uses the pair of itemset and it's TIDs for this purpose. It works efficiently in later passes. The best features of both the algorithms were combined in another variant of Apriori, presented in the same paper called **AprioriHybrid.** In Apriori, the problem is that the database of transactions is scanned entirely for each pass. For AprioriTid algorithm, the database is not scanned after the 1st pass. Rather, the transaction id & candidate large k-itemsets present in each transaction are generated in every pass. But AprioriTid's performance is not better than Apriori's in initial stages, as there are too many candidate k-itemsets to be tracked during the early stages of the process. The AprioriHybrid algorithm is presented as a solution comprising of the best features of Apriori and AprioriTid, but the challenge is to determine the switch over point between the two algorithms.

The algorithm **DHP** (standing for **D**irect **H**ashing and **P**runing) is presented in [10], which is an extension of Apriori. It is confined to the generation of large itemsets only. Its details are given in section 2.1.4. It also uses the Apriori_gen() and Subset() function as were used in the Apriori.

PARTITION algorithm is presented in [11]. This algorithm takes two database scans. Firstly, for generation of all potentially large itemsets and store it as a set, this set is the superset of all the large itemsets. Secondly, to measure the support of these itemsets and storing them in their respective counters created. The detailed study of PARTITION Algorithm is presented in section 2.1.5. The main problem with PARTITION algorithm is to find out the accurate number of partitions for the given memory. So this must also be cater for while implementing.

The mining of association rules in distributed environment is discussed in section 2.2, which is an important and emerging field of database scenarios. An algorithm **DMA** (**D**istributed **M**ining of **A**ssociation rules), is presented in [13], which is an adaptation of Apriori in parallel systems. The performance of algorithm is compared with another algorithm, CD

(Count Distribution), which is the adaptation of Apriori in share nothing parallel systems. The details of DMA are given in section 2.2.2. It takes just a single database scan in the course of its execution.

The literature proves that the generation of large itemsets is the main problem in generating the association rules in large databases. It involves many problems like the candidate sets created for generation of large itemsets are very large; the database is supposed to be scanned again and again in order to create candidate sets and to generate their support counts. Also the pruning of the itemsets from the candidate sets, that are not large, is also a problem.

1.6 Scope of the project

The Association Rule Mining is a very effective research area, which helps in facilitating not only the retail industry problems of hidden association between different itemsets, but also in different other areas like telecommunications, effective advertising, targeted marketing, and inventory control.

In the centralized environment, the problem is to take the huge database and apply the association rule mining algorithm on that database, which definitely is not easy to be handled efficiently. Secondly, the extra database scans in that case is also a big problem in both the cases of time and space complexities. In this case huge memory is required to scan the entire database, which is either in partitions or in a single large database. So it needs resource-hungry architecture.

Chapter 2

Literature Survey

2. Literature Survey

One of the reasons behind maintaining any database is to enable the user to find interesting patterns and trends in the data. For example, in a supermarket, the user can figure out which items are being sold most frequently. But this is not the only type of 'trend' which one can possibly think of. The goal of database mining is to automate this process of finding interesting patterns and trends. Once this information is available, we can perhaps get rid of the original database. The output of the data-mining process should be a "summary" of the database. This goal is difficult to achieve due to the vagueness associated with the term 'interesting'. The solution is to define various types of trends and to look for only those trends in the database. One such type constitutes the Association Rules.

Association rule mining finds interesting association or correlation relationships among a large set of data items. With massive amounts of data continuously being collected and stored, many industries are becoming interested in mining association rules from their databases. The discovery of interesting relationships among huge amounts of business transaction records can help in many business decision making processes, such as catalog design, cross-marketing, and loss-leader analysis.

A typical example of association rule mining is **Market Basket Analysis**. This process analyzes customer buying habits by finding associations between the different items that customers place in their "shopping baskets". The discovery of such associations can help retailers develop marketing strategies by gaining insight into which items are frequently purchased together by customers. For instance, if customers are buying milk, how likely are they to also buy bread (and what kind of bread) on the same trip to the supermarket? Such information can lead to increased sales by helping retailers do selective marketing and plan their shelf space. For example, placing milk and bread within close proximity may further encourage the sale of these items together within single visits to the store. So association rule mining has always been a hot topic for the researchers, as it addresses the problem area of the retail industry, from where every one today is effected to some extent. The amount of research going on in the area predicts that in near future, we should expect computers to help each and every business of any volume, in every aspect. No matter whether it is shelf placement, Human Resource Management, customer's trend analysis, and introduction of new products based on those trends analyzed and what not.

2.1 Association Rule Mining in Centralized Architecture

In centralized environment, there is one, huge centralized database, from which the task is to identify the most frequently occurring together items. So to generate association rules in such an environment, not only the data to be examined is important, but also the size and amount of data. It requires large disks, for scanning the data for candidate set generation and support count calculation. Also, the efficiency of algorithm to be designed for centralized environment is very crucial.

Today, the data warehouses are mostly developed on centralized architecture, as they contain massive amount of data of the tens of years, so the organizations often dedicate one big machine with huge memory and fastest processing capabilities and RAM for the data warehouse at a single node. When data warehouses are centralized, so are the data mining algorithms, and so are the researches undertaken on the data mining. Association rule mining, too, is mostly done in centralized environment. Different techniques of centralized mining of association rules are devised so far, like Apriori, Sampling, Partitioning, Data parallelism, Task parallelism etc. In the following, we give the study of few notorious research papers of the centralized architecture of the association rule mining.

2.1.1 Mining Association Rules Between Sets of Item In Large Databases.

Agarawal et al., [7] presented in this paper an algorithm called **AIS**, for the mining of Association rules of items in large databases. According to algorithm, the database is scanned and candidate sets of frequently occurring itemsets are created. Their support is counted with database scan again. It reads the transaction and determines which large items of previous pass are present in that transaction. New candidate itemsets are generated by extending these large itemsets with other items in the transaction. The candidates generated from a transaction are added to the set of candidate itemsets for the pass, or the counts of the corresponding entries are increased if they were created by an earlier transaction.

2.1.2 Set-Oriented Mining for Association Rules in Relational Databases

Houtsma and Swami., [8] presented in this paper an algorithm, called **SETM**. It works in the same way as **AIS,** but SQL is used to compute large itemsets. The candidate set generations separate in SETM than counting. Candidate itemsets and the TIDs of the transactions containg it are saved in same sequential structure, which is sorted and aggregated at the end of pass in order to determine the support count of candidate set. The small candidate itemsets are pruned out after counting, as they do not have minimum support specified by the user. Then the set of the candidate set is again sorted on the basis of TIDs preparing it to be used in the next pass for generating candidate set.

2.1.3 Fast Algorithms for Mining Association Rules

Agarawal and Srikant., [9] presented in this paper an algorithm, called **Apriori**. It is termed as the pioneer work in their area. All other subsequent algorithms are the adoption of Apriori to some extents. This algorithm counts item occurrences from the database to determine large 1-itemset in first pass. In the next pass, the algorithm
1).Generates candidate itemsets
2).Checks the support count.

The algorithm uses a special function, apriori-gen () and uses the candidate itemsets of previous pass to generate large itemsets. It joins the previously determined large itemsets to make the candidate itemsets. It stores the candidate itemsets in a Hash tree, a special structure. The candidate itemsets generated by this algorithm is smaller one than that created by SETM and AIS. Also, it generates multiple consequent association rules, which AIS and SETM do not.

Another version of Apriori, **AprioriTid,** is also presented by authors in this paper, which works in the same way as Apriori, but does not use the database for counting support after the first scan, instead it uses the pair of itemset and its TIDs for this purpose. It works efficiently in later passes.

AprioriHybrid is another variant of the same algorithms which combines the best features of both Apriori and AprioriTid, i.e., using Apriori in earlier iterations and AprioriTid in later ones, to enjoy more benefits from both the algorithms. Basically in AprioriHybrid, it switches to AprioiTid when it expects that the set of candidate item sets at the end of the pass will fit in memory. Experimental results show that AprioriHybrid has excellent performance over large databases.

2.1.4 An Effective Hash-Based algorithm for Mining Association Rules

Park et al., [10] give in this paper, the algorithm **DHP** (standing for **D**irect **H**ashing and **P**runing), which is an extension of Apriori. It uses the hashing technique. It is confined to the generation of large itemsets, the step one of the mining association rules. It deals with the itemset generation up to 2-itemsets. It also uses the Apriori_gen() and Subset() function as were used in the base algorithm, Apriori. The major features of DHP are:
1).It generates large itemsets more efficiently.
2).It also reduces the transaction database.

It uses hashing technique for generation of candidate itemsets, especially for 2-itemsets and for reducing the database size. It uses the effective pruning techniques to progressively shrink the size of transaction database. The candidate itemset generated by previous algorithms was very large so tracking of k-itemsets in each transaction was not easy. Where as DHP trims the database right after generation of large 2-itemsetm, so the cost of computation is reduced for the subsequent iterations. As DHP generates Hash table in its first pass for storage of candidate sets, so it takes a bit longer time in first pass than Apriori. But its execution time for later iterations is much faster than Apriori.

2.1.5 An Efficient Algorithm for Mining Association Rules in Large Databases

Savasere et al., [11] presented **PARTITION** algorithm in this paper. This algorithm takes two database scans. Firstly, for generation of all potentially large itemsets and store it as a set, this set is the superset of all the large itemsets. Secondly, to measure the support of these itemsets and storing them in their respective counters created.

The working of this algorithm is divided in two phases. In phase I, the database is logically divided into non-overlapping partitions, which are considered one by one and large itemsets for each partition are generated and at the end all itemsets are merged to form the set of all potentially large itemsets. The phase II generates the actual support of these itemsets, to identify large itemsets. The partition sizes are chosen such that each partition can be accommodated in the main memory so that the partitions are read only once in each phase.
The database is read once in phase I and once in phase II. The small itemsets are pruned out. For each itemset, is associated its sorted TID list. A TID list for itemset l contains the TIDs of all transactions that contain the itemset 1 within a given partition. To count the support of all itemsets in a partition, this algorithm divides the cardinality of TID list by the total number of transactions in that partition. Initially, the tidlists for 1-itemsets are generated directly by reading the partition. The tidlist for a candidate k-itemset, is generated by joining the tidlists of the two (k-1)-itemsets that were used to generate the candidate k-itemset.

2.1.6 Mining Association Rules: Anti- Skew Algorithms

Lin and Dunham, [14] presented a series of algorithms, called AS-CPA, in which different algorithms were presented. In case a random sample can be drawn, the RSAS-CPA is presented, whereas in case a random sample is not to be drawn without scanning the database from the beginning, a SSAS-CPA is applied.
AS-CPA is the variation of the SPINC algorithm, which makes use of the cumulative count of each candidate itemset to achieve the illusion of a large partition. AS-CPA provides several effective techniques to filter out false candidate itemsets at an earlier stage, as compare to SPINC algorithm.

2.2 Association Rule Mining in Distributed Architecture

In distributed environment, the data is either horizontally or vertically distributed across different nodes of a network, so the problem of one single huge database is solved to some extends, but it gives rise to another problem, i.e., the itemset found to be large at one node, need not be so in the entire network. Such problems are faced because of data skew property. So in order to determine that whether the locally large itemset is also globally large or not, all the nodes broadcast its large itemsets across the network. But it results in increased message passing, so this is also a problem to be solved.

In Distributed architecture, a data warehouse is divided amongst the local nodes of the distributed system, so each node maintains its own data warehouse. This trend is mainly supportive for the big chains, having customers and branches all over the world. Centralized data warehouses for such organizations are not a sane idea, so the data warehouse is distributed among each node.

In distributed data warehouses, the data mining techniques for association rule mining are mostly adapted from centralized environment, and modified in order to meet the needs of the distributed architecture. Although a little, but the comprehensive work done so far in the distributed environment is also considered in this research. The literature surveyed for distributed association rule mining is as follows:

2.2.1 A Fast Distributed Algorithm for Mining Association Rules.

Cheung et al., [12] discussed in this paper the mining of association rules in distributed environment, as it is an important and emerging field of database scenarios. An algorithm **FDM** (standing for **F**ast **D**istributed algorithm for **M**ining) is proposed in this paper which in an adoption of DHP in parallel environment, with three variations i.e., FDM-LP, FDM-LUP and FDM-LPP. The performance of algorithm is compared with another algorithm, CD (Count Distribution), which is the adoption of Apriori in share nothing parallel systems.

As FDM is designed mainly for parallel systems (which can be easily extended to distributed systems) so the task of finding large itemset is divided into two subtasks:
1).To determine the locally large itemsets
2).To determine the globally large itemsets

The FDM has some distinct features. Firstly, the candidate sets are generated at each site using technique of Apriori. Secondly some itemsets from the candidate sets generated are pruned away using two different techniques, the local pruning and global pruning, to prune away those itemsets that are not truly the large itemsets. Thirdly, to determine the support counts to check whether the itemset is large or not, O(n) messages are required for each itemset. Whereas the straight forward adoption of some sequential technique requires $O(n^2)$ messages. To ensure O(n) messages, a technique count polling is introduced. In this technique, each candidate itemset is assigned to a polling site, which will determine whether that itemset is globally large or not by broadcasting polling requests for that particular itemset and collecting the local support counts for it and then determining the global support count. At each site, the large itemsets along with their support counts are sent to their respective polling site before polling request. The polling site then sends polling request to rest of the sites to collect the support counts and when all the support counts are received, the globally large itemsets are determined and broadcast to all the sites along with their global support counts. In this paper, the authors compared their technique with anther algorithm, the Count Distribution. The details of CD algorithm are given below.

CD (Count Distribution)

It generates the candidate itemsets using apriori-gen () function, at each site on the large itemsets found in previous pass. Then local support counts for all candidate itemsets are computed at each site, and then broadcasted to all the sites then the globally large itemsets are computed and so on. Same process is repeated for all the iterations. The number of messages to compute support count for each candidate itemsets in CD is $O(n^2)$.

2.2.2 Efficient Mining of Association Rules in Distributed Databases

Cheung et al., [13] enhanced their previous work discussed in [13] in this paper. The claim of O(n) messages passed to collect support count for a candidate itemset was not achieved by FDM successfully, and it remained O(n^2) messages. To cope with this problem, another algorithm DMA (Distributed Mining of Association rules), is presented. The performance of DMA is tested against CD algorithm. In DMA, to generate candidate itemsets, apriori-gen () is not applied directly, rather it is applied in such a way that it minimizes the candidate set to a greater extend than in the case of direct application of apriori-gen().
DMA also uses polling site technique to determine heavy itemsets. After local pruning, each site computes candidate set itemsets along with their support counts and sends it to their corresponding polling site. After receiving it, polling site sends polling request to the rest of sites to send the support count for that itemsets. As all the sites are already having the support counts of all itemsets so reply to polling request is sent by those remaining sites as well. The polling site then computes the global count, determines the heavy itemsets and broadcasts those heavy itemsets along with their global support counts to all the sites. In the whole procedure, the database partition at each site is scanned only once for calculating the support count and then those counts are stored in hash tree. This hash tree contains the support counts of both the heavy itemsets at that site and heavy itemsets at some other site. The later are stored in order to entertain the polling requests made by the remote sites so one database scan is done to compute the support counts of itemsets and are stored in the hash tree and retrieved from that hash tree when required. This optimizes the database scanning required for count exchange, which is equivalent as that done in the sequential algorithms.

As the candidate sets at all the sites are the same, so during polling requests, instead of sending the name of itemset, only their respective position is sent to optimize the size of the message over the network.
The algorithms tested for its performance in two ways i.e.
1).With fixed number of sites and varying support threshold and the size of database.
2).With different number of sites and fixed support threshold and database size.
In both cases DMA performs better than CD.

2.3 Problem Statement

In mining the association rules, the problem is decomposed in to two steps:

1. All the itemsets that have **support** above the user specified minimum support are generated. These itemsets are called **large itemsets.** All the others are said to be **small.**
2. Then from these large itemsets generate the association rules.

From the literature surveyed, it is obvious that the first step dominates the efficiency of the whole algorithm. If the number of candidate itemset is small and to-the-point, the rules generated on the basis of these itemsets would be the exact ones. Their will be a little effort spent on pruning the small itemstes during the iteration of algorithm in k steps.

After the creation of large itemsets, it is straightforward to generate rule out of it, using the above mentioned formula. So it is the step1 which determines the efficiency of the algorithm as the database is scanned for many times during this step. Firstly, the items are taken from the database, then their support is counted from the database, then 2-itemsets are created, and till k-itemset creation the database is scanned again and again, i.e., the same is done for the every iteration. As it is obvious that the mining of association rules is not done on a small database rather it could be on a huge database, or a Data warehouse, or some distributed database with multiple nodes. So this multiple scan of database physically means a lot. Excessive research has been done on this area.

An earlier work in this area was done in paper [7], but it is confined and restricted to only the single consequent rule generation. Secondly the larger candidate set generated is its major drawback.

The technique discussed in paper [8] is also a single consequent rule generation technique. Its disadvantage is due to the larger size of candidate set generated. For each candidate itemset, the candidate generated has many entries as the number of transactions in which the candidates are present.

Also to count the support for candidate itemsets, the candidate set is in wrong order and needs to be sorted on itemsets. After counting and pruning out small candidate itemstes that don not have minimum support, the resulting set of large items needs another sort on TID before it can be used for generating candidates in the subsequent passes.

For Apriori, discussed in [9], the database of transactions is scanned entirely for each pass, while in first pass the database is scanned twice, so it means for 10 iterations, 10+1 scans of the entire database.

For AprioriTid algorithm, the database is not scanned after the 1st pass. Rather, the transaction id & candidate large k-itemsets present in each transaction are generated in every pass. But AprioriTid's performance is not better than Apriori's in initial stages, as there are too many candidate k-itemsets to be tracked during the early stages of the process.

The Hybrid algorithm is presented as a solution comprising of the best features of Apriori and AprioriTid, but the challenge is to determine the switch over point between the two algorithms.

The problem of efficient itemset generation is tackled up to 2-itemset generation only in [10].

PARTITION algorithm is presented in [11]. The main problem with PARTITION algorithm is to find out the accurate number of partitions for the given memory. So this must also be cater for while implementing.

Itemsets are reduced in the second iteration in paper [12]. So other iterations might have a larger set of candidate itemsets. Also O(n) messages claim is not met. Actually it remained $O(n^2)$.
The problem with AS-CPA algorithm family, presented in [14] is that it takes (2n-1)/n scans over the database, where n is the number of partitions. So as the number of partitions increase, the number of scans over the database also increases.

The literature survey proves that the generation of large itemsets is the main problem in generating the association rules in large databases. It involves many problems like the candidate sets created for generation of large itemsets are very large; the database is supposed to be scanned again and again in order to create candidate sets and to generate their support counts. Also the pruning of the itemsets from the candidate sets, that are not large, is also a problem. So in order to get efficient results, DMA presented in [13] should be implemented in centralized environment.

Chapter 3

Problem Domain and Proposed Solution

3. Problem Domain and Proposed Solution

Association rule mining is grabbing more and more attention of the researchers of the Data Mining because this is the area of the common interest of every one. It mainly addresses the problem that the retail community faces, which affects almost every body. Association rule mining is basically to find out those special and interesting relationships between different items which can never be targeted out by any other means. The Association model is often associated with "market basket analysis", which is used to discover relationships or correlations in a set of items. It is widely used in data analysis for target direct marketing, catalog design, and other business decision-making processes. A typical association rule of this kind asserts the likelihood that, for example, "70% of the people who buy spaghetti, cola drink, and sauce also buy garlic bread."

Association models capture the co-occurrence of items or events in large volumes of customer transaction data. Because of progress in bar-code technology, it is now possible for retail organizations to collect and store massive amounts of sales data, referred to as "basket data." Association models were initially defined on basket data, even though they are applicable in several other applications. Finding all such rules is valuable for cross-marketing and mail-order promotions, but there are other applications as well: catalog design, add-on sales, store layout, customer segmentation, web page personalization, and target marketing.

Traditionally, association models are used to discover business trends by analyzing customer transactions. However, they can also be used effectively to predict Web page accesses for personalization. For example, assume that after mining the Web access log, Company X discovered an association rule "A and B implies C," with 80% confidence, where A, B, and C are Web page accesses. If a user has visited pages A and B, there is an 80% chance that he/she will visit page C in the same session. Page C may or may not have a direct link from A or B. This information can be used to create a dynamic link to page C from pages A or B so that the user can "click-through" to page C directly. This kind of information is particularly valuable for a Web server supporting an e-commerce site to link the different product pages dynamically, based on the customer interaction.

Since association rule algorithms work by iterative enumeration, they work best for sparse data sets, that is, data sets where each record contains only a small fraction of the total number of possible items (if the total number of items is very large). Algorithm performance degrades exponentially with increasing number of frequent items per record. Therefore, to get good runtime performance, one of the following conditions should hold:

- If the data set is dense, the number of possible items (candidate itemset) is small.
- If the number of possible items is large, the data set is sparse.
- The data set becomes progressively sparser with increasing item set length due to the application of the minimum support threshold.

The last condition holds for higher minimum support values.

3.1 Problem Domain

There are some issues in association rule mining that must be considered while devising an efficient association rule mining algorithm, because it can degrade the performance of the algorithm if not catered for. Some of the issues are:

3.1.1 Multiple Database Scans

So, with the help of association rule mining, the aim is to find out the hidden associations between items and generate association rules, which facilitates the retailers to place the items of high confidence value together. The problem with the association rule mining is the multiple database scans. For example, at first, different items are examined as whether they are frequently sold items or not. Then the frequently sold items are examined together to find out the confidence between two large itemsets. For the large 2-itemsets, confidence level is measured, and then large 3-itemsets are found, and so on, up till large k-itemsets. Before generating a particular large itemset, its respective candidate itemset is generated first. During all this process, the database is scanned again and again. Multiple disk I/Os are never appreciated in the computing world; no matter how much smaller is the size of that file. While talking about data mining disk I/Os, we always refer to a huge data warehouse, having data of over decades of years. So in data mining, the primary task should always be to minimize the database scans, the disk I/Os, and not the query execution time.

3.1.2 Large Candidate Set Size

Large itemsets are generated on the basis of its respective candidate sets. For example, in order to generate large 3-itemset, its candidate 3-itemset is considered, which is created from large 2-itemset. So candidate set plays the vital role in the efficiency of the algorithm. The large sized candidate sets result into the wasted efforts of considering false candidates as the large itemsets. So in order to have an efficient algorithm for association rule mining, efforts should be made to generate accurate and smaller candidate sets, having almost all the candidate items that should be the large items as well.

3.1.3 Algorithm Execution Time

Although the algorithm execution time has never been an issue in data warehousing queries, because these queries run on huge data gluts, having data of twenty to forty years, so obviously data warehouse queries are time consuming. In fact the data mining queries are asked to help in reaching some decision. These decisions can change the over all shape of the business in question, hence has a long lasting effect on it. So the obvious delays in query response time are tolerated with patience in business community. To understand this scenario, let's take the example of ABC TV manufacturing company.

ABC TVs want to establish a new branch of their company in a neighboring/ European country. Before going to take such a decision and invest in a foreign country, they want to know is it safe and sane to invest there or not? Will it proved to be a good decision? To answer such queries, they need to consider few factors. Like, is TV-Watching the favorite and popular time passing activity there or not? What is the annual sale of TV's there? What is the standard of TV channels, and the programs they telecast? And what are the average prices of TV's in the market out there? To get such queries answered, a huge database needs to be considered, having records of some 10-15 years, and telling about the sales transactions of TV sets. Manually, such queries would be taking at least a month to be answered.

Whereas, using some data mining algorithm, results could be efficiently and quickly generated. So, if this data mining algorithm takes some time, of course longer than that of the usual operational queries, it is just nothing as comparing it with the benefits they are getting from it. The algorithm will tell them, after analyzing the TV customer's data, about the average sale prices of TV sets, the average number of TV sets sold there, the growth in sales of TV sets before some big sporting event (like FIFA world cup, Olympics, etc.), and will finally help the top management of company to decide whether to invest there or not, in the time span that is much much shorter than that would have been taken by the management manually, and with much accuracy that is intractable to be achieved through manual system.

3.1.4 Accurate Number of Partitions

Partitioning is the technique of dividing the dense centralized database into a number of partitions, and then executing the algorithm on each partition by bringing the partitions one by one in the memory. A partition once brought into the memory for execution must not be brought again. Also care must be taken while making partitions so that the made partition should fit well in to the memory, and hence the need for bringing a partition rapidly into the memory is minimized.

3.1.5 Data for Association Models

Association models are designed to use sparse data. Sparse data is data for which only a small fraction of the attributes are non-zero or non-null in any given row. Examples of sparse data include market basket and text mining data. For example, a market basket problem, there might be 1,000 products in the company's catalog, and the average size of a basket (the collection of items that a customer purchases in a typical transaction) is 20 products. In this example, a transaction/case/record has on average 20 out of 1000 attributes that are not null. This implies that the fraction of non-zero attributes on the table (or the density) is 20/1000, or 2%. This density is typical for market basket and text processing problems. Data that has a significantly higher density can require extremely large amounts of temporary space to build associations.

Association models treat NULL values an indication of sparse data. The algorithm doesn't handle missing values. If the data is not sparse and the NULL values are indeed missing at random, it is necessary to perform missing data imputation (that is, "treat" the missing values) and substitute the NULL values for a non-null value.

3.1.6 Data Skew

One of the major problems of the Partition algorithm is data skew, which refers to the irregularity of data distribution over the entire database. For example, the October earthquake of Pakistan caused a drastic demand for tents. Per month demands of the tents for the affected areas were about 200,000, and the situation remained so throughout the winters. By using the sequential approach for partitioning the centralized database, if we take the data of a quarter of the year as one partition, then according to the 4th partition of 2005's data, tent will emerge as a large itemset, which in fact is not. As a contrary, the annual sale of tents in Pakistan, before the October earthquake, is 25 tents per year. This is the outlier, caused by the data skew of the abnormal circumstances. And if we take the transactions randomly from the database to build different partitions, the October-Dec'05's tragic condition's transactions will be distributed throughout the database, so no problem of outliers will emerge.

Data skew can cause the algorithm to generate many false candidate itemsets. One way to overcome the data skew is the randomness of data across all partitions. However, this conflicts with the goal of exploiting sequential I/O to speed up reading the database. Even without data skew, unless each item is distributed rather uniformly over database, and the size of each partition is large enough to capture this uniformness, the chance of a local large itemset being a global large itemset can be small, and the number of candidate itemset generated can be very large. However, this conflicts with the main idea of partitioning: processing one partition in memory at a time to avoid multiple scans over database from disk.

3.2 Proposed Solution

Efficient association rule mining algorithm can be devised by finding solutions to some of the above mentioned factors. Like the algorithm must not take frequent database scans, should create accurate and precise candidate sets with lowest possible number of false candidates, should consider records at random to avoid data skew, and be a bit quicker as well. The functionality of C-ARMing project is shown in fig.3-1.

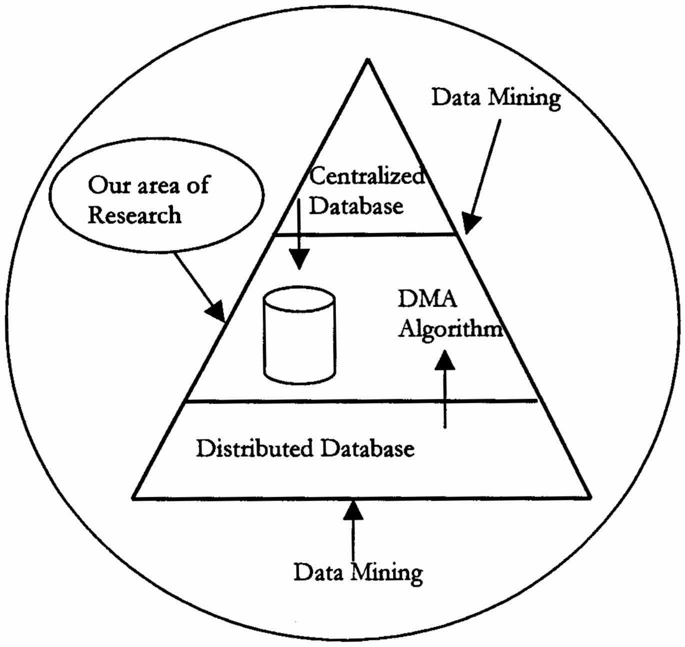


Figure 3-1: Functionality of C-ARMing

In the following, we present the factors by which we have improved our algorithm and minimized the overheads of the techniques in the market.

3.2.1 Decreased Disk I/Os

Like other data mining techniques that must process enormous databases, association rule mining is inherently disk-I/O intensive. It takes multiple database scans in order to calculate the support counts of the candidate itemsets, to approve them as large itemsets or not. Which in large databases, means a lot of extra work to be done. These I/O costs can be reduced in two ways: by reducing the number of times the database needs to be scanned, or through parallelization, by partitioning the database between several machines which then perform a parallel association rule mining algorithm. In recent years much progress has been made in both directions.

In our devised algorithm, the algorithm takes only a single database scan over each partition to create large itemsets. We are using a formula, as is used by Cheung et al [13] in DMA algorithm, to check the support of an itemset. To calculate locally large itemsets, $s * D_i$ is used, where s is the support of the itemset, set by the user, and D_i is the size of each partition. For global support count, $s * D$ is used, where s is same support, while D is the size of entire central database. So now, instead of calculating the occurrence frequency of an itemset by counting its actual occurrences in the database, it is simply counted through this formula, and hence results in the decreased disk /Os.

3.2.2 Dataset

The data used is synthetic data, which is created exclusively for this study. Each tupple of the database is of the form <TID> <A, B, C, D,...> where TID is the Transaction Identifier, which is the primary key to uniquely identify each tupple, and the literals represent the retail store's items bought together in each transaction. For example, take a transaction <T1004> <A, C, D> for example, where A represents butter, C represents bread, and D represents milk, so we can say that transaction number 1004 represents the purchase of breakfast items by a customer during his visit to the store. The size of the database used is 100,000 transactions. This synthetic data represents different transaction data items with literals, which helps reducing the size of the database and improves the algorithm processing speed.

3.2.3 Randomization of the Partitions

To calculate large itemsets, the database should be partitioned randomly, i.e., for each partition, transactions should be picked from the database randomly. We devised a random function, which randomly picks transactions and store them in partitions, and at the end, equal-sized, n number of partitions are created. These partitions are random, equal-sized, and non-overlapping. By non-overlapping, we mean that a transaction once picked for a particular partition, have 0% probability to be picked up again for any other partition.

4. System Design

The project takes MS Excel sheet as an input. The database stored in that Excel sheet is the exclusively created synthetic database, which is used to represent the actual transaction database's items with literals. We know that association rule mining is a two-step process:

1. **Finding all frequent itemsets:** by definition, each of those itemsets will occur at least as frequently as a pre-determined minimum support count.
2. **Generating strong association rules from the frequent itemsets:** By definition, these rules must satisfy minimum support and minimum confidence [2].

   The association rules could be simply generated with the help of the formula presented in section 1.3.1.

Our research is confined to the step 1, as the step 2 is trivial after calculation of accurate large itemsets.

4.1 Major Modules & Architectural Diagram

The main modules of our project are given below, while the Architectural Diagram is mentioned in the fig. 4-1.

- Database Acquisition
- Database Randomization
- Partition Creation
- Partition Loading
- Locally Large Itemset Creation
- Globally Large Itemset Creation

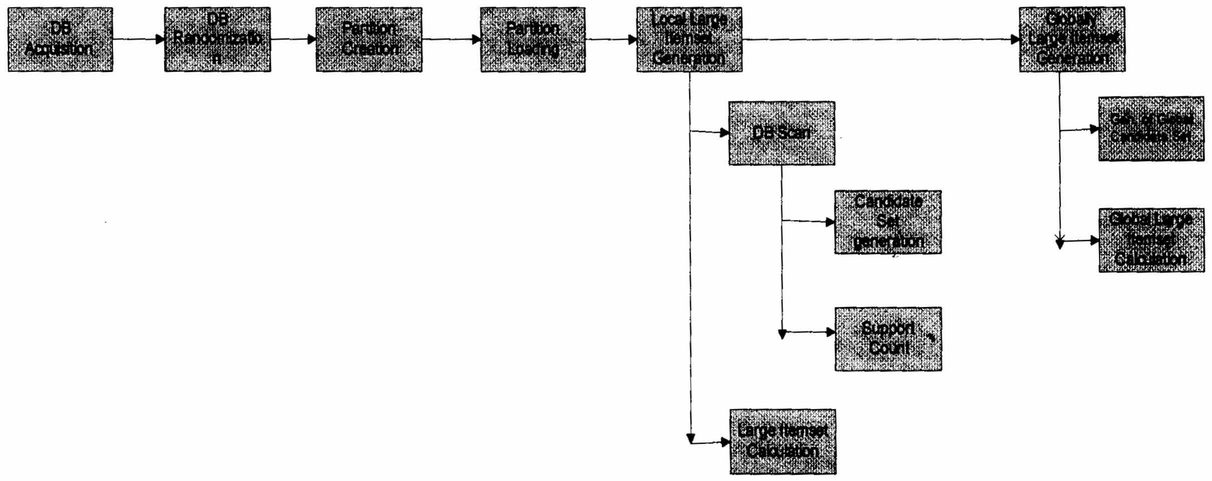

DB Acquisition
DB Randomizatio n
Partition Creation
Partition Loading
Local Large Itemset Generation
Globally Large Itemset Generation
DB Scan
Candidate Set generation
Support Count
Large Itemset Calculation
Gen. of Global Candidate Set
Global Large Itemset Calculation

Figure 4-1: Architectural Diagram

The detailed description of modules is given below.

4.2 Database Acquisition

This project takes as an input the synthetic database, which is specially crated for this research. The database is created in MS Excel. The size of the database is 100,000 tuples. Records are in the form of <TID> <list of items>, where TID is the transaction identifier of each record for its unique identification in the database, the primary key. The synthetic database is actually the representative of the transaction database because it represents the items that are purchased together in a single transaction. Instead of representing those transactions in the form of the actual items (like Milk, Butter, Tea bags etc.), synthetic database represents these items in the form of literals. So a transaction of the Transaction database will be represented in the following way in the Synthetic database:

Transaction database record= <T10056><Burger, Amrat, French Fries, Shashlik>.
Corresponding Synthetic database record= <T10056><A, B, D, E>.

So throughout the database, A, B, D, and E will be used to represent Milk, Butter, Bread, and Corn Flakes, respectively.

4.3 Database Randomization

Transactions in a database are stored as the transactions physically occur. So we can say that a page from the database is the snapshot of the transactions of some specific time period. For example, it is observed that the sales of eggs are higher during winters and falls during summer. So transactions occurred during winters will clearly show us this observation. This is not recommended for analysis, because the database is going to be partitioned before analysis, and sequential reading of database will render us with the items which might be frequently purchased during some specific time period, but actually are not the frequently purchased items. Example could be taken as that of the sales of the tents during earthquake 2005, as mentioned in chapter 3. To avoid such outliers, we have randomized our database. This randomization of the records results in a database, having the records of same climatic conditions scattered over the database, so only the true frequently sold items of the database are marked as frequent items.

4.4 Partition Creation

Partitioning is the technique of dividing the huge centralized database into a number of partitions in order to speed up the specified processing. It is basically to help in memory management in case of having huge data to be processed with low minimum primary memory. So, the huge data will be processed in that memory.

The algorithm divides the database into four, non-overlapping partitions. The partition creation module stores its output in four different columns of a separate MS Excel sheet.

4.5 Partition Loading

Previously, in the centralized databases, the entire database was supposed to be loaded into the memory for the creation of large itemsets in the database. We have adopted the technique

4.6.1.1 Support Count

In his step, the algorithm counts the occurrences of each item in the loaded partition. These occurrences are actually the supports of these items, as defined in chapter 1. On the basis of these supports, the items are qualified as whether the large or the small itemsets.

4.6.1.2 Candidate Set Generation

Large itemsets are created on the basis of their respective candidate sets. Candidate set is composed of the items that might possibly be the actually large itemsets. The candidate set generation and the support count is done during the single database scan.

4.6.2 Large Items Calculation

For the creation of Large-1 itemsets, the candidate set is matched against the supports of the items. If support of the item is greater than the minimum support specified by the user, then the item is marked as a Large Itemset. All the rest of the items are pruned away, hence minimizing the chance for the false candidates.

For large 2-itemset calculation, candidate 2-itemset is created first. Candidate 2-itemset is created by joining large 1-itemset with itself. It is denoted with l_i * l_i. This joining will result into a set of 2-items like AB, BD, CE etc. then their confidence is measured, and qualifying items are marked as large 2-itemsets, and so on till k-itemsets.

4.7 Globally Large Itemset Creation

All the large itemsets from all the partitions are then considered globally, i.e., whether the locally large itemsets also the globally larger or not? This module helps in finding the most accurate itemsets so that the association rules could only be found between those itemsets only.

The sub modules of this module are as follow:

4.7.1 Global Candidate Set Generation

In order to find out large itemsets, we know that its respective candidate set is generated first. Locally, it is created by scanning the database. For global candidate set, scanning the database again is contrary to the idea of the partitioning technique. We use the locally large itemsets as the candidate set for the calculation of global large itemsets. For example, in order to have candidate 1-itemset, the large 1-itemsets from all the partitions are considered as the candidate 1-itemset. Large 2-itemsets of all the partitions are considered as candidate 2-itemsets, and so on.

4.7.2 Global Large Itemset Calculation

All the items of candidate 1-itemset are then considered for the globally large 1-itemset. The supports of all the items are merged from their respective supports, from each partition. Then the items are matched against the condition **X. Support ≥ min_support,** where support is now the support of the item in the entire database. For all the items for which the condition holds true are marked as globally large 1-itemsets. Same is done for all the itemset, till the creation of globally large k-itemset.

Chapter 5

Implementation

5. Implementation

Implementation includes all the details that were required to make the system operational. The development tools and technologies to implement the system and also reasons for selecting particular tool is discussed. Then the modules being translated into the implementation tool will be descried.

Software selection is very important step in developing a computer based system. The software that is used is capable of meeting the requirement of the proposed system. After considering the number of tools available these days such as Visual Basic, Visual C++.Net, Visual C, Java we choose MS Excel, Visual Basic.Net and MATLAB.

5.1 MS Excel

Microsoft Excel (full name **Microsoft Office Excel**) is a spreadsheet program written and distributed by Microsoft for computers using the Microsoft Windows operating system and for Apple Macintosh computers. It features an intuitive interface and capable calculation and graphing tools which, along with aggressive marketing, have made Excel one of the most popular microcomputer applications to date. It is overwhelmingly the dominant spreadsheet application available for these platforms and has been so since version 5 in 1993 and its bundling as part of Microsoft Office.

MS Excel provides us its built-in workbooks for storage of huge data in any or all of the three worksheets of workbook, as well as provides us the facility to add more worksheet in a workbook to fulfill the requirement of data. Then we can perform mathematical, statistical or accounting functions. Our dataset that we used for finding large item set is in two excel sheets, as it consists of 1 lac records. After randomizing the data and creating partitions our final data is in sheet 3, this data set is then provided to MATLAB for large item set.

5.2 Visual Basic .NET (VB.NET)

Visual Basic .Net is an object-oriented computer language that can be viewed as an evolution of Microsoft's Visual Basic (VB) implemented on the Microsoft .NET framework. Its introduction has been controversial, as significant changes were made that broke backward compatibility with VB and caused a rift within the developer community.

The great majority of VB.NET developers use Visual Studio .NET as their integrated development environment (IDE). SharpDevelop provides an open-source alternative IDE.

Like all .NET languages, programs written in VB.NET require the .NET framework to execute.

Microsoft .Net is an umbrella term that applies to a collection of products and technologies from Microsoft. All have in common a dependence on the Microsoft .NET Framework, a component of the Windows operating system. For more details about .Net Framework, see Appendix.

5.2.1 Creation of Randomized Partitions

The following code creates the randomized partitions for the C-ARMing project.

```
Public Sub fLoadExcelFile()

    Dim objwb As Object
    Dim objws As Object
    Dim partNo As String
    Dim irow, icol, i, j, k As Integer
    Dim sXLSPath As String
    sXLSPath = TextBox1.Text
    Dim obj As New Excel.Application
    dt1.Columns.Add("Col1", System.Type.GetType("System.String"))
    dt1.Columns.Add("Col2", System.Type.GetType("System.String"))
```

```
dt2.Columns.Add("Col3", System.Type.GetType("System.String"))
dt2.Columns.Add("Col4", System.Type.GetType("System.String"))
Try
  'Get open application
  If File.Exists(sXLSPath) = False Then
    MsgBox("XLS fiel Not Found")
  End If
  objwb = obj.Workbooks.Open(sXLSPath)
  Dim R As Excel.Range
  Dim R1 As Int64
  R = objwb.Worksheets(1).usedrange()
  Dim r2 As Excel.Range
  r2 = objwb.Worksheets(2).usedrange
  Dim max As Int64
  Dim totRow As Int64
  totRow = R.Rows.Count + r2.Rows.Count
  Dim arr(totRow + 2, 2) As Object
  Dim arrInd As Int64
  arrInd = 0
  ProgressBar1.Maximum = R.Rows.Count
  max = R.Rows.Count
  ProgressBar1.Value = 0
        For i = 1 To R.Rows.Count
        Dim ar() As Object = {objwb.Worksheets(1).Cells(i, 1).Value,
        objwb.Worksheets(1).Cells(i, 2).Value}
    Dim newRow As DataRow = dt1.NewRow
    Dim a As Int64
    Dim b As Int64
    a = 1
    b = 0
    For j = 0 To UBound(ar)
```

```
      newRow(j) = ar(j)
      arr(arrInd, a) = ar(j)
      a = a + 1
    Next
    arrInd = arrInd + 1
    dt1.Rows.Add(newRow)
    ProgressBar1.Value = ProgressBar1.Value + 1
            Label1.Text = "Records " & ProgressBar1.Value & " Of
            Sheet 1 are Processed"
    Application.DoEvents()
Next
R = objwb.Worksheets(2).usedrange
ProgressBar1.Maximum = R.Rows.Count
ProgressBar1.Value = 0
For i = 1 To R.Rows.Count
        Dim ar() As Object =  {objwb.Worksheets(2).Cells(i,   1).Value,
        objwb.Worksheets(2).Cells(i, 2).Value}
    Dim newRow As DataRow = dt2.NewRow
    Dim a As Int64
    Dim b As Int64
    a = 1
    b = 0
    For j = 0 To UBound(ar)
      newRow(j) = ar(j)
      If ar(j) = "" Then
        b = 1
      Else
        b = 0
        arr(arrInd, a) = ar(j)
        a = a + 1
      End If
    Next
    If b = 0 Then
      arrInd = arrInd + 1
    End If
    ProgressBar1.Value = ProgressBar1.Value + 1
        Label1.Text = "Records " & ProgressBar1.Value & " Of  Sheet 2 are
        Processed"
    Application.DoEvents()
  Next
  Dim MyValue, maxVal, part As Integer
  ProgressBar1.Maximum = arrInd
  ProgressBar1.Value = 0
  maxVal = arrInd
  part = maxVal / 4
  For i = 0 To part – 1
  ' Initialize random-number generator.
         Randomize(0)
  ' Generate random value between 1 and 100000.
         MyValue = CInt(Int((maxVal * Rnd()) + 0))
         With objwb.worksheets(3)
            .Cells(i + 1, 1).Value = arr(MyValue, 1)
            .Cells(i + 1, 2).Value = arr(MyValue, 2)
   ' Initialize random-number generator.
         Randomize(0)
  'Generate random value between 1 and 100000.
         MyValue = CInt(Int((maxVal * Rnd()) + 0))
         .Cells(i + 1, 3).Value = arr(MyValue, 1)
         .Cells(i + 1, 4).Value = arr(MyValue, 2)
  ' Initialize random-number generator.
         Randomize(0)
   'Generate random value between 1 and 100000.
```

These are comprehensive collections of MATLAB functions (M-files) that extend the MATLAB environment to solve particular classes of problems. Areas in which toolboxes are available include signal processing, image processing, image acquisition, control systems, neural networks, fuzzy logic, wavelets, simulation, and many others.

The code for the modules describe in previous chapter is as follows.

5.3.1 Partition Loading

```
for i=1:4


  if i==1
    data_set='B1:B25000'
      elseif i==2
        data_set='D1:D25000'
      elseif i==3
        data_set='F1:F25000'
      elseif i==4
        data_set='H1:H25000'
  end]
  [n,T]= xlsread('dataset.xls',3,data_set);
```

5.3.2 Finding Locally Large 1-itemset

```
      % find A
      idx = strfind(T, 'A');
      idx(:,1);
      countA=cal_count(idx);
      countA
      aa(q)=countA;
      q=q+1;
      aa;
```

```
                MyValue = CInt(Int((maxVal * Rnd()) + 0))
               .Cells(i + 1, 5).Value = arr(MyValue, 1)
               .Cells(i + 1, 6).Value = arr(MyValue, 2)
         ' Initialize random-number generator.
               Randomize(0)
         'Generate random value between 1 and 100000.
               MyValue = CInt(Int((maxVal * Rnd()) + 0)) '
               .Cells(i + 1, 7).Value = arr(MyValue, 1)
               .Cells(i + 1, 8).Value = arr(MyValue, 2)
           End With
           ProgressBar1.Value = ProgressBar1.Value + 1
           Label1.Text = " Writing Records in Sheet 3"
           Application.DoEvents()
         Next
         Label1.Text = "Process Completed"
         obj.Workbooks.Close()
       Catch ex As Exception
         MsgBox(ex.Message)
       End Try
   End Sub
```

5.3 MATLAB 7.0:

MATLAB integrates mathematical computing, visualization, and a powerful language to provide a flexible environment for technical computing. The open architecture makes it easy to use MATLAB and its companion products to explore data, create algorithms, and create custom tools that provide early insights and competitive advantages.
MATLAB has evolved over a period of years with input from many users. In industry, MATLAB is the tool of choice for high-productivity research, development, and analysis.
MATLAB features a family of add-on application-specific solutions called *toolboxes*. Moreover toolboxes in MATLAB, allow one to *learn* and *apply* specialized technology.

```
% find B
idx = strfind(T, 'B');
idx(:,1);
size(idx);
countB=cal_count(idx);
countB
bb(c)=countB;
c=c+1;
bb;

% find C
idx = strfind(T, 'C');
idx(:,1);
size(idx);
countC=cal_count(idx);
countC
cc(g)=countC;
g=g+1;
cc;

% find D
idx = strfind(T, 'D');
idx(:,1);
size(idx);
countD=cal_count(idx);
countD
dd(b)=countD;
b=b+1;
dd;

 %  find E
```

```
        idx = strfind(T, 'E');
        idx(:,1);
        size(idx);
        countE=cal_count(idx);
        countE
        ee(y)=countE;
        y=y+1;
        ee;
    % ************************find large 1-itemset************************

        s=struct('ts',{});

        s2=struct('ts',{});
        lit_ind=1;
        %  s=55%
        %  D=25000
        % thus using formula
        %  count=s*D=55%*2500 = 13500
            if countA>13500
                cha=1;
                large_1(lit_ind)='A';
                lit_ind=lit_ind+1;
                msgbox('A is Large 1-itemset')
            end
            if countB>13500
                chb=1;
                large_1(lit_ind)='B';
                lit_ind=lit_ind+1;
               msgbox('B is Large 1-itemset')
            end
             if countC>13500
            chc=1;
            large_1(lit_ind)='C';
            lit_ind=lit_ind+1;
            msgbox('C is Large 1-itemset')
          end
        if countD>13500
            chd=1;
            large_1(lit_ind)='D';
            lit_ind=lit_ind+1;
            msgbox('D is Large 1-itemset')
           end
        if countE>13500
        che=1;
        large_1(lit_ind)='E';
        lit_ind=lit_ind+1;
        msgbox('E is Large 1-itemset')
      end
```

5.3.3 Finding Locally Large 2-itemset

```
%%%%%%%%%%%%%%%%    large 2 item set    %%%%%%%%%%%%%%%%%%%%%%

 lit_ind
 large_1
 vt=1;
 rt=1;
 for z=1:lit_ind-1
   for k=z+1:lit_ind-1
     large_2=strcat(large_1(z),',');
     large_2=strcat(large_2,large_1(k));
     s2(rt).ts=large_2;
     rt=rt+1;
   end
```

```
end
a=1;
for p=1:rt-1
   large_2=s2(p).ts
   idx = strfind(T, large_2);
   idx(:,1);
   size(idx);
   count_larg_2=cal_count(idx);
   if (count_larg_2>13500)
      ch_21=1;
      fi_count(a)=count_larg_2
      ch_2(l)=count_larg_2;
      l=l+1;
      ch_2;

      s(a).ts=large_2;
      a=a+1;
      msgbox(large_2)
   end
   if(a>2)
      vt=size(s)
      var=1;
      for r=1:vt(2)-1
         for u=r+1:vt(2)
            cand_st = strcat(s(r).ts,',')
            cand_st = strcat(cand_st,s(u).ts)
         end
      end
```

5.3.4 Finding Locally Large k-itemset

```
[t1,t2,t3,T]=p1('B1:B25000');
f1=1;f2=1;f3=1;f4=1;cnd_item_st=1;f6=1;cnd_item=1;f8=1;sum_21=zeros(5);sum_21a=0;su
m_22a=0;sum_23a=0;sum_44a=0;
sum_21b=0;sum_22b=0;sum_23b=0;sum_44b=0;
sum_21=0;sum_22=0;sum_33=0;,sum_44=0;
if nargin<3
   t3.cnt=0;
   t3.ele=' ';
end

sz1=size(t1);
for kl=1:sz1(2)
   t{kl}=t1(kl).ele;
   s11(kl)=t1(kl).ele;
   cnt1(kl)=t1(kl).cnt;
end

sz=size(t2);
if sz~=0
f_2=1
if sz(2)>1
   for kj=1:sz(2)
      kj
      t{kl+kj}=t2(kj).ele;

      cnt12(kj)=t2(kj).cnt

      item(f1)=t2(kj).cnt;
      b=t2(kj).ele
      [c1,c2,c3]=check_1(b)
      f1=f1+1;
     item(f1)=c1
      f1=f1+1;
      item(f1)=c2
      f1=f1+1;
      item(f1)=c3
      f1=f1+1;
sum_21=0;
      for x=1:4

         sum_21=sum_21+item(kj-1+x)
      end

      sum_21a(f_2)=sum_21;
      f_2=f_2+1;
   end
else
   kj=1;
   t{kl+kj}= t2.ele;
   cnt12(kj)=t2.cnt;
   item(f1)=t2.cnt;
   b=t2.ele;
   [c1,c2,c3]=check_1(b)
   f1=f1+1;
   item(f1)=c1;
   f1=f1+1;
   item(f1)=c2;
   f1=f1+1;
   item(f1)=c3;
   f1=f1+1;
   sum_21=0;
   for x=1:4
      sum_21=sum_21+item(x)
   end

 sum_21a(f_2)=sum_21;
end
sz3=size(t3);
f1b=1;f_2b=1;
if (sz3(2))==1

   t{kl+kj+1}=t3.ele;
   item_b(f1b)=t3.cnt;
   b=t3.ele;
   [c1,c2,c3]=check_1(b)
   f1b=f1b+1;
   item_b(f1b)=c1;
   f1b=f1b+1;
   item_b(f1b)=c2;
   f1b=f1b+1;
   item_b(f1b)=c3;
   f1b=f1b+1;
   sum_21b=0;
   for x=1:4
      sum_21b=sum_21b+item_b(x)
   end

 sum_21b(f_2b)=sum_21b;

elseif sz3(2)>1
   f1b=1;f_2b=1;
   for km=1:sz3(2)
      t{kl+kj+km}=t3(km).ele;
      cnt31(km)=t3(km).cnt;

      item_b(f1b)=t3(km).cnt;
      b=t3(km).ele
      [c1,c2,c3]=check_1(b)
      f1b=f1b+1;
      item_b(f1b)=c1
      f1b=f1b+1;
      item_b(f1b)=c2
      f1b=f1b+1;
      item_b(f1b)=c3
      f1b=f1b+1;
      sum_21b=0;
      for x=1:4

         sum_21b=sum_21b+item_b(km-1+x)
      end

      sum_21b(f_2b)=sum_21b;
      f_2b=f_2b+1;
   end

end
end
sx=size(t);
    str=strrep(t,'A','Burger')
  str=strrep(str,'B','Amrat')
  str=strrep(str,'C','Chilli Sauce')
  str=strrep(str,'D','French fries')
  str=strrep(str,'E','Shashlik')
str

set(handles.listbox13,'String',str)

set(handles.listbox2,'String',T)
[t12,t22,t32]=p1('D1:D25000');

if nargin<3
   t33.cnt=0;
   t33.ele=' ';
end

sz11=size(t12);
for kl1=1:sz11(2)
```

```
   t_1{kl1}=t12(kl1).ele;
   s21(kl1)=t12(kl1).ele;
   cnt21(kl1)=t12(kl1).cnt;

end
sz2=size(t22);
if sz2~=0

f3=1;f_4=1;
if sz2(2)>1
   for kj2=1:sz2(2)
      kj2
      t_1{kl1+kj2}=t22(kj2).ele;

      cnt22(kj)=t22(kj2).cnt

      item_1(f3)=t22(kj2).cnt;
      b=t22(kj2).ele
      [c1,c2,c3]=check_2(b)
      f3=f3+1;
      item_1(f3)=c1
      f3=f3+1;
      item_1(f3)=c2
      f3=f3+1;
      item_1(f3)=c3
      f3=f3+1;
sum_22=0;;

      for x=1:4

         sum_22=sum_22+item_1(kj2-1+x)
      end

      sum_22a(f_4)=sum_22
      f_4=f_4+1;
   end
else
   kj2=1;
   t_1{kl1+kj2}= t22.ele;
   cnt22(kj2)=t22(kj2).cnt;
   item_1(f3)=t22(kj2).cnt;
    b=t22.ele;
   [c1,c2,c3]=check_2(b)
   f3=f3+1;
   item_1(f3)=c1;
   f3=f3+1;
```

```
  item_1(f3)=c2;
  f3=f3+1;
  item_1(f3)=c3;

  for x=1:4
     sum_22=sum_22+item_1(x)
  end
   sum_22a(f_4)=sum_22
end


f3b=1;f_4b=1;
if (sz3(2))==1

  t_1{kl+kj+1}=t32.ele;
  item_b(f3b)=t32.cnt;
  b=t32.ele;
  [c1,c2,c3]=check_2(b)
  f3b=f3b+1;
  item_b(f3b)=c1;
  f3b=f3b+1;
  item_b(f3b)=c2;
  f3b=f3b+1;
  item_b(f3b)=c3;
  f3b=f3b+1;
  sum_22b=0;
  for x=1:4
     sum_22b=sum_22b+item_b(x)
  end

 sum_22b(f_4b)=sum_22b;

elseif sz3(2)>1
  f3b=1;f_4b=1;
  for km=1:sz3(2)
    t_1{kl+kj+1}=t32(km).ele;
  item_b(f3b)=t32(km).cnt;
  b=t32.ele;
  [c1,c2,c3]=check_2(b)
  f3b=f3b+1;
  item_b(f3b)=c1;
  f3b=f3b+1;
  item_b(f3b)=c2;
  f3b=f3b+1;
  item_b(f3b)=c3;
  f3b=f3b+1;
  sum_22b=0;
  for x=1:4
     sum_22b=sum_22b+item_b(x)
  end

 sum_22b(f_4b)=sum_22b;

     f_4b=f_4b+1;
  end

end
end

sx1=size(t);
  str1=strrep(t,'A','Burger')
  str1=strrep(str1,'B','Amrat')
  str1=strrep(str1,'C','Chilli Sauce')
  str1=strrep(str1,'D','French fries')
  str1=strrep(str1,'E','Shashlik')

str1

set(handles.listbox14,'String',str1)

[t13,t23,t33]=p1('F1:F25000');
t13
t23
t33
if nargin<3
   t33.cnt=0;
   t33.ele=' ';
end

sz13=size(t13);
for kl1=1:sz11(2)
   t_3{kl1}=t13(kl1).ele;
   s31(kl1)=t13(kl1).ele;
   cnt31(kl1)=t13(kl1).cnt;
end

sz2=size(t23)
if sz2~=0
   f_4=1;
   f3=1
   if sz2(2)>1
      for kj2=1:sz2(2)
```

```
        kj2
        t_3{kl1+kj2}=t23(kj2).ele;

        cnt23(kj)=t23(kj2).cnt

        item_2(f3)=t23(kj2).cnt;
        b=t23(kj2).ele
        [c1,c2,c3]=check_3(b)
        f3=f3+1;
        item_2(f3)=c1
        f3=f3+1;
        item_2(f3)=c2
        f3=f3+1;
        item_2(f3)=c3
        f3=f3+1;
        sum_23=0;

        for x=1:4

           sum_23=sum_23+item_2(kj2-1+x)
        end

        sum_23a(f_4)=sum_23
        f_4=f_4+1;
     end
  else
     kj2=1;
     t_3{kl1+kj2}= t23.ele;
     cnt23(kj)=t23(kj2).cnt;
     item_2(f3)=t23(kj2).cnt;
     b=t23.ele;
     [c1,c2,c3]=check_3(b)
     cnd_item_st=cnd_item_st+1;
     item_2(cnd_item_st)=c1;
     cnd_item_st=cnd_item_st+1;
     item_2(cnd_item_st)=c2;
     cnd_item_st=cnd_item_st+1;
     item_2(cnd_item_st)=c3;
     sum_23=0;
     for x=1:4
        sum_23=sum_23+item_2(x)
     end
     sum_23a(f_4)=sum_23
  end

  sz3=size(t33);
```

```
if (sz3(2))==1

   cnd_item_st_1=1;f_6b=1;
   if (sz3(2))==1

      t_3{kl+kj+1}=t33.ele;
      item_b(cnd_item_st_1)=t33.cnt;
      b=t33.ele;
      [c1,c2,c3]=check_3(b)
      cnd_item_st_1=cnd_item_st_1+1;
      item_b(cnd_item_st_1)=c1;
      cnd_item_st_1=cnd_item_st_1+1;
      item_b(cnd_item_st_1)=c2;
      cnd_item_st_1=cnd_item_st_1+1;
      item_b(cnd_item_st_1)=c3;
      cnd_item_st_1=cnd_item_st_1+1;
      sum_23b=0;
      for x=1:4
         sum_23b=sum_23b+item_b(x)
      end

      sum_23b(f_6b)=sum_23b;
   elseif sz3(2)>1
      cnd_item_st_1=1;f_6b=1;
      for km3=1:sz3(2)
         t_3{kl+kj+1}=t33(km3).ele;
         item_b(cnd_item_st_1)=t33(km3).cnt;
         b=t33.ele;
         [c1,c2,c3]=check_3(b)
         cnd_item_st_1=cnd_item_st_1+1;
         item_b(cnd_item_st_1)=c1;
         cnd_item_st_1=cnd_item_st_1+1;
         item_b(cnd_item_st_1)=c2;
         cnd_item_st_1=cnd_item_st_1+1;
         item_b(cnd_item_st_1)=c3;
         cnd_item_st_1=cnd_item_st_1+1;
         sum_23b=0;
         for x=1:4
            sum_23b=sum_23b+item_b(x)
         end

         sum_23b(f_6b)=sum_23b;
         f_6b=f_6b+1;
      end

   end
   end
end

sx2=size(t);


  str2=strrep(t,'A','Burger')
  str2=strrep(str2,'B','Amrat')
  str2=strrep(str2,'C','Chilli Sauce')
  str2=strrep(str2,'D','French fries')
  str2=strrep(str2,'E','Shashlik')
str2
   set(handles.listbox15,'String',str2)
[t14,t24,t34]=p1('H1:H25000');

if nargin<3
   t34.cnt=0;
   t34.ele=' ';
end

sz11=size(t14);
for kl1=1:sz11(2)
   t_4{kl1}=t14(kl1).ele;
   s41(kl1)=t14(kl1).ele;
   cnt41(kl1)=t14(kl1).cnt;
end

sz2=size(t24);
if sz2~=0
   cnd_item=1;

   if sz2(2)>1
      for kj2=1:sz(2)
         t_4{kl1+kj2}= t24(kj2).ele;
         cnt24(km3)=t24(km3).cnt;
         item_1(cnd_item)=t22(kj2).cnt;
         [c1,c2,c3]=check_4(t2(kj.ele))
         cnd_item=cnd_item+1;
         item_1(cnd_item)=c1;
         cnd_item=cnd_item+1;
         item_1(cnd_item)=c2;
         cnd_item=cnd_item+1;
         item_1(cnd_item)=c3;

         for x=1:4
            sum_44=sum_44+item_1(kj2-1+x)
```

```
          end
          sum_44a(f8)=sum_44;
          f8=f8+1;
       end
   else
       kj2=1;
       t_4{kl1+kj2}= t24.ele;
       cnt24(kj2)=t24.cnt;
       item_1(cnd_item)=t24(kj2).cnt;
       b=t24.ele;
       [c1,c2,c3]=check_4(b)
       cnd_item=cnd_item+1;
       item_1(cnd_item)=c1;
       cnd_item=cnd_item+1;
       item_1(cnd_item)=c2;
       cnd_item=cnd_item+1;
       item_1(cnd_item)=c3;

       for x=1:4
          sum_44=sum_44+item_1(x)
       end
       sum_44a(f8)=sum_44;
   end
   sz3=size(t34);
   if (sz3(2))==1
       cnd_item2=1;f_8b=1;
       km3=1;
       t_4{kl+kj+1}=t34(km3).ele;
       item_b(cnd_item2)=t34(km3).cnt;
       b=t33.ele;
       [c1,c2,c3]=check_4(b)
       cnd_item2=cnd_item2+1;
       item_b(cnd_item2)=c1;
       cnd_item2=cnd_item2+1;
       item_b(cnd_item2)=c2;
       cnd_item2=cnd_item2+1;
       item_b(cnd_item2)=c3;
       cnd_item2=cnd_item2+1;
       sum_44b=0;
       for x=1:4
          sum_44b=sum_44b+item_b(x)
       end

       sum_44b(f_8b)=sum_44b;
   elseif sz3(2)>1
```

```
          cnd_item2=1;f_8b=1;
      for km3=1:sz3(2)
         t_4{kl+kj+1}=t34(km3).ele;
         item_b(cnd_item2)=t34(km3).cnt;
         b=t33.ele;
         [c1,c2,c3]=check_4(b)
         cnd_item2=cnd_item2+1;
         item_b(cnd_item2)=c1;
         cnd_item2=cnd_item2+1;
         item_b(cnd_item2)=c2;
         cnd_item2=cnd_item2+1;
         item_b(cnd_item2)=c3;
         cnd_item2=cnd_item2+1;
         sum_44b=0;
         for x=1:4
            sum_44b=sum_44b+item_b(x)
         end

         sum_44b(f_8b)=sum_44b;
         f_8b=f_8b+1;
      end

   end
end
sx3=size(t);
  str3=strrep(t,'A','Burger')
  str3=strrep(str3,'B','Amrat')
  str3=strrep(str3,'C','Chilli Sauce')
  str3=strrep(str3,'D','French fries')
  str3=strrep(str3,'E','Shashlik')
str3
set(handles.listbox16,'String',str3)

sum_a=0;
sum_b=0;
sum_c=0;
sum_d=0;
sum_e=0;sum_2=0;sum_3=0;

sa=size(s11)
sb=size(cnt1)
a1=0;b1=0;c1=0;d1=0;e1=0;
a2=0;b2=0;c2=0;d2=0;e2=0;
a3=0;b3=0;c3=0;d3=0;e3=0;
a4=0;b4=0;c4=0;d4=0;e4=0;
for w=1:sa(2)
```

```
   if s11(w)=='A'
      cha=1;
      a1=cnt1(w)
   elseif s11(w)=='B'
      chb=1;
      b1=cnt1(w)
   elseif s11(w)=='C'

      c1=cnt1(w)
   elseif s11(w)=='D'
      chd=1;
      d1=cnt1(w)
   elseif s11(w)=='E'
      che=1;
      e1=cnt1(w)
   end
end


sa=size(s21)
sb=size(cnt21)

for w=1:sa(2)
   if s21(w)=='A'

      a2=cnt21(w)
   elseif s21(w)=='B'

      b2=cnt21(w)
   elseif s21(w)=='C'

      c2=cnt1(w)
   elseif s21(w)=='D'

      d2=cnt21(w)
   elseif s21(w)=='E'

      e2=cnt21(w)
   end
end
sa=size(s31)
sb=size(cnt31)

for w=1:sa(2)
   if s31(w)=='A'
      cha=1;
      a3=cnt31(w)
   elseif s31(w)=='B'
      chb=1;
      b3=cnt31(w)
   elseif s31(w)=='C'

      c3=cnt31(w)
   elseif s31(w)=='D'
      chd=1;
      d3=cnt31(w)
   elseif s31(w)=='E'
      che=1;
      e3=cnt31(w)
   end
end
sa=size(s41)
sb=size(cnt41)

for w=1:sa(2)
   if s41(w)=='A'
      cha=1;
      a4=cnt41(w)
   elseif s41(w)=='B'
      chb=1;
      b4=cnt41(w)
   elseif s41(w)=='C'

      c4=cnt41(w)
   elseif s41(w)=='D'
      chd=1;
      d4=cnt41(w)
   elseif s41(w)=='E'
      che=1;
      e4=cnt41(w)
   end
end
```

5.3.5 Finding Globally Large Item set

```
sum_a= a1+a2+a3+a4;
sum_b= b1+b2+b3+b4;
sum_c= c1+c2+c3+c4;
sum_d= d1+d2+d3+d4;
sum_e= e1+e2+e3+e4;
```

```
gla=1;
if sum_a>13500
   var='A'
   glar{gla}=var;
   gla=gla+1;

   msgbox('A is globally large 1-itemset')
end
if sum_b>13500
   var='B'
   glar{gla}=var;
   gla=gla+1;
   msgbox('B is globally large 1-itemset')
end
if sum_c>13500
   var='C'
   glar{gla}=var;

   gla=gla+1;

   msgbox('C is globally large 1-itemset')
end
if sum_d>13500
   var='D'
   glar{gla}=var;
   gla=gla+1;

   msgbox('D is globally large 1-itemset')
end
if sum_e>13500
   var='E'
   glar{gla}=var;

   gla=gla+1;

   msgbox('E is globally large 1-itemset')
end
sum_21a
sz21= size(sum_21a)

if(sz21(2)>1)
   for u=1:sz21(2)
      if sum_21a(u)>55500
         var=t2(u).ele;
         glar{gla}=var
```

```
            gla=gla+1;
            msgbox(var)
        end
    end
else

    if((sum_21a)>55500)
        var=t2.ele;
        glar{gla}=var
        gla=gla+1
        msgbox(var)

    end
end
sz21= size(sum_22a)

if(sz21(2)>1)
    for u=1:sz21(2)
        if sum_22a(u)>55500
            var=t22(u).ele;
            glar{gla}=var
            gla=gla+1;
            msgbox(var)
        end
    end
else

    if((sum_22a)>55500)
        var=t22.ele;
        glar{gla}=var
        gla=gla+1
        msgbox(var)

    end
end
sz21= size(sum_23a)
if(sz21(2)>1)
    for u=1:sz21(2)
        if sum_23a(u)>55500
            var=t23(u).ele;
            glar{gla}=var
            gla=gla+1
            msgbox(var)
        end
    end
else
```

```
   if((sum_23a)>55500)
      var=t23.ele;
      glar{gla}=var
      gla=gla+1
       msgbox(var)
   end
end
sz21= size(sum_44a)
if(sz21(2)>1)
   for u=1:sz21(2)
      if sum_44a(u)>55500
         var=t24(u).ele;
         glar{gla}=var
         gla=gla+1
          msgbox(var)
      end
   end
else

   if((sum_44a)>55500)
      var=t24.ele;
      glar{gla}=var
      gla=gla+1
       msgbox(var)

   end
end

   sz21=size(sum_21b);
if(sz21(2)>1)
   for u=1:sz21(2)
      if sum_21b(u)>55500
         var=t3(u).ele;
         glar{gla}=var
         gla=gla+1;
         msgbox(var)
      end
   end
else

   if((sum_21b)>55500)
      var=t3.ele;
      glar{gla}=var
      gla=gla+1
      msgbox(var)
```

```
  end
end
sz21= size(sum_22b)

if(sz21(2)>1)
  for u=1:sz21(2)
    if sum_22b(u)>55500
      var=t23(u).ele;
      glar{gla}=var
      gla=gla+1;
      msgbox(var)
    end
  end
else

  if((sum_22b)>55500)
    var=t23.ele;
    glar{gla}=var
    gla=gla+1
    msgbox(var)

  end
end
sz21= size(sum_23b)
if(sz21(2)>1)
  for u=1:sz21(2)
    if sum_23b(u)>55500
      var=t32(u).ele;
      glar{gla}=var
      gla=gla+1
      msgbox(var)
    end
  end
else

  if((sum_23b)>55500)
    var=t32.ele;
    glar{gla}=var
    gla=gla+1
     msgbox(var)
  end
end
sz21= size(sum_44b)
if(sz21(2)>1)
  for u=1:sz21(2)
    if sum_44b(u)>55500
```

```
          var=t34(u).ele;
          glar{gla}=var
          gla=gla+1
           msgbox(var)
      end
   end
else

   if((sum_44b)>55500)
      var=t34.ele;
      glar{gla}=var
      gla=gla+1
       msgbox(var)

   end
end
glar
ah='';
ha='';
fg=1;
sx=size(glar);
for t=1:sx(2)-1

  ha=glar{t}
  ah=glar{t+1}
   if strcmp(ha,ah)
continue;
   end
   finl{fg}=ha;
   fg=fg+1;
end
finl{fg}=ah;

sx4=size(t);

  str4=strrep(t,'A','Burger')
  str4=strrep(str4,'B','Amrat')
  str4=strrep(str4,'C','Chilli Sauce')
  str4=strrep(str4,'D','French fries')
  str4=strrep(str4,'E','Shashlik')
str4
set(handles.listbox8,'String',str4)
```

Chapter 6

Results

6. Results

To illustrate the performance of our devised algorithm, the CMA algorithm, we give here an example, in which at first PARTITION Algorithm is applied on the central database. It divides the database into three distinct, non-overlapping, and random partitions, and then calculates the locally large k-itemsets. And finally computes globally large itemsets of size 1,2,…,k. In the second part of the example, CMA Algorithm is applied on the same database, which also divides the central database into three distinct, non-overlapping, and random partitions, and then calculates locally large itemsets by bringing each partition one by one into memory. Globally large itemsets are calculated at the end. Instead of counting the itemsets in the transactions of the actual database, CMA Algorithm uses the formula of $s * D_i$ (as is used by Cheung et al in DMA Algorithm presented in [13]) for counting support within each partition, where s is the support of the itemset, set by the user, like 30%, 60%, 75% etc.(50% in this example), and D_i is the size of each partition database(3 in this example). For global support count, it uses the formula, $s * D$, where D is the size of the entire database (12 in this example).

The dataset for this example is presented in table 6-1.

TID	Itemset
T1	A,B,C
T2	A,B,D
T3	A,D,E
T4	A,B,D
T5	B,C,D
T6	A,B,E
T7	A,D,E
T8	B,C,E
T9	A,B,C
T10	A,C,D
T11	A,D,E

T12	C,D,E

Table 6-1: the dataset

6.1 Working of PARTITION Algorithm

First, PARTITION Algorithm is applied to the database, which works on random partitions, so the original database, given in table 6-11, is divided into three distinct, non-overlapping, and random partitions, which are presented in table 6-2.

Partition 1		Partition 2		Partition 3	
TID	Itemset	TID	Itemset	TID	Itemset
T2	A,B,D	T1	A,B,C	T3	A,D,E
T6	A,B,E	T4	A,B,D	T5	B,C,D
T9	A,B,C	T7	A,D,E	T8	B,C,E
T12	C,D,E	T11	A,D,E	T10	A,C,D

Table 6-2: Three Partition created after dividing the database.

6.1.1 Processing on Partition 1

It then generates the candidate 1-itemset, and then counts its support. The support taken in this example is 50%. So the large itemsets are created and stored along with their TIDs, as shown in table 6-3.

TID	1 Itemset
T2	A
T2	B
T2	D
T6	A

T6	B
T6	E
T9	A
T9	B
T9	C
T12	C
T12	D
T12	E

Table 6-3: Candidate 1-itemsets of partition 1.

Then candidate 2-itemset is generated by multiplying large 1-itemset with itself, like $l_1 * l_1$.The candidate 2-itemsets of partition 1 are {AB, AC, AD, AE, BC, BD, BE, CD, CE, DE}. Then the TIDs of the transactions containing these 2-itemsets are stored against each 2-itemsets. The number of TIDs against each itemset shows its occurrence in the partition. For example, in partition 1, AB is present in T2, T6, and T9, so it means it has occurrence of 3 out of 4 transactions, i.e., ¾=0.75, which means 75% transactions in partition 1 contains AB, which is greater than 50%, so AB is a locally large 2-itemset in partition 1. The supports calculated for each candidate 2-itemset is given below:

AB={2,6,9}, so support is AB=3/4= 0.75

AC={9}, so support is AC=1/4=0.25

AD= {2}, so support is AD=1/4= 0.25

AE= {2}, so support is AE=1/4=0.25

BC= {9}, so support is BC=1/4=0.25

BD= {2}, so support is BD=1/4=0.25

BE= {6}, so support is BE=1/4=0.25

CD= {12}, so support is CD=1/4=0.25

CE= {0}, so support is CE=0

DE= {12}, so support is DE=1/4=0.25

So the large 2-itemsets in Partition 2 are {AB, AD, AE, DE}. From this we will have candidate 3-itemset of {ABD, ADE, ABE, BDE}. But as only 1 large 2-itemset is present in partition 1, so no candidate 3-itemset is possible. Same is repeated with partition 2 and partition 3, which are illustrated below.

6.1.2 Processing on Partition 2

The large itemsets in partition 2 are calculated the same way as are calculated in the partition 1.

TID	Itemset
T1	A
T1	B
T1	C
T4	A
T4	B
T4	D
T7	A
T7	D
T7	E
T11	A
T11	D
T11	E

Table 6-4: Candidate 1-itemset of partition 2.

The candidate 2-itemset is, $l_1 * l_1$ ={AB,AC, AD,AE ,BC,BD, BE,CD, CE, DE}

AB={T1, T4}, so the support is= AB=2/4= 0.5

AC={T1}, so the support is=1/4=0.25

AD={T4,T7,T11}, so the support is= 3/4= 0.75

AE={T7,T11}, so the support is= 2/4=0.5

BC={T1}, so the support is=1/4=0.25

BD={T4}, so the support is=1/4=0.25

BE={0}, so the support is=0

CD={0}, so the support is=0

CE={0}, so the support is=0

DE={T7,T11}, so the support is= 2/4=0.5

Candidate 3-itemsets are:

ABD={T4}, so the support is=1/4=0.25

ADE={T7,T11}, so the support is=2/4=0.5

ABE={0}, so the support is=0

BDE={0}, so the support is=0

So in the end we have only ADE as large 3-itemset in Partition 2.

6.1.3 Processing on Partition 3

Similarly, large itemsets of partition 3 are calculated which are given below.

TID	Itemset
T3	A
T3	D
T3	E
T5	B
T5	C
T5	D
T8	B
T8	C
T8	E
T10	A

T10	C
T10	D

Table 6-5: Candidate 1-itemset of partition 2.

Supports of the 1-itemsets are

A=4,B=2,C= 1,D=3,E=2

Candidate 2-itemset= {AB, AC, AD, AE, BC, BD, BE, CD,CE, DE }

AB={0}, so the support is=0

AC={T10}, so the support is=1/4=0.25

AD={T3,T10}, so the support is=2/4= 0.5

AE={T3}, so the support is=1/4=0.25

BC={T5,T8}, so the support is=2/4=0.5

BD={T5}, so the support is=1/4=0.25

BE={T8}, so the support is=1/4=0.25

CD={T5,T10}, so the support is=2/4=0.5

CE={T8}, so the support is=1/4=0.25

DE={T3}, so the support is=1/4=0.25

As we have only one large 2-itemset so no candidate 3-itemset is possible.

6.1.4 Globally Large Itemset Calculation

Now, we combine the large itemsets of all the partitions and merge them in a single global set.

TID	Itemset
1	A
1	B
1	C

2	A
2	B
2	D
3	A
3	D
3	E
4	A
4	B
4	D
5	B
5	C
5	D
6	A
6	B
6	E
7	A
7	D
7	E
8	B
8	C
8	E
9	A
9	B
9	C
10	A
10	C
10	D
11	A
11	D

11	E
12	C
12	D
12	E

Table 6-6: Globally large 1-itemsets.

Supports of the 1-itemsets are

A=9,B=7,C= 6,D=7,E=6

TID	2-itemset
T1,T2,T4	AB
T2,T3,T4	AD
T2,T4	BD
T5,T8	BC
T6,T7	AE
T6,T8	BE
T9,T10	AC
T10,T11	AD
T10,T12	CD
T11,T12	DE

Table 6-7: Globally large 2-itemsets.

Supports of the large 2-itemsets are:

AB= 7,AD= 2,BD= 1,BC= 1,AE= 1,BE= 1,AC= 1,AD= 1,CD= 1,DE= 1.

TID	Itemset
T2,T4	ABD

Table 6-8: Globally large 3-itemsets.

And the support of the 3-itemset is 1, so no globally large 3-itemset is present in the database taken.

6.2 Working of CMA Algorithm

Now, we apply CMA Algorithm on the same database, which also works by randomizing the database, and divides the database into three distinct, non-overlapping, and random partitions, which are presented in table 6-9.

Partition 1		Partition 2		Partition 3	
TID	Itemset	TID	Itemset	TID	Itemset
T2	A,B,D	T4	A,B,D	T3	A,D,E
T9	A,B,C	T11	A,D,E	T5	B,C,D
T6	A,B,E	T1	A,B,C	T8	B,C,E
T12	C,D,E	T7	A,D,E	T10	A,C,D

Table 6-9: 3 Partitions created by CMA

We consider in this example, minimum support count, s=50%. So the candidate 1-itemsets are presented in table 6-10.

Partition1		Partition2		Partition3	
Candidate 1-Itemset	Local Count (X.support >=s*D_i)	Candidate 1-Itemset	Local Count (X.support >=s*D_i)	Candidate 1-Itemset	Local Count (X.support >=s*D_i)
A	3>=0.5*4	A	4>=0.5*4	A	2>=0.5*4
B	3>=0.5*4	B	2>=0.5*4	B	2>=0.5*4
C	2>=0.5*4	C	1>=0.5*4	C	3>=0.5*4
D	2>=0.5*4	D	3>=0.5*4	D	3>=0.5*4
E	2>=0.5*4	E	2>=0.5*4	E	2>=0.5*4

Table 6-10: Candidate 1-itemsets.

In Partition 1 the large 1-itemsets are {A, B, C, D, E}. In Partition 2 the large 1-itemsets are {A, B, D, E}. While in Partition 3 the large 1-itemsets are {A, B, C, D, E}.
The candidate 2-itemsets in Partition 1 are {AB, AC, AD, AE, BC, BD, BE, CD,CE, DE}, in Partition 2 are {AB, AD, AE, BD, BE, DE} and in Partition 3, are {AB, AC, AD, AE, BC, BD, BE,CD,CE, DE}. The large 2-itemsets of three partitions are presented in table 6-11.

Partition 1		Partition 2		Partition 3	
CL	X.sup	CL	X.sup	CL	X.sup
AB	3	AB	2	AB	0
AC	1	AD	3	AC	1
AD	1	AE	2	AD	2
AE	1	BD	1	AE	1
BC	1	BE	0	BC	2
BD	1	DE	2	BD	1
BE	1			BE	1

CD	1			CD	2
CE	1			CE	1
DE	1			DE	1

Table 6-11: Large 2-itemsets and their supports.

As shown in the table 6-11, large 2-itemset in Partition 1 is {AB}, in Partition 2 are {AB, AD, AE, DE} and in Partition 3 are {AD, BC, CD}. The table also shows that no candidate 3-itemset in Partition 1 is possible. Whereas in Partition 2 the candidate 3-itemset are {ABD, ADE, ABE} and in Partition 3 {ABC, ABD, ACD, BCD}.

The support counts of candidate 3-itemsets of three partitions are given below in the table 6-12:

Partition 2		Partition [illegible]	
C_3	X.Sup	C[illegible]	X.Sup
ABD	1	ABC	0
ADE	2	ABD	0
ABE	0	ACD	1
		BCD	1

Table 6-12: Large 3-itemsets and their supports.

So the only large 3-itemset found is {ADE}.

In order to check whether the locally large itemsets are actually large in database or not, we check each locally large itemset globally with the help of formula X.Sup>= s*D, where D is the size of the entire database, which in this case is 12.

Locally large candidate set	$X.sup^1$	$X.sup^2$	$X.sup^3$	X.Sup >= s*D
A	3	4	2	9>=6
B	3	2	2	7>=6
C	2	1	3	6>=6
D	2	3	3	8>=6
E	2	2	2	6>=6
AB	3	2	0	5<6
AC	1	0	1	2<6
AD	1	3	2	6>=6
AE	1	2	1	4<6
BC	1	0	2	3<6
BD	1	1	1	3<6
BE	1	0	1	2<6
CD	1	0	2	3<6
CE	1	0	1	2<6
DE	1	2	1	4<6

Table 6-13: Global Support count of candidate itemsets.

From the table 6-13, the only globally large 2-itemset is {AD}, and sine we have a single globally large 2-itemset so no candidate 3-itemset is possible and the process stops here.

Experiments show that the database scanning is reduced to half with CMA algorithm, and the early pruning also decreased the chances for the false candidates.

Chapter 7

Conclusion & Future Enhancements

7. Conclusion & Future Enhancements

We have implemented our C-ARMing technique's simulation software on the Pentium Machine with limited memory and slow processing speed. But there is always a chance for improvement in every software. So we discuss below the conclusion and future enhancements to our C-ARMing model. Better results can be obtained if the server machine is used with Xeon processor or Dual processing power system. It would further reduce the time complexity of the algorithm.

7.1 Conclusion

Previously, the database was used to be scanned again and again for the generation of candidate itemsets, and then for the actually large itemsets, which in the case of huge database means a lot. Which resulted in higher CPU costs and large number of I/Os as well. The technique we adopted not only reduces the CPU cost, but also decreases the database scanning with the percentage of 50%, which is an achievement, with special reference to the Data mining queries. Our technique, the C-ARMing, also results in the efficient processing speed, and hence Association Rules could easily be generated from them. The database scans taken by the different algorithms are shown in the fig. 7-1.

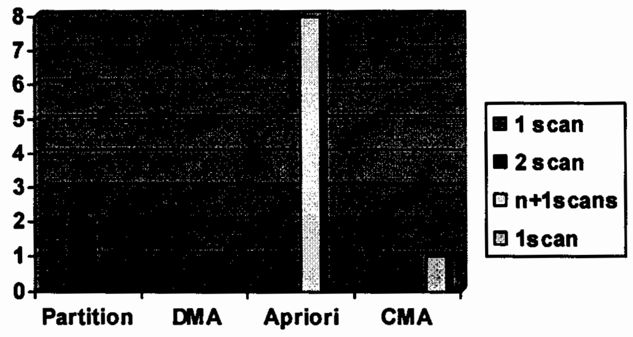


Figure 7-1 Database scans taken by different algorithms

7.2 Future Enhancements

During our project, we have implemented our C-ARMing technique with the synthetic database, created exclusively for this study. The size of the database was 100,000 tuples. In future we will compare PARTITION and our devised CMA Algorithm by implementing the PARTITION Algorithm as well. CMA Algorithm could also be tested with transactional databases, and on different systems, other than Pentium machines, to conduct a comparative study and test the functionality of the algorithm. Development of same project in some other tool such as VC++ or entirely in Dot net frame work can improve the efficiency by reducing its time complexity. In future we are also planning to test the same algorithm with the database size exceeding 1million transactions.

Appendix A

Reffernces

References

[1]. "**Decision Support Systems and intelligent Systems**". Fifth edition. By Efraim Turban, Jaye. Aronson. Pg 34.

[2]. "**Data Mining: Concepts and Techniques**". By Jiawei Han, Micheline Kamber. Pg 15, 226, 227.

[3]. "**Data Mining: Introductory and Advanced Topics**". By Margaret H. Dunham. Pg 5, 6, 38, 166.

[4]. "**Database Systems: A Practical Approach to Design, Implementation, and Management**". By Thomas Connolly, Carolyn Begg. Pg 1047.

[5]. "**Data Warehousing**". By BRB Publications. Pg 79.

[6]. "**Data Mining: Typical Data Mining Process for Predictive Modelling**". By BPB Publication. Pg 3, 5.

[7]. Rakesh Agarawal, Tomasz, Imielinski, and Arun Swami. "**Mining Association Rules Between Sets of Item In Large Databases**". Proceedings of the ACM International Conference on Management of Data,1993.

[8]. Maurice Houtsma, Arun Swami."**Set-Oriented Mining for Association Rules in Relational Databases**". Proceedings of the IEEE International Conference on Data Engineering, 1995.

[9]. Rakesh Agarawal, Ramakrishnan Srikant. **"Fast Algorithms for Mining Association Rules"**. Proceedings of the International Very large Databases Conference, 1994.

[10]. Jong Soo Park, Ming-Syan Chen and Philip S. Yu. **"An Effective Hash-Based algorithm for Mining Association Rules".** Proceedings of the ACM International Conference on Management of Data, 1995.

[11]. Ashoke Savasere, Edward Omiecinski and Shamkant Navathe. **"An Efficient Algorithm for Mining Association Rules in Large Databases".** Proceedings of the 21s VLDB Conference, Zurich, Switzerland,1995.

[12]. David W. Cheung, Jiawei Han, Vincent T. Ng, Ada W Fu, and Yongjian Fu. **"A Fast Distributed Algorithm for Mining Association Rules".**

[13]. David W. Cheung, Vincent T. Ng, Ada W. Fu, and Yongjian Fu. **"Efficient Mining of Association Rules in Distributed Databases"**. IEEE Transactions on Knowledge and Data Engineering, 1996.

[14]. Jun-Lin Lin, Margaret H. Dunham. **"Mining Association Rules: Anti-Skew Algorithms"**. Department of Computer Science and Engineering, Southern Methodist University, Dallas, Texas.

[15]. http://en.wikipedia.org/wiki/Microsoft_.NET_Framework.

[16]. http://msdn.microsoft.com/netframework/technologyinfo/overview/default.aspx.

Appendix B

Publication

Journal Name
ISSN
Copy Right

Association Rule Mining in Centralized Databases

Saleha Jamshaid, Zakia Jalil, Malik Sikander Hayat Khiyal and Muhammad Imran Saeed
Faculty of Applied Sciences, International Islamic University, Islamabad, Pakistan

Abstract: Mining of Association Rules between the items of a huge centralized Database is very interesting and important research area. Its importance becomes more significant in case of sales transaction. There are a number of algorithms working on this specialized research area. The algorithm presented in this study, (Centralized Mining of Association-Rules), CMA is more efficient than the previous existing algorithms, as it not only reduces the overhead of frequent disk I/Os and the CPU cost, but it also reduces the database scan to the half. The CMA algorithm, presented in this study, basically takes the best features of two state-of-the-art algorithms in the area, i.e., the technique of PARTITION algorithm of centralized database area is taken for partitioning the huge Database and then, the DMA algorithm of distributed database environment is applied on each partition. The large itemsets to be found at the end of operation at each partition are to be merged together and then the actual set of large itemsets is finally created.

Key words: Data warehousing, data mining, association rules

INTRODUCTION

With the growing competition in retail industry, it has been observed that along with other factors, proper placement of different items together at shelves is also a major factor to increase the sales of a store. Because some items have special sort of relationships among each other, which can never be targeted out simply with the help of Entity-Relationship Diagram (ERD) or some mathematical formulae, as these relationships are not the casual relationships. To represent these relationships between data items, association rules are used. For example, if a person buys a computer, he is likely to be buying some software CDs as well. If we offer our customers free Operating System installation and place some antivirus CDs on the prominent and nearest shelves to the counter, the customer couldn't stop himself from noticing the importance of the antivirus software for his system's protection as well. And if we offer this antivirus to him with some special concession package, he will definitely be induced to buy those as well. It means that not only we are running our computer business with distinction from our competitors, but also establishing a side business of CD shop. This is what we call Association between two items; the Association between computer and CDs and the association between computers and the virus-guard software. And all such business decisions are supported well with the help of finding out these Associations. And to find these Associations, we use Association Rules.

Association rule Mining is an important research area in Databases. It usually involves huge amount of data glut, out of which, useful information is to be extracted in the form of association rules. The task of Mining association rules is to analyze the entire Database and find out the association rules for different sets of items. It requires multiple Database scans for the rule generation.

The algorithm presented in this study CMA, generates large itemsets by only taking a single scan of the database after the creation of the candidate sets. It divides the centralized database into a number of partitions, which are not taken sequentially from the database, but the partitions are created by taking random tuples from the database and then collecting them together in different partitions. Each partition is loaded into the memory one by one and then the large itemsets are created from each partition. At the end, the large itemsets of all the partitions are collected and then examined that whether these are actually large itemsets in the entire database or not. Within each partition, the large itemsets are created by taking a single database scan after creation of the candidate sets, thus minimizing the disk I/Os to the half, as that of the PARTITION Algorithm presented by Savarse *et al.* (1995). The database used for the experiments of CMA Algorithm is synthetic data, created exclusively for this study.

BASIC CONCEPT

In the present study we give the basic description of this problem area, which is mostly based on the description given by Agarawal *et al.* (1993). The problem

Corresponding Author: Malik Sikander Hayat Khiyal, Faculty of Applied Sciences, International Islamic University, Islamabad, Pakistan

of association rule mining is tackled in many research papers. Many algorithms are devised to solve this problem. From the literature, it has been observed that problem can be divided into two sub problems:

- Find all frequent/large itemsets
- Generate strong association rules from these frequent/large itemsets (Han, 2001).

Here, by large itemsets, we mean the items that frequently occur together in different transactions. We define a threshold number of occurrences for a given itemset, if the number of occurrences is greater than that threshold value, then the item is a larger one, otherwise a small one. It is done by scanning the entire Database and calculating the number of occurrences of each itemset. This individual number of occurrences of an itemset is called the support of that itemset. And the threshold value of support, specified by the user is called the minimum_support. Like support another feature used is called confidence, which actually determines the strength of the rule. To understand it lets take an example. In order to measure the confidence of two items, say burger and the drink, the number of times the burger appears in the transaction, so is the drink. This is called the confidence between two items. It can have any value, 60 and 70% etc. association rules are the implication of the form $X \rightarrow Y$. Here X is called the *antecedent* and Y is called *consequent* of the rule. The rule $X \rightarrow Y$ has confidence c in the transaction set D if c is the percentage of the transaction in D containing X that also contain Y (Han, 2001).

The literature surveyed and the study with different algorithms, has shown that in order to have an efficient algorithm for the association rule mining, it is required to emphasize a lot over the generation of frequent/large itemsets, the first sub problem. The quicker the algorithm at step1, the efficient will be the association rule Mining process. So othe present study is confined with the step1 only.

The functionality, followed so far for generation of large itemsets is as follows:

Let $I = \{i_1, i_2, \ldots, i_m\}$ be the set of items, DB be the transactional database, T the set of items such that $T \subset I$. So the rule $X \rightarrow Y$ means $X \subset I$, $Y \subset I$ and $X \cap Y = \varphi$. As mentioned before, it is the step1 which determines the efficiency of the algorithm as the database is scanned for many times during this step. Firstly, the items are taken from the database, then their support is counted from the database, then 2-itemsets are created and up to k-itemset creation (k-itemset is an itemset of size k), the database is scanned again and again, i.e., the same is done for the every iteration. As it is obvious that the mining of association rules is not done on a small database, rather it could be on a huge database, or a data warehouse, or some distributed database with multiple nodes. So this multiple scan of database physically means a lot. Excessive study has been done on this area.

An earlier study in this area is done by Agrawal *et al.* (1993), in which AIS algorithm is presented. This algorithm scans the database to create the candidate sets of frequently occurring itemsets. The second database scan counts the support of these candidate sets. The candidates generated from a transaction are added to the set of candidate itemsets for the pass, or the counts of the corresponding entries are increased if they were created by an earlier transaction. The problem with this algorithm is that, it is confined to only the single consequent rule generation and creates larger candidate set.

An algorithm SETM is presented by Houtsma and Swami (1995). It uses SQL for computing large itemsets. Candidate itemsets and the corresponding TIDs are saved together and are sorted and aggregated at the end of pass to determine the support count. The small candidate itemsets are pruned out. Then the set of the candidate set is again sorted on the basis of TIDs for next pass. SETM also uses the single consequent rule generation technique. It also generates large candidate itemsets. For each candidate itemset, the candidate generated has many entries as the number of transactions in which the candidates are present. Also to count the support for candidate itemsets, the candidate set is in wrong order and needs to be sorted on TIDs. After counting and pruning out small candidate itemstes, the resulting set of large items needs another sort on TID, to be used in the subsequent passes.

The pioneer work is presented by Agrawal and Srikant (1993) in which notorious Apriori algorithm is presented. All other subsequent algorithms are the adaptation of Apriori to some extents. This algorithm counts item occurrences from the database to determine large 1-itemset in first pass. In the next pass, the algorithm first generates candidate itemsets, using apriori-gen () and then checks the support count. It stores the candidate itemsets in a special structure, the Hash Tree. The candidate itemsets generated by this algorithm is smaller than that created by SETM and AIS. Also, unlike the AIS and SETM, it generates multiple consequent association rules. Another version of Apriori, AprioriTid, is also presented in this study, which does not use the database for support counting after the first scan, instead it uses the pair of itemset and it's TIDs for this purpose. It works efficiently in later passes. AprioriHybrid is another variant of the same algorithms which uses Apriori in earlier iterations and AprioriTid in later ones, to enjoy more benefits from both the algorithms. In Apriori, the problem is that the database is scanned entirely for each pass, so it means for 10 iterations, 10 scans of the entire database. For AprioriTid algorithm, its performance is not better than Apriori's in initial stages, as there are too many candidate k-itemsets to be tracked during the early stages of the process. The challenge in AprioriHybrid is to determine the switch over point between the two algorithms.

Partition algorithm, presented by Savasere *et al.* (1995), takes two database scans. Firstly, for generation of potentially large itemsets and store it as a set, which is the superset of all the large itemsets. Secondly, to measure the support of these itemsets and storing them in their respective counters created.

The working of this algorithm is divided in two phases. In phase I, the database is logically divided into non-overlapping partitions, which are considered one by one and large itemsets for each partition are generated and at the end all itemsets are merged to form the set of all potentially large itemsets. The phase II generates the actual support of these itemsets, to identify large itemsets. The database is read once in phase I and once in phase II. The small itemsets are pruned out. For each itemset, is associated its sorted TID list. To count the support of all itemsets in a partition, this algorithm divides the cardinality of TID list by the total number of transactions in that partition. Initially, the tidlists for 1-itemsets are generated directly by reading the partition. The TID list for a candidate k-itemset, is generated by joining the TID lists of the two (k-1)-itemsets that were used to generate the candidate k-itemset. The main problem with PARTITION algorithm is to find out the accurate number of partitions for the given memory.

The mining of association rules in distributed environment is discussed by Cheung *et al.* (1996). An algorithm Distributed Mining of Association rules, (DMA) is presented, which is an adaptation of Apriori in parallel systems. In DMA, to generate candidate itemsets, apriori-gen() is not applied directly, rather it is applied in such a way that it minimizes the candidate set to a greater extend than in the case of direct application of apriori-gen(). It uses polling site technique to determine heavy itemsets, after local pruning. In the whole procedure, the database partition at each site is scanned only once for calculating the support count and then those counts are stored in Hash Tree. This Hash Tree contains the support counts of both the heavy itemsets at that site and heavy itemsets at some other site. The later are stored in order to entertain the polling requests made by the remote sites so one database scan is done to compute the support counts of itemsets and are stored in the Hash Tree and retrieved from that Hash Tree when required. This optimizes the database scanning required for count exchange.

The literature proves that the generation of large itemsets is the main problem in generating the association rules in large databases. It involves many problems like the candidate sets created for generation of large itemsets are very large; the database is supposed to be scanned again and again in order to create candidate sets and to generate their support counts. Also the pruning of the itemsets from the candidate sets, that are not large, is also a problem.

MATERIALS AND METHODS

The algorithm DMA requires only one database scan at each site in order to calculate the heavy itemsets. So far, the best work done on centralized databases, the Partition algorithm, achieves the same with two database scans. So to apply DMA algorithm on each partition (obviously excluding the concept of polling site, which is specific for distributed databases), the large itemsets could be found with a single database scan.

So the major task is to divide the database into a number of partitions. These partitions are logical and non-overlapping, i.e., the transactions found in one partition should not also be present in other partition. Similarly, the partitions are not to be taken sequentially from the centralized database, these are to be created by taking the transactions randomly from the database to avoid the outliers (i.e., itemsets caused due to data skewness). For example, the October earthquake of Pakistan caused a drastic demand for tents. Per month demands of the tents for the affected areas were about 200,000 and the situation remained so throughout the winters. By using the sequential approach for partitioning the centralized database, if we take the data of a quarter of the year as one partition, then according to the 4th partition of 2005's data, tent will emerge as a large itemset, which in fact is not. As a contrary, the annual sale of tents in Pakistan, before the October earthquake, is 25 tents per year. This is the outlier, caused by the data skewness of the abnormal circumstances. And if we take the transactions randomly from the database to build different partitions, the October-Dec'05's tragic condition's transactions will be distributed throughout the database, so no problem of outliers will emerge. This task is achieved with the specially devised function, the Random (), to randomly take transactions from the database and group them together in partitions. Each partition is supposed to contain D_i number of transactions, after which the counter is initialized and then selected transactions are to be stored in second partition and so on. Then each partition is brought into memory one by one. For each partition, candidate 1-itemset is created, from which, large 1-itemset is generated. Then candidate 2-itemset is generated using apriori-gen () and from that candidate set, large 2-itemset is created and so on up to k-itemsets. Same is done with each partition.

Dataset: The functionality of the CMA algorithm has been tested on extensive experiments. The dataset used in these experiments is synthetic data, created exclusively for these experiments. Each tupple of the database is of the form <TID> <A, B, C, D,...>, where TID is the transaction identifier, which is the primary key to uniquely identify each tupple and the literals represent the retail store's items bought together in each transaction. For example, take a transaction <T1004> <A, C, D> for example, where A represents butter, C represents bread and D represents milk, so we can say that transaction number 1004 represents the purchase of breakfast items by a customer during his visit to the store. The size of the database used is 100, 000 transactions.

Random function: To calculate large itemsets, CMA Algorithm partitions the database randomly, i.e., for each partition, transactions are picked from the database randomly. It is also possible to avoid this overhead of randomizing the partitions by picking up sequential partitions, i.e., if the size of the database is 20 MB, then take first five MB data in partition 1, next five MB data as partition 2, next five MB data as partition 3 and the last five MBs as partition 4. But the problem with sequential partitioning is that there might be some item that are excessively bought during some specific time period, under some specific abnormal circumstances, which actually are not the large itemsets, but within a partition containing those transactions of that particular time period, it appear as large itemsets. Such situations cause into wasted efforts for considering something as large itemsets, which factually are not. Let's take October 8th earthquake example once again. During those 3-4 months, sweaters, jackets and blankets were extensively purchased by the people of the saved areas, in order to gift those to the people of devastated areas, to help them fight the extreme climatic conditions. Under normal circumstances, these items are not heavily bought items, because people usually buy two or three sweaters during the whole winters. And blankets are the least purchased items, as those are purchased after some five years. So, by taking sequential partitions, the partition consisting of transactions of the last quarter of the year 2005 will result blankets, sweaters and jackets as heavily bought items, which actually are not and also no other partition will approve those as the heavy itemsets. So it means we are wasting our resources on wrong candidates, in other words, false positives. To overcome these problems caused by the sequential partitioning, random partitions are used. We devised a random function, which randomly picks transactions and store them in partitions and at the end, equal-sized, n number of partitions are created. These partitions are random, equal-sized and non-overlapping. By non-overlapping, we mean that a transaction once picked for a particular partition, have 0% probability to be picked up again for any other partition. The functionality of random function is tested over a number of experiments, with varying number of transactions different datasets and it proved its performance in every case.

Decreased disk I/Os: Earlier, algorithms were taking multiple database scans in order to calculate the support counts of the candidate itemsets, to approve them as large itemsets or not. Which in large databases, means a lot of extra work to be done. In CMA algorithm, we are using a formula (as is used by Cheung ***et al.*** (1996) in DMA algorithm) to check the support of an itemset. To calculate locally large itemsets, s * D_i is used, in which s is the support of the itemset, set by the user (50% in these experiments) and D_i is the size of each partition. For global support count, s * D is used, where s is same support, while D is the size of entire central database.

The size of the D_i is selected by keeping in view the hardware requirements of the system in use, so that each partition fits well in the memory, keeping enough space for the L1_List, L_List. And other Operating System processes to run, simultaneously. After the partitioning, each partition is to be loaded into the memory and then the DMA algorithm (with due amendments) is to be applied on it. As the database to be used in this study is a centralized database, so there is no need of polling site technique or polling requests etc. Similarly, only one set of support count is to be generated, instead of the two sets generated in DMA. The database to be used for the experiments is the synthetic data, created exclusively for this study. The functionality of our devised CMA algorithm is presented in the Fig. 1. When a partition is loaded into the memory, the partition is scanned and supports for the candidate itemsets are counted and stored in the L1_List. To find out the large itemsets of the partition, each itemset is checked for the following condition:

```
If X.sup_i >= s * D_i then
Add(X, X.sup_i, partition#) into L_List
```

Where D_i is the number of transactions in ith partition.The itemsets not meeting this condition are pruned away to avoid extra processing. In the next iteration, the function apriori_gen () is applied on the large itemsets created in the previous iteration and so on, till either no new itemset is added to the L_List, or no itemsets are there to be the input for the apriori_gen () in the next iteration. At the end of calculations in all the partitions, the itemsets in the G_List are now checked again, this time with the D, the number of transactions in the entire database. The condition is:

```
For all X ∈ L_List
        {$X.sup = \sum^{n}_{i=1} X.sup_i$;
        If X.sup>s * D then
        {Insert X into L_List AND
        Prune away rest of the itemsets;
        }
    }
```

So only the actually large itemsets are left in the L_List. CMA algorithm is presented in Fig. 1.

```
Partition = Random (Centralized DB); //n number of
partitions are generated
P = partition-database (D)
N = number of partitions
For i = 1 to n begin
   Read-in-partition (pi∈P) // load partition
     If k = 1 then
         Scan $DB_i$ to compute L1_List;
     Else
         {
$C_k = U^{n}_{i=1} C^{i}_{k} = U^{n}_{i=1}$ Apriori_gen
         ($L1\_List^{i}_{k-1}$)
     Scan $DB_i$ for support and put in $C_k$;
         }
//calculation of locally large itemset
for all X ∈ $C_k$ do
         {
         If $X.sup_i \geq s * D_i$ then
         Add(X, $X.sup_i$, partition#) into
         L_List;
         }
//calculation of actually large itemset
For all X ∈ L_List
         {
         $X.sup = \sum^{n}_{i=1} X.sup_i$;
         If $X.sup \geq s * D$ then
                 {
                 Insert X into L_List AND
                 Prune away rest of the itemsets;
                 }
         }
```

Fig. 1 CMA-Algorithm

Example: To illustrate the performance of CMA algorithm, we give in the following section an example, in which at first Partition algorithm is applied on the central database, which divides the database into three distinct, non-overlapping and random partitions, and then calculates the locally large k-itemsets, and finally computes globally large itemsets of size 1,2,...,k. In the second part of the example, CMA Algorithm is applied on the same database, which also divides the central database into three distinct, non-overlapping and random partitions and then calculates locally large itemsets by bringing each partition one by one into memory. Globally large itemsets are calculated at the end. Instead of counting the itemsets in the transactions of the actual database, CMA Algorithm uses the formula of $s*D_i$ (as is used by Cheung *et al.* (1996) for counting support within each partition, where s is the support of the itemset, set by the user, like 30, 60 and 75% etc. (50% in this example) and D_i is the size of each partition database (3 in this example). For global support count, it uses the formula, s * D, where D is the size of the entire database (12 in this example). The dataset is presented below:

TID	Itemset
T1	A,B,C
T2	A,B,D
T3	A,D,E
T4	A,B,D
T5	B,C,D
T6	A,B,E
T7	A,D,E
T8	B,C,E
T9	A,B,C
T10	A,C,D
T11	A,D,E
T12	C,D,E

First, Partition algorithm is applied to the database, which works on random partitions, so the original database, given in dataset presentation, is divided into three distinct, non-overlapping and random partitions, which are presented in Table 1.

It then generates the candidate 1-itemset and then counts its support. The support taken in this example is 50%. So the large itemsets are created and stored along with their TIDs, as shown in Table 2.

Then candidate 2-itemset is generated by multiplying large 1-itemset with itself, like $l_1 * l_1$.The candidate 2-itemsets of partition 1 are {AB, AC, AD, AE, BC, BD, BE, CD, CE, DE}. Then the TIDs of the transactions containing these 2-itemsets are stored against each 2-itemsets. The number of TIDs against each itemset shows its occurrence in the partition. For example, in partition 1, AB is present in T2, T6 and T9, so it means it has occurrence of 3 out of 4 transactions, i.e., 3/4 = 0.75, which means 75% transactions in partition 1 contains AB, which

Table 1: Three Partitions created after dividing the database

Partition 1		Partition 2		Partition 3	
TID	Itemset	TID	Itemset	TID	Itemset
T2	A,B,D	T1	A,B,C	T3	A,D,E
T6	A,B,E	T4	A,B,D	T5	B,C,D
T9	A,B,C	T7	A,D,E	T8	B,C,E
T12	C,D,E	T11	A,D,E	T10	A,C,D

Table 2: Candidate 1-itemsets of partition 1

TID	1-itemset
T2	A
T2	B
T2	D
T6	A
T6	B
T6	E
T9	A
T9	B
T9	C
T12	C
T12	D
T12	E

is greater than 50%, so AB is a locally large 2-itemset in partition 1. The supports calculated for each candidate 2-itemset is given below:

AB = {T2,T6,T9}, so support is AB = 3/4 = 0.75
AC = {T9}, so support is AC =1/4 = 0.25
AD = {T2}, so support is AD =1/4 = 0.25
AE = {T2}, so support is AE =1/4 = 0.25
BC = {T9}, so support is BC =1/4 = 0.25
BD = {T2}, so support is BD = 1/4 = 0.25
BE = {T6}, so support is BE = 1/4 = 0.25
CD = {T12}, so support is CD = 1/4 = 0.25
CE = {0}, so support is CE = 0
DE = {T12}, so support is DE = 1/4 = 0.25

So the large 2-itemsets in Partition 2 are {AB, AD and AE, DE}. From this we will have candidate 3-itemset of {ABD, ADE, ABE, BDE}. But as only 1 large 2-itemset is present in partition 1, so no candidate 3-itemset is possible. Same is repeated with partition 2 and partition 3, which are illustrated in Table 3 and 4.

Partition = 2

Candidate 2-itemset= $l_1 * l_1$ ={AB,AC, AD, AE, BC, BD, BE,CD, CE, DE}

AB = {T1T,4}, so the support is= AB= 2/4 = 0.5
AC = {T1}, so the support is=1/4 = 0.25
AD = {T4,T7,T11}, so the support is= 3/4 = 0.75
AE = {T7,T11}, so the support is= 2/4 = 0.5
BC = {T1}, so the support is=1/4 = 0.25
BD = {T4}, so the support is=1/4 = 0.25
BE = {0}, so the support is = 0
CD = {0}, so the support is = 0
CE = {0}, so the support is = 0
DE = {T7,T11}, so the support is= 2/4 = 0.5

Table 3: Candidate 1-itemset of partition 2

TID	1-itemset
T1	A
T1	B
T1	C
T4	A
T4	B
T4	D
T7	A
T7	D
T7	E
T11	A
T11	D
T11	E

Table 4: Candidate 1-itemset of partition 2

TID	1-itemset
T3	A
T3	D
T3	E
T5	B
T5	C
T5	D
T8	B
T8	C
T8	E
T10	A
T10	C
T10	D

Candidate 3-itemsets are

ABD = {T4}, so the support is = 1/4 = 0.25
ADE = {T7,T11}, so the support is = 2/4 = 0.5
ABE = {0}, so the support is = 0
BDE = {0}, so the support is = 0

So at the end we have only ADE as large 3-itemset in Partition 2.

Partition = 3
Supports of the 1-itemsets are

A = 4, B = 2, C = 1, D = 3, E = 2
Candidate 2-itemset = {AB, AC, AD, AE, BC, BD, BE, CD,CE, DE }
AB = {0}, so the support is = 0
AC = {T10}, so the support is = 1/4 = 0.25
AD = {T3,T10}, so the support is = 2/4 = 0.5
AE = {T3}, so the support is = 1/4 = 0.25
BC = {T5,T8}, so the support is = 2/4 = 0.5
BD = {T5}, so the support is = 1/4 = 0.25
BE = {T8}, so the support is = 1/4 = 0.25
CD = {T5,T10}, so the support is = 2/4 = 0.5
CE = {T8}, so the support is =1/4 = 0.25
DE = {T3}, so the support is =1/4 = 0.25

As we have only one large 2-itemset so no candidate 3-itemset is possible.

Now, we combine the large itemsets of all the partitions and merge them in a single global set (Table 5-7).

Table 5: Globally large 1-itemsets

TID	1-itemset
1	A
1	B
1	C
2	A
2	B
2	D
3	A
3	D
3	E
4	A
4	B
4	D
5	B
5	C
5	D
6	A
6	B
6	E
7	A
7	D
7	E
8	B
8	C
8	E
9	A
9	B
9	C
10	A
10	C
10	D
11	A
11	D
11	E
12	C
12	D
12	E

Table 6: Globally large 2-itemsets

TID	2-itemset
T1,T2,T4	AB
T2,T3,T4	AD
T2,T4	BD
T5,T8	BC
T6,T7	AE
T6,T8	BE
T9,T10	AC
T10,T11	AD
T10,T12	CD
T11,T12	DE

Table 7: Globally large 3-itemsets

TID	3-itemset
T2,T4	ABD

Supports of the 1-itemsets are

A = 9, B = 7, C = 6, D = 7, E = 6

Supports of the large 2-itemsets are

AB = 7, AD = 2, BD = 1, BC = 1, AE = 1, BE = 1, AC = 1, AD = 1, CD = 1, DE = 1.

And the support of the 3-itemset is 1, so no globally large 3-itemset is present in the database taken.

Now, we apply CMA Algorithm on the same database, which also works by randomizing the database and divides the database into three distinct, non-overlapping and random partitions, which are presented in Table 8.

We consider in this example, s = 50%. So the candidate 1-itemsets are presented in Table 9.

In Partition 1 the large 1-itemsets are {A, B, C, D, E}. In Partition 2 the large 1-itemsets are {A, B, D, E}. While in Partition 3 the large 1-itemsets are {A, B, C, D, E}.

The candidate 2-itemsets in Partition 1 are {AB, AC, AD, AE, BC, BD, BE, CD, CE, DE}, in Partition 2 are {AB, AD, AE, BD, BE, DE} and in Partition 3, are {AB, AC, AD, AE, BC, BD, BE, CD, CE, DE}. The large 2-itemsets of three partitions are presented in Table 10.

As shown in the Table 10, large 2-itemset in Partition 1 is {AB}, in Partition 2 are {AB, AD, AE, DE} and in

Table 8: 3 Partitions created by CMA

Partition 1		Partition 2		Partition 3	
TID	Itemset	TID	Itemset	TID	Itemset
T2	A,B,D	T4	A,B,D	T3	A,D,E
T9	A,B,C	T11	A,D,E	T5	B,C,D
T6	A,B,E	T1	A,B,C	T8	B,C,E
T12	C,D,E	T7	A,D,E	T10	A,C,D

Table 9: Candidate 1-itemsets

Partition 1		Partition 2		Partition 3	
Candidate 1-Itemset	Local count (X.support $\geq s^*D_i$)	Candidate 1-Itemset	Local count (X.support $\geq s^*D_i$)	Candidate 1-Itemset	Local count (X.support $\geq s^*D_i$)
A	$3 \geq 0.5^*4$	A	$4 \geq 0.5^*4$	A	$2 \geq 0.5^*4$
B	$3 \geq 0.5^*4$	B	$2 \geq 0.5^*4$	B	$2 \geq 0.5^*4$
C	$2 \geq 0.5^*4$	C	$1 \geq 0.5^*4$	C	$3 \geq 0.5^*4$
D	$2 \geq 0.5^*4$	D	$3 \geq 0.5^*4$	D	$3 \geq 0.5^*4$
E	$2 \geq 0.5^*4$	E	$2 \geq 0.5^*4$	E	$2 \geq 0.5^*4$

Table 10: Large 2-itemsets and their supports

P1		P2		P3	
CI_2^1	$X.sup^1$	CI_2^2	$X.sup^2$	CI_2^3	$X.sup^3$
AB	3	AB	2	AB	0
AC	1	AD	3	AC	1
AD	1	AE	2	AD	2
AE	1	BD	1	AE	1
BC	1	BE	0	BC	2
BD	1	DE	2	BD	1
BE	1			BE	1
CD	1			CD	2
CE	1			CE	1
DE	1			DE	1

Table 11: Large 3-itemsets and their supports

P2		P3	
CI_3^2	$X.sup^2$	CI_3^3	$X.sup^3$
ABD	1	ABC	0
ADE	2	ABD	0
ABE	0	ACD	1
		BCD	1

Table 12: Global Support count of candidate itemsets

Locally large candidate set	X.sup[1]	X.sup[2]	X.sup[3]	X.Sup $\geq$s*D
A	3	4	2	9≥6
B	3	2	2	7≥6
C	2	1	3	6≥6
D	2	3	3	8≥6
E	2	2	2	6≥6
AB	3	2	0	5<6
AC	1	0	1	2<6
AD	1	3	2	6≥6
AE	1	2	1	4<6
BC	1	0	2	3<6
BD	1	1	1	3<6
BE	1	0	1	2<6
CD	1	0	2	3<6
CE	1	0	1	2<6
DE	1	2	1	4<6

Partition 3 are {AD, BC, CD}. Table 10 shows that no candidate 3-itemset in Partition 1 is possible. Whereas in Partition 2 the candidate 3-itemset are {ABD, ADE, ABE} and in Partition 3 {ABC, ABD, ACD, BCD}.

The support counts of candidate 3-itemsets of three partitions are given in the Table 11.

So the only large 3-itemset found is {ADE}: In order to check whether the locally large itemsets are actually large in database or not, we check each locally large itemset globally with the help of formula X.Sup≥ s*D, where D is the size of the entire database, which in this case is 12.

From the Table 12, the only globally large 2-itemset is {AD} and sine we have a single globally large 2-itemset so no candidate 3-itemset is possible and the process stops here.

CONCLUSIONS

Previously, the database was used to be scanned again and again for the generation of candidate itemsets and then the actually large itemsets, which in the case of huge database means a lot. Which resulted in higher CPU costs and large number of I/Os as well. CMA algorithm presented, not only reduces it, but also decreases the database scanning with the percentage of 50%, which is an achievement. It also results in the efficient processing speed and hence association rules could easily be generated from them. The database scans taken by the different algorithms are given in Fig. 2.

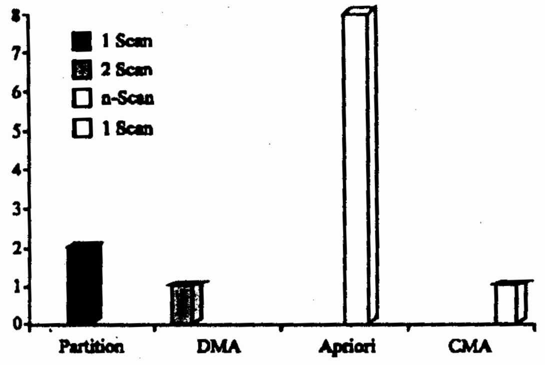


Fig. 2: Database scans taken by different algorithms

We have implemented CMA Algorithm, but used only with our synthetic dataset, created exclusively for the study. In future we will compare PARTITION and our designed CMA Algorithm with different datasets and systems to conduct a comparative study.

REFERENCES

Agarawal, R., T. Imielinski and A. Swami, 1993. Mining association rules Between Sets of item in large databases. Proceedings of the ACM International Conference on Management of Data, pp: 207-216.

Agarawal, R. and R. Srikant, 1994. Fast algorithms for mining association rules. Proceedings of the International Very large Databases Conference, pp: 487-499.

Cheung, D.W., V.T. Ng, A.W. Fu and Y. Fu, 1996. Efficient mining of association rules in distributed databases. IEEE Trans. on Know. Data Engineering, pp: 911-921.

Han, J.M., Kamber and Simon Fraser University, 2001. Data Mining: Concepts and Techniques, Kaufmann Publishers.

Houtsma, M. and A. Swami, 1995. Set-Oriented mining for association rules in relational databases. Proceedings of the IEEE Intl. Conference on Data Engineering, pp: 25-32.

Savasere, A., E. Omiecinski and S. Navathe, 1995. An Efficient algorithm for mining association rules in large databases. Proceedings of the 21st VLDB Conference, Zurich, Switzerland, pp: 432-443.

Farrukh (July 19, 2006)

Appendix C

.NET Technology

C-1 Microsoft .NET

- Microsoft .NET is software that connects information, people, systems, and devices. It spans clients, servers, and developer tools, and consists of:
- The .NET Framework 1.1, used for building and running all kinds of software, including Web-based applications, smart client applications, and XML Web services components that facilitate integration by sharing data and functionality over a network through standard, platform-independent protocols such as XML (Extensible Markup Language), SOAP, and HTTP.
- Developer tools, such as Microsoft Visual Studio® .NET 2003 which provides an integrated development environment (IDE) for maximizing developer productivity with the .NET Framework.
- A set of servers, including Microsoft Windows® Server 2003, Microsoft SQL Server™, and Microsoft BizTalk® Server, that integrates, runs, operates, and manages Web services and Web-based applications.
- Client software, such as Windows XP, Windows CE, and Microsoft Office XP, that helps developers deliver a deep and compelling user experience across a family of devices and existing products.

C-2 .NET Framework

- The .NET Framework is an integral Windows component for building and running the next generation of software applications and Web services. The .NET Framework:
- Supports over 20 different programming languages.
- Manages much of the plumbing involved in developing software, enabling developers to focus on the core business logic code.
- Makes it easier than ever before to build, deploy, and administer secure, robust, and high-performing applications.

- The .NET Framework is composed of the common language runtime and a unified set of class libraries

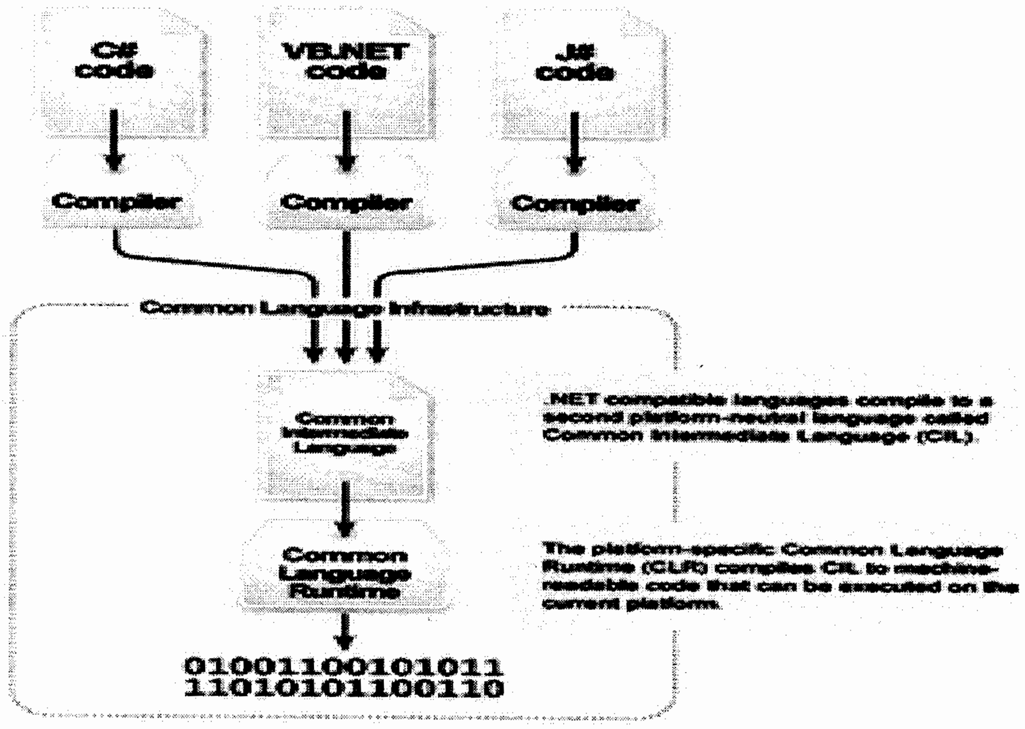
C-3 Design Goals and Principal Features

The .NET Framework was designed with several intentions:

- **Interoperability** - Because so many COM libraries have already been created, the .NET Framework provides methods for allowing interoperability between new code and existing libraries.
- **Common Runtime Engine** - Programming languages on the .NET Framework compile into an intermediate language known as the Common Intermediate Language, or CIL; Microsoft's implementation of CIL is known as Microsoft Intermediate Language, or MSIL. In Microsoft's implementation, this intermediate language is not interpreted, but rather compiled in a manner known as just-in-time compilation (JIT) into native code. The combination of these concepts is called the Common Language Infrastructure (CLI), a specification; Microsoft's implementation of the CLI is known as the Common Language Runtime (CLR).
- **Language Independence** - The .NET Framework introduces a Common Type System, or CTS. The CTS specification defines all possible datatypes and programming constructs supported by the CLR and how they may or may not interact with each other. Because of this feature, the .NET Framework supports development in multiple programming languages. This is discussed in more detail in the .NET languages section below.
- **Base Class Library** - The Base Class Library (BCL), sometimes referred to as the Framework Class Library (FCL), is a library of types available to all languages using the .NET Framework. The BCL provides classes which encapsulate a number of common functions such as file reading and writing, graphic rendering, database interaction, XML document manipulation, and so forth.

- **Simplified Deployment** - Installation and deployment of Windows applications has been the bane of many developers' existence. Registry settings, file distribution and DLL hell have been nearly completely eliminated by new deployment mechanisms in the .NET Framework.
- **Security** - .NET allows for code to be run with different trust levels without the use of a separate sandbox.
- The design of the .NET Framework is such that it supports platform independence. That is, a program written to use the framework should run without change on any type of computer for which the framework is implemented. At present, Microsoft has implemented the full framework only on the Windows operating system. Microsoft and others have implemented portions of the framework on non-Windows systems, but to date those implementations are not widely used.

C-4 NET Framework architecture



.NET compatible languages compile to a second platform-neutral language called Common Intermediate Language (CIL).

The platform-specific Common Language Runtime (CLR) compiles CIL to machine-readable code that can be executed on the current platform.

Visual overview of the Common Language Infrastructure (CLI)

C-5 Common Language Infrastructure (CLI)

The most important component of the .NET Framework lies in the Common Language Infrastructure, or CLI. The purpose of the CLI is to provide a language agnostic platform for application development, including, but not limited to, components for: exception handling, garbage collection, security, and interoperability. **Microsoft's** implementation of the CLI is called the Common Language Runtime, or **CLR**. The CLI is composed of five primary parts:

- Common Type System (CTS)
- Common Language Specification (CLS)
- Common Intermediate Language (CIL)
- Just-in-Time Compiler (JIT)
- Virtual Execution System (VES)

C-6 Assemblies

The intermediate MSIL code is housed in .NET assemblies, which for the Windows implementation means a Portable Executable (PE) file (EXE or DLL). Assemblies are the .NET unit of deployment, versioning and security. The assembly can be made up of one or more files, but one of these must contain the manifest, which has the metadata for the assembly. The complete name of an assembly contains its simple text name, version number, culture and public key token; it must contain the name, but the others are optional. The public key token is generated when the assembly is created, and is a value that uniquely represents the name and contents of all the assembly files, and a private key known only to the creator of the assembly. Two assemblies with the same public key token are guaranteed to be identical. If an assembly is tampered with (for example, by hackers), the public key can be used to detect the tampering.

C-7 Metadata

All CIL is self-describing through .NET metadata. The CLR checks on metadata to ensure that the correct method is called. Metadata is usually generated by language compilers but developers can create their own metadata through custom attributes.

C-8 Base Class Library (BCL)

The Base Class Library (BCL), sometimes incorrectly referred to as the Framework Class Library (FCL) (which is a superset including the Microsoft.* namespaces), is a library of types available to all languages using the .NET Framework. The BCL provides classes which encapsulate a number of common functions such as file reading and writing, graphic rendering, database interaction, XML document manipulation, and so forth. The BCL is much larger than other libraries, but has much more functionality in one package.

C-9 Security

.NET has its own security mechanism, with two general features: Code Access Security (CAS), and validation and verification. Code Access Security is based on evidence that is associated with a specific assembly. Typically the evidence is the source of the assembly (whether it is installed on the local machine, or has been downloaded from the intranet or Internet). Code Access Security uses evidence to determine the permissions granted to the code. Other code can demand that calling code is granted a specified permission. The demand causes the CLR to perform a call stack walk: every assembly of each method in the call stack is checked for the required permission and if any assembly is not granted the permission then a security exception is thrown.

When an assembly is loaded the CLR performs various tests. Two such tests are validation and verification. During validation the CLR checks that the assembly contains valid metadata and CIL, and it checks that the internal tables are correct. Verification is not so exact. The verification mechanism checks to see if the code does anything that is 'unsafe'. The algorithm used is quite conservative and hence sometimes code that is 'safe' is not verified. Unsafe code will only be executed if the assembly has the 'skip verification' permission, which generally means code that is installed on the local machine.

The languages supported by the .Net framework are listed below:

Supported programming languages		
APL	Fortran	Pascal
C++	Haskell	Perl
C#	Java Language	Python
COBOL	Microsoft JScript®	RPG
Component Pascal	Mercury	Scheme
Curriculum	Mondrian	SmallTalk
Eiffel	Oberon	Standard ML
Forth	Oz	Microsoft Visual Basic®