Comparative Study of Randomized PARTITION with CMA

T-4036

T-4036

Developed by
Fakhra Razi
Samina Kausar

Supervised by
Prof. Dr. Malik Sikander Hayat Khiyal
Mr. Muhammad Imran Saeed

Department of Computer Science
Faculty of Basic and Applied Sciences

International Islamic University, Islamabad.

(2007)

161711:0

MS
006.3
SAC

بسم الله الرحمن الرحيم

In The Name of

ALLAH ALMIGHTY

The Most Merciful, The Most Beneficent

Department of Computer Science

International Islamic University, Islamabad.

22nd Sept, 2007

Final Approval

It is certified that we have read this project report submitted by Miss Fakhra Razi and Miss Samina Kausar. It is our Judgment that this report is sufficient standard to warrant its acceptance by International Islamic University, Islamabad for the MS Degree in Computer Science.

Committee

1. External Examiner

Dr. Tasneem Shah
Prof., Computer Science Department,
Air University, Rawalpindi.

[illegible]

2. Internal Examiners

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

[illegible]

3. Supervisors

Prof. Dr. Sikander Hayat Khiyal
Fatima Jinnah University, Rawalpindi.

[illegible]

Mr. Muhammad Imran Saeed
Assistant Professor,
Faculty of Basic and Applied Sciences,
International Islamic University, Islamabad.

[illegible]

A dissertation submitted to the

Department of Computer Science,

Faculty of Basic and 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 عيه 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.
&
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.

Fakhra Razi

Samina Kausar

Acknowledgment

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 **Prof. Dr. Malik Sikander Hayat Khiyal** who kept our morale high by his suggestions and appreciation.

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

Finally we must mention that it was mainly due to our parents' 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 and family, who mean most to us, for their prayers and support and are more precious before any treasure on the earth.

We are also thankful to **Mr. Kashif Jamil** and **Mr. Hafiz Abdurraheem** who mean the most to us, and whose prayers have always been a source of determination for us, for their moral support and help at every step. Without which we were unable to complete our project.

Fakhra Razi

Samina Kausar

Project In Brief

Project Title	Comparative Study of Randomized PARTITION with CMA
Organization	International Islamic University, Islamabad.
Undertaken By	Fakhra Razi Samina Kausar
Supervised By:	Prof. Dr. Malik Sikander Hayat Khiyal Asst. Prof. Mr. Muhammad Imran Saeed
Operating System:	Windows XP
Tools used:	Matlab 7, MS Excel 2000
System Used	Pentium IV
Starting Date	February, 2007.
End Date	July, 2007.

Abstract

Recently the environment that enterprises face has become more and more competitive. The proliferation of computer in today's dynamic business environment has increased the demand and urgency of successful business organization to be able to react rapidly to the changing market demands both locally and globally, by utilizing the Information technology such as the latest data mining techniques of extracting previously unknown and potentially useful information from vast resources of raw data. Data Mining and Knowledge Discovery in transactional and relational databases has been of great interest in recent years for customer-based applications.

Association Rule Mining is one of the core tasks 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 Algorithms find these associations between the frequently sold items, so that the user could put such associated items with in close proximity.

The Randomized PARTITION algorithm implemented in this study is more efficient than CMA due to its efficient partitioning technique. This version of Randomized PARTITION algorithm is also more efficient than previously implemented PARTITION algorithm as it reduces the memory usage by simply storing counts for each large itemset instead of storing the TIDs in hash tree. By using counts the time efficiency of Randomized PARTITION algorithm is also increased than previously implemented PARTITION algorithm. The experiments are performed over synthetic database, created exclusively for this project.

TABLE OF CONTENTS

Chapter 1

Introduction

1. Introduction

The past two decades has seen a dramatic increase in the amount of information or data being stored in electronic format. This accumulation of data has taken place at an explosive rate. It has been estimated that the amount of information in the world doubles every 20 months and the size and number of databases are increasing even faster. The amount of raw data stored in corporate databases is exploding. From trillions of point-of-sale transactions and credit card purchases to pixel-by-pixel images of galaxies, databases are now measured in gigabytes and terabytes. Raw data by itself, however, does not provide much information. In today's fiercely competitive business environment, companies need to rapidly turn these terabytes of raw data into significant insights into their customers and markets to guide their marketing, investment, and management strategies. The increase in use of electronic data gathering devices such as point-of-sale or remote sensing devices has contributed to this explosion of available data. Data storage became easier as the availability of large amounts of computing power at low cost i.e. the cost of processing power and storage is falling, made data cheap. There was also the introduction of new machine learning methods for knowledge representation based on logic programming etc. in addition to traditional statistical analysis of data. The new methods tend to be computationally intensive hence a demand for more processing power. Having concentrated so much attention on the accumulation of data the problem was what to do with this valuable resource? It was recognized that information is at the heart of business operations and that decision-makers could make use of the data stored to gain valuable insight into the business. Database Management systems gave access to the data stored but this was only a small part of what could be gained from the data. The drop in price of data storage has given companies willing to make the investment a tremendous resource: Data about their customers and potential customers stored in "Data Warehouses."

Data warehouses are becoming part of the technology. They are used to consolidate data located in disparate databases. A data warehouse stores large quantities of data by specific categories so it can be more easily retrieved, interpreted, and sorted by users. Warehouses enable executives and managers to work with vast stores of transactional or other data to respond faster to markets and make more informed business decisions. It has been predicted that every business will have a data warehouse within ten years, but merely storing data in a data warehouse does a company little good. Companies will want to learn more about that data to improve knowledge of customers and markets. Traditional on-line transaction processing systems, OLTPs, are good at putting data into databases quickly, safely and efficiently but are not good at delivering meaningful analysis in return. Analyzing data can provide further knowledge about a business by going beyond the data explicitly stored to derive knowledge about the business. This is where Data Mining or Knowledge Discovery in Databases (KDD) has obvious benefits for any enterprise.

1.1 Data Mining

Data Mining grew as a direct consequence of the availability of large reservoirs of data. Data collection in digital form was already underway by the 1960s, allowing for retrospective data analysis via computers. Relational Databases arose in the 1980s along with Structured Query Languages (SQL), allowing for dynamic, on-demand analysis of data. The 1990s saw an explosion in growth of data. Data warehouses were beginning to be used for storage of data. Data Mining thus arose as a response to challenges faced by the database community in dealing with massive amounts of data, application of statistical analysis to data and application of search techniques from Artificial Intelligence to these problems.

Generally, data mining (sometimes called data or knowledge discovery) is the process of analyzing data from different perspectives and summarizing it into useful information that can be used to increase revenue, cuts costs, or both. . Seeking knowledge from massive data is one of the most desired attributes of data mining. Data could be large in two senses.

- In terms of size, e.g. image data
- In terms of dimensionality, e.g. gene expression data.

Technically, data mining is the process of finding correlations or patterns among dozens of fields in large relational databases.

It is the art and science of discovering and exploiting new, useful, and profitable relationships in data. It is a powerful new technology with great potential to help companies focus on the most important information in the data they have collected about the behavior of their customers and potential customers. It discovers information within the data that queries and reports can't effectively reveal. Data mining derives its name from the similarities between searching for valuable information in a large database and mining a mountain for a vein of valuable ore. Both processes require either sifting through an immense amount of material, or intelligently probing it to find where the value resides. Data mining is the process of extracting knowledge hidden from large volumes of raw data. It automates the process of finding relationships and patterns in raw data and delivers results that can be either utilized in an automated decision support system or assessed by a human analyst. Data mining techniques can be implemented rapidly on existing software and hardware platforms to enhance the value of existing information resources, and can be integrated with new products and systems as they are brought on-line. When data mining tools are implemented on high performance parallel processing systems, they can analyze massive databases in minutes. Faster processing means that users can automatically experiment with more models to understand complex data. High speed makes it practical for users to analyze huge quantities of data. Larger databases, in turn, yield improved predictions.

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. 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 [1].

1.2 Foundations of Data Mining

Data mining techniques are the result of a long process of research and product development. This evolution began when business data was first stored on computers, continued with improvements in data access, and more recently, generated technologies that allow users to navigate through their data in real time. Data mining takes this evolutionary process beyond retrospective data access and navigation to prospective and proactive information delivery. Data mining is ready for application in the business community because it is supported by three technologies that are now sufficiently mature:

- Massive data collection
- Powerful multiprocessor computers
- Data mining algorithms

1.3 Potential Applications of Data Mining

Although data mining is still in its infancy, companies in a wide range of industries including retail, finance, heath care, manufacturing transportation, and aerospace are already using data mining tools and techniques to take advantage of historical data. By using pattern recognition technologies and statistical and mathematical techniques to sift through warehouse information, data mining helps analysts recognize significant facts, relationships, trends, patterns, exceptions and anomalies that might otherwise go unnoticed.

For businesses, data mining is used to discover patterns and relationships in the data in order to help make better business decisions. A wide range of companies have deployed successful applications of data mining. While early adopters of this technology have tended to be in information-intensive industries such as financial services and direct mail marketing, the technology is applicable to any company looking to leverage a large data warehouse to better manage their customer relationships. Two critical factors for success with data mining are: a large, well-integrated data warehouse and a well-defined understanding of the business process within which data mining is to be applied (such as customer prospecting, retention, campaign management, and so on). Data mining can help spot sales trends, develop smarter marketing campaigns, and accurately predict customer loyalty. Specific uses of data mining include:

1.3.1 Market Segmentation

Identify the common characteristics of customers who buy the same products from the company, e.g. large consumer package goods, company, can apply data mining to improve its sales process to retailers. Data from consumer panels, shipments, and competitor activity can be applied to understand the reasons for brand and store switching. Through this analysis, the manufacturer can select promotional strategies that best reach their target customer segments.

1.3.2 Interactive Marketing

Predict what each individual accessing a website is most likely interested in seeing. A credit card company can leverage its vast warehouse of customer transaction data to identify customers most likely to be interested in a new credit product. Using a small test mailing, the attributes of customers with an affinity for the product can be identified. Recent projects have indicated more than a 20-fold decrease in costs for targeted mailing campaigns over conventional approaches.

1.3.3 Market Basket Analysis

Market basket analysis is an algorithm that examines a long list of transactions in order to determine which items are most frequently purchased together. It takes its name from the idea of a person in a supermarket throwing all of their items into a shopping cart (a "market basket"). The results can be useful to any company that sells products, whether it's in a store, a catalog, or directly to the customer.

Market basket analysis is used to determine which products sell together. As such, the input to a market basket analysis is normally a list of sales transactions, where each column represents a product and each row represents either a sale or a customer, depending on whether the goal of the analysis is to find which items sell together at the same time, or to the same person. [2].

In retailing, most purchases are bought on impulse according to models of consumer behavior. Market basket analysis gives clues as to what a customer might have bought if the idea had occurred to them. It can be used as a first step in deciding the location and promotion of goods inside a store.

Example

If, as has been observed, purchasers of Barbie dolls are more likely to buy candy, then high-margin candy can be placed near to the Barbie doll display. Customers who would have bought candy with their Barbie dolls had they thought of it will now be suitably tempted. This, however, is only the first level of analysis.

Consider sitting in bar and buying a coke not a bar meal. While servicing the request, the barkeep asks if one is interested in a bag of chips as well. Why would the keep ask such a question? Because it is the goal of the barkeep, in some regards, to be profitable and maximize the amount of revenue per transaction. By asking if one wanted chips, the barkeep may make a bigger tip or the bar may make more revenue. The barkeep knew to ask this question, and knew there was a good chance (a high probability) that one would also take the chips. The barkeep had this knowledge from experience, specifically from previous interactions with customers.

Similarly, the association rule finding algorithm is trained on historical data, i.e. past transactions. The data contains checkout information and a list of products that were purchased in each transaction, perhaps along with other information (volume, sale amount, although in many cases just the presence or absence of a product in a transaction is sufficient). While training, the algorithm may identify a relationship (a form of an association) between coke and no bar meals, and predict the customers are more likely to buy crisps over someone not identified with that relationship.

Typically the relationship will be in the form of a rule such as:

IF {coke, no bar meal} THEN {crisps}

Knowledge Extraction from Market Basket Analysis

In market basket analysis, extracting knowledge means constructing models of purchase behavior, e.g. from supermarket sales data. In market-basket data mining, a critical step is finding of frequent itemsets (sets of items that are frequently bought in a single market basket). An efficient method for finding such itemsets along with their frequencies (number of occurrences in the dataset) is essential for enabling subsequent analysis of the data. Most data mining methods are sensitive to input parameters that must be carefully selected for each dataset (e.g., the support threshold for market-basket analysis). The market-basket problem assumes there are some large number of items, e.g., “bread," "milk" Customers fill their market baskets with some subset of the items, and the shopkeeper get to know that what items people buy together, even if he don't know who they are. Shopkeeper uses this information to position items, and control the way a typical customer traverses the store. In addition to the marketing application, the same sort of question has the following uses:

- Baskets = documents; items = words.

  Words appearing frequently together in documents may represent phrases or linked concepts, can be used for intelligence gathering.
- Baskets = sentences, items = documents.

  Two documents with many of the same sentences could represent plagiarism or mirror sites on the web.

News Filtering and Document Classification

Market basket analysis can also be used in the news filtering and document classification problems. Instead of just dealing with individual words, it is useful to identify 'word phrases' like "Los Angeles" which should be treated as a unit. If each pair of adjacent words is considered a "basket", then market basket analysis tells us which words occur together more often than would be expected by chance. Incorporating word phrases in this way can improve retrieval performance. This idea has also been applied to automatic transcription and indexing of medical reports, so that the computer can identify salient technical phrases used by doctors. Association rules identify collections of data attributes that are statistically related in the underlying data.

1.3.4 Banking

Detect patterns of fraudulent credit card use, Identify `loyal' customers, Predict customers likely to change their credit card affiliation, determine credit card spending by customer groups, find hidden correlations between different financial indicators and identify stock trading rules from historical market data.

1.3.5 Insurance and Health Care

Claims analysis, i.e., which medical procedures are claimed together, predict which customers will buy new policies, identify behavior patterns of risky customers, identify fraudulent behavior.

1.3.6 Transportation

Determine the distribution schedules among outlets, analyse loading patterns e.g. a diversified transportation company with a large direct sales force can apply data mining to identify the best prospects for its services. Using data mining to analyze its own customer experience, this company can build a unique segmentation identifying the attributes of high-value prospects.

1.3.7 Medicine

Characterize patient behavior to predict office visits, identify successful medical therapies for different illnesses, drug side effects, hospital cost analysis, genetic sequence analysis, prediction etc. A pharmaceutical company can analyze its recent sales force activity and their results to improve targeting of high-value physicians and determine which marketing activities will have the greatest impact in the next few months. The data needs to include competitor market activity as well as information about the local health care systems. The results can be distributed to the sales force via a wide-area network that enables the representatives to review the recommendations from the perspective of the key attributes in the decision process. The ongoing, dynamic analysis of the data warehouse allows best practices from throughout the organization to be applied in specific sales situations.

1.3.8 Knowledge Acquisition

Expert systems are models of real world processes so much of the information is available straight from the process e.g. in production systems, data is collected for monitoring the system, knowledge can be extracted using data mining tools, experts can verify the knowledge.

In the short-term, the results of data mining will be in profitable in ordinarily business related areas. Micro-marketing campaigns will explore new niches. Advertising will target potential customers with new precision. In the medium term, data mining may be as common and easy to use as e-mail. We may use these tools to find the best airfare to New York, root out a phone number of a long-lost classmate, or find the best prices on lawn mowers. The long-term prospects are truly exciting. Computers may reveal new treatments for diseases or new insights into the nature of the universe.

1.4 Data Mining Models

Modeling is simply the act of building a model (a set of examples or a mathematical relationship) based on data from situations where the answer is known and then applying the model to other situations where the answers aren't known. Data mining is the discovery of structures and patterns in large data sets. There are two aspects to data mining: model building and pattern detection. Model building in data mining is very similar to statistical modeling, although new problems do arise because of the large sizes of the data sets and the fact that data mining is secondary data analysis. Pattern detection seeks anomalies or small local structures in data, with the vast mass of the data being irrelevant.

1.4.1 Verification Model

The verification model takes a hypothesis from the user and tests the validity of it against the data. The emphasis is with the user who is responsible for formulating the hypothesis and issuing the query on the data to affirm or negate the hypothesis. The problem with this model is the fact that no new information is created in the retrieval process but rather the queries will always return records to verify or negate the hypothesis.

1.4.2 Discovery Model

The discovery model differs in its emphasis in that it is the system automatically discovering important information hidden in the data. The data is sifted in search of frequently occurring patterns, trends and generalizations about the data without intervention or guidance from the user. The discovery or data mining tools aim to reveal a large number of facts about the data in as short a time as possible.

An example of such a model is a bank database which is mined to discover many groups of customers to target for a mailing campaign. The data is searched with no hypothesis in mind other than for the system to group the customers according to the common characteristics found.

1.5 Scope of Data Mining

Data mining derives its name from the similarities between searching for valuable business information in a large database for example, finding linked products in gigabytes of store scanner data and mining a mountain for a vein of valuable ore. Both processes require either sifting through an immense amount of material, or intelligently probing it to find exactly where the value resides. Given databases of sufficient size and quality, data mining technology can generate new business opportunities by providing these capabilities:

1.5.1 Automated Prediction of Trends and Behaviors

Data mining automates the process of finding predictive information in large databases. Questions that traditionally required extensive hands-on analysis can now be answered directly from the data quickly. A typical example of a predictive problem is targeted marketing. Data mining uses data on past promotional mailings to identify the targets most likely to maximize return on investment in future mailings. Other predictive problems include forecasting bankruptcy and other forms of default, and identifying segments of a population likely to respond similarly to given events.

1.5.2 Automated Discovery of Previously Unknown Patterns

Data mining tools sweep through databases and identify previously hidden patterns in one step. An example of pattern discovery is the analysis of retail sales data to identify seemingly unrelated products that are often purchased together. Other pattern discovery problems include detecting fraudulent credit card transactions and identifying anomalous data that could represent data entry keying errors.

Data mining techniques can yield the benefits of automation on existing software and hardware platforms, and can be implemented on new systems as existing platforms are upgraded and new products developed. When data mining tools are implemented on high performance parallel processing systems, they can analyze massive databases in minutes. Faster processing means that users can automatically experiment with more models to understand complex data. High speed makes it practical for users to analyze huge quantities of data. Larger databases, in turn, yield improved predictions.

1.6 Data Mining Functions

Data mining methods may be classified by the function they perform or according to the class of application they can be used in. Some of the main techniques used in data mining are as follows:

1.6.1 Classification

Data mine tools have to infer a model from the database, and in the case of supervised learning this requires the user to define one or more classes. The database contains one or more attributes that denote the class of a tuple and these are known as predicted attributes whereas the remaining attributes are called predicting attributes. A combination of values for the predicted attributes defines a class.

Once classes are defined the system should infer rules that govern the classification therefore the system should be able to find the description of each class. The descriptions should only refer to the predicting attributes of the training set so that the positive examples should satisfy the description and none of the negative. A rule said to be correct if its description covers all the positive examples and none of the negative examples of a class.

1.6.2 Associations

Given a collection of items and a set of records, each of which contain some number of items from the given collection, an association function is an operation against this set of records which return affinities or patterns that exist among the collection of items. These patterns can be expressed by rules such as "72% of all the records that contain items A, B and C also contain items D and E." The specific percentage of occurrences (in this case 72) is called the confidence factor of the rule. Also, in this rule, A,B and C are said to be on an opposite side of the rule to D and E. Associations can involve any number of items on either side of the rule.

A typical application, identified by IBM that can be built using an association function is Market Basket Analysis. This is where a retailer runs an association operator over the point of sales transaction log, which contains among other information, transaction identifiers and product identifiers. The set of products identifiers listed under the same transaction identifier constitutes a record. The output of the association function is, in this case, a list of product affinities. Thus, by invoking an association function, the market basket analysis application can determine affinities such as "20% of the time that a specific brand toaster is sold, customers also buy a set of kitchen gloves and matching cover sets."

1.6.3 Sequential/Temporal Patterns

Sequential pattern mining functions are quite powerful and can be used to detect the set of customers associated with some frequent buying patterns. Use of these functions on for example a set of insurance claims can lead to the identification of frequently occurring sequences of medical procedures applied to patients which can help identify good medical practices as well as to potentially detect some medical insurance fraud.

1.6.4 Clustering/Segmentation

Clustering according to similarity is a very powerful technique. When learning is unsupervised then the system has to discover its own classes i.e. the system clusters the data in the database. The system has to discover subsets of related objects in the training set and then it has to find descriptions that describe each of these subsets. A task of identifying groups of records those are similar between themselves but different from the rest of the data.

1.7 Association Rules Mining

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, one 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 rule.

Association rule mining finds interesting associations and/or correlation relationships among large set of data items. Association rules show attributes value conditions that occur frequently together in a given dataset. A typical and widely-used example of association rule mining is Market Basket Analysis.

For example, data are collected using bar-code scanners in supermarkets. Such 'market basket' databases consist of a large number of transaction records. Each record lists all items bought by a customer on a single purchase transaction. Managers would be interested to know if certain groups of items are consistently purchased together. They could use this data for adjusting store layouts (placing items optimally with respect to each other), for cross-selling, for promotions, for catalog design and to identify customer segments based on buying patterns.

1.7.1 How Association Rules Work?

The usefulness of this technique to address unique data mining problems is best illustrated in a simple example. Suppose the data is to be collected at the check-out cash registers at a large book store. Each customer transaction is logged in a database, and consists of the titles of the books purchased by the respective customer, perhaps additional magazine titles and other gift items that were purchased, and so on. Hence, each record in the database will represent one customer (transaction), and may consist of a single book purchased by that customer, or it may consist of many (perhaps hundreds of) different items that were purchased, arranged in an arbitrary order depending on the order in which the different items (books, magazines, and so on) came down the conveyor belt at the cash register. The purpose of the analysis is to find associations between the items that were purchased together, i.e., to derive association rules that identify the items and co-occurrences of different items that appear with the greatest (co-frequencies). For example, to learn which books are likely to be purchased by a customer who is already known to purchase (or is about to purchase) a particular book. This type of information could then quickly be used to suggest to the customer those additional titles. One may already be "familiar" with the results of these types of analyses, if he is a customer of various on-line (Web-based) retail businesses; many times when making a purchase on-line, the vendor will suggest similar items (to the ones purchased) at the time of "check-out", based on some rules such as "customers who buy book title *A* are also likely to purchase book title *B*," and so on.

In the present context, an association rule tells us about the association between two or more items. For example: In 80% of the cases when people buy bread, they also buy milk. This tells us of the association between bread and milk. It is represented as -

Bread => milk | 80%

This should be read as - "Bread means or implies milk, 80% of the time." Here 80% is the "confidence factor" of the rule.

Association rules can be between more than 2 items. For example -

Bread, milk => jam | 60%

Bread => milk, jam | 40%

Given any rule, its confidence can easily be found. For example, for the rule

Bread, milk => jam

Count the number say n_1, of records that contain bread and milk. Of these, how many contain jam as well? Let this be n_2. Then required confidence is n_2/n_1.

1.7.2 Basic Concept

Let I = $\{i_1, i_2, \ldots, i_m\}$ be the set of items. Let D, the task-relevant data, is 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,

$$\text{Support } (A \Rightarrow B) = P(A \cup B).$$

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

Rules that satisfy both a minimum support threshold (min_sup) and a minimum confidence threshold (min_conf) are called **strong**. By convention, support and confidence value are written 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* 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:

- **Find all frequent itemsets:** By definition, each of those itemsets will occur at least as frequently as a pre-determined minimum support count.
- **Generate strong association rules from the frequent itemsets:** By definition these rules must satisfy minimum support and minimum confidence [3].

  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.

Strength of a rule is measured by using support/confidence as shown in the following example:

Support shows the frequency of the patterns in the rule; it is the percentage of transactions that contain both A and B, i.e.

Support = Probability (A and B)

Support = (# of transactions involving A and B) / (total number of transactions).

Confidence is the strength of implication of a rule; it is the percentage of transactions that contain B if they contain A, i.e.

Confidence = Probability (B if A) = P (B/A)

Confidence = (# of transactions involving A and B) / (total number of transactions that have A).

Example:

Customer	Item purchased	Item purchased
1	pizza	juice
2	salad	soda
3	pizza	soda
4	salad	tea

If A is "purchased pizza" and B is "purchased soda" then

Support = P (A and B) = ¼

Confidence = P (B / A) = ½

Confidence does not measure if the association between A and B is random or not.

For example, if milk occurs in 30% of all baskets, information that milk occurs in 30% of all baskets with bread is useless, but if milk is present in 50% of all baskets that contain coffee, that is significant information.

Support allows to weed out most infrequent combinations – but sometimes they should not be ignored, for example, if the transaction is valuable and generates a large revenue, or if the products repel each other.

Example: Measure the following:

P (Coke in a basket) = 50%

P (Pepsi in a basket) = 50%

P (coke and peps in a basket) = 0.001%

What does this mean? If Coke and Pepsi were independent, it would be expected that

P (coke and Pepsi in a basket) = .5*0.5 = 0.25.

The fact that the joint probability is much smaller says that the products are dependent and that they repel each other.

1.8 Association Rule Mining Environments

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

1.8.1 Centralized Environment

In Centralized Environment, there is one, large centralized database, from which the task is to identify the most frequently occurring item sets. 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 has also an important impact. 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.

1.8.2 Distributed Environment

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 extent, 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 property. So, in order to determine whether the locally large itemset is also globally large or not, all the nodes broadcast their large itemsets across the network, but this result in increased communication cost.

1.9 Existing Techniques

A lot of work has been done to overcome the association rule mining problem. Researchers introduced many techniques to handle this particular task. The basic purpose of each technique is to minimize the database scans and generate more and more true-positives. The techniques are as follows:

1.9.1 AIS Algorithm

Many algorithms have been discovered in this area .The earliest work for mining of association rules is presented in paper [4]. In this paper an algorithm AIS is introduced. It takes a separate database scan for almost every step, like candidate itemsets 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 generation of larger candidate itemsets is its major drawback.

1.9.2 SETM Algorithm

SETM algorithm is presented in paper [5]. The working of this algorithm is same as AIS; the only difference is that it uses SQL to compute large itemsets. 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.

1.9.3 Apriori Algorithm

Apriori algorithm is presented in paper [6]. It is called the pioneer work in this area. All other subsequent algorithms are the adaptation of Apriori to some extents. Due to some shortcomings in Apriori algorithm another version of Apriori called AprioriTid is presented. 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 its TIDs for this purpose. Working of this algorithm is efficient in later passes. Another variant of the same algorithm i.e. AprioriHybrid is also presented in the same paper. It combines the best features of both algorithms i.e. Apriori and AprioriTid. 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. 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.

1.9.4 DHP (Direct Hashing and Pruning) Algorithm

The algorithm DHP (Direct Hashing and Pruning) is presented in [7], which is an extension of Apriori. It is confined to the generation of large itemsets only. It also uses the Apriori_gen () and Subset () function as were used in the Apriori.

1.9.9 Survey of Parallel and Distributed Association Rule Mining Algorithms

The survey of parallel and distributed association rule mining algorithms is presented in [12]. These algorithms are divided into groups according to the techniques utilized. Existing mining methods are described according to the database structure, search, techniques used complexity and either they are specified on all or maximal patterns. The aim of this paper is to provide a reference and describes the challenges and problems in the fields of association rule mining.

1.9.10 CMA (Centralized Mining of Association-Rules) Algorithm

CMA (Centralized Mining of Association-Rules) Algorithm for mining of association rules is presented in [13]. The technique of PARTITION algorithm [3] of centralized database area is taken for partitioning the huge database and then, the DMA algorithm [4] of distributed database environment is applied on each partition. This technique improves the database scans as it takes just a single database scan over each partition.

The study of above techniques shows that the generation of large itemsets is the main problem in generating the association rules in large databases. Further it involves many problems like generation of large candidate itemsets and database involving multi scans to create candidate sets and to generate their support counts. Pruning of the item set is also a problem.

1.10 Conclusion

The Association Rule Mining is an important research area, which helps not only solving the retail industry problems by discovering hidden association between different itemsets, but also in different other areas like insurance and health care, medicine, telecommunications, effective advertising, targeted marketing, and inventory control, knowledge acquisition and finance.

When association rule mining algorithms are applied in centralized environment, they cannot be easily handled due to large database. Secondly, the extra database scans in that case are 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, either it is in partitions or a single large database.

Chapter 2

Literature Survey

2. Literature Survey

Algorithms for finding rules or affinities between items in a database are well known and well documented in the knowledge discovery community. The algorithms for performing market basket analysis are fairly straightforward. The complexities mainly arise in exploiting taxonomies, avoiding combinatorial explosions (a supermarket may stock 10,000 or more line items), and dealing with the large amounts of transaction data that may be available. A prototypical application of such affinity algorithms is in "market basket analysis" – the application of affinity rules to analyzing consumer purchases. Such analyses are of particular importance to the retail and consumer packaged goods industry. Maintenance of an existing customer base is more important than growing entirely new customers. This new phase of retail growth is based upon selling more and a greater variety of products to pre-existing consumers. The most profitable retailers are those that are able to maintain or reduce their operating costs. Economics of scope, not scale, determine profitability.

Data warehousing is one of the foremost technological means of increasing operational efficiency. Efficient consumer response systems, based upon data warehouses, are expected to save the industry $30 billion a year. A class of applications, based on intensive analysis of large amounts of data, is becoming of uttermost importance. The strong demand for intelligent data analysis comes from the increasing availability of massive information sources produced by organizations and stored in databases and/or made available through the Internet. The database technology, which is nowadays mature, easily and efficiently supports data warehousing and data aggregation/reporting, but hardly supports intelligent data analysis and knowledge extraction.

Data mining is an important application of the above mentioned class. A simple example of data mining is its use in a retail sales department. If a store tracks the purchases of a customer and notices that a customer buys a lot of silk shirts, the data mining system will make a correlation between that customer and silk shirts. The sales department will look at that information and begin direct mail marketing of silk shirts to that customer. In this case, the data mining system used by the retail store discovered new information about the customer that was previously unknown to the company.

Market basket analysis (also known as association-rule mining) is a useful method of discovering customer purchasing patterns by extracting associations or co-occurrences from stores' transactional databases. Because the information obtained from the analysis can be used in forming marketing, sales, service, and operation strategies, it has drawn increased research interest.

Current business processes often use data from several sources. Data is characterized to be heterogeneous, incomplete and usually involves a huge amount of records. This implies that data must be transformed in a set of patterns, rules or some kind of formalism, which helps to understand the underlying information. The participation of several organizations in this process makes the assimilation of data more difficult. Data mining is a widely used approach for the transformation of data to useful patterns, aiding the comprehensive knowledge of the concrete domain information. Data mining is emerging as one of the key features of many homeland security initiatives. Often used as a means for detecting fraud, assessing risk, and product retailing, data mining involves the use of data analysis tools to discover previously unknown, valid patterns and relationships in large data sets. In the context of homeland security, data mining is often viewed as a potential means to identify terrorist activities, such as money transfers and communications, and to identify and track individual terrorists themselves, such as through travel and immigration records.

The general goal of data mining is to extract interesting correlated information from large collection of data. A key computationally intensive sub problem of data mining involves finding frequent sets in order to help mine association rules for market basket analysis. Given a bag of sets and a probability, the frequent set problem is to determine which subsets occur in the bag with some minimum probability.

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 have become interested in mining association rules from their databases. The discovery of interesting relationships among huge amounts of business transaction records helps in many business decision making processes, such as catalog design, cross-marketing, and loss-leader analysis.

Association rule discovery techniques are generally applied to databases of transactions where each transaction consists of a set of items. In such a framework the problem is to discover all associations and correlations among data items where the presence of one set of items in a transaction implies (with a certain degree of confidence) the presence of other items.

2.1 Centralized Architecture of Association Rule Mining

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. 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. Today there are several efficient algorithms that cope with the popular and computationally expensive task of association rule mining. Actually, these algorithms are more or less described on their own. The study of few research papers of the centralized architecture of the association rule mining is given below:

2.1.1 Fast Algorithms for Mining Association Rules

Agarawal and Srikant; (1994) discussed Apriori algorithm [6] in this paper. It is termed as the pioneer work. All other subsequent algorithms are the adaptation of Apriori to some extent. This algorithm counts item occurrences from the database to determine large 1-itemset in first pass. In the next pass, the algorithm generates candidate itemsets and checks the support count.

This algorithm uses a special function, apriori-gen ( ) and 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. The node of Hash tree consists of either a list of itemsets i.e. a leaf node or a hash table i.e. an interior node .All initially created nodes are in the form of leaf nodes. Itemsets are stored in leaves. In a leaf node when the number of itemsets increases a predefined threshold then the leaf node is changed into an interior node Also, it generates multiple consequent association rules. It works efficient in earlier passes.

Another version of Apriori, AprioriTid, is also discussed in this paper, which works in the same way as Apriori, but does not use the database for counting support after the first pass. Instead it uses the pair of itemset and their TIDs for this purpose. Two additional fields are maintained for each candidate itemsets i.e. generators and extensions. It works efficiently in later passes. When Apriori switches to AprioriTid, it involves a cost.

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 algorithms. Basically in AprioriHybrid, it switches to AprioiTid when it expects that the set of candidate itemsets at the end of the pass will fit in memory. Experimental results show that AprioriHybrid has excellent performance and good linearly scale-up properties over large databases in real applications. In addition the execution time decreases a little as the number of items in the database increases. As the average transaction size increases (while keeping the database size constant), the execution time increases only gradually. The experiments showed the feasibility of using AprioriHybrid in real applications involving very large databases.

2.1.2 An Effective Hash-Based Algorithm for Mining Association Rules

Park et al; (1995) presented DHP (Direct Hashing and Pruning) algorithm [7], which is an extension of Apriori. It is confined to the generation of large itemsets, the step one of the mining association rules. It deals with the itemsets generation up to 2-itemsets. It also uses the Apriori_gen ( ) and Subset ( ) function as used in the base algorithm, Apriori. Efficient generation of large itemsets more efficiently and reduction of the transaction database are the major features of this algorithm.

It uses hashing technique for generation of candidate itemsets, especially for 2-itemsets and for reducing the database size. This technique filters out unnecessary itemsets for next candidate itemset generation. When the support of candidate k-itemsets is counted by database scanning, DHP accumulates information about candidate (k+1) itemsets in advance. After some pruning all possible (k+1) itemsets of each transaction are hashed to a hash table. Hash table building is a unique feature of DHP that is used by the next pass. It uses the effective pruning techniques to progressively shrink the size of transaction database. The candidate itemsets generated by previous algorithms was very large so tracking of k-itemsets in each transaction was not easy. Whereas DHP trims the database right after generation of large 2-itemsets, 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, it takes a bit longer time in first pass than Apriori, but its execution time for later iterations is much faster than Apriori.DHP is particularly powerful to determine large itemsets more efficiently in early stages to improve performance bottleneck.

2.1.3 An Efficient Algorithm for Mining Association Rules in Large Databases

Savasere et al., (1995) presented PARTITION algorithm [8] 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, PARTITION algorithm divides database into non-overlapping partitions. 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 l 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.

In phase I the algorithm takes partitions one by one and applies the function gen_large_itemsets ( ) to generate local large item sets as the output. All the local itemsets of the same length from the partitions are combined to generate the global candidate itemsets in the merge phase. In phase II the algorithm sets up counters for each global candidate itemset and counts their support for the entire database and generates the global large itemsets. The algorithm reads the entire database once during phase I and once during phase II.

2.2 Distributed Architecture of Association Rule Mining

In distributed environment, the data distribution is done either homogeneously (horizontally partitioned) or heterogeneously (vertically partitioned) across the different nodes of a network, so the problem that arises in a single huge database is solved to some extent. But due to data skewness property 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. So in order to determine that whether the locally large itemset is also globally large or not, all the nodes broadcast their large itemsets across the network. This process results in increased message passing which increases the cost factor. Typically communication is a bottleneck. It is commonly assumed that each site stores its data in tables. Both assumptions i.e. homogenous and heterogeneous adopt the conceptual viewpoint that tables at each site are partitions of a single global table. In the homogeneous case the global table is horizontally partitioned. The tables at each site are subsets of the global table. They have exactly the same attributes. In the heterogeneous case, the table vertically partitioned, each site contains a collection of columns (sites do not have the same attributes).however each tuple at each site is assumed to contain a unique identifier to facilitate matching across sites. Matched tuples contain the same identifiers.

Distributed/parallel ARM algorithms may entail much exchange of data (messaging) as the algorithm proceeds. Messaging represents a significant computational overhead (in some cases out weighing any other advantage gained). Usually, the number of messages sent is a much more significant performance factor than the size of the content of the message. It is therefore expedient, in the context of the techniques described, to minimize the number of messages that are required to be sent.

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 Fast Sequential and Parallel Algorithms for Association Rule Mining: A Comparison

Andreas Mueller (1995) presented a comparison of sequential and parallel algorithms [9] of association rule mining. In the first part of this research work a non_partitioning algorithm SEAR (Sequential Efficient Association Rules) having a new prefix-data structure is compared with a TID-list and partitioning based algorithm called SPTID (Sequential Partitioning with TID).According to this research partitioning algorithms reduces the I/O cost but when the number partitions is increased CPU overhead is also increased. As most algorithms are CPU bound, they make the performance slower as a result an increase in execution time. TID-list algorithm performs efficiently with partitioning, but partitioning overhead makes the performance slower. As compared to TID-lists, ITEM-lists by using pass bundling can minimize I/O and CPU overhead. They implemented parallel algorithms PPAR and SPEAR. They showed that these algorithms are not only easier to parallelize but obtained good speed-ups and scale up results, further when parallel version of SEAR is compared with SPEAR, SEAR performed better than parallel SPEAR at the expense of communication cost.

2.2.2 Efficient Mining of Association Rules in Distributed Databases

Cheung et al. (1996) enhanced their previous work discussed in [10] in this paper. DMA (Distributed Mining of Association Rules) algorithm is presented to cope with the previous problem. The performance of DMA is compared with 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( ).

In DMA, polling site technique is used 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.

Performance of DMA is tested 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.

2.2.3 Parallel Mining of Association Rules

Agrawal and Shafer;(1996) described association rule-mining problem on a shared-nothing multiprocessor in this paper [11]. Three parallel algorithms were studied for experiments based upon the best sequential algorithm called Apriori for experiments. These algorithms are the Count distribution, the Data distribution and the Candidate distribution. These algorithms were compared according to communication, computation, memory usage, synchronization and problem specific information usage. Count distribution uses replication, as a result of replication communication cost is reduced. Data distribution algorithm takes a specific portion of the candidate set by using the counter approach. This makes the aggregate-memory usage high but at the expense of high communication cost. Candidate distribution algorithm increases aggregate memory usage, decreases high communication cost and removes the synchronization costs completely. These three algorithms were implemented on 32-node SP2 parallel machine. In performance results the Count distribution showed the linear scale up, good size up and speed-up behavior, this makes it as the best choice as compared to other two parallel algorithms.

2.2.4 Parallel and Distributed Association Mining: A Survey

T-4036

J.Zaki; (1999) presented the survey of parallel and distributed association rule mining algorithms [12] in this paper. After detail study these algorithms are divided into groups according to the techniques utilized. Existing mining methods are described according to the database structure, search, techniques used complexity and either they are specified on all or maximal patterns. Further this paper provides the design space of parallel and distributed algorithms either they are implemented on distributed or shared–memory architecture, parallelism applied i.e. task or data and the load balancing strategy either it is static or dynamic. Shared memory architecture gives programming simplicity, but at the expense of scalability. A distributed memory, message-passing architecture removes the scalability problem, but at the cost of programming simplicity. The aim of this paper is to provide a reference and describes the challenges and problems in the fields of association rule mining.

2.2.5 Association Rule Mining in Centralized Databases

Jamshaid et al; (2007) presented CMA (Centralized Mining of Association-Rules) Algorithm [13] in this paper. The technique of PARTITION algorithm [9] of centralized database area is taken for partitioning the huge database and then, the DMA algorithm [10] of distributed database environment is applied on each partition. This technique improves the database scans as it takes just a single database scan over each partition.

2.3 Problem Statement

Association rule mining is decomposed into two steps:
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.** From these large itemsets, generate the association rules as follows:

For a large itemset X, and any $Y \subset X$, if support(X)/support(X-Y) $\geq$minimum_confidence then the rule $X\text{-}Y \Rightarrow Y$ is a valid rule.

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. There will be a little effort spent on pruning the small itemsets 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.
Firstly, the items are extracted from the database, then their support is counted from the database, then 2-itemsets are created, and upto k itemset creation the database is scanned again and again, i.e., the same is done for 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.
For Apriori discussed in [6], the database of transactions is scanned entirely for each pass, as in first pass database is scanned twice, so it means for 10 iterations, 10 +1 scans of the entire database are done.
AprioriTid algorithm which is the variant of Apriori does not scan the database 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.

Apriori Hybrid algorithm combines the best features of both Apriori and AprioriTid i.e. Apriori in early iterations and AprioriTid in later ones, but still 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 [7].

According to literature surveyed for n iterations Apriori algorithm takes n+1 database scans, PARTITION algorithm takes two database scans and CMA does only one database scan over each partition.

In literature survey it is proved that large itemsets generation is the main problem in generation of 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.

Chapter 3

Problem Domain and Proposed Solution

3. Problem Domain and Proposed Solution

In data mining and treatment learning, association rules are used to discover elements that co-occur frequently within a data set consisting of multiple independent selections of elements (such as purchasing transactions), and to discover rules, such as implication or correlation, which relate co-occurring elements. Questions such as "if a customer purchases product A, how likely is he to purchase product B?" and "What products will a customer buy if he buys products C and D?" are answered by association-finding algorithms. This application of association rule learners is also known as market basket analysis. As with most data mining techniques, the task is to reduce a potentially huge amount of information to a small, understandable set of statistically supported statements. Association rules are "if-then rules" with two measures which quantify the support and confidence of the rule for a given data set.

Association analysis has been broadly used in many application domains. One of the best known is the business field where the discovering of purchase patterns or associations between products is very useful for decision making and for effective marketing. In the last few years the application areas have increased significantly.

The market-basket problem represents an attempt by a retail store to learn what items its customers frequently purchase together. The goal is an understanding of the behavior of typical customers as they navigate the aisles of the store. For instance, if customers frequently buy hamburgers and ketchup together, then it might be supposed that many customers will walk from onto the other. If the store owner puts high-prot items tempting to such customers, e.g., relish, between, and then they might induce more impulse buying and thus increase ports.

Market basket analysis (MBA) is a tool used in data mining to find non-obvious statistical relationships between items by looking at how often two or more items appear in the same context or were interacted with during the same browsing or shopping session.

In market basket analysis, there is a collection of sets ("baskets") and the elements which often occur together in these sets is to be found. For example, grocery stores want to know what items a person is likely to buy within the same trip. They can put these items closer together in the store or in some cases farther apart so that more of the store is seen. They can put certain items on sale, with the expectation that people will buy certain other items which are not on sale. If the store has a way of identifying, such as with an advantage card, a market basket could extend over multiple purchases. These associations are used to trigger coupon suggestions. In e-commerce, the store always knows who you are, so this data is easy to gather. They can dynamically modify their web site to suggest other items which are likely to be bought.

By definition, association rule mining discovers frequent patterns with frequency above the minimum support threshold. Therefore, in order to find associations involving rare events, the algorithm must run with very low minimum support values. However, doing so could potentially explode the number of enumerated item sets, especially in cases with large number of items. That could increase the execution time significantly. Therefore, association rule mining is not recommended for finding associations involving "rare" events in problem domains with large number of items. However, there are ways to restrict the item set enumeration to a smaller set if the "rare" events of interest are known. One could also use association rules to perform "partial classification" of the rare events.

3.1 Problem Domain

There are some important issues in association rule mining those should be considered while devising an efficient association rule mining algorithm as the performance of the algorithm may affect .Some of the issues are:

3.1.1 Size of Data Sets

A large data set might contain a few hundred points. Certainly a data set of a few thousand would be large. Modern databases often contain millions of records. Indeed, nowadays gigabyte or terabyte databases are by no means uncommon. Human genome project has already collected gigabytes of data.

Numbers like these clearly put into context the futility of standard statistical techniques. Something new is called for. Data sets of these sorts of sizes lead to problems with which statisticians have not typically had to concern themselves in the past. An obvious one is that the data will not at all fit into the main memory of the computer, despite the recent dramatic increases in capacity. This means that, if all of the data is to be processed during an analysis, adaptive or sequential techniques have to be developed. Adaptive and sequential estimation methods have been of more central concern to non statistical communities. Especially to those working in pattern recognition and machine learning.

Data sets may be large because the number of records is large or because the number of variables is large. (Of course, what is a record in one situation may be a variable in another, depending on the objectives of the analysis.) When the number of variables is large, the curse of dimensionality really begins to bite- with 1,000 binary variables. The problem of limited computer memory is just the beginning of the difficulties that follow from large data sets. Perhaps the data are stored not as the single flat file as beloved of statisticians, but as multiple interrelated flat files.

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 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.

3.1.2 Contaminated Data

Clean data is a necessary prerequisite for most statistical analysis. Entire books, not to mention careers, have been created around the issues of outlier detection and missing data. An ideal solution, when questionable data items arise, is to go back and check the source. In the data mining context, however, when the analysis is necessarily secondary, this is impossible. Moreover, when the data sets are large, it is practically certain that some of the data will be invalid in some way. This is especially true when the data describe human interactions of some kind, such as marketing data, financial transaction data, or human resource data. Contamination is also an important problem when large data sets, in which seeking weak relationships, are involved. Suppose, for example, that one in a thousand records have been drawn from some distribution other than that which has been drawn from. One-tenth of 1% of the data from another source would have little impact in conventional statistical problems, but in the context of a billion records this means that a million are drawn from this distribution. This is sufficient that they cannot be ignored in the analysis.

another, depending on the objectives of the analysis.) When the number of variables is large, the curse of dimensionality really begins to bite- with 1,000 binary variables. The problem of limited computer memory is just the beginning of the difficulties that follow from large data sets. Perhaps the data are stored not as the single flat file as beloved of statisticians, but as multiple interrelated flat files.

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 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.

3.1.2 Contaminated Data

Clean data is a necessary prerequisite for most statistical analysis. Entire books, not to mention careers, have been created around the issues of outlier detection and missing data. An ideal solution, when questionable data items arise, is to go back and check the source. In the data mining context, however, when the analysis is necessarily secondary, this is impossible. Moreover, when the data sets are large, it is practically certain that some of the data will be invalid in some way. This is especially true when the data describe human interactions of some kind, such as marketing data, financial transaction data, or human resource data. Contamination is also an important problem when large data sets, in which seeking weak relationships, are involved. Suppose, for example, that one in a thousand records have been drawn from some distribution other than that which has been drawn from. One-tenth of 1% of the data from another source would have little impact in conventional statistical problems, but in the context of a billion records this means that a million are drawn from this distribution. This is sufficient that they cannot be ignored in the analysis.

3.1.3 Data Quality

Data quality is a multifaceted issue that represents one of the biggest challenges for association rule mining. Data quality refers to the accuracy and completeness of the data. It can also be affected by the structure and consistency of the data being analyzed. The presence of duplicate records, the lack of data standards, the timeliness of updates, and human error can significantly impact the effectiveness of the more complex data mining techniques, which are sensitive to subtle differences that may exist in the data. To improve data quality, it is sometimes necessary to "clean" the data, which can involve the removal of duplicate records, normalizing the values used to represent information in the database (e.g., ensuring that "no" is represented as a 0 throughout the database, and not sometimes as a 0, sometimes as a N, etc.), accounting for missing data points, removing unneeded data fields, identifying anomalous data and standardizing data formats.

3.1.4 Interoperability

Related to data quality, is the issue of interoperability of different databases and data mining software. Interoperability refers to the ability of a computer system and/or data to work with other systems or data using common standards or processes. Interoperability is a critical part of the larger efforts to improve interagency collaboration and information sharing through e-government and homeland security initiatives. For data mining, interoperability of databases and software is important to enable the search and analysis of multiple databases simultaneously, and to help ensure the compatibility of data mining activities of different agencies. Data mining projects that are trying to take advantage of existing legacy databases or that are initiating first-time collaborative efforts with other agencies or levels of government (e.g., police departments in different states) may experience interoperability problems. Similarly, as agencies move forward with the creation of new databases and information sharing efforts, they will need to address interoperability issues during their planning stages to better ensure the effectiveness of their data mining projects.

3.1.5 Multiple Database Scans

As association rule mining helps to find the hidden associations between items and generate association rules. Firstly 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. As in data mining, data is extracted from a huge database so the primary task should always be to minimize the database scans, the disk I/Os, and not the query execution time.

3.1.6 Large Candidate Set Size

Generation of large itemsets is found on the basis of their 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. This shows that candidate set plays a vital role in calculating the efficiency of the algorithm. 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.

3.1.7 Data for Association Models

Association models are designed to use sparse data. Sparse data is the 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.

3.1.8 Algorithm Execution Time

As the data warehouse contains the huge data gluts more than of twenty years so the algorithm execution time is not an issue. As the data mining queries are helpful in decision making. A decision whether it is good or bad can change the over all shape of the business, hence has a long lasting effect on it. When a company wants to invest its products in a foreign country, the businessmen want to know whether it will be a profitable decision or not? To answer such type of queries, they need to consider few factors. Like, what is the annual sale of those products there and what is their standard? To get such queries answered, a huge database needs to be considered; having records of some 10-20 years, and such queries would take at least a month to be answered. Whereas, using some data mining algorithm, results could be efficiently and quickly generated. The algorithm will tell them the required details quickly which will finally help the businessmen to decide whether to invest there or not, in a very short time as compared to manual calculation.

3.1.9 Accurate Number of Partitions

Partition splits the database, whereas this optimization helps to cope with large databases it adds the additional overhead of an extra pass to determine the globally frequent itemsets. In other words 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. In the case of partitions that can hold many possible data values analysis is more complex, because query predicate values are not guaranteed to coincide with partition boundary values. Thus in the first case, in order to determine an appropriate number of partitions for a column one is led to analyze how the average scan time of a set of range queries of a given size varies with number of partitions.

3.1.10 Data Skew

Skewness is a measure of symmetry, or more precisely, the lack of symmetry. A distribution, or data set, is symmetric if it looks the same to the left and right of the center point. One of the major problems of the PARTITION algorithm is data skew, which refers to the irregularity of data distribution over the entire database.

3.1.10.1 Outliers

An outlier is an observation that lies outside the overall pattern of a distribution. Usually, the presence of an outlier indicates some sort of problem. This can be a case which does not fit the model under study or an error in measurement. Outliers are uncommon observations that are clearly separated from the bulk of the data. Outliers in the data may be due to recording errors or system noise of various kinds, and as such need to be cleaned as part of the extract, transform, clean and load (ETCL) phase of the data mining/KDD process. On the other hand an outlier or small group of outliers may be quite error-free recordings that represent the most important part of the data that deserve further careful inspection, e.g., an outlier might represent an unusually high response to a particular advertising campaign, or an unusually effective dose-response combination in a drug therapy. Either way, it is quite important in data mining to detect outliers in large amounts of highly multi-dimensional data. The multidimensional aspect of the data makes this task particularly challenging. This is because highly important and influential outliers can be completely hidden in one-dimensional views of the data, which renders ineffective one-dimensional outlier detection based on scanning one field (variable, attribute) at a time. Furthermore, classical statistical methods and most traditional data mining methods lack robustness toward outliers, and have very little power to detect outliers.

3.1.10.2 How to Overcome Data Skew?

Data skew can cause the algorithm to generate many false candidate itemsets. One way to overcome the data skew is the randomization 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.

The skewness for a normal distribution is zero, and any symmetric data should have skewness near zero. Negative values for the skewness indicate data that are skewed left and positive values for the skewness indicate data that are skewed right. By skewed left, it means that the left tail is long relative to the right tail. Similarly, skewed right means that the right tail is long relative to the left tail. Some measurements have a lower bound and are skewed right. For example, in reliability studies, failure times cannot be negative.

3.2 Proposed Solution

In this study, it is desired to be investigated that although the database scans are reduced to half by CMA algorithm, but do it also perform efficiently like the previous work in the field, i.e., the PARTITION algorithm. It means, it is aimed in this study to find out that whether the computational efficiency of the CMA algorithm is better than that of PARTITION algorithm or not.

PARTITION algorithm [8] was previously implemented on a Silicon Graphics Indy R4400SC workstation with a clock rate of 150 MHz and 32 Mbytes of main memory. The data resided on 1GB SCSI disk. Whereas CMA algorithm [13] was implemented on Pentium IV system with 256 MB main memory, 40 GB disk capacity and in an environment of Windows XP and MATLAB 7 and VB.NET.

In this study, Randomized PARTITION algorithm is implemented in the same environment as CMA for comparison purpose. The same synthetic database is used as was used for the implementation of CMA by Jamshaid et. al; (2007).

Following modules are the basic parts of this study:

- Implementation of Randomized PARTITION Algorithm
- Comparison of No. of Large Itemsets of Sequential PARTITIONS with No. of Large Itemsets of Randomized Partitions
- Comparison of previously implemented PARTITION Algorithm with Randomized PARTITION Algorithm
- Comparison of Randomized PARTITION Algorithm with CMA

In this study, the basic aim is to figure out the best work in the research area of Association Rule Mining. It is one of the most interesting research area in Data Mining, because it targets the problems of the retail industry directly, so the research should be made in such a way that most benefits could be gained within little time span, but the problem is that, association rule mining is done for searching the interesting patterns and associations among different data items. Apparently or through the use of some CASE tool, these data items might not seem associated with each other. At the same time, these data associations could not be over looked as these can cause a handsome increment in the sales of the retail store for example. So it is to be searched for these patterns in the entire data warehouse, containing the data of decades of the years. In this way, by analyzing the conceptual resources, these data associations can be generated and can be implemented for the good of the business. But the problem arises here that these association finding techniques, or the association rule algorithms, take a lot of time and resources for execution. So it requires a handsome financing for its execution, which every retailer cannot afford. So the aim must always be to minimize the resource and time consumption from these algorithms. For example, the AIS algorithm presented by R. Agarwal et al. (1993) [4] takes n+1 database scans for n iterations in the execution of the algorithm for the large itemsets generation. This requires larger RAM and disk space to accommodate the data in the RAM for processing. As the large disks are unavoidable in this scenario, so the efforts remain focus on minimizing the database scans taken by the algorithm. Then comes Partitioning technique presented by Savasere et al. (1995), which divides the entire large database in the number of partitions to find the large itemsets from each partition one after another. So the requirement for the larger RAM capacities was thus excluded.

The limitation with the PARTITION algorithm is that, although the authors have discussed the importance of randomization of the database to avoid the data skewness, but it is not mentioned in algorithm. Also in PARTITION algorithm hash structures are used which are much time consuming and requires more memory for storage.

3.2.1 Randomization

Randomization of the database means to randomize the database from its original sequential order. It means that, as the items purchased from a retail store are recorded in the sequence of their actual sale transactions, so are available for analysis purposes in the sequential order. There might come some severe conditions in which the sales of some particular items boost, which are not heavily bought items in the normal circumstances. By taking sequential partitions, we might get such items as large itemsets of that particular partition, whereas in other partitions these items do not qualify to be the large itemsets. So these large itemsets of one partition, produced as a result of abnormal conditions, are termed as Outliers, which results in data skewness.

To avoid such outliers, the database needs to be randomized. By randomization of the database, the entire database containing the data of decades of years, is shuffled so that the records containing some abnormal data is randomized all around the database, and hence eliminating the chance of data skewness.

3.2.3 PARTITION Algorithm

The idea behind PARTITION algorithm is as follows. Recall that the reason the database needs to be scanned multiple number of times is because the number of possible itemsets to be tested for support is exponentially large if it must be done in a single scan of the database. PARTITION algorithm accomplishes this in two scans of the database. In one scan it generates a set of all potentially large itemsets by scanning the database once. This set is a superset of all large itemsets, i.e., it, may contain false positives. But no false negatives are reported. During the second scan, counters for each of these itemsets are set up and their actual support is measured in one scan of the database. The algorithm executes in two phases. In the first phase, the PARTITION algorithm logically divides the database into a number of non-overlapping partitions. The partitions are considered one at a time and all large itemsets for that partition are generated. At the end of phase I, these large itemsets are merged to generate a set of all potential large itemsets. In phase II, the actual support for these itemsets is generated and the large itemsets are identified. 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 algorithm reads the entire database once during phase I and once during phase II.

3.2.2 CMA Algorithm

The CMA algorithm presented by Jamshaid et al; (2007) also divides the database into a number of partitions after the randomization of the database. The large itemsets creation and support count calculation, both are done in the single scan of the database in each partition. The globally large itemsets are then calculated from those locally large itemsets. This minimizes the database scans to the half, i.e., 50% database scans are decreased as compared to the previous work done in this field.

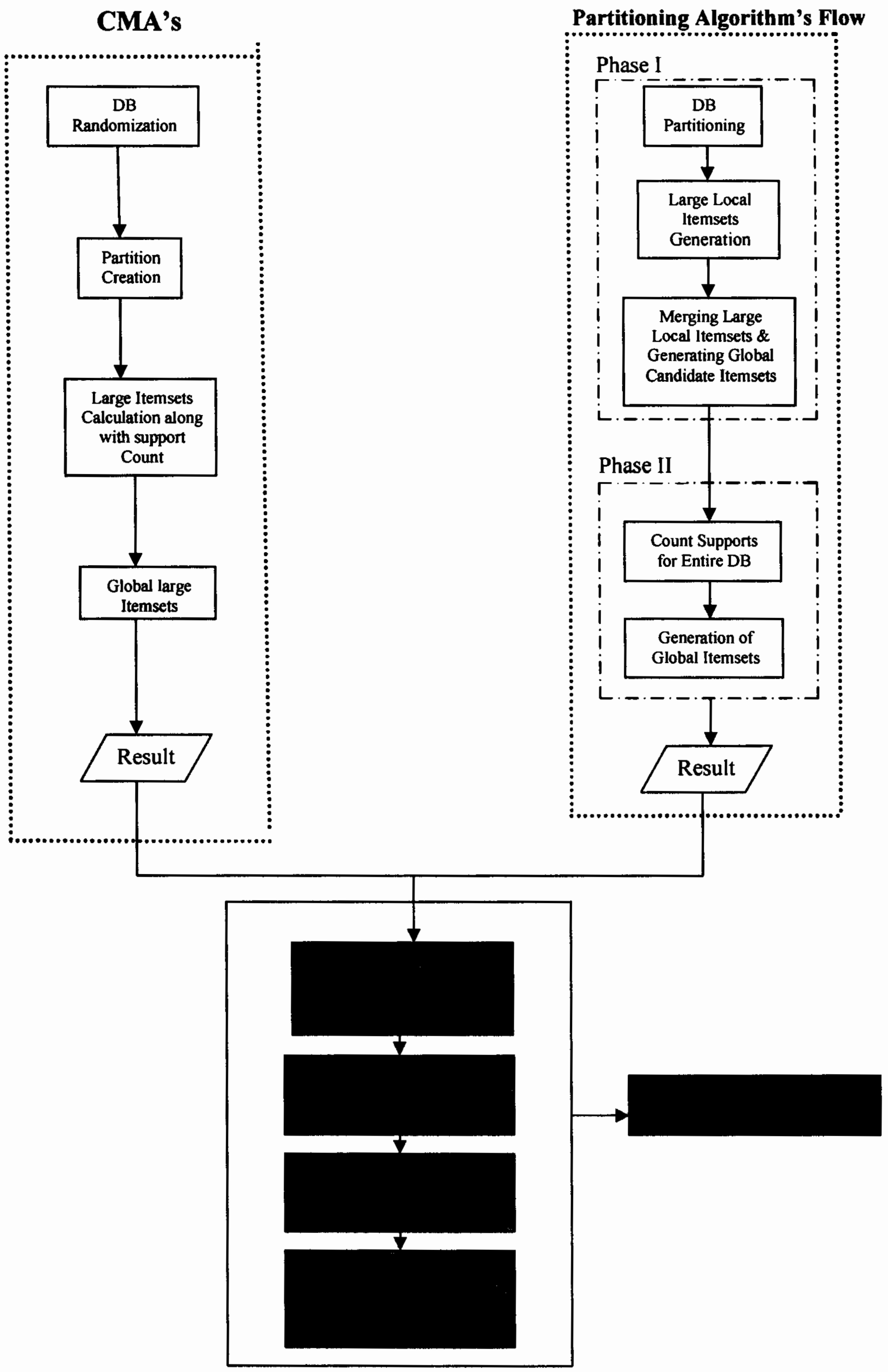

CMA's
DB Randomization
Partition Creation
Large Itemsets Calculation along with support Count
Global large Itemsets
Result
Partitioning Algorithm's Flow
Phase I
DB Partitioning
Large Local Itemsets Generation
Merging Large Local Itemsets & Generating Global Candidate Itemsets
Phase II
Count Supports for Entire DB
Generation of Global Itemsets
Result
[redacted]
[redacted]
[redacted]
[redacted]
[redacted]

Figure 1.1 Flow Charts of PARTITION, CMA, and Proposed Algorithms

Chapter 4

Research Methodology

4. Research Methodology

Association rule mining is a two-step process:

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

4.1 Architectural Diagram

Architectural diagram of this study is given below:

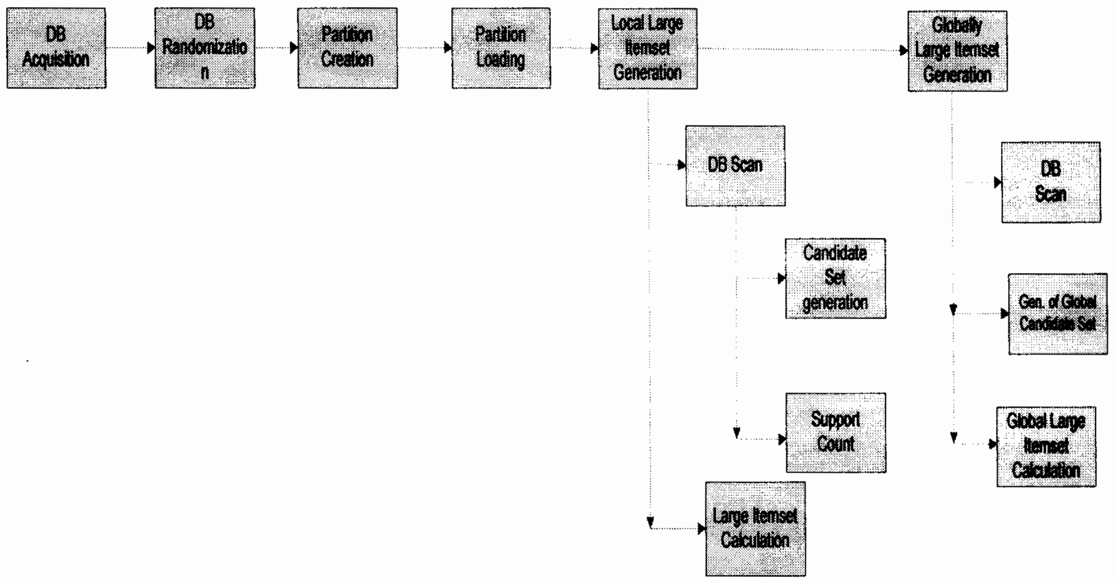

DB Acquisition
DB Randomization
Partition Creation
Partition Loading
Local Large Itemset Generation
Globally Large Itemset Generation
DB Scan
Candidate Set generation
Support Count
Large Itemset Calculation
DB Scan
Gen. of Global Candidate Set
Global Large Itemset Calculation

Figure 4.1 Architectural Diagram

4.2 Main Modules

The main modules are as follows:

- Database Acquisition
- Database Randomization
- Partition Creation
- Local Large Itemset Generation
- Global Large Itemset Generation

- Comparison of Randomized PARTITION Algorithm with Previously Implemented PARTITION Algorithm
- Comparison of Randomized PARTITION Algorithm with CMA Algorithm

4.2.1 Database Acquisition

Synthetic database is used for this study. This is the same database as was used by CMA algorithm. 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, in the form of literals. Instead of representing those transactions in the form of the actual items (like Bread, Milk, and Butter 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.

4.2.2 Database Randomization

As transactions in a database are stored as the transactions physically occur, so this means a page from the database is the snapshot of the transactions of some specific time period. For example, during winters it is observed that the sales of blankets are much higher than summer. So transactions occurred during winters will obviously demonstrate this observation. This is not recommended for analysis. In this study, the database is going to be randomized before analysis as sequential reading of database will render with the items which might be frequently purchased during some specific time period, but actually are not the frequently purchased items. To avoid such outliers, the database is randomized. By randomization of the database, the entire database containing the data of decades of years, is shuffled so that the records containing some abnormal data is randomized all around the database, and hence eliminating the chance of data skewness. 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.

4.2.3 Partition Creation

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. In other words partitioning is the technique of dividing the huge centralized database into a number of partitions in order to speed up the specified processing. 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. 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, logical partitions. The partition creation module stores its output in four different columns of a separate MS Excel sheet.

4.2.4 Local Large Itemset Generation

This module creates the frequently occurred itemsets of the loaded partition. All the itemsets, for which the condition **X. Support ≥ min_support** holds, are marked as large itemsets of the loaded partition. Major steps involved in the generation of local large itemsets are:

4.2.4.1 Database Scan

In this step, the database is scanned for the first time in the course of execution of the algorithm. Here, the database is the partition, just loaded into the memory. Database scan performs the following two major tasks:

- **Support Count**

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

- **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.

4.2.4.2 Large Items Computation

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 large 1-itemset. All the rest of the items are pruned away, hence minimizing the chance for the false candidates. The large items and their counts are then stored in a structure.

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. E.g. if items {A, B, D, E} is a set of large 1-itemset then candidate 2-itemset is {AB, AD, AE, BD, BE, DE}. Then their confidence is measured, and qualifying items are marked as large 2-itemsets. Suppose itemset {AB, BD, DE} is large 2-itemset. Then for the creation of large-3itemset, candidate 3-itemset will be generated as {ABD, ADE, BDE}. From this candidate 3-itemset, large 3-itemset will be generated and so on till k-itemset generation.

4.2.5 Global Large Itemset Generation

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

4.2.5.1 Global Candidate Set Generation

In this step the large itemsets of same lengths for each partition, are merged together to form global candidate itemsets of all lengths. If there are same large itemset in more than one partition then their counts are added, making sure that each large itemset is stored only once in each global candidate itemset. e.g. in partition 1 the large 1 itemset contains A=50, partition 2 contains A=55,D=70 as large 1-itemsets, partition 3 contains B=60 as large 1-itemset, partition 4 contains B=65, E=85. Then the global candidate large 1-itemset will be {A=105, B=125, D=70, E=85}. Similarly global candidate 2-itemset is generated by merging the large 2-itemsets of all partitions then global candidate 3-itemset is generated by merging large 3-itemsets of all partitions and so on till global candidate k-itemset generation.

4.2.5.2 Global Large Itemset Computation

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 $\geq$ 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 lengths of itemset, till globally large k-itemset generation.

4.2.6 Comparison of Large Itemsets of Sequential and Randomized Partitions

The large itemsets of sequential partitions are compared with large itemsets of randomized partitions by randomizing database again and again and creating large itemsets for different randomized partitions to see the effect of randomization on large itemsets.

4.2.7 Comparison of Randomized PARTITION with Previously Implemented PARTITION Algorithm

In this study, an improved version of PARTITION algorithm is implemented hence improving the efficiency of previously implemented PARTITION algorithm and minimizing the memory usage. In previous PARTITION algorithm the benefits of randomization of database were discussed in detail but the algorithm was not doing randomization of database. One more drawback was that previously implemented PARTITION algorithm was using hash trees. i.e. if item A is large 1-itemset of partition 1 then the TIDs of all transactions containing item A in partition 1 will be stored against A in a hash tree. Same process is done with all the large 1-itemsets of all other partitions and then for all lengths of itemsets of all partitions. This consumes much memory. Time efficiency of algorithm also reduces due to this large storage. To avoid this memory usage count structure is used in the improved version of PARTITION algorithm. i.e. instead of storing all the TIDs of each large itemsets their count is stored in a structure, reducing the memory usage and increasing time efficiency for better performance and efficient results.

In this study, Randomized PARTITION algorithm has been implemented. Logical non overlapping partitions are created with both random and non random data. Results of both randomized and non randomized partitions are compared to see the effect of data skew on both locally and globally large itemsets.

4.2.8 Comparison of Randomized PARTITION with CMA Algorithm

The efficiency of Randomized PARTITION algorithm is compared with CMA.
PARTITION algorithm was previously implemented in Silicon Graphics Indy R4400SC workstation with a clock rate of 150 MHz and 32 Mbytes of main memory. The data resided on 1GB SCSI disk.
In this study Randomized PARTITION algorithm has been implemented in the same environment as CMA except the partitioning of database that has been done in different tool (i.e. CMA used DOTNET while Randomized PARTITION algorithm is using MATLAB) for achievement of better performance and efficient results. This change in partitioning technique increases the time efficiency of algorithm as compared to CMA. In today's world, time is an important factor. Along with the need of accurate results, time efficiency matters a lot. It is important to have better results in less time and minimum cost. Synthetic database is used for this study which is the same database as was used by CMA.

4.3 Algorithm

```
//Read database sequentially
Read_Database
Function [P]=Create_Partitions (rand)
For x=1 to P
Begin
        Function Generate_Candidate_Itemsets Ci for $p_x$
        For i=1 to k
        Begin
                //Generate local large itemsets Li for pi from Ci and store item and its
                count in structure Si
                Function [Si]=Gen_Local_Large_Itemset Li
        End
End
//Merge local large itemsets to form global candidate itemsets Gi_C
```

```
Function [Gi_C]=Merge_large_itemsets(Li)
//Generate globally large itemsets Gi from Gi_C
Gi=Gen_Global_Large_itemset (Gi_C)
```

Algorithm for Database Partitioning

```
If rand=0
        Create logical sequential partitions
Else
        Randomize database
        Create logical non-overlapping randomized partitions
End
```

4.3.1 Notations Used

Notation	Definition
Px	Partitions (x=1 to 4)
Ci_L	Local Candidate itemset
Li	Local large itemset
Si	Structure for storing local large itemsets along with their counts in particular partition pi
Gj_C	Global candidate itemset containing local large itemsets alongwith their counts
Gi	Globally large itemsets

Chapter 5

Implementation

5. Implementation

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

Software selection is very important step in developing a computer based system. PARTITION algorithm was previously implemented in Silicon Graphics Indy R4400SC workstation with a clock rate of 150 MHz and 32 Mbytes of main memory. The data resided on 1GB SCSI disk.

As this is the comparative study so comparison could only be done if both algorithms run on same environment. For this purpose the Randomized PARTITION algorithm is implemented in the same environment as CMA to check the efficiency of both algorithms. MS Excel and MATLAB 7 were chosen for implementation. The only difference in environment of CMA and Randomized PARTITION algorithm was in the partitioning technique used for partitioning the database. CMA used VB.Net for partitioning while Randomized PARTITION algorithm has used MATLAB 7 for partitioning in order to achieve efficient results.

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 its built-in workbooks for storage of huge data in any or all of the worksheets of workbook, as well as provides the facility to add more worksheet in a workbook to fulfill the requirement of data. Then the mathematical, statistical or accounting functions can be performed. The dataset that was used in this study for finding large itemsets is in two excel sheets, as it consists of 1 lac records. After randomizing the data and creating partitions the final data is stored in sheet 3, this data set is then provided to MATLAB for large itemset creation.

5.2 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. 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.
MATLAB works best with arrays so it makes it much easier to deal with database records from Excel sheet.

5.3 Working of Randomized PARTITION Algorithm

The number of records to be read is taken from user. Then local minimum support and global minimum support is taken from user in percentage and actual local and global minimum support is calculated on the basis of number of records to be read. For example if the user wants 100,000 records to be read and gives the local minimum support as 50% and global minimum support as 50%. The database is read in sequential order first. After this actual local and global minimum support is calculated. That makes local minimum support as 12500 and global minimum support as 50000. Then logical non overlapping randomized or non randomized partitions are created depending upon user's choice. Partition size is chosen in such a way that for a specific time a whole partition can easily reside in memory. Partition size should not be very small or very large, as small partitions are negatively affected by data skew and in large partitions for intermediate results processing buffer requirements can exceed the available space, so risk is involved in both cases. In this study, the number of partitions is fixed i.e. four partitions are created. Then candidate large 1 itemsets are generated for all partitions. Then local large 1 itemsets are generated containing items having their support greater than user defined minimum support within each partition and the candidate itemsets whose minimum support is less than user defined minimum support are pruned away. i.e. Itemsets whose transations are greater than 12500 within an individual partition are considered as local large itemsets. Here a count structure is used for large itemsets rather than storing TIDs in a Hash tree. Storage of TIDs in hash tree is wastage of time and memory that's why count structure is used for finding large itemsets directly. Large 1 itemsets of all partitions are then merged and stored in global candidate 1 itemset along with their counts for the generation of globally large itemsets. The counts of same itemsets are added. From large 1 itemsets, candidate large 2 itemsets are generated from which local large 2 itemsets are generated for each partition. The process continues upto locally large k itemset generation. Finally globally large itemsets are generated and global candidate itemsets having minimum support less than user defined minimum support are pruned away. The database used for this study is the one that was used by CMA(Centralized Mining of Association Rules) presented by Jamshaid et. al (2007) for comparison purpose. The database contains 5 items and 100,000 records.

The code and its description is given below.

5.3.1 Read Database

This function reads the entire database sequentially and returns the transactions in an array T. Code of this function is as follows:

```
function [T]=read_database(no_of_records)
global T
T='';
% Read data from excel
if (no_of_records>0) & (no_of_records<=65536)
   range=strcat('A1:B',num2str(no_of_records));
   [n,T]=xlsread('dataset1.xls',1,range);
else
   no_of_records=no_of_records-65536;
   range=strcat('A1:B',num2str(no_of_records));
   [n,T1]=xlsread('Dataset1.xls',1,'A1:B65536');
   [n,T2]=xlsread('Dataset1.xls',2,range);
```

```
  T=vertcat(T1,T2);
End
```

5.3.2 Create Partitions

This function takes rand as argument. If rand=0 then sequential, non overlapping partitions are created otherwise T is first randomized and then logical non overlapping partitions are created and an array of partition P is returned. Code of this function is given as follows:

```
function [ partitions ]=create_partitions(rand_partition)
global T
global partitions
if rand_partition==0
%   data not randomized
   B=T;
else
%   randomize data
   B=random_array(T);
end
x=max(size(B));
partition_size=x/4;
%Create Partitions
range=strcat('A1:B',num2str(partition_size));
xlswrite('Dataset2.xls',partitions{1,1},1,range);
range=strcat('C1:D',num2str(partition_size));
xlswrite('Dataset2.xls',partitions{2,1},1,range);
range=strcat('E1:F',num2str(partition_size));
xlswrite('Dataset2.xls',partitions{3,1},1,range);
range=strcat('G1:H',num2str(partition_size));
xlswrite('Dataset2.xls',partitions{4,1},1,range);
```

5.3.3 Calculate User Defined Local min_sup

```
function txt_minsup_Callback(hObject, eventdata, handles)
global no_of_records;
global min_sup;

user_entry = str2double(get(hObject,'string'));
if isnan(user_entry) |(user_entry<1) |(user_entry>100)
   errordlg('Please enter a number b/w 1 and 100','Bad Input','modal')
   set(handles.txt_minsup, 'String' ,'');
else
     min_sup=(user_entry/100)*(no_of_records/4);
end
```

5.3.4 Calculate User Defined Global min_sup

```
function txt_g_minsup_Callback(hObject, eventdata, handles)
global no_of_records;
global g_min_sup;

set(handles.btn_read, 'Enable' ,'off');
user_entry = str2double(get(hObject,'string'));
if isnan(user_entry) |(user_entry<1) |(user_entry>100)
   errordlg('Please enter a number b/w 1 and 100','Bad Input','modal')
   set(handles.txt_g_minsup, 'String' ,'');
else
     g_min_sup=(user_entry/100)*(no_of_records);
     set(handles.btn_read, 'Enable' ,'on');
end
```

5.3.5 Finding Locally Large 1-itemset

This function takes partition as argument and creates large 1-itemset for that partition. Code for this function is given as follows:

```
function [ struct_large1 ] = large_1itemset( p1 )
        s_cand_item=struct('it',{'A','B','C','D','E'});
        struct_large1=find_item(p1,s_cand_item,1);
```

5.3.6 Finding Locally Large 2-Itemset

This function takes the partition, and large-1 itemset as argument and creates large 2-itemset for that partition. Code for this function is given as follows:

```
function [ struct_large2 ] = large_2itemset( p1,large_1 )
        %creates large 2-itemset
        s_cand_item=struct('it',{});
        struct_large2=struct('it',{},'cnt',{});
        k=0;
        lsize =max(size(large_1));
        if (lsize >1)
                % Create candidate itemset
                for i=1: lsize -1
                        for j=i+1:indx
                                l_itemset=strcat(large_1(i).it,',', large_1(j).it);
                                k=k+1;
                                s_cand_item(k).it=l_itemset;
                                l_itemset='';
                        end
                end
                %Create large 2 itemset
                struct_large2=find_item(p1,s_cand_item,2);
        end
```

5.3.7 Finding Locally Large 3-Itemset

This function takes partition and large 2-itemset as an argument and creates large 3-itemset for that partition. Code for this function is given below:

```
function [ struct_large3 ] = large_3itemset( p1,large_2 )
        %creates large 3-itemset
        s_cand_3item=struct('it',{});
        struct_large3=struct('it',{},'cnt',{});
        n=0;
        lsize=max(size(large_2));
        if lsize>1
                chkflag=1;
                for i=1:lsize-1
                        for j=i+1:lsize
                                str1=strcat(large_2(i).it,large_2(j).it);
                                str1=sort(str1);
                                str1=strrep(str1,',','');
                                for k=1:max(size(str1))-1
                                        if (str1(k)==str1(k+1))
                                                strtemp=str1(k);
                                                str1=strrep(str1,str1(k),'');
                                                str1=strcat(str1,strtemp);
                                                str1=sort(str1);
                                                p=1;
                                                break;
                                        else
                                                p=0;
                                                continue;
                                        end
                                end
                                if (p==0)
                                        str2=strcat(str1(1),',',str1(2),',',str1(3));
                                        chkmerge=merge_in_global(s_cand_3item,str2);
                                        if chkmerge ==1
                                        n=n+1;
                                        s_cand_3item(n).it=str2;
                                end
                                str2=strcat(str1(1),',',str1(2),',',str1(4));
                                chkmerge=merge_in_global(s_cand_3item,str2);
                                if chkmerge ==1
                                        n=n+1;
                                        s_cand_3item(n).it=str2;
                                end
                                str2=strcat(str1(1),',',str1(3),',',str1(4));
                                chkmerge=merge_in_global(s_cand_3item,str2);
                                if chkmerge ==1
                                        n=n+1;
                                        s_cand_3item(n).it=str2;
                                end
                        else
                                str2=strcat(str1(1),',',str1(2),',',str1(3));
                                chkmerge=merge_in_global(s_cand_3item,str2);
                                if chkmerge ==1
                                        n=n+1;
                                        s_cand_3item(n).it=str2;
                                end
                        end
                end
        end
        struct_large3=find_item(p1,s_cand_3item,3);
end
```

5.3.8 Finding Locally Large 4-Itemset

This function takes partition and large 3-itemset as an argument and creates large 4-itemset. Code for this function is given below:

```
function [ struct_large4 ] = large_4itemset( p1,large_3 )
        % creates large 4-itemset
        s_cand_4item=struct('it',{});
        struct_large4=struct('it',{},'cnt',{});
        n=0;
        lsize=max(size(large_3));
        if lsize>1
                chkflag=1;
                for i=1:lsize-1
                        for j=i+1:lsize
                                str4=strcat(large_3(i).it,large_3(j).it);
                                str4=sort(str4);
                                str4=strrep(str4,',','');
                                k=1;
                                while k<max(size(str4))
                                        if (str4(k)==str4(k+1))
                                                strtemp=str4(k);
                                                str4=strrep(str4,str4(k),'');
                                                str4=strcat(str4,strtemp);
                                                str4=sort(str4);
                                        end
                                        k=k+1;
                                end
                                if (max(size(str4))>4)
                                        str2=strcat(str4(1),',',str4(2),',',str4(3),',',str4(4));
                                        chkmerge=merge_in_global(s_cand_4item,str2);
                                        if chkmerge ==1
                                                n=n+1;
                                                s_cand_4item(n).it=str2;
                                        end
```

```
                    str2=strcat(str4(1),',',str4(2),',',str4(3),',',str4(5));
                    chkmerge=merge_in_global(s_cand_4item,str2);
                    if chkmerge ==1
                        n=n+1;
                        s_cand_4item(n).it=str2;
                    end
                    str2=strcat(str4(1),',',str4(2),',',str4(4),',',str4(5));
                    chkmerge=merge_in_global(s_cand_4item,str2);
                    if chkmerge ==1
                        n=n+1;
                        s_cand_4item(n).it=str2;
                    end
                    str2=strcat(str4(1),',',str4(3),',',str4(4),',',str4(5));
                    chkmerge=merge_in_global(s_cand_4item,str2);
                    if chkmerge ==1
                        n=n+1;
                        s_cand_4item(n).it=str2;
                    end
                    str2=strcat(str4(2),',',str4(3),',',str4(4),',',str4(5));
                    chkmerge=merge_in_global(s_cand_4item,str2);
                    if chkmerge ==1
                        n=n+1;
                        s_cand_4item(n).it=str2;
                    end
                elseif (max(size(str4))==4)
                    str2=strcat(str4(1),',',str4(2),',',str4(3),',',str4(4));
                    chkmerge=merge_in_global(s_cand_4item,str2);
                    if chkmerge ==1
                        n=n+1;
                        s_cand_4item(n).it=str2;
                    end
                end
            end
        end
```

```
            struct_large4=find_item(p1,s_cand_4item,4);
        end
```

5.3.9 Finding Locally Large k-Itemset

This function takes partition and large 4-itemset as an argument and creates large k-itemset. Code for this function is given below:

```
function [ struct_largek ] = large_kitemset( T,large_4 )
% creates large k-itemset
s_cand_kitem=struct('it',{});
struct_largek=struct('it',{},'cnt',{});

lsize=max(size(large_4));
if lsize>1
  chkflag=1;
  for i=1:lsize-1
    for j=i+1:lsize
      str5=strcat(large_4(i).it,large_4(j).it);
      str5=strrep(str5,',','');
      k=1;
       while k<max(size(str5))
         if (str5(k)==str5(k+1))
           strtemp=str5(k);
           str5=strrep(str5,str5(k),'');
           str5=strcat(str5,strtemp);
           str5=sort(str5);
         end
         k=k+1;
       end
      if (max(size(str5))==5)
         str5=strcat(str5(1),',',str5(2),',',str5(3),',',str5(4),',',str5(5));
         s_cand_kitem(1).it=str5;
      end
    end
  end
```

```
        struct_largek=find_item(T,s_cand_kitem,5);
end
```

5.3.10 Finding Global Large Itemset

This function is used to create globally large itemsets. It takes global candidate itemset as an argument and creates globally large itemset. The global candidate itemset was created by merging the large itemsets of same lengths of all partitions.

```
function [ g_large ] = global_itemsets(g_cand)
% generates globally large itemsets
global g_min_sup;

g_large=struct('it',{});
n=0;
for i=1:max(size(g_cand))
    if g_cand(i).cnt>g_min_sup
        n=n+1;
        g_large(n).it=g_cand(i).it;
    end
end
```

5.3.11 Call_Back Function of Sequential Button

This function calls the function to create sequential partitions and show them in list boxes on the form. Code for the call back function of Sequential button is as follows:

```
function btn_sequential_Callback(hObject, eventdata, handles)
set(handles.txtStatus, 'String' ,'Creating non-randomized partitions, please wait...');
tic;
partitions=create_partitions(0);
partition1=partitions{1,1};
set(handles.lst_p1, 'String' ,strcat((partition1(:,1)),' : ',(partition1(:,2))));
partition2=partitions{2,1};
set(handles.lst_p2, 'String' ,strcat((partition2(:,1)),' : ',(partition2(:,2))));
partition3=partitions{3,1};
set(handles.lst_p3, 'String' ,strcat((partition3(:,1)),' : ',(partition3(:,2))));
partition4=partitions{4,1};
set(handles.lst_p4, 'String' ,strcat((partition4(:,1)),' : ',(partition4(:,2))));
t=toc;
set(handles.txtStatus, 'String' ,'Sequential partitions created!    Time taken to create
sequential partitions:');
set(handles.txt_time, 'String' ,t);
set(handles.txt_sec, 'String' ,'Secs');
set(handles.btn_large, 'Enable' ,'on');
```

5.3.12 Call_Back Function of Randomized Button

This function calls the function to create randomized partitions and show them in list boxes on the form. Code for the call back function of Randomized button is as follows:

```
set(handles.txtStatus, 'String' ,'Creating randomized partitions...');
tic
partitions=create_partitions(1);
partition1=partitions{1,1};
set(handles.lst_p1, 'String' ,strcat((partition1(:,1)),' : ',(partition1(:,2))));
partition2=partitions{2,1};
set(handles.lst_p2, 'String' ,strcat((partition2(:,1)),' : ',(partition2(:,2))));
partition3=partitions{3,1};
```

```
set(handles.lst_p3, 'String' ,strcat((partition3(:,1)),' : ',(partition3(:,2))));
partition4=partitions{4,1};
set(handles.lst_p4, 'String' ,strcat((partition4(:,1)),' : ',(partition4(:,2))));
t=toc;
set(handles.txtStatus, 'String' ,'Randomized partitions created!   Time taken to create
randomized partitions:');
set(handles.txt_time, 'String' ,t);
set(handles.txt_sec, 'String' ,'Secs');
set(handles.btn_large, 'Enable' ,'on');
```

5.3.13 Call_Back Function of Create Large Itemsets Button

When this button is pressed the local and global large itemsets are created and shown in list boxes on the form.

function btn_large_Callback(hObject, eventdata, handles)

```
set(handles.txtStatus, 'String' ,'Creating large itemsets, Please wait...');
global partitions;
% global no_of_records;
s_gcand_large1=struct('it',{},'cnt',{});
s_gcand_large2=struct('it',{},'cnt',{});
s_gcand_large3=struct('it',{},'cnt',{});
s_gcand_large4=struct('it',{},'cnt',{});
s_gcand_largek=struct('it',{},'cnt',{});

tic
% Local Large 1 itemset for partition 1
p1=partitions{1,1};
struct_large1_p1=large_1itemset( p1 );
if isempty(struct_large1_p1)
   s1='No large1 Itemsets';
else
   s_gcand_large1=struct_large1_p1;
   s1=rmfield(struct_large1_p1,'cnt');
   s1=struct2cell(s1);
end
```

```
set(handles.lst_l1_p1, 'String' ,s1);
s1='';

% Local Large 2 itemset for partition 1
struct_large2_p1=large_2itemset( p1,struct_large1_p1 );
if isempty(struct_large2_p1)
   s2='No large2 Itemsets';
else
   s_gcand_large2=struct_large2_p1;
   s2=rmfield(struct_large2_p1,'cnt');
   s2=struct2cell(s2);
end
set(handles.lst_l2_p1, 'String' ,s2);
s2='';

% Local Large 3 itemset for partition 1
struct_large3_p1=large_3itemset( p1,struct_large2_p1 );
if isempty(struct_large3_p1)
   s3='No large3 Itemsets';
else

   s_gcand_large3=struct_large3_p1;
   s3=rmfield(struct_large3_p1,'cnt');
   s3=struct2cell(s3);
end
set(handles.lst_l3_p1, 'String' ,s3);
s3='';

% Local Large 4 itemset for partition 1
struct_large4_p1=large_4itemset( p1,struct_large3_p1 );
if isempty(struct_large4_p1)
   s4='No large4 Itemsets';
else
   s_gcand_large4=struct_large4_p1;
```

```
  s4=rmfield(struct_large4_p1,'cnt');
  s4=struct2cell(s4);
end
set(handles.lst_l4_p1, 'String' ,s4);
s4='';

% Local Large k itemset for partition 1
struct_largek_p1=large_kitemset( p1,struct_large4_p1 );
if isempty(struct_largek_p1)
  sk='No large k Itemsets';
else
  s_gcand_largek=struct_largek_p1;
  sk=rmfield(struct_largek_p1,'cnt');
  sk=struct2cell(sk);
end
set(handles.lst_lk_p1, 'String' ,sk);
sk='';

% Local Large itemset for partition 2
p2=partitions{2,1};

struct_large1_p2=large_1itemset( p2 );
if isempty(struct_large1_p2)
  s1='No large1 Itemsets';
else
  s_gcand_large1=global_candidate(s_gcand_large1,struct_large1_p2);
  s1=rmfield(struct_large1_p2,'cnt');
  s1=struct2cell(s1);
end
set(handles.lst_l1_p2, 'String' ,s1);
s1='';

struct_large2_p2=large_2itemset( p2,struct_large1_p2 );
if isempty(struct_large2_p2)
```

```
   s2='No large2 Itemsets';
else
   s_gcand_large2=global_candidate(s_gcand_large2,struct_large2_p2);
   s2=rmfield(struct_large2_p2,'cnt');
   s2=struct2cell(s2);
end
set(handles.lst_l2_p2, 'String' ,s2);
s2='';

struct_large3_p2=large_3itemset( p2,struct_large2_p2 );
if isempty(struct_large3_p2)
   s3='No large3 Itemsets';
else

   s_gcand_large3=global_candidate(s_gcand_large3,struct_large3_p2);
   s3=rmfield(struct_large3_p2,'cnt');
   s3=struct2cell(s3);
end
set(handles.lst_l3_p2, 'String' ,s3);
s3='';

struct_large4_p2=large_4itemset( p2,struct_large3_p2 );
if isempty(struct_large4_p2)
   s4='No large4 Itemsets';
else
   s_gcand_large4=global_candidate(s_gcand_large4,struct_large4_p2);
   s4=rmfield(struct_large4_p2,'cnt');
   s4=struct2cell(s4);
end
set(handles.lst_l4_p2, 'String' ,s4);
s4='';

% Local Large k itemset for partition 2
struct_largek_p2=large_kitemset( p2,struct_large4_p2 );
```

```
if isempty(struct_largek_p2)
  sk='No large k Itemsets';
else
  s_gcand_largek=global_candidate(s_gcand_largek,struct_largek_p2);
  sk=rmfield(struct_largek_p2,'cnt');
  sk=struct2cell(sk);
end
set(handles.lst_lk_p2, 'String' ,sk);
sk='';

% Local Large itemset for partition 3
p3=partitions{3,1};

struct_large1_p3=large_1itemset( p3 );
if isempty(struct_large1_p3)
  s1='No large1 Itemsets';
else
  s_gcand_large1=global_candidate(s_gcand_large1,struct_large1_p3);
  s1=rmfield(struct_large1_p3,'cnt');
  s1=struct2cell(s1);
end
set(handles.lst_l1_p3, 'String' ,s1);
s1='';

struct_large2_p3=large_2itemset( p3,struct_large1_p3 );
if isempty(struct_large2_p3)
  s2='No large2 Itemsets';
else
  s_gcand_large2=global_candidate(s_gcand_large2,struct_large2_p3);
  s2=rmfield(struct_large2_p3,'cnt');
  s2=struct2cell(s2);
end
set(handles.lst_l2_p3, 'String' ,s2);
 s2='';
```

```
struct_large3_p3=large_3itemset( p3,struct_large2_p3 );
if isempty(struct_large3_p3)
   s3='No large3 Itemsets';
else
   s_gcand_large3=global_candidate(s_gcand_large3,struct_large3_p3);
   s3=rmfield(struct_large3_p3,'cnt');
   s3=struct2cell(s3);
end
set(handles.lst_l3_p3, 'String' ,s3);
s3='';

struct_large4_p3=large_4itemset( p3,struct_large3_p3 );
if isempty(struct_large4_p3)
   s4='No large4 Itemsets';
else
   s_gcand_large4=global_candidate(s_gcand_large4,struct_large4_p3)
   s4=rmfield(struct_large4_p3,'cnt');
   s4=struct2cell(s4);
end
set(handles.lst_l4_p3, 'String' ,s4);
s4='';

% Local Large k itemset for partition 3
struct_largek_p3=large_kitemset( p3,struct_large4_p3 );
if isempty(struct_largek_p3)
   sk='No large k Itemsets';
else
   s_gcand_largek=global_candidate(s_gcand_largek,struct_largek_p3);
   sk=rmfield(struct_largek_p3,'cnt');
   sk=struct2cell(sk);
end
set(handles.lst_lk_p3, 'String' ,sk);
sk='';
```

```
% Local Large itemset for partition 4
p4=partitions{4,1};

struct_large1_p4=large_1itemset( p4 );
if isempty(struct_large1_p4)
   s1='No large1 Itemsets';
else
   s_gcand_large1=global_candidate(s_gcand_large1,struct_large1_p4);
   s1=rmfield(struct_large1_p4,'cnt');
   s1=struct2cell(s1);
end
   set(handles.lst_l1_p4, 'String' ,s1);
s1='';

struct_large2_p4=large_2itemset( p4,struct_large1_p4 );
if isempty(struct_large2_p4)
   s2='No large2 Itemsets';
else
   s_gcand_large2=global_candidate(s_gcand_large2,struct_large2_p4);
   s2=rmfield(struct_large2_p4,'cnt');
   s2=struct2cell(s2);
end
set(handles.lst_l2_p4, 'String' ,s2);
s2='';

struct_large3_p4=large_3itemset( p4,struct_large2_p4 );
if isempty(struct_large3_p4)
   s3='No large3 Itemsets';
else
   s_gcand_large3=global_candidate(s_gcand_large3,struct_large3_p4);
   s3=rmfield(struct_large3_p4,'cnt');
   s3=struct2cell(s3);
end
set(handles.lst_l3_p4, 'String' ,s3);
```

```
s3='';

struct_large4_p4=large_4itemset( p4,struct_large3_p4 );
if isempty(struct_large4_p4)
   s4='No large4 Itemsets';
else
   s_gcand_large4=global_candidate(s_gcand_large4,struct_large4_p4);
   s4=rmfield(struct_large4_p4,'cnt');
   s4=struct2cell(s4);
end
set(handles.lst_l4_p4, 'String' ,s4);
s4='';

% Local Large k itemset for partition 4
struct_largek_p4=large_kitemset( p4,struct_large4_p4 );
if isempty(struct_largek_p4)
   sk='No large k Itemsets';
else
   s_gcand_largek=global_candidate(s_gcand_largek,struct_largek_p4);
   sk=rmfield(struct_largek_p4,'cnt');
   sk=struct2cell(sk);
end
set(handles.lst_lk_p4, 'String' ,sk);
sk='';

%Adding local large itemsets to make global large itemset

g_large=global_itemsets(s_gcand_large1);
if isempty(g_large)
   g_large='No Global Large1 Itemsets';
else
   g_large=struct2cell(g_large);
end
set(handles.lst_g1, 'String',g_large);
```

```
g_large='';

g_large=global_itemsets(s_gcand_large2);
if isempty(g_large)
   g_large='No Global Large2 Itemsets';
else
   g_large=struct2cell(g_large);
end
set(handles.lst_g2, 'String',g_large);
g_large='';

g_large=global_itemsets(s_gcand_large3);
if isempty(g_large)
   g_large='No Global Large3 Itemsets';
else
   g_large=struct2cell(g_large);
end
set(handles.lst_g3, 'String',g_large);
g_large='';

g_large=global_itemsets(s_gcand_large4);
if isempty(g_large)
   g_large='No Global Large4 Itemsets';
else
   g_large=struct2cell(g_large);
end

set(handles.lst_g4, 'String',g_large);
g_large='';

g_large=global_itemsets(s_gcand_largek);
if isempty(g_large)
   g_large='No Global Large k Itemsets';
else
```

```
  g_large=struct2cell(g_large);
end
set(handles.lst_gk, 'String',g_large);
g_large='';

t=toc;
set(handles.txtStatus, 'String' ,'Large itemsets created!   Time taken to create large
itemsets:');
set(handles.txt_time, 'String' ,t);
set(handles.txt_sec, 'String' ,'Secs');
set(handles.btn_large, 'Enable' ,'off');
```

Chapter 6

Results

6. Results

To illustrate the efficiency of Randomized PARTITION algorithm different experiments were performed. First of all the working of Randomized PARTITION algorithm is explained by an example and then number of large itemsets of randomized and non randomized partitions were compared. After that the number of large itemsets with varying size of data was compared. Then Randomized PARTITION algorithm was compared with previously implemented PARTITION algorithm and in the end the efficiency comparison of Randomized PARTITION algorithm with CMA was done.

6.1 Explanation of Randomized PARTITION Algorithm with Example

Performance of Randomized PARTITION algorithm can be best demonstrated by the case study. For testing the algorithm, many experiments were performed by taking both random and non random data. At initial stage the database is read sequentially. And its local and global itemsets are calculated. Then the database is randomized and various results are observed after doing randomization again and again. In the current example after randomization the algorithm creates four non_overlapping partitions (P_1, P_2, P_3, P_4) each containing 25 records. Total number of items is 5. After partition creation, the Randomized PARTITION algorithm is applied in such away that it finds local candidate itemsets for the generation of locally large1- itemsets in each partition. Then candidate itemset for locally large 2-itemset is generated from which locally large 2-itemset is generated for each partition. This process continues unless up to k large itemsets are found. The algorithm stores counts of the locally large itemsets in a structure rather using a hash tree for this process. This change reduces the high memory usage as compared to previously implemented PARTITION algorithm. Then global candidate itemsets are found and finally the globally large itemsets are generated from the candidate sets. Table 6.6 shows the detailed results. In this example it calculated up to 2-large itemsets. Many experiments have been done with different datasets in which algorithm calculates 3-large itemsets, 4-large itemsets and up to k item sets.

The dataset used for this study is shown in table 6.1.1

PARTITIONS			
P_1	P_2	P_3	P_4
T_1: A,B,C	T_{26}: A,D,E	T_{51}: A,E	T_{76}: A,C,D,E
T_2: A,B,D	T_{27}: A,B,C,D,E	T_{52}: A,B,E	T_{77}: B,C,E
T_3: B,D,E	T_{28}: C,D,E	T_{53}: B,D,E	T_{78}: C,D,E
T_4: A,B,E	T_{29}: B,C,D,E	T_{54}: B,C,E	T_{79}: B,C,D,E
T_5: A,C,D	T_{30}: A,D,E	T_{55}: C,D	T_{80}: A,D,E
T_6: A,C,E	T_{31}: A,B,E	T_{56}: A,D,E	T_{81}: C,D,E
T_7: A,C,D	T_{32}: D,E	T_{57}: A,D,E	T_{82}: B,D,E
T_8: B,D,E	T_{33}: A	T_{58}: A,C	T_{83}: A,B,C,E
T_9: C,D	T_{34}: C,D	T_{59}: A,B,C,E	T_{84}: B,C,E
T_{10}: C,D,E	T_{35}: C,D,E	T_{60}: A,B,C,D	T_{85}: B,C,D
T_{11}: A,C,D	T_{36}: B,C	T_{61}: A,C,D	T_{86}: C,D,E
T_{12}: A,B,E	T_{37}: A,C	T_{62}: A,C,D,E	T_{87}: B,E
T_{13}: B,D,E	T_{38}: A,C,D,E	T_{63}: D,E	T_{88}: A,B,E
T_{14}: D,E	T_{39}: D,E	T_{64}: C,D,E	T_{89}: A,B,C
T_{15}: C,D,E	T_{40}: B,D,E	T_{65}: A,B,D	T_{90}: A,C,E
T_{16}: A,D,E	T_{41}: A,B	T_{66}: B,D,E	T_{91}: B,C,E
T_{17}: A,B,C,D	T_{42}: A,C	T_{67}: C,D,E	T_{92}: B,D,E
T_{18}: B,C,E	T_{43}: A,B,C	T_{68}: A,D,E	T_{93}: A,C,E
T_{19}: A,C,E	T_{44}: B,C,D	T_{69}: B,D,E	T_{94}: B,C,E
T_{20}: B,C,D	T_{45}: D,E	T_{70}: A,C	T_{95}: D,E
T_{21}: A,B,D,E	T_{46}: B,C,D,E	T_{71}: A,B,C	T_{96}: C,E
T_{22}: C,D,E	T_{47}: C,D,E	T_{72}: A,C,D,E	T_{97}: B,D,E
T_{23}: B,E	T_{48}: D,E	T_{73}: B,C,D,E	T_{98}: A,B,D,E
T_{24}: A,C,D	T_{49}: C,D	T_{74}: A,D,E	T_{99}: D,E
T_{25}: B,D,E	T_{50}: A,D	T_{75}: B,C,D,E	T_{100}: A,C,D,E

Table 6.1.1 Sequential Dataset

The above dataset was randomized as shown in table 6.1.2 and then was used for the case study.

PARTITIONS			
P_1	P_2	P_3	P_4
T_{24}: A,C,D	T_{95}: D,E	T_{70}: A,C	T_{81}: A,C,D
T_{92}: B,D,E	T_{53}: B,D,E	T_{54}: B,C,E	T_{23}: B,E
T_{31}: A,B,E	T_{49}: C,D	T_{77}: B,C,E	T_{65}: A,B,D
T_8: B,D,E	T_{72}: A,C,D,E	T_{74}: A,D,E	T_9: C,D
T_{42}: A,C	T_{69}: B,D,E	T_{60}: A,B,C,D	T_{63}: D,E
T_{21}: A,B,D,E	T_{67}: C,D,E	T_{28}: C,D,E	T_{45}: D,E
T_{99}: D,E	T_{22}: C,D,E	T_3: B,D,E	T_{41}: A,B
T_{91}: B,C,E	T_{75}: B,C,D,E	T_{11}: A,C,D	T_{73}: B,C,D,E
T_{25}: B,D,E	T_{57}: A,D,E	T_{80}: A,D,E	T_{37}: A,C
T_{55}: C,D	T_{44}: B.C.D	T_{64}: C,D,E	T_{59}: A,B,C.E
T_{85}: B,C,D	T_{16}: A,D,E	T_{66}: B,D,E	T_{58}: A,C
T_{15}: C,D,E	T_{19}: A,C,E	T_{96}: C,E	T_{89}: A,B,C
T_{50}: A,D	T_{36}: B,C	T_{40}: B,D,E	T_{84}: B,C,E
T_{51}: A,E	T_{48}: D,E	T_{43}: A,B,C	T_5: A,C,D
T_{27}: A,B,C,D,E	T_{10}: C,D,E	T_{52}: A,B,E	T_{20}: B,C,D
T_{30}: A,D,E	T_{78}: C,D,E	T_{79}: B,C,D,E	T_{93}: A,C,E
T_{94}: B,C,E	T_{33}: A	T_{56}: A,D,E	T_{62}: A,C,D,E
T_{39}: D,E	T_7: A,C,D	T_{76}: A,C,D,E	T_{18}: B,C,E
T_{26}: A,D,E	T_{35}: C,D,E	T_{47}: C,D,E	T_{13}: B,D,E
T_2: A,B,D	T_{98}: A,B,D,E	T_{71}: A,B,C	T_{34}: C,D
T_{88}: A,B,E	T_4: A,B,E	T_{90}: A,C,E	T_{17}: A,B,C,D
T_{87}: B,E	T_{61}: A,C,D	T_{14}: D,E	T_1: A,B,C
T_{29}: B,C,D,E	T_{46}: B.C.D.E	T_{32}: D.E	T_{82}: B,D,E
T_{97}: B,D,E	T_{83}: A,B,C,E	T_6: A,B,C	T_{86}: C,D,E
T_{68}: A,D,E	T_{38}: A,C,D,E	T_{12}: A,B,E	T_{100}: A,C,D.E

Table 6.1.2 Randomized Dataset

The number of local and global large itemsets generated from above randomized data are given below in table 6.1.3

LOCAL LARGE ITEMSETS (L_I)				GLOBAL LARGE ITEMSET (G_I)
P_1 Item=Count	P_2 Item=Count	P_3 Item=Count	P_4 Item=Count	G_i Item=Count
L_1:B=14, D=18, E=19 L_2:* L_3:* L_4:* L_5:*	L_1:C=16, D=19, E=19 L_2:DE=17 L_3:* L_4:* L_5:*	L_1:C=15,D=15, E=20 L_2:* L_3:* L_4:* L_5:*	L_1:C=18,D=15,E=14 L_2:* L_3:* L_4:* L_5:*	G_1:D=52, E=72 G_2:* G_3:* G_4:* G_5:*

Table 6.1.3 Results of Local and Global Large Itemset Generation

6.2 Comparison of Large Itemsets of Randomized and Non Randomized Partitions

Table 6.2 shows the experimental results of randomized and non-randomized partitions with 100 transactions having Mininmum_Support=50% of total records in each partition and Global_Support=50 % of all records of entire database.

Non Randomized (Sequential)		**Randomized 1**		**Randomized 2**		**Randomized 3**	
ΣL_i P	**G_i**	**ΣL_i P**	**G_i**	**ΣL_i P**	**G_i**	**ΣL_i P**	**G_i**
L_1=14	G_1=3	L_1=12	G_1=2	L_1=12	G_1=2	L_1=11	G_1=2
L_2= 1	G_2=*	L_2= 1	G_2=*	L_2= 1	G_2=*	L_2= 2	G_2=*
L_3= *	G_3=*	L_3= *	G_3=*	L_3= *	G_3=*	L_3= *	G_3=*
L_4= *	G_4=*	L_4= *	G_4=*	L_4= *	G_4=*	L_4= *	G_4=*
L_5= *	G_5=*	L_5= *	G_5=*	L_5= *	G_5=*	L_5= *	G_5=*

Table 6.2 Comparison of Randomized and Non Randomized Data sets

The comparison of results show that in non randomized or sequential partitions, there is data skew involved which causes outliers. While randomization of partitions removes these outliers.

6.3 Comparison of No. of Large Itemsets with Varying Size of Data

Many experiments were performed using varying size of datasets. Table 6.3 shows some of the experimental results with 20, 100, 1000, 10,000 transactions. Number of locally large and globally large itemsets are given in the table.

DATASETS		NO. OF LOCAL LARGE ITEMSETS				SUM	NO. OF GLOBAL ITEMSETS	
Transactions	m_sup	P_1	P_2	P_3	P_4	ΣL_i	m_ sup	G_i
		L_1=2	L_1=5	L_1=3	L_1=4	14		G_1=2
		L_2=1	L_2=1	L_2=2	L_2=5	9		G_2=1
20	3	L_3=*	L_3=*	L_3=*	L_3=2	2	10	G_3=*
		L_4=*	L_4=*	L_4=*	L_4=*	*		G_4=*
		L_5=*	L_5=*	L_5=*	L_5=*	*		G_5=*
		L_1=3	L_1=3	L_1=4	L_1=4	14		G_1=3
		L_2=*	L_2=1	L_2=1	L_2=1	3		G_2=*
100	13	L_3=*	L_3=*	L_3=*	L_3=*	*	50	G_3=*
		L_4=*	L_4=*	L_4=*	L_4=*	*		G_4=*
		L_5=*	L_5=*	L_5=*	L_5=*	*		G_5=*
		L_1=4	L_1=4	L_1=4	L_1=4	16		G_1=4
		L_2=1	L_2=1	L_2=1	L_2=1	4		G_2=1
1000	250	L_3=*	L_3=*	L_3=*	L_3=*	*	500	G_3=*
		L_4=*	L_4=*	L_4=*	L_4=*	*		G_4=*
		L_5=*	L_5=*	L_5=*	L_5=*	*		G_5=*
		L_1=4	L_1=4	L_1=4	L_1=3	15		G_1=3
		L_2=1	L_2=1	L_2=1	L_2=1	4		G_2=1
1,0000	2500	L_3=*	L_3=*	L_3=*	L_3=*	*	5000	G_3=*
		L_4=*	L_4=*	L_4=*	L_4=*	*		G_4=*
		L_5=*	L_5=*	L_5=*	L_5=*	*		G_5=*

Table 6.3 Comparison with Varying Size of Data Sets

6.4 Comparison of Randomized PARTITION with CMA

Comparison of CMA with randomized PARTITION algorithm was done on the basis of computational complexity as well as time efficiency. Results of comparisons are given below

6.4.1 Computational Complexity Comparison

Result of comparison on the basis of computational complexity of both algorithms are shown in table 6.4.1

CMA	Randomized PARTITION
Computational complexity of Partitioning database=O(n^3) Computational complexity of algorithm after partitioning=O(n^3)	Computational complexity of Partitioning database=O(n) Computational complexity after partitioning=O(n^2)

Table 6.4.1 Comparison of Computational Complexity of Randomized PARTITION with CMA

6.4.2 Comparison of Time Efficiency of Database Partitioning

Both algorithms were run on the same system one by one and their running time was observed to see which one is more efficient and gives results in less time. Results of this comparison are shown below

No. of Records	Database Partitioning (time in sec)	
	CMA	Randomized PARTITION
20	3.875	23.645
100	6.54	27.545
1,000	29.102	31.335
10,000	338.622	38.305
25,000	921.646	104.34
50,000	1364.73	304.839
75,000	1752.67	503.227
100,000	2195.42	658.307

Table 6.4.2 Comparison of Time Efficiency of Randomized PARTITION with CMA

Table 6.4.2 shows that although CMA performs better with more efficiency with less number of records but as the number of records is increased, its efficiency decreases to a greater extent. While Randomized PARTITION performs much better than CMA as the number of records is increased. The time variation in partitioning technique of

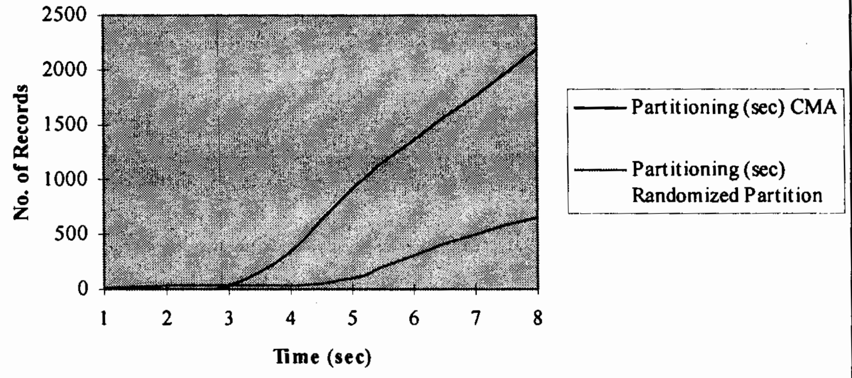
Randomized PARTITION is much less than that of CMA which proves that larger number of records, Randomized PARTITION is much better than CMA.

Threshold

The threshold here is 10,000. As the performance of CMA is much better than Randomized PARTITION with number of records less than 10,000. The efficiency of CMA is best with less number of records but as the number of records is increased its efficiency decreases. While the efficiency of randomized PARTITION algorithm is much better than CMA as the number of records is increased. jn

Graphical representation of the above table is given in figure 6.4.2(a)

Figure 6.4.2a Time Efficiency Comparison of Partitioning of Randomized PARTITION with CMA



6.4.3 Comparison of Time Efficiency of Large Itemset Creation

The time taken by Randomized PARTITION to create large itemsets for varying size of data was compared with the time taken by CMA. The results of this comparison are shown in table 6.4.3.

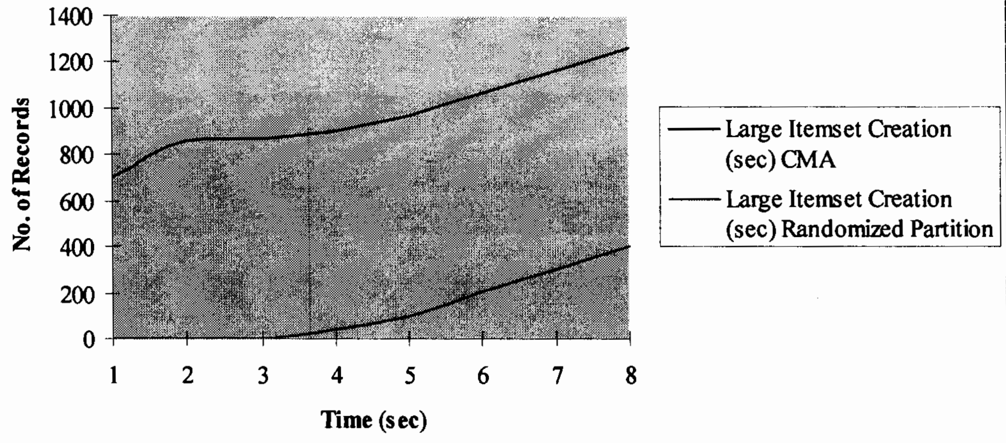
No. of Records	Large Itemset Creation (sec)	
	CMA	Randomized PARTITION
20	700.12	1.192
100	860	1.332
1000	866	4.045
10000	904	42.13
25000	963	101.8126
50000	1068	206.1212
75000	1165	303.102
100000	1260	398.342

Table 6.4.2 Comparison of Time Efficiency of Large Itemset Creation of Randomized PARTITION with CMA

The above table shows that Randomized PARTITION algorithm creates large itemsets more efficiently than CMA in much less time than CMA.

The graphical representation is shown in figure 6.4.3 (a)

Figure 6.4.3 Time efficiency Comparison of Large Itemset Creation



6.5 Comparison of Randomized PARTITION with Previous PARTITION Algorithm

Previous PARTITION	Randomized PARTITION
Memory usage=increased Hash tree storage	Memory usage=decreased Structure with count

Table 6.5 Comparison of Randomized PARTITION with Previous PARTITION Algorithm

Experiments proved that Randomized PARTITION algorithm is more efficient than CMA due to its efficient partitioning. Its time complexity is O (n^2) much better than the time complexity of CMA that is O (n^3).

This version of Randomized PARTITION algorithm is also more efficient than previously implemented PARTITION algorithm as it reduces the memory usage by simply storing counts for each large itemset instead of storing the TIDs in hash tree. By using counts the time efficiency of Randomized PARTITION algorithm is also increased than previously implemented PARTITION algorithm.

Chapter 7

Conclusion & Future Enhancements

7. Conclusion & Future Enhancements

This improved version of Randomized PARTITION algorithm is more efficient than CMA as its partitioning technique takes much less time than the one used by CMA. The computational complexity is O (n) which is much better than the computational complexity of CMA that is O (n^3). Its time efficiency of database partitioning as well as large itemset creation is greater than CMA. Randomized PARTITION algorithm works in a much better way as the number of records is increased as compared to CMA.

This version of Randomized PARTITION algorithm is also more efficient than previously implemented PARTITION algorithm as it reduces the memory usage by simply storing counts for each large itemset instead of storing the TIDs in hash tree. This has also increased the time efficiency of Randomized PARTITION algorithm as compare to previously implemented PARTITION algorithm.

During this study, Randomized PARTITION algorithm has been implemented with synthetic database, as was used by CMA, for comparison purpose. The size of the database was 100,000 tuples. In future it is planned to test the same algorithm with the database size exceeding 1million transactions, further it is planned to implement this algorithm in parallel environment for better efficiency.

Appendix A

References

Appendix A. References

[1]. “**Data Mining: Typical Data Mining Process for Predictive Modelling**”. By BPB Publication. Pg 3, 5.

[2].**Data Mining** by Doug Alexander **dea@tracor.com** .

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

[4]. 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.

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

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

[7]. 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.

[8]. 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.

[9]. 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

.

[10] Andreas Mueller. **"Fast Sequential and Parallel Algorithms for Association Rule Mining: A Comparison"**. Department of Computer Science University of Maryland-College Park. College Park ,MD 20742.

[11] Rakesh Agrawal John C. Shafer. **"Parallel Mining of Association Rules"**. IBM Almaden Research Center 650 Harry Rd., San Jose,CA 95120 ragrawal,shafer}@almaden.IBM.com Tel: (408) 927-1734 Fax: (408) 927-3215.

[12] Mohammed J. Zaki. **" Parallel and Distributed Association Mining: A Survey"**. Department of Computer Science Rensselaer Polytechnic Institute Troy, NY 12180-3590 zaki@cs.rpi.edu

[13] Saleha Jamshaid, Zakia Jalil, Malik Sikander Hayat Khiyal and Muhammad Imran Saeed. **"Association Rule Mining in Centralized Databases"**.Information Technology Journal 6(2) 2007.

Appendix B

Publication

Appendix B. Publication

Paper_PARTITION_vs_CMA Research 4 thesis.pdf

Appendix B

Publication

Comparative Study of Randomized PARTITION Algorithm with CMA Algorithm

Fakhra Razi[1], Samina Kausar[1], Malik Sikander Hayat Khiyal[1] and Muhammad Imran Saeed[1].

Dept. of Computer Sciences, Faculty of Applied and Basic Sciences, International Islamic University, H-10 Islamabad, Pakistan[1].

***Abstract*-Association Rule Mining is an important research area in the field of Data Mining especially in case of 'Sales transactions'. A number of algorithms have been presented in this regard. In this paper a comparison of PARTITION algorithm with CMA algorithm is presented after improving the PARTITION algorithm. In this study, randomized partitioning of database is done. The database is randomized so that real random data is available for better results. The randomized partitioning of database has been implemented in different tool, i.e., MATLAB, as compared to CMA, which uses VB.Net for randomization so as to achieve better performance and efficient results. In the end it has been proved with extensive experiments that although Randomized PARTITION algorithm takes two database scans as compared to CMA that takes single database scan, still it gives better results with more efficiency than CMA.**

Keywords: Data Mining, Association Rule Mining, Randomized PARTITION Algorithm, CMA Algorithm.

I. Introduction

In the past few years the amount of data in business organizations has been increased to a greater extent, so the extraction of useful information from such databases is an up hill and challenging task. Data mining techniques can be applied in various fields like sales transaction, marketing, finance, insurance, medicine and fraud detection etc. Interesting association relationships are much helpful in taking good business decisions like how to increase sales or purchases process. Its basic aim is to analyze entire huge database to find the frequently occurring itemsets together. For the rule generation it takes one or multiple scans of the whole database. Association rule mining can be best explained by a simple example of market basket analysis that basically determines which groups of products or items are sold or purchased together most frequently at the same time. Customer buying behavior can be best analyzed by this process, as it helps in finding association among different itemsets. For example if customers buy soft drink and chips together mostly with in the same visit to the superstore then using this information for the next time the shop keeper will place the soft drink and chips in the nearer shelves. Such shelf arrangement of these items will increase the sales process in future visits to the store. Similarly if a person orders a birthday cake to some bakery, then he is likely to be purchasing some birthday candles as well. If the baker is aware of the association between candles and the birthday cake then he must be offering his customers to purchase candles from him, hence supporting his candle business as well. Market basket analysis is helpful for the retailers in the best adjustments of catalogue design and store layouts etc.

In this study an improved version of PARTITION algorithm is presented and it has been proved through various experiments that its performance is much better than previously implemented PARTITION algorithm as it reduces the memory usage as well as time. Its efficiency is also compared with previously implemented CMA algorithm and it has been proved that due to its efficient partitioning technique of database it gives better results than CMA in terms of time efficiency.

II. Association Rule Mining

Association rule mining searches for the interesting relationships among the items in a given dataset [1]. For example, if the support = 5%, confidence=70%. Association rule for the last discussed example of birthday cake and candle is:

Birthday cake => candles

Rule interestingness can be measured well by support and confidence. Support=5% shows that for the above discussed rule, the birthday cake and candles are bought together in all transactions. Confidence=70% means that 70% of the customers who buy a birthday cake also purchased birthday candles.

III. Basic Concept

Association rule mining basic concept is as illustrated in [1]. Let $I = \{i_1, i_2, \ldots, i_m\}$ be the set of items. Let D, the task-relevant data, is a set of database transactions where each transaction T is a set of items such that $T \subset I$. 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 $A \subset T$. 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,

$$\text{Support } (A \Rightarrow B) = P\ (A \cup B). \tag{1}$$

$$\text{Confidence } (A \Rightarrow B) = P\ (B|A). \tag{2}$$

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.

Association rule mining is a two-step process. Firstly, find all frequent itemsets: by definition, each of those itemsets will occur at least as frequently as a pre-determined minimum support count. Secondly, generate strong association rules from the frequent itemsets: by definition, these rules must satisfy minimum support and minimum confidence. The association rules are generated simply using the following formula:

$$\text{If} \quad \frac{(\text{support}(\{Y, X\}))}{\text{support}(\{X\})} \geq \text{min\_conf.} \qquad (3)$$

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.

IV. Previous Work

Association rule mining for the first time was introduced by [2]. It has remained highly in use by the pundits and practitioners of the data mining for more research since its inception. Many algorithms have been discovered in this field. The pioneer work in this area was presented by [3]. They discussed Apriori algorithm. Apriori is termed as the best base algorithm for all other subsequent algorithms. Apriori was an iterative algorithm which used complete bottom search to find out large 1-itemset in first pass. Next pass generated candidate itemsets and checked the support. The process repeated until all large itemsets were found. Other variants of Apriori, AprioriTid and AprioriHybrid were also discussed by them in the same paper. In Apriori for ***n*** iterations, ***n*** scans of the entire database were done. AprioriTid overcame this problem in later iterations, as it did not use the whole database for counting support after the first pass. Best features of both Apriori and AprioriTid were combined in AprioriHybrid which showed good performance than the other two in real applications.

DHP (Direct Hashing and Pruning) algorithm was presented by [4] in 1995, which was an extension of Apriori algorithm. It was confined to the generation of large itemsets, the step one of the mining association rules. The problem with this algorithm was that database pruning benefit was quite ambiguous.

Reference [5] presented the survey of parallel and distributed association rule mining algorithms. These algorithms were divided into groups according to the techniques utilized, database structure and search techniques etc. This paper also provided the design space of parallel and distributed algorithms either implemented on distributed or shared–memory architecture. The aim of this paper was to provide a reference for more research. It also described the challenges and problems in the field of association rule mining.

Reference [6] presented PARTITION algorithm that worked in two phases. Logical division of horizontal database into non-overlapping partitions was done. Then locally large itemsets were found for each partition. For each locally large itemset, the TIDs were stored in hash tree. Further potentially large itemsets were obtained by merging the locally large itemsets at the end of phase I. To find the globally large itemsets actual support of these itemsets was measured in phase II. For the available memory to uncover accurate no. of partitions was unsolvable by this algorithm.

Reference [7] implements CMA (Centralized Mining of Association-Rules) algorithm for association rule mining. In this algorithm the database was divided into logical non overlapping partitions and then DMA algorithm was applied on each partition to find local and global large itemsets. This algorithm took just a single database scan over each partition for the creation of large itemsets.

From the literature review it is concluded that despite the recent advances in association rule mining algorithms, there are still some problems like in large databases, scanning is much expensive and the resulting candidate itemsets are too large to fit in the aggregate memory. Data skew, data size, multiple scans and pruning techniques needs further study.

V. Randomized Partition Algorithm

In this paper, Randomized PARTITION algorithm has been implemented. Logical non overlapping partitions are created with both random and non random data. Results of both randomized and non randomized partitions are compared to see the effect of data skew on both locally and globally large itemsets. Then the efficiency of Randomized PARTITION algorithm is compared with CMA.

PARTITION algorithm was previously implemented in Silicon Graphics Indy R4400SC workstation with a clock rate of 150 MHz and 32 Mbytes of main memory. The data resided on 1GB SCSI presented by [6].

In this study Randomized PARTITION algorithm has been implemented in the same environment as CMA except the partitioning of database that has been done in a different tool (i.e. CMA has used .NET while Randomized PARTITION algorithm is using MATLAB) for achievement of better performance and efficient results. This change in partitioning technique increases the time efficiency of algorithm as compared to CMA. In today's world, time is an important factor. Along with the need of accurate results, time efficiency matters a lot. It is important to have better results in less time and minimum cost. Randomized PARTITION algorithm has also been improved by using count structure rather storing TIDs in Hash tree as was done in previously implemented PARTITION algorithm. This change reduces the high memory usage and time as compared to previously implemented PARTITION algorithm.

Synthetic database is used for this study which is the same database as was used by CMA. It is also assumed that transactions are in the form (TID, i_1, i_2, i_3). The items are assumed to be kept sorted in lexicographic order. Similar assumption is also made in [1].

A. Algorithm

```
//Read database sequentially
Read_Database
P=Create_Partitions (rand)
For x=1 to P
Begin
        Generate_Candidate_Itemsets $C_i$ for $p_x$
        For i=1 to k
        Begin
                //Generate local large itemsets $L_i$ for $p_i$ from
                $C_i$ and store items and their count in structure $S_i$
                $S_i$=Gen_Local_Large_Itemset $L_i$
        End
        //store value in global candidate structure
        Merge local large itemsets to form global candidate
        itemsets $G_i$_C
End
//Generate globally large itemsets $G_i$ from $G_i$_C
$G_i$=Gen_Global_Large_itemset ($G_i$_C)
```

TABLE I
NOTATIONS USED

Notation	Definition
P_x	Partitions (x=1 to 4)
C_i_L	Local candidate itemsets
L_i	Local large itemsets
S_i	Structure for storing local large itemsets along with their counts in particular partition p_i
Gi_C	Global candidate itemsets containing local large itemsets along with their counts
Gi	Globally large itemsets
*	Not possible

B. Working of Randomized Partition Algorithm

Working of Randomized PARTITION algorithm is explained as follows:

First of all database is read in sequential order. Then logical non overlapping randomized or non randomized partitions are created. Partition size is chosen in such a way that for a specific time a whole partition can easily reside in memory. Partition size should not be very small or very large, as small partitions are negatively affected by data skew and in large partitions for intermediate results processing buffer requirements can exceed the available space, so risk is involved in both cases. Candidate large 1 itemsets are generated for all partitions. Then local large 1 itemsets are generated containing items having their support greater than user defined minimum support within each partition and the candidate itemsets whose minimum support is less than user defined minimum support are pruned away. Here a count structure is used for large itemsets rather than storing TIDs in a Hash tree. Storage of TIDs in hash tree is wastage of time and memory that's why count structure is used for finding large itemsets directly. These large itemsets are then merged and stored in global candidate 1 itemset along with their counts for the generation of globally large itemsets. From large 1 itemsets, candidate large 2 itemsets are generated from which local large 2 itemsets are generated for each partition. The process continues upto locally large k itemset generation. Finally globally large itemsets are generated and global candidate itemsets having minimum support less than user defined minimum support are pruned away. Randomized PARTITION algorithm reduces the time complexity up to O (n) as compared to CMA whose time complexity is O (n^3). Experimental results proved that Randomized PARTITION is 3 times more efficient than CMA.

C. Functions of Randomized PARTITION Algorithm

Function [T] = Read_Database

Reads the entire database sequentially and returns the transactions in an array.

Function [P] = Create_Partitions (rand)

This function takes rand as argument. If rand=0 then sequential (non randomized) non overlapping partitions are created otherwise T is first randomized and then logical non overlapping partitions are created and an array of partition P is returned.

Function [LocalLargeItemset] = Generate_Large_Itemsets (p_i)

This function creates candidate itemsets for all partitions and then from these candidate itemsets local large itemsets are generated for each partition. This function returns locally large itemsets for each partition.

Function [GlobalCandidate] = Generate_Global_Candidate (LocalLargeItemsets)

This function takes LocalLargeItemsets as an argument, merge them and generates global candidate itemset.

Function [GlobalLargeItemset] = Generate_Global_Itemsets (GlobalCandidate)

This function takes GlobalCandidate as an argument and generates globally large itemsets RIP.

D. Explanation of Randomized PARTITION Algorithm with Example

Performance of Randomized PARTITION algorithm can be best demonstrated by the case study. Following dataset was used for the case study:

The dataset shown in Table II was randomized as shown in Table III to find out large itemsets. The results generated by this dataset are shown in Table IV.

For testing the algorithm, many experiments were performed by taking both random and non random data. Number of items in the given dataset was 5. Initially, the database was read sequentially and four logical non overlapping partitions were created. Then local and global itemsets for sequential database partitions were generated. In the second step, the database was randomized and again logical non overlapping partitions (P_1, P_2, P_3, P_4) were created. After partition creation, the Randomized PARTITION algorithm was applied to find locally large 1-itemsets for each partition. From this, candidate 2-itemset was generated and then locally large 2-itemset for each partition was created. This process continued unless up to k large itemsets were found. The algorithm stored counts of the locally large itemsets in a structure rather than using a hash tree for this process.

TABLE II
SEQUENTIAL DATASET

Partitions			
P_1	P_2	P_3	P_4
T_{1}: A,B,C	T_{26}: A,D,E	T_{51}: A,E	T_{76}: A,C,D,E
T_{2}: A,B,D	T_{27}: A,B,C,D,E	T_{52}: A,B,E	T_{77}: B,C,E
T_{3}: B,D,E	T_{28}: C,D,E	T_{53}: B,D,E	T_{78}: C,D,E
T_{4}: A,B,E	T_{29}: B,C,D,E	T_{54}: B,C,E	T_{79}: B,C,D,E
T_{5}: A,C,D	T_{30}: A,D,E	T_{55}: C,D	T_{80}: A,D,E
T_{6}: A,C,E	T_{31}: A,B,E	T_{56}: A,D,E	T_{81}: C,D,E
T_{7}: A,C,D	T_{32}: D,E	T_{57}: A,D,E	T_{82}: B,D,E
T_{8}: B,D,E	T_{33}: A	T_{58}: A,C	T_{83}: A,B,C,E
T_{9}: C,D	T_{34}: C,D	T_{59}: A,B,C,E	T_{84}: B,C,E
T_{10}: C,D,E	T_{35}: C,D,E	T_{60}: A,B,C,D	T_{85}: B,C,D
T_{11}: A,C,D	T_{36}: B,C	T_{61}: A,C,D	T_{86}: C,D,E
T_{12}: A,B,E	T_{37}: A,C	T_{62}: A,C,D,E	T_{87}: B,E
T_{13}: B,D,E	T_{38}: A,C,D,E	T_{63}: D,E	T_{88}: A,B,E
T_{14}: D,E	T_{39}: D,E	T_{64}: C,D,E	T_{89}: A,B,C
T_{15}: C,D,E	T_{40}: B,D,E	T_{65}: A,B,D	T_{90}: A,C,E
T_{16}: A,D,E	T_{41}: A,B	T_{66}: B,D,E	T_{91}: B,C,E
T_{17}: A,B,C,D	T_{42}: A,C	T_{67}: C,D,E	T_{92}: B,D,E
T_{18}: B,C,E	T_{43}: A,B,C	T_{68}: A,D,E	T_{93}: A,C,E
T_{19}: A,C,E	T_{44}: B,C,D	T_{69}: B,D,E	T_{94}: B,C,E
T_{20}: B,C,D	T_{45}: D,E	T_{70}: A,C	T_{95}: D,E
T_{21}: A,B,D,E	T_{46}: B,C,D,E	T_{71}: A,B,C	T_{96}: C,E
T_{22}: C,D,E	T_{47}: C,D,E	T_{72}: A,C,D,E	T_{97}: B,D,E
T_{23}: B,E	T_{48}: D,E	T_{73}: B,C,D,E	T_{98}: A,B,D,E
T_{24}: A,C,D	T_{49}: C,D	T_{74}: A,D,E	T_{99}: D,E
T_{25}: B,D,E	T_{50}: A,D	T_{75}: B,C,D,E	T_{100}: A,C,D,E

TABLE III
RANDOMIZED DATASET

Partitions			
P_1	P_2	P_3	P_4
T_{24}: A,C,D	T_{95}: D,E	T_{70}: A,C	T_{81}: A,C,D
T_{92}: B,D,E	T_{53}: B,D,E	T_{54}: B,C,E	T_{23}: B,E
T_{31}: A,B,E	T_{49}: C,D	T_{77}: B,C,E	T_{65}: A,B,D
T_{8}: B,D,E	T_{72}: A,C,D,E	T_{74}: A,D,E	T_{9}: C,D
T_{42}: A,C	T_{69}: B,D,E	T_{60}: A,B,C,D	T_{63}: D,E
T_{21}: A,B,D,E	T_{67}: C,D,E	T_{28}: C,D,E	T_{45}: D,E
T_{99}: D,E	T_{22}: C,D,E	T_{3}: B,D,E	T_{41}: A,B
T_{91}: B,C,E	T_{75}: B,C,D,E	T_{11}: A,C,D	T_{73}: B,C,D,E
T_{25}: B,D,E	T_{57}: A,D,E	T_{80}: A,D,E	T_{37}: A,C
T_{55}: C,D	T_{44}: B.C.D	T_{64}: C,D,E	T_{59}: A,B,C.E
T_{85}: B,C,D	T_{16}: A,D,E	T_{66}: B,D,E	T_{58}: A,C
T_{15}: C,D,E	T_{19}: A,C,E	T_{96}: C,E	T_{89}: A,B,C
T_{50}: A,D	T_{36}: B,C	T_{40}: B,D,E	T_{84}: B,C,E
T_{51}: A,E	T_{48}: D,E	T_{43}: A,B,C	T_{5}: A,C,D
T_{27}:A,B,C,D,E	T_{10}: C,D,E	T_{52}: A,B,E	T_{20}: B,C,D
T_{30}: A,D,E	T_{78}: C,D,E	T_{79}: B,C,D,E	T_{93}: A,C,E
T_{94}: B,C,E	T_{33}: A	T_{56}: A,D,E	T_{62}: A,C,D,E
T_{39}: D,E	T_{7}: A,C,D	T_{76}: A,C,D,E	T_{18}: B,C,E
T_{26}: A,D,E	T_{35}: C,D,E	T_{47}: C,D,E	T_{13}: B,D,E
T_{2}: A,B,D	T_{98}: A,B,D,E	T_{71}: A,B,C	T_{34}: C,D
T_{88}: A,B,E	T_{4}: A,B,E	T_{90}: A,C,E	T_{17}: A,B,C,D
T_{87}: B,E	T_{61}: A,C,D	T_{14}: D,E	T_{1}: A,B,C
T_{29}: B,C,D,E	T_{46}: B.C.D.E	T_{32}: D.E	T_{82}: B,D,E
T_{97}: B,D,E	T_{83}: A,B,C,E	T_{6}: A,B,C	T_{86}: C,D,E
T_{68}: A,D,E	T_{38}: A,C,D,E	T_{12}: A,B,E.	T_{100}:A,C,D.E

TABLE IV
RESULTS OF LOCAL AND GLOBAL ITEMSET GENERATION

Local Large itemsets (L_i)				Global Large Item-set (G_i)
P_1 Item=Count	P_2 Item=Count	P_3 Item=Count	P_4 Item=Count	G_i Item=Count
L_1:B=14 D=18, E=19 L_2:* L_3:* L_4:* L_5:*	L_1:C=16 D=19,E=19 L_2:DE=17 L_3:* L_4:* L_5:*	L_1:C=15 D=15,E=20 L_2:* L_3:* L_4:* L_5:*	L_1:C=18 D=15 E=14 L_2:* L_3:* L_4:* L_5:*	G_1:D=52 E=72 G_2:* G_3:* G_4:* G_5:*

This change reduced the high memory usage as compared to previously implemented PARTITION algorithm. Then large itemsets of all lengths were merged to produce global candidate itemsets from which the globally large itemsets of all lengths were generated.

Table V shows the experimental results of randomized and non-randomized partitions with 100 transactions. The comparison of results show that in non randomized or sequential partitions, there was data skew involved which caused outliers while randomization of partitions removed the outliers.

TABLE V
COMPARISON OF NO. OF LARGE ITEMSETS OF RANDOMIZED AND NON RANDOMIZED DATASETS

Sequential		Randomized 1		Randomized 2		Randomized 3	
ΣL_i P	G_i	ΣL_i P	G_i	ΣL_i P	G_i	ΣL_i P	G_i
L_1=14 L_2= 1 L_3= * L_4= * L_5= *	G_1=3 G_2=* G_3=* G_4=* G_5=*	L_1=12 L_2= 1 L_3= * L_4= * L_5= *	G_1=2 G_2=* G_3=* G_4=* G_5=*	L_1=12 L_2= 1 L_3= * L_4= * L_5= *	G_1=2 G_2=* G_3=* G_4=* G_5=*	L_1=11 L_2= 2 L_3= * L_4= * L_5= *	G_1=2 G_2=* G_3=* G_4=* G_5=*

TABLE VI
COMPARISON OF RANDOMIZED PARTITION WITH CMA

CMA	Randomized PARTITION
Time complexity of Partitioning database=O(n^3) Time complexity of algorithm after partitioning=O(n^3)	Time complexity of Partitioning database=O(n) Time complexity of algorithm after partitioning=O(n)

Table VI shows the results of comparison of Randomized PARTITION algorithm with CMA. Experimental results show that although Randomized PARTITION algorithm takes two database scans for large itemset creation still its efficiency is better than CMA which takes one database scan for large itemset creation. The time complexity of partitioning technique as well as the rest of the algorithm is much better than CMA.

TABLE VII
COMPARISON OF NO. OF LARGE ITEMSETS WITH VARYING SIZE OF DATA

Datasets		No. of Local Large itemsets				Sum	G_i	
Trans	m_sup	P_1	P_2	P_3	P_4	ΣL_i	m_sup	G_i
20	3	L_1=2	L_1=5	L_1=3	L_1=4	14	10	G_1=2
		L_2=1	L_2=1	L_2=2	L_2=5	9		G_2=1
		L_3=*	L_3=*	L_3=*	L_3=2	2		G_3=*
		L_4=*	L_4=*	L_4=*	L_4=*	*		G_4=*
		L_5=*	L_5=*	L_5=*	L_5=*	*		G_5=*
100	13	L_1=3	L_1=3	L_1=4	L_1=4	14	50	G_1=3
		L_2=*	L_2=1	L_2=1	L_2=1	3		G_2=*
		L_3=*	L_3=*	L_3=*	L_3=*	*		G_3=*
		L_4=*	L_4=*	L_4=*	L_4=*	*		G_4=*
		L_5=*	L_5=*	L_5=*	L_5=*	*		G_5=*
1000	250	L_1=4	L_1=4	L_1=4	L_1=4	16	500	G_1=4
		L_2=1	L_2=1	L_2=1	L_2=1	4		G_2=1
		L_3=*	L_3=*	L_3=*	L_3=*	*		G_3=*
		L_4=*	L_4=*	L_4=*	L_4=*	*		G_4=*
		L_5=*	L_5=*	L_5=*	L_5=*	*		G_5=*
1,0000	2500	L_1=4	L_1=4	L_1=4	L_1=3	15	5000	G_1=3
		L_2=1	L_2=1	L_2=1	L_2=1	4		G_2=1
		L_3=*	L_3=*	L_3=*	L_3=*	*		G_3=*
		L_4=*	L_4=*	L_4=*	L_4=*	*		G_4=*
		L_5=*	L_5=*	L_5=*	L_5=*	*		G_5=*

Many experiments were performed using varying size of datasets. Table VII shows some of the experimental results with 20, 100, 1000, 10,000 transactions. Given dataset calculates upto large 3-itemsets. Randomized PARTITION algorithm can calculate upto k itemsets with other datasets.

TABLE VIII
COMPARISON OF PREVIOUS PARTITION WITH RANDOMIZED PARTITION ALGORITHM

Previous PARTITION	Randomized PARTITION
Memory usage=increased Hash tree storage	Memory usage=decreased Structure with count

Randomized PARTITION algorithm was also compared with previously implemented PARTITION algorithm and it was observed that its efficiency is better in terms of memory usage and time as shown in Table VIII. Extensive experiments were performed with different datasets for large itemset generation upto k itemset to check the efficiency of algorithm.

VI. CONCLUSION

This improved version of Randomized PARTITION algorithm is more efficient than CMA as its partitioning technique takes much less time than the one used by CMA. Its time complexity is O (n) much better than the time complexity of CMA that is O (n^3). Its efficiency is greater than CMA due to its changed partitioning technique.

This version of Randomized PARTITION algorithm is also more efficient than previously implemented PARTITION algorithm as it reduces the memory usage by simply storing counts for each large itemset instead of storing the TIDs in hash tree. This has also increased the time efficiency of Randomized PARTITION algorithm as compare to previously implemented PARTITION algorithm.

In future we have a plan to implement this algorithm in parallel environment for better efficiency.

ACKNOWLEDGMENT

We wish to thank Mr. Muhammad Imran Saeed and Mr. Abdussalam for their sincere guidance and help throughout our research.

REFERENCES

[1] J. Han, M. Kamber, "Data Mining: Concepts and Techniques," Kaufmann Publishers, San Francisco, California, USA, 2001.

[2] R. Agrawal, T. Imielinski, and A. Swami, "Mining Association Rules between Sets of Items in Large Databases," *In Proceedings of the ACM SIGMOD International Conference on Management of Data,* Washington, D.C. USA, May 1993, pp.207 – 216.

[3] R. Agarawal and R. Srikant, "Fast Algorithms For Mining Association Rules," *In Proceedings of the 20th International Conference on Very Large Data Bases*, Santiago de Chile, Chile, September 1994, edited by J. B. Bocca, M. Jarke, and C. Zaniolo, "Very Large Data Bases." Morgan Kaufmann Publishers, San Francisco, CA, USA, pp. 487 – 499.

[4] J. Soo Park, M. Chen and P.S. Yu, "An Effective Hash-Based Algorithm for Mining Association Rules," *In Proceedings of the ACM SIGMOD International Conference on Management of Data*, May 1995, pp. 175 – 186.

[5] M. J. Zaki, 'Parallel and Distributed Association Mining: A Survey," *In IEEE Concurrency Journal,* Special issue on Parallel Mechanisms for Data Mining, Vol. 7, No. 4, December 1999, pp. 14 – 25.

[6] A. Savasere, E. Omiecinski and S. Navathe, "An Efficient Algorithm for Mining Association Rules in Large Databases," *In Proc. of the 21st Int'l VLDB Conference*, Zurich, Switzerland, September 1995, pp. 432 – 444.

[7] S. Jamshaid, Z. Jalil, M. Khiyal and M. Saeed, "Association Rule Mining in Centralized Databases," *Information Technology Journal*, Vol. 6, Issue 2, 2007, pp. 174 – 181.