1. Introduction

We are in an age often referred to as the information age. In this information age, because we believe that information leads to power and success, from the technologies such as computers, satellites, etc., we have been collecting tremendous amounts of information. Initially, with the advent of computers and means for mass digital storage, we started collecting and storing all sorts of data, counting on the power of computers to help sort through this amalgam of information. Unfortunately, these massive collections of data stored on disparate structures very rapidly became overwhelming. This initial chaos has led to the creation of structured databases and database management systems (DBMS). The efficient database management systems have been very important assets for management of a large corpus of data and especially for effective and efficient retrieval of particular information from a large collection whenever needed. The proliferation of database management systems has also contributed to recent massive gathering of all sorts of information. Today, we have far more information than we can handle, from business transactions and scientific data, to satellite pictures, text reports and military intelligence. Information retrieval is simply not enough anymore for decision-making. Confronted with huge collections of data, we have now created new needs to help us make better managerial choices. These needs are automatic summarization of data, extraction of the "essence" of information stored, and the discovery of patterns in raw data.

1.1 Kind of information collection

We have been collecting a myriad of data, from simple numerical measurements and text documents, to more complex information such as spatial data, multimedia channels, and hypertext documents. Here is a non-exclusive list of a variety of information collected in digital form in databases and in flat files.

•**Business transactions**: Every transaction in the business industry is (often) "memorized" for perpetuity. Such transactions are usually time related and can be inter-business deals such as purchases, exchanges, banking, stock, etc., or intra-business operations such as management of in-house wares and assets. The effective use of the data in a reasonable time frame for competitive decision-making is definitely the most important problem to solve for businesses that struggle to survive in a highly competitive world.

•**Scientific data**: There is amassing colossal amounts of scientific data that need to be analyzed. Unfortunately, we can capture and store more new data faster than we can analyze the old data already accumulated.

•**Medical and personal data**: From government census to personnel and customer files, very large collections of information are continuously gathered about individuals and groups. Governments, companies and organizations such as hospitals, are stockpiling very important quantities of personal data to help them manage human resources, better understand a market.

•**Surveillance video and pictures**: Videotapes from surveillance cameras are usually recycled and thus the content is lost. However, there is a tendency today to store the tapes and even digitize them for future use and analysis.

•**Satellite sensing**: There are a countless number of satellites around the globe, but all are sending a non-stop stream of data to the surface. Many satellite pictures and data are made public as soon as they are received in the hopes that other researchers can analyze them.

•**Digital media**: Many radio stations; television channels and film studios are digitizing their audio and video collections to improve the management of their multimedia assets.

•**CAD and Software engineering data**: There are a multitude of Computer Assisted Design (CAD) systems for architects to design buildings or engineers to conceive system components or circuits. These systems are generating a tremendous amount of data. Moreover, software engineering is a source of considerable similar data with code, function libraries, objects, etc., which need powerful tools for management and maintenance.

•**The World Wide Web repositories**: Since the inception of the World Wide Web in 1993, documents of all sorts of formats, content and description have been collected and inter-connected with hyperlinks making it the largest repository of data ever built. Despite its dynamic and unstructured nature, its heterogeneous characteristic, and its very often redundancy and inconsistency, the World Wide Web is the most important data collection regularly used for reference because of the broad variety of topics covered and the infinite contributions of resources and publishers. Many believe that the World Wide Web will become the compilation of human knowledge.

1.2 Data Mining and Knowledge Discovery

With the enormous amount of data stored in files, databases, and other repositories, it is increasingly important, if not necessary, to develop powerful means for analysis and perhaps interpretation of such data and for the extraction of interesting knowledge that could help in decision-making. *Data Mining*, also popularly known as *Knowledge Discovery in Databases* (KDD), refers to the nontrivial extraction of implicit, previously unknown and potentially useful information from data in databases. The following figure (Figure 1.1) shows data mining as a step in an iterative knowledge discovery process.

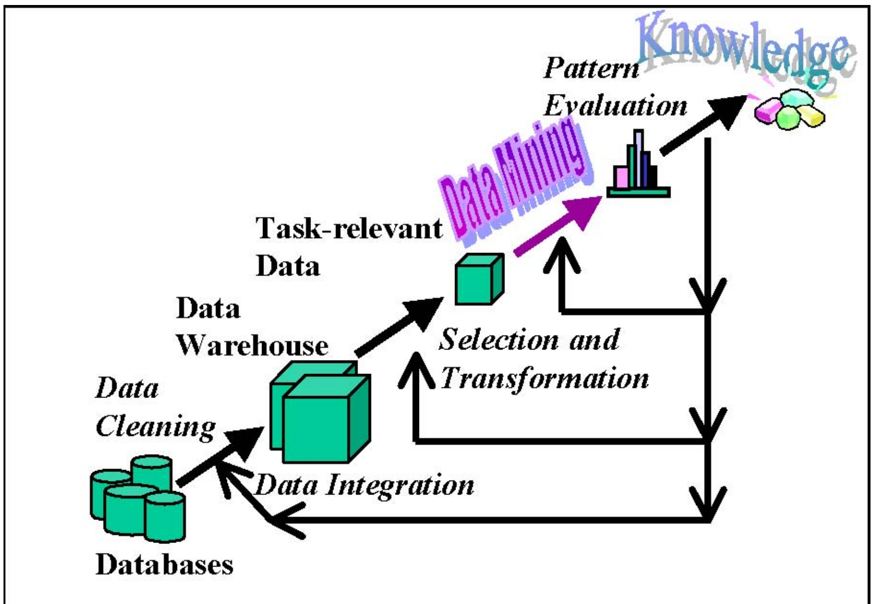


Figure 1.1: Data Mining is the core of Knowledge Discovery process

The Knowledge Discovery in Databases process comprises of a few steps leading from raw data collections to some form of new knowledge. The iterative process consists of the following steps:

•**Data cleaning**: also known as data cleansing, it is a phase in which noise data and irrelevant data are removed from the collection.

•**Data integration**: at this stage, multiple data sources, often heterogeneous, may be combined in a common source.

•**Data selection**: at this step, the data relevant to the analysis is decided on and retrieved from the data collection.

•**Data transformation**: also known as data consolidation, it is a phase in which the selected data is transformed into forms appropriate for the mining procedure.

•**Data mining**: it is the crucial step in which clever techniques are applied to extract patterns potentially useful.

•**Pattern evaluation**: in this step, strictly interesting patterns representing knowledge are identified based on given measures.

•**Knowledge representation**: is the final phase in which the discovered knowledge is visually represented to the user. This essential step uses visualization techniques to help users understand and interpret the data mining results.

1.3 Characteristics of the data mining system

- Large quantity of data

  Volume of data so great it has to be analyzed by automated technique ie. Satellite information, credit card transactions etc.

- Noisy, incomplete data
  1. Imprecise data is characteristics of all data collection.
  2. Databases-usually contaminated by errors, cannot assume that the data they contain correct e.g. some attributes depends on the subjective or measurement judgments.
- Complex data structure- conventional statistical analysis not possible.
- Data stored in the legacy system.

1.4 Kind of Data to be mined

Data mining is not specific to one type of media or data. Data mining should be applicable to any kind of information repository. Data mining is being put into use and studied for databases, including relational databases, object-relational databases and object-oriented databases, data warehouses, transactional databases, repositories such as the World Wide Web, multimedia databases, time-series databases and textual databases, and even flat files. Here are some examples in more detail:

•**Flat files**: Flat files are actually the most common data source for data mining algorithms, especially at the research level. Flat files are simple data files in text or binary format with a structure known by the data-mining algorithm to be applied. The data in these files can be transactions, time-series data, scientific measurements, etc.

•**Relational Databases**: Briefly, a relational database consists of a set of tables containing either values of entity attributes, or values of attributes from entity relationships. Tables have columns and rows, where columns represent attributes and rows represent tuples. A tuple in a relational table corresponds to either an object or a relationship between objects and is identified by a set of attribute values representing a unique key.

The most commonly used query language for relational database is SQL, which allows retrieval and manipulation of the data stored in the tables, as well as the calculation of aggregate functions such as average, sum, min, max and count.

Data mining algorithms using relational databases can be more versatile than data mining algorithms specifically written for flat files, since they can take advantage of the structure inherent to relational databases. While data mining can benefit from SQL for data selection, transformation and consolidation, it goes beyond what SQL could provide, such as predicting, comparing, detecting deviations, etc.

•**Data Warehouses**: A data warehouse as a storehouse, is a repository of data collected from multiple data sources (often heterogeneous) and is intended to be used as a whole under the same unified schema. A data warehouse gives the option to analyze data from different sources under the same roof. To facilitate decision-making and multi-dimensional views, data warehouses are usually modeled by a multi-dimensional data structure

•**Transaction Databases**: A transaction database is a set of records representing transactions, each with a time stamp, an identifier and a set of items. Associated with the transaction files could also be descriptive data for the items. Since relational databases do not allow nested tables (i.e. a set as attribute value), transactions are usually stored in flat files or stored in two normalized transaction tables, one for the transactions and one for the transaction items. One typical data mining analysis on such data is the so-called market basket analysis.

•**Multimedia Databases**: Multimedia databases include video, images, audio and text media. They can be stored on extended object-relational or object-oriented databases, or simply on a file system. Multimedia is characterized by its high dimensionality, which makes data mining even more challenging. Data mining from multimedia repositories may require computer vision, computer graphics, image interpretation, and natural language processing methodologies.

•**Spatial Databases**: Spatial databases are databases that, in addition to usual data, store geographical information like maps, and global or regional positioning. Such spatial databases present new challenges to data mining algorithms.

•**Time-Series Databases**: Time-series databases contain time related data such stock market data or logged activities. These databases usually have a continuous flow of new data coming in, which sometimes causes the need for a challenging real time analysis. Data mining in such databases commonly includes the study of trends and correlations between evolutions of different variables, as well as the prediction of trends and movements of the variables in time.

•**World Wide Web**: The World Wide Web is the most heterogeneous and dynamic repository available. A very large number of authors and publishers are continuously contributing to its growth and metamorphosis, and a massive number of users are accessing its resources daily. Data in the World Wide Web is organized in inter-connected documents. These documents can be text, audio, video, raw data, and even applications. Conceptually, the World Wide Web is comprised of three major components: The content of the Web, which encompasses documents available; the structure of the Web, which covers the hyperlinks and the relationships between documents; and the usage of the web, describing how and when the resources are accessed. A fourth dimension can be added relating the dynamic nature or evolution of the documents. Data mining in the World Wide Web, or web mining, tries to address all these issues and is often divided into web content mining, web structure mining and web usage mining.

1.5 Things discovered

The kinds of patterns that can be discovered depend upon the data mining tasks employed. By and large, there are two types of data mining tasks: *descriptive data mining* tasks that describe the general properties of the existing data, and *predictive data mining* tasks that attempt to do predictions based on inference on available data.

•**Association analysis**: Association analysis is the discovery of what are commonly called *association rules*. It studies the frequency of items occurring together in transactional databases, and based on a threshold called *support*, identifies the frequent item sets. Another threshold, *confidence*, which is the conditional probability than an item appears in a transaction when another item appears, is used to pinpoint association rules. Association analysis is commonly used for market basket analysis. The discovered association rules are of the form: $P \rightarrow Q$ [s,c], where P and Q are conjunctions of attribute value-pairs, and s (for support) is the probability that P and Q appear together in a transaction and c (for confidence) is the conditional probability that Q appears in a transaction when P is present. For example, the hypothetic association rule:

1.6 Data Mining Tools

Any data-mining algorithm is composed of 3 general parts:

Model representation, model evaluation and search method. The model should represents flexible limits and assumptions clearly, so that patterns can be discovered; the model should have predictive validity-which can be based on cross validation: and the search should optimize the model evaluation criteria given the data and model representation

The mining tools are usually automated programs, which detect predefined patterns. Various types of tools used are:

1) NEURAL NETWORKS:

They are a collection of connected nodes with inputs, outputs and processing at each node. A number of hidden layers exist between the visible input and output layers. They can’t be trained on very large databases but with sampling method they can produce reasonable accuracy on small and mid size datasets. In their case no explanation of the results is provided.

2) DECISION TREE:

They segregate the data based on values of variables. They use a hierarchy of if – then statements to classify data. They are faster and easily understandable. But in this case data type has to be interval or categorical.

3) RULE INDUCTION:

In this case a set of nonhierarchical sets of conditions will be generated which is used to predict values for new data items .The rules used for the prediction are more general and more powerful than decision trees.

4) DATA VISUALIZATION:

It makes possible to gain a deeper, intuitive understanding of the data presenting a picture of users to see rather automating the process.

5) GENETIC ALGIRITHM:

Optimization technique that use process such as genetic combination, mutation and natural selection in a design based on the concepts of evolution.

1.7 Classification of data mining systems

There are many data mining systems available or being developed. Some are specialized systems dedicated to a given data source or are confined to limited data mining functionalities, other are more versatile and comprehensive. Data mining systems can be categorized according to various criteria among other classification are the following:

•**Classification according to the type of data source mined**: This classification categorizes data mining systems according to the type of data handled such as spatial data, multimedia data, time-series data, text data, World Wide Web, etc.

•**Classification according to the data model drawn on**: This classification categorizes data mining systems based on the data model involved such as relational database, object-oriented database, data warehouse, transactional, etc.

•**Classification according to the king of knowledge discovered**: This classification categorizes data mining systems based on the kind of knowledge discovered or data mining functionalities, such as characterization, discrimination, association, classification, clustering, etc. Some systems tend to be comprehensive systems offering several data mining functionalities together.

•**Classification according to mining techniques used**: Data mining systems employ and provide different techniques. This classification categorizes data mining systems according to the data analysis approach used such as machine learning, neural networks, genetic algorithms, statistics, visualization, database-oriented or data warehouse-oriented, etc. The classification can also take into account the degree of user interaction involved in the data mining process such as query-driven systems, interactive exploratory systems, or autonomous systems. A comprehensive system would provide a wide variety of data mining techniques to fit different situations and options, and offer different degrees of user interaction.

1.8 Issues in Data Mining

Data mining algorithms embody techniques that have sometimes existed for many years, but have only lately been applied as reliable and scalable tools that time and again outperform older classical statistical methods. While data mining is still in its infancy, it is becoming a trend and ubiquitous. Before data mining develops into a conventional, mature and trusted discipline, many still pending issues have to be addressed. Some of these issues are addressed below. Note that these issues are not exclusive and are not ordered in any way.

Security and social issues: Security is an important issue with any data collection that is shared and is intended to be used for strategic decision-making. In addition, when data is collected for customer profiling, user behavior understanding, correlating personal data with other information, etc., large amounts of sensitive and private information about individuals or companies is gathered and stored. This becomes controversial given the confidential nature of some of this data and the potential illegal access to the information. Moreover, data mining could disclose new implicit knowledge about individuals or groups that could be against privacy policies, especially if there is potential dissemination of discovered information. Another issue that arises from this concern is the appropriate use of data mining. Due to the value of data, databases of all sorts of content are regularly sold, and because of the competitive advantage that can be attained from implicit knowledge discovered, some important information could be withheld, while other information could be widely distributed and used without control.

User interface issues: The knowledge discovered by data mining tools is useful as long as it is interesting, and above all understandable by the user. Good data visualization eases the interpretation of data mining results, as well as helps users better understand their needs. The major issues related to user interfaces and visualization are "screen real-estate", information rendering, and interaction. Interactivity with the data and data mining results is crucial since it provides means for the user to focus and refine the mining tasks, as well as to picture the discovered knowledge from different angles and at different conceptual levels.

Mining methodology issues: These issues pertain to the data mining approaches applied and their limitations. Topics such as versatility of the mining approaches, the diversity of data available, the dimensionality of the domain, the broad analysis needs (when known), the assessment of the knowledge discovered, the exploitation of background knowledge and metadata, the control and handling of noise in data, etc. are all examples that can dictate mining methodology choices. For instance, it is often desirable to have different data mining methods available since different approaches may perform differently depending upon the data at hand. Moreover, different approaches may suit and solve user's needs differently.

Performance issues: Many artificial intelligence and statistical methods exist for data analysis and interpretation. However, these methods were often not designed for the very large data sets data mining is dealing with today. Terabyte sizes are common. This raises the issues of scalability and efficiency of the data mining methods when processing considerably large data. Algorithms with exponential and even medium-order polynomial complexity cannot be of practical use for data mining. Linear algorithms are usually the norm. In same theme, sampling can be used for mining instead of the whole dataset. However, concerns such as completeness and choice of samples may arise. Other topics in the issue of performance are *incremental updating*, and parallel programming. There is no doubt that parallelism can help solve the size problem if the dataset can be subdivided and the results can be merged later. Incremental updating is important for merging results from parallel mining, or updating data mining results when new data becomes available without having to re-analyze the complete dataset.

1.9 Scope of Data Mining

Data mining derives its name from the similarity between searching for valuable business information in a large database e.g. finding liked products in gigabytes. As real time systems have database of sufficient size and quality, data mining technology can be used for these purpose:

- **Automated discovery of previously unknown patterns**:

  Data mining sweeps through the database and identify previously hidden pattern in one step. For ex consider the case of retail sales. We have to identify seemingly unrelated products that are often purchased together. This could be like detecting fraud credit cards and anomalous data that could represent entry while entering data.

- **Automated prediction of trends and behavior:**

  Data mining automates the process of finding predictive information in large databases. Problems that require a lot of human analysis earlier can be easily answered using data mining methods. Data mining can be used in target marketing, forecasting bankruptcy and identifying segments of a population likely to respond similarly in given events.

  Data mining techniques can yield the benefits of automation on existing software and hardware platforms and can be implemented on the new systems, they can analyze massive database in less time. High speed makes the use of large database practical, which in turn point to improved predictions. Data mining has many fields of application. Some of which are:

Marketing:

1. Identifying buying patterns of customers.
2. Find association among customers demographic characteristic
3. Predict response to mailing campaigns
4. Market basket analysis.

Banking

1. Detect pattern of fraud credit card usage.
2. Identifying loyal customers
3. Identify stock market rules from historical market data.
4. Finding the hidden correlations between different financial indicators.
5. Determine credit card spending by customer group.
6. Predict customers likely to change their credit card company.

Medicine

1. Characterize patient behavior to predict clinical visit.
2. Identifying successful medical therapies for different illnesses.

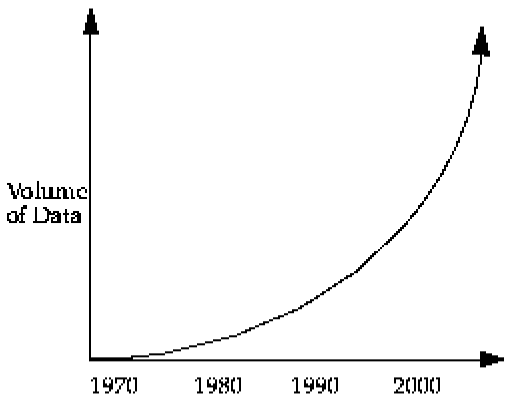
Insurance and Health Care

1. Claiming analysis that means which medical procedures are claimed together
2. Predict which customers will buy new policies.
3. Identifying behavior patterns of risky customer
4. Identifying fraud behavior

This dissertation is organized as follows. A detailed survey of earlier association rule algorithms is given in chapter 2. Then the basics of association rule data mining are given in chapter 3.chapter 4 gives the details of Signature method and Database pruning method algorithms proposed by us. Software requirement specification is given in chapter 5. Performance results are presented in chapter 6.chapter 7 contains the conclusion and the chapter 8 contains the further directions.

2. A SURVEY ON ASSOCIATION RULES

The past two decades has seen a dramatic increase in the amount of information or data being stored in electronic format. 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 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. Below graph illustrates the data explosion.



The Growing Base of Data

Data storage became easier and cheaper as the cost of computing power and electronic data storage devices decreased rapidly. Having concentrated so much attention on the accumulation of data the problem for organizations was what to do with this valuable resource. It was recognised 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. Traditional on-line transaction processing systems, 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.

Data Mining, or Knowledge Discovery in Databases (KDD) as it is also known, is the nontrivial extraction of implicit, previously unknown, and potentially useful information from data. This encompasses a number of different technical approaches, such as clustering, data summarization, learning classification rules, finding dependency net works, analyzing changes, and detecting anomalies.

Data mining analysis tends to work from the data up and the best techniques are those developed with an orientation towards large volumes of data, making use of as much of the collected data as possible to arrive at reliable conclusions and decisions. The analysis process starts with a set of data, uses a methodology to develop an optimal representation of the structure of the data during which time knowledge is acquired.

Once knowledge has been acquired this can be extended to larger sets of data working on the assumption that the larger data set has a structure similar to the sample data.

Again, this is analogous to a mining operation where large amounts of low-grade materials are sifted through in order to find something of value. Association rules are one of the most researched areas of data mining and have recently received much attention from the database community. They have proven to be quite useful in the marketing and retail communities as well as other more diverse fields.

Data Mining is the discovery of hidden information found in databases and can be viewed as a step in the knowledge discovery process. Data mining functions include clustering, classification, prediction, and link analysis (associations). One of the most important data mining applications is that of mining association rules. Association rules are used to identify relationships among a set of items in a database. These relationships are not based on inherent properties of the data themselves (as with functional dependencies), but rather based on co-occurrence of the data items. Association rules are often used by retail stores to analyze market basket transactions. The discovered association rules can be used by management to increase the effectiveness (and reduce the cost) associated with advertising, marketing, inventory, and stock location on the floor. Association rules are also used for other applications such as prediction of failure in telecommunications networks by identifying what events occur before a failure.

2.1 ASSOCIATION RULE PROBLEM:

Definition 1: Let I ={I1, I2, … , Im} be a set of m distinct attributes, also called *literals*. Let D be a database, where each record (tuple) T has a unique identifier, and contains a set of items such that $T \subseteq I$ An *association rule* is an implication of the form $X \Rightarrow Y$, where X, $Y \subset I$, are sets of items called *item sets*, and $X \cap Y = \varphi$. Here, X is called antecedent, and Y consequent.

Two important measures for association rules, support (s) and confidence (α), can be defined as follows.

Definition 2: The *support (s)* of an association rule is the ratio (in percent) of the records that contain $X \cup Y$ to the total number of records in the database.

Therefore, if we say that the support of a rule is 5% then it means that 5% of the total records contain $X \cup Y$.

Definition 3: For a given number of records, *confidence (α)* is the ratio (in percent) of the number of records that contain $X \cup Y$ to the number of records that contain X.

Thus, if we say that a rule has a confidence of 85%, it means that 85% of the records containing X also contain Y. The confidence of a rule indicates the degree of correlation in the dataset between X and Y. Confidence is a measure of a rule's strength. Often a large confidence is required for association rules.
Mining of association rules from a database consists of finding all rules that meet the user-specified threshold support and confidence. The problem of mining association rules can be decomposed into two sub problems.

Algorithm 1. Basic:

Input: I, D, s, α
Output: Association rules satisfying s and α
Algorithm:

1) Find all sets of items which occur with a frequency that is greater than or equal to the user-specified threshold support, s.

2) Generate the desired rules using the large itemsets, which have user-specified threshold confidence, α.

The first step in Algorithm 1 finds *large* or *frequent itemsets*. Itemsets other than those are referred as *small itemsets*. Here an item set is a subset of the total set of items of

interest from the database. An interesting (and useful) observation about large itemsets is that:
If an item set X is small, any superset of X is also small. Of course the contra positive of this statement (If X is a large item set than so is any subset of X) is also important to remember.

Algorithm 2. Find Association Rules Given Large Itemsets:
Input: I, D, s, α, L
Output: Association rules satisfying s and α
Algorithm:

1) Find all nonempty subsets, *x*, of each large item set, $l \in L$

3) For every subset, obtain a rule of the form $x \Rightarrow (l\text{-}x)$ if the ratio of the frequency of Occurrence of *l* to that of x is greater than or equal to the threshold confidence.

Since finding large itemsets in a huge database is very expensive and dominates the overall cost of mining association rules, most research has been focused on developing efficient algorithms to solve step 1 in Algorithm 1.

2.2 BASIC ALGORITHMS

In most cases, it is assumed that the itemsets are identified and stored in lexicographic order (based on item name). This ordering provides a logical manner in which itemsets can be generated and counted.

2.2.1 Sequential Algorithms

2.2.1.1 AIS

This technique is limited to only one item in the consequent. That is, the association rules are in the form of $X \Rightarrow Ij \mid \alpha$, where X is a set of items and Ij is a single item in the domain I, and α is the confidence of the rule.

The AIS algorithm makes multiple passes over the entire database. During each pass, it scans all transactions. In the first pass, it counts the support of individual items and determines which of them are large or frequent in the database. Large itemsets of each pass are extended to generate candidate itemsets. After scanning a transaction, the common itemsets between large itemsets of the previous pass and items of this transaction are determined. These common itemsets are extended with other items in the transaction to generate new candidate itemsets. A large item set *l* is extended with only those items in the transaction that are large and occur in the lexicographic ordering of items later than any of the items in *l*. To perform this task efficiently, it uses an estimation tool and pruning technique. The estimation and pruning techniques determine candidate sets by omitting unnecessary itemsets from the candidate sets. Then, the Support of each candidate set is computed. Candidate sets having supports greater than or equal to min support are chosen as large itemsets. These large itemsets are extended to generate candidate sets for the next pass. This process terminates when no more large itemsets are found.

2.2.1.2 Apriori

It is by far the most well known association rule algorithm. This technique uses the property that any subset of a large item set must be a large item set. Also, it is assumed that items within an item set are kept in lexicographic order. The Apriori generates the candidate itemsets by joining the large itemsets of the previous pass and deleting those subsets, which are small in the previous pass without considering the transactions in the database. By only considering large itemsets of the previous pass, the number of candidate large itemsets is significantly reduced.

In the first pass, the itemsets with only one item are counted. The discovered large itemsets of the first pass are used to generate the candidate sets of the second pass using the apriori_gen() function. Once the candidate itemsets are found, their supports are counted to discover the large itemsets of size two by scanning the database. In the third pass, the large itemsets of the second pass are considered as the candidate sets to discover large itemsets of this pass. This iterative process terminates when no new large itemsets are found. Each pass *i* of the algorithm scans the database once and determines large itemsets of size *i*. Li denotes large itemsets of size *i*, while Ci is candidates of size *i*.

The apriori_gen() function has two steps. During the first step, L_{K-1} is joined with itself to obtain Ck. In the second step, apriori_gen() deletes all itemsets from the join result, which have some (k-1)–subset that is not in L_{K-1}. Then, it returns the remaining large k-itemsets.

```
Method: apriori_gen()
Input: set of all large (k-1)-itemsets $L_{K-1}$
Output: A superset of the set of all large k-itemsets
//Join step
Ii = Items i
insert into $C_K$

 Select $p.I_1, p.I_2, \ldots\ldots, p.I_{K-1}, q.I_{K-1}$
 From $L_{K-1}$ is p, $L_{K-1}$ is q
 Where $p.I_1 = q.I_1$ and …… and $p.I_{K-2} = q.I_{K-2}$ and $p.I_{K-1} < q.I_{K-1}$.
 //pruning step
 forall itemsets $c \in C_K$ do
 forall (k-1)-subsets s of c do
 If ($s \notin L_{K-1}$) then
 delete c from $C_K$
```

The subset () function returns subsets of candidate sets that appear in a transaction. Counting support of candidates is a time-consuming step in the algorithm.

Algorithm 3 shows the Apriori technique. As mentioned earlier, the algorithm proceeds iteratively.

```
Function Count(C: a set of itemsets ,D :database)
    Begin
        for each transaction $T \in D = \cup D^i$ do begin
            forall subsets $x \subseteq T$ do
             If $x \in C$ then
                 x.count++;
        end
    end
```

Algorithm 3:**Apriori**
Input:
I , D , s
Output:
L

```
Algorithm:
//procedure large Itemsets
1)$C_1$:=I; //Candidate 1-Itemsets
2) Generate $L_1$by traversing database and counting each occurrence of an attribute in a transaction
3) for(K=2;$L_{k-1} \neq \varphi$ ;k++)
//Candidate itemsets generation
//New K-Candidate itemsets are generated from (k-1) large itemsets
4) $C_k$=apriori_gen($L_{k-1}$)
// counting support of $C_k$
5) count ( $C_k$,D)
6) $L_k = \{C \in C_k | \text{c.count} \geq \text{minsup}\}$
7) end
8) L:= $\cup_k L_k$
```

Apriori always outperforms AIS. Apriori incorporates buffer management to handle the fact that all the large itemsets L_{K-1} and the candidate itemsets C_K need to be stored in the candidate generation phase of a pass k may not fit in the memory. A similar problem may arise during the counting phase where storage for C_K and at least one page to buffer the database transactions are needed. We considered two approaches to handle these issues. At first they assumed that L_{K-1} fits in memory but C_K does not. The authors resolve this problem by modifying apriori_gen() so that it generates a number of candidate sets C_K' which fits in the memory. Large itemsets L_K resulting from C_K' are written to disk, while small itemsets are deleted. This process continues until all of C_K has been measured. The second scenario is that L_{K-1} does not fit in the memory. This problem is handled by sorting L_{K-1} externally. A block of L_{K-1} is brought into the memory in which the first (k-2) items are the same. Blocks of L_{K-1} are read and candidates are generated until the memory fills up. This process continues until all C_K has been counted

2.2.1.3 Sampling Algorithm:

It is used to facilitate efficient counting of itemsets with large database. It reduces the number of database scans to one in the best case and two in the worst case. Here first any algorithm like Apriori is used to find the large itemsets in the sample .The set of large itemsets is used as to find the large itemsets in the sample. The set of large itemsets is used as a set of candidates during a scan of the entire database. This results in counting of all candidates. During the second scan, additional candidates are generated and counted. This is done to ensure that all large itemsets are found.

2.2.1.4 Partitioning

PARTITION reduces the number of database scans to 2. It divides the database into small partitions such that each partition can be handled in the main memory. Let the partitions of the database be D^1, D^2, ..., D^p. In the first scan, it finds the *local large itemsets* in each partition D^i ($1 \leq i \leq p$), i.e. $\{X \mid X.count \geq s \times |D^i|\}$. The local large itemsets, L^i, can be found by using a level-wise algorithm such as Apriori. Since each partition can fit in the main memory, there will be no additional disk I/O for each partition after loading the partition into the main memory. In the second scan, it uses the property that a large itemsets in the whole database must be locally large in at least one partition of the database. Then the union of the local large itemsets found in each partition is used as the candidates and are counted through the whole database to find all the large itemsets.

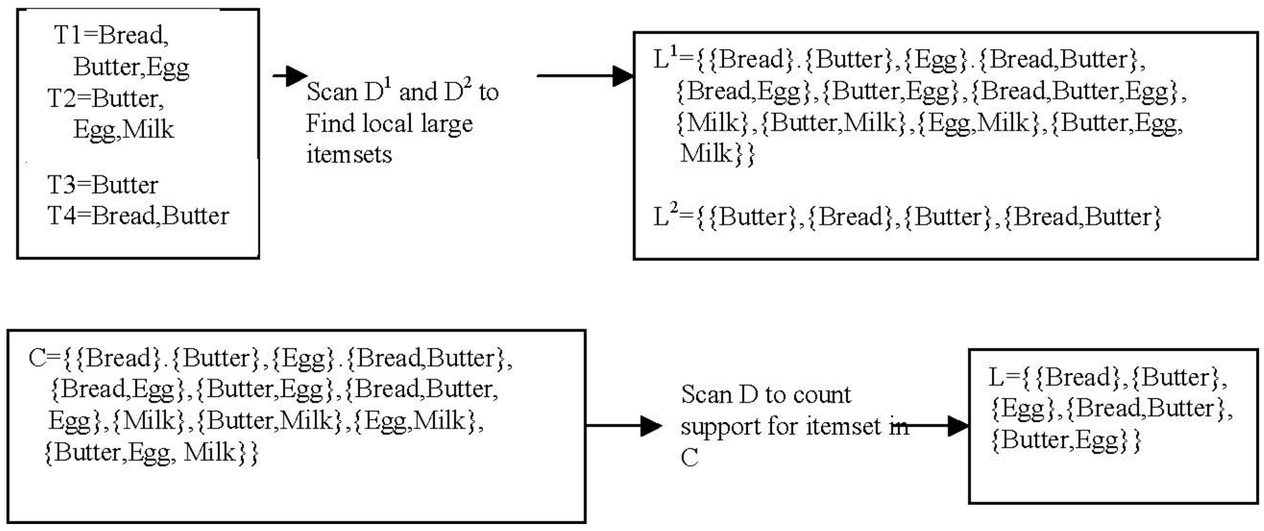

T1=Bread, Butter,Egg
T2=Butter, Egg,Milk
T3=Butter
T4=Bread,Butter
Scan D1 and D2 to Find local large itemsets
L1={{Bread}.{Butter},{Egg}.{Bread,Butter}, {Bread,Egg},{Butter,Egg},{Bread,Butter,Egg}, {Milk},{Butter,Milk},{Egg,Milk},{Butter,Egg, Milk}}
L2={{Butter},{Bread},{Butter},{Bread,Butter}
C={{Bread}.{Butter},{Egg}.{Bread,Butter}, {Bread,Egg},{Butter,Egg},{Bread,Butter, Egg},{Milk},{Butter,Milk},{Egg,Milk}, {Butter,Egg, Milk}}
Scan D to count support for itemset in C
L={{Bread},{Butter}, {Egg},{Bread,Butter}, {Butter,Egg}}

Figure 2 Discovering Large Itemsets using the PARTITION Algorithm

If the database is divided into two partitions, with the first partition containing the first two transactions and the second partition the remaining two transactions. Since the minimum support is 40% and there are only two transactions in each partition, an item set, which occurs once, will be large. Then the local large itemsets in the two partitions are just all subsets of the transactions. Their union is the set of the candidate itemsets for the second scan. The algorithm is shown in Algorithm 4. Note that we use superscripts to denote the database partitions, and subscripts the sizes of the itemsets.

Algorithm4.PARTITION

Input**:**

$I, s, D^1, D^2, ..., D^p$

Output:

L

Algorithm:

```
//scan one computes the local large itemsets in each partition
1) for i from 1 to p do
2) $L^i$ = Apriori (I,$D^i$,s); //$L^i$ are all local large itemsets(all sizes) in $D^i$
   //scan two counts the union of the local large itemsets in all partitions
3) C = $\cup$i $L^i$;
4) count(C, D) = $\cup$ $D^i$;
5) return L = {x | x$\in$ C, x.count $\geq$s $\times$ |D|};
```

PARTITION favors a homogeneous data distribution. That is, if the count of an item set is evenly distributed in each partition, then most of the itemsets to be counted in the second scan will be large. However, for a skewed data distribution, most of the itemsets in the second scan may turn out to be small, thus wasting a lot of CPU time counting false itemsets.

2.2.2 Parallel and Distributed Algorithms

The current parallel and distributed algorithms are based on the serial algorithm Apriori. An excellent survey classifies the algorithms by load-balancing strategy, architecture and parallelism. Here we focus on the parallelism used: *data parallelism* and *task parallelism*. The two paradigms differ in whether the candidate set is distributed across the processors or not. In the data parallelism paradigm, each node counts the same set of candidates. In the task parallelism paradigm, the candidate set is partitioned and distributed across the processors, and each node counts a different set of candidates. The database, however, may or may not be partitioned in either paradigm theoretically. In practice for more efficient I/O it is usually assumed the database is partitioned and distributed across the processors.

In the data parallelism paradigm, a representative algorithm is the count distribution algorithm. The candidates are duplicated on all processors, and the database is distributed across the processors. Each processor is responsible for computing the *local support counts* of all the candidates, which are the support counts in its database partition. All processors then compute the *global support counts* of the candidates, which are the total support counts of the candidates in the whole database, by exchanging the local support counts (Global Reduction). Subsequently, large itemsets are computed by each processor independently. The data parallelism paradigm is illustrated in Figure 2. The four transactions are partitioned across the three processors with processor 3 having two transactions T3 and T4, processor 1 having transaction T1 and processor 2 having transaction T2. The three candidate itemsets in the second scan are duplicated on each processor. The local support counts are shown after scanning the local databases.

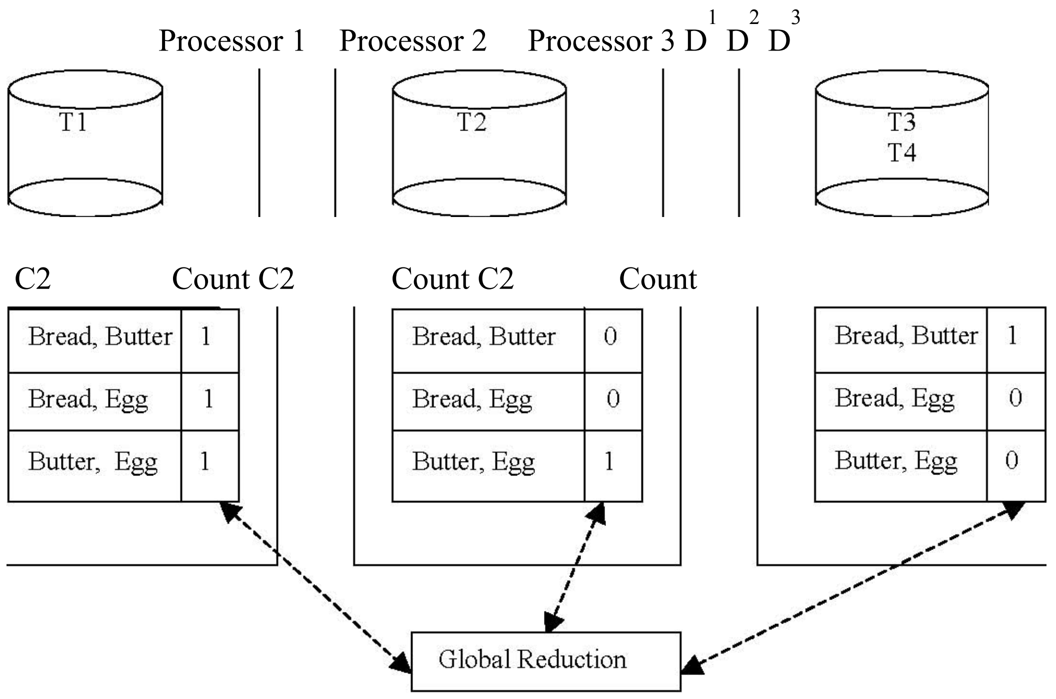


Figure 2 Data Parallelism Paradigm

In the task parallelism paradigm, the candidate set is partitioned and distributed across the processors, as is the database. Each processor is responsible for keeping the global support counts of only a subset of the candidates. This approach requires two rounds of communication at each iteration. In the first round, every processor sends its database partition to all the other processors. In the second round, every processor broadcasts the large itemsets that it has found to all the other processors for computing the candidates for the next iteration. The task parallelism paradigm is shown in Figure 3 . The four transactions are partitioned as in data parallelism. The three candidate itemsets are partitioned across the processors with each processor having one candidate item set. After scanning the local database and the database partitions broadcast from the other processors, the global count of each candidate is shown.

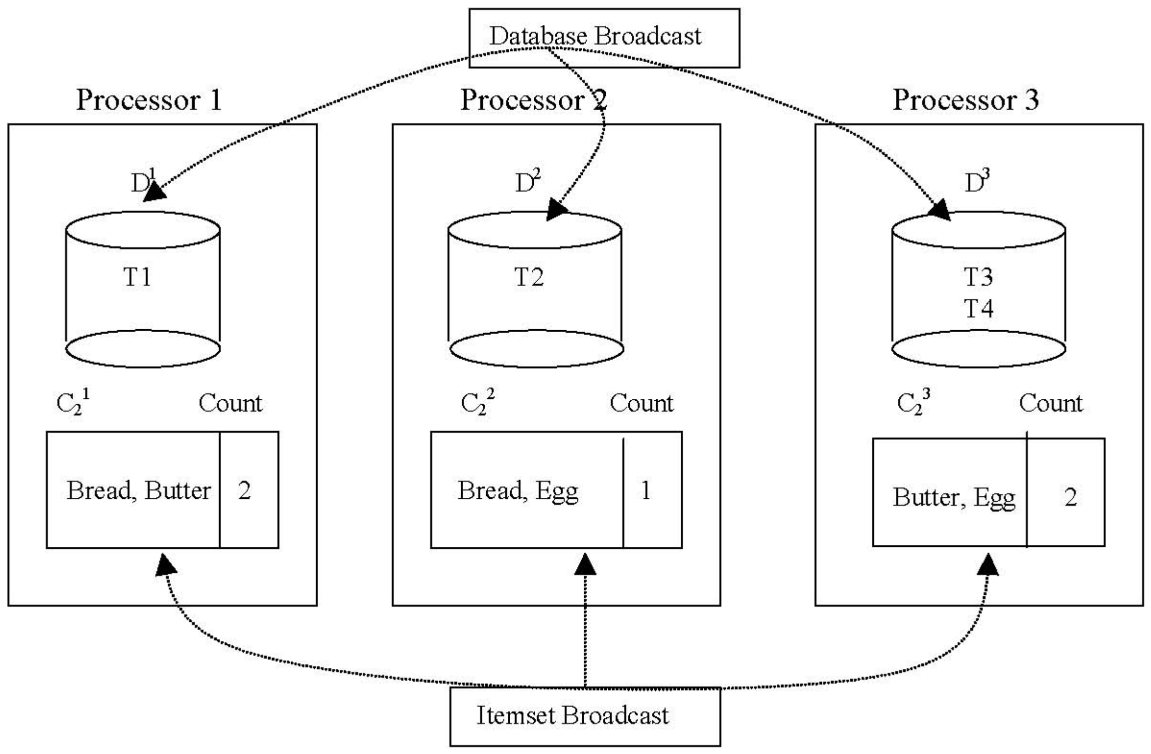


Figure 3 Task parallelism paradigms

Discussion

Both data and task paradigms have advantages and disadvantages. They are appropriate for certain situations. The data parallelism paradigm has simpler communication and thus less communication overhead; it only needs to exchange the local counts of all candidates in each iteration. However, the data parallelism paradigm requires that all the candidates fit into the main memory of each processor.

The task parallelism paradigm was initially proposed to efficiently utilize the aggregate main memory of a parallel computer. It partitions and distributes the candidates among the processors in each iteration, so it utilizes the aggregate main memory of all processors and may not have the insufficient main memory problem with the number of processors increasing. Therefore, it can handle the mining problem with a very low minimum support. However, the task parallelism paradigm requires movement of the database partitions in addition to the large item set exchange. Usually the database to be mined is very large, so the movement of the database will introduce tremendous communication overhead. Thus, it may be a problem when the database is very large. In the basic data distribution algorithm, as the database partition on each processor is broadcasted to all others, the total message for database movement is $O(p^2)$, where p is the number of processors involved.

2.2.3 EXTENDED ASSOCIATION RULES

Association rule algorithms presented in previous sections generate all association rules satisfying given confidence and support requirements. There have been algorithms, which either generate rules under other requirements or extend the basic definition of what an association rule is. We examine the body of work to extend the basic algorithms in this section.

2.2.3.1 Generalized Association Rules

Generalized association rules use the existence of a *hierarchical taxonomy (concept hierarchy)* of the data to generate different association rules at different levels in the taxonomy. Figure 4 shows an example of taxonomy on market basket data. Here beverage is further divided into coffee, tea, soft drinks, and juice. Juice is divided into orange, apple, cranberry, and grape. When association rules are generated, we could generate them at any of the hierarchical levels present. As would be expected, when rules are generated for items at a higher level in the taxonomy, both the support and confidence increase. In a given transaction database, there may be multiple taxonomies for different items and even multiple taxonomies for the same item. A *generalized association rule*, $X \Rightarrow Y$, is defined identically to that of regular association rule, except that no item in Y can be an ancestor of any in X. An *ancestor* of an item is one, which is above it in some taxonomy. A supermarket may want to find associations_relating to soft drinks in general or may want to identify those for a specific brand or type of soft drink (such as a cola). The generalized association rules allow this to be accomplished and also ensure that all association rules (even those across levels in different taxonomies are found.

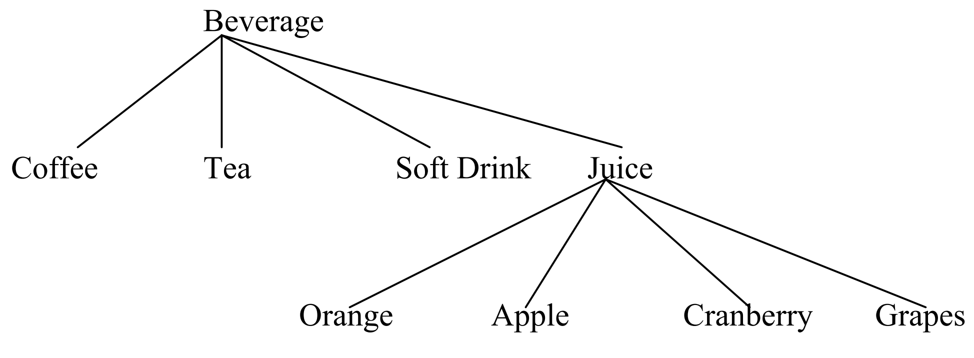


Figure 4 Market Basket Taxonomy

The generalized association rule problem is to generate association rules for all levels of all taxonomies. One approach to do this would be to take each transaction and expand each item to include all items above it in any hierarchy. This naïve approach is quite

Expensive and other more efficient algorithms have been proposed. One algorithm, Cumulate, uses several optimization strategies to reduce the number of ancestors which need to be added to each transaction. Another approach, Stratification, counts itemsets by their levels in the taxonomy and uses relationships about items in taxonomy to reduce the number of items to be counted. Several parallel *algorithms* to generate generalized association rules have also been proposed.

When association rules are generated from across different levels in the concept hierarchy, they are called *multiple-level association rules*.

3. Basics of Association Rule Mining

•Search patterns given as association rules of the form

Body $\Rightarrow$Head [support, confidence]

• Examples

– Buys (x, “diapers”) $\Rightarrow$ buys (x, “beers”) [0.5%, 60%]

– Major (x, “CS”) $\wedge$ takes (x, “DB”) $\Rightarrow$ grade (x , “A”) [1%, 75%]

• Problem: Given

(1) Database of transactions

(2) Each transaction is a list of items

Find: *all* rules that correlate the presence of one set of items with that of another set of items.

Association rule mining is a technique for discovering unsuspected data dependencies and is one of the best-known data mining techniques. The basic idea is to identify from a given database, consisting of itemsets (e.g. shopping baskets), whether the occurrence of specific items, implies also the occurrence of other items with a relatively high probability. In principle the answer to this question could be easily found by exhaustive exploration of all possible dependencies, which is however prohibitively expensive. Association rule mining thus solves the problem of how to search efficiently for those dependencies.

3.1 Single vs. Multidimensional Association Rules

- Single-dimensional rules

  buys (X, “milk”)$\Rightarrow$ buys (X, “bread”)

- Multi-dimensional rules: more than 2 dimensions or predicates

  age (X,”19-25”) $\wedge$ buys(X, “popcorn”) $\Rightarrow$ buys(X, “coke”)

- Transformation into single-dimensional rules: use predicate/value pairs as items

  customer (X, [age, ”19-25”]) $\wedge$ customer(X, [buys, “popcorn”]) $\Rightarrow$ customer(X, [buys ,“coke”])

- Simplified Notation for single dimensional rules

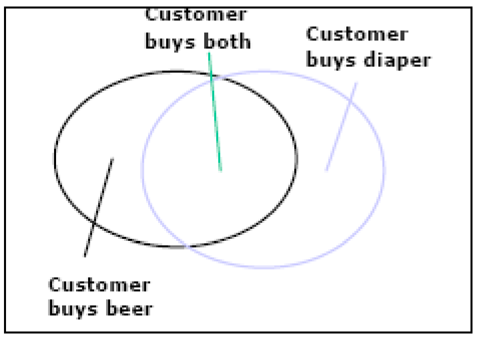
  { milk } $\Rightarrow$ { bread }

  {[age, ”19-25”], [buys, “popcorn”]} $\Rightarrow$ {[buys ,“coke”]}

In the "logical" notation we have used before in order to express association rules, it was possible to establish dependencies among different types of predicates applied to the items. These general types of rules are called multidimensional association rules. However, it is straightforward to transform multi-dimensional association rules into single-dimensional rules, by considering different predicates applied to the same items as different items. Therefore in the following we will only consider single-dimensional association rules.

3.2 Support and Confidence:

Support and Confidence



Transaction ID	Items Bought
2000	beer, diaper, milk
1000	beer, diaper
4000	beer, milk
5000	milk, eggs, apple

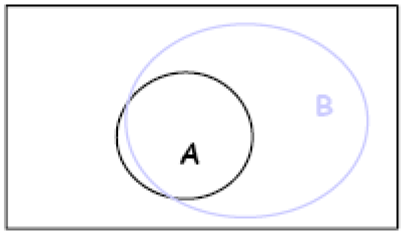
Let minimum support 50%,
and minimum confidence 50%

buys(x,"beer") ⇒ buys(x,"diaper") [50%, 66.6%]
buys(x,"diaper") ⇒ buys(x,"beer") [50%, 100%]

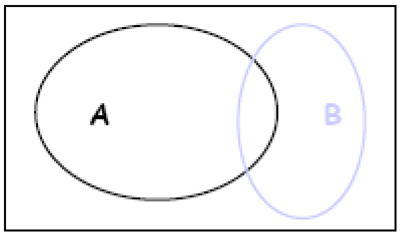
This example illustrates the basic notions used in association rule mining: transactions, item sets, support and confidence. Transaction consists of a transaction identifier and an item set. The item set is the set of items that occur jointly in a transactions (e.g. the items bought).
Support is the number of transactions in which the association rule holds, i.e. in which all items of the rule occur (e.g. both beer and diaper). If this number is too small, probably the rule is not important or accidentally true.
Confidence is the probability that in case the head of the rule (the condition) is satisfied also the body of the rule (the conclusion) is satisfied. This indicates to which degree the rule is actually true, in the cases where the rule is applicable.

Support and Confidence: Possible Situations

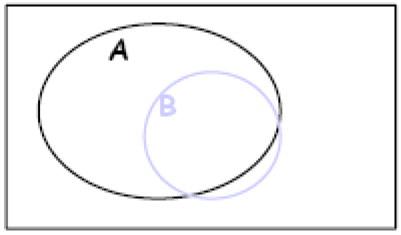
- Assume support for A ∪ B is high (above threshold)



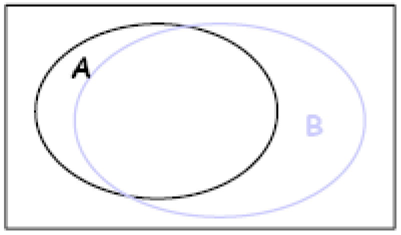
Conf(A ⇒ B) high
Conf(B ⇒ A) low



Conf(A ⇒ B) low
Conf(B ⇒ A) low



Conf(A ⇒ B) low
Conf(B ⇒ A) high



Conf(A ⇒ B) high
Conf(B ⇒ A) high

This figure illustrates the meaning and importance of the "directionality" of association rules. We assume that in all cases the intersection areas (i.e. the support) are above the required threshold. Then four cases are possible as shown. Thus association rules not only express a high probability of co-occurrence of items, such as in the last case, but also conditional dependencies among the occurrences of items (or inclusion relationships).

3.3 Definition of Association rules:

Terminology and notation:
-Set of all items I , subset of I is called itemsets.
-Transaction (tid , T), T $\subseteq$ I itemsets , transaction identifier tid
-Set of all transactions D(database), Transaction T$\in$ D.

Definition of association rules A $\Rightarrow$ B [s,c]
A,B itemsets (A,B $\subseteq$ I)
A $\cap$ B = φ
Support s = probability that a transaction contain A $\cup$ B
= P (A $\cup$ B)
Confidence c =conditional probability that a transaction having A also contain B
=P (B|A)

Example: Items I ={apple, beer, diaper, eggs, milk}
Transaction (2000, {beer, diaper, eggs, milk})
Association Rules {beer} $\Rightarrow$ {diaper} [0.5, 0.66]

This is a summary of the basic notations and notions used in association rule mining.

3.4 Frequent Item Sets:

Transaction ID	Item Bought
2000	beer, diaper, milk
1000	beer, diaper
4000	beer, milk
5000	milk, egg, apple

Frequent itemsets	Support
{beer}	0.75
{milk}	0.75
{diaper}	0.5
{beer, milk}	0.5
{beer, diaper}	0.5

1.A-> B can only be an association rule if A$\cup$B is a frequent item set
-Search for the frequent item set

2.Any subset of a frequent item set is also a frequent item set
(Apriori Property->Apriori Algorithm)
- e.g., if { beer , diaper is a frequent item set , both {beer}and {diaper} are frequent itemsets.

3.Therefore iteratively find frequently itemsets with increasing cardinality from 1 to K (K- itemsets).
- Reduces the number of possible candidates in search for large frequent item set.

Here we summarize the most important ideas that will allow searching for Association rules efficiently.
First, a necessary condition for finding an association rule of forms A->B is sufficiently high support. Therefore, for finding such rules, we have first to find item sets within the transactions that occur sufficiently frequent. These are called *frequent item sets*.
Second we can observe that any subset of a frequent item set is necessarily also a frequent item set (this is called the apriori property).
Third, we can exploit this observation in order to reduce the number of item sets that need to be considered in the search. Once frequent item sets of lower cardinality are found, only item sets of larger cardinality need to be considered that contain one of the frequent item sets already found. This allows reducing the search space drastically.

3.5 Apriori Algorithm:

Given a set of items I={I1,I2, ………,Im} and a database of transaction D={t1,t2…..tn} where ti={Ii1,Ii2,Ii3,……Iik} and Iij $\in$ I , an association rule is an implication of the form X=>where X,Y $\subset$ I are sets of itemsets and X $\cap$Y = ϕ .The standard measures to access association rules are the support and confidence of a rule, both of which are computed from the support of certain item sets.
Given a set of transaction D , the problem of mining association rules is to generate all association rules that have support and confidence greater than user specified minimum support (called minsup) and minimum confidence (called mincof) respectively.
The main problem of association rules induction is that there are so many possible rules. For example, for the product range of a supermarket, which may consist of several thousands different products, there are billions of possible association rules. It is obvious that such a vast amount of rules cannot be processed by inspecting each one in turn.
Therefore efficient algorithms are needed that restrict the search space and check only a subset of all rules, but, if possible, without missing important rules.

Problem Decomposition:
The problem of discovering all association rules can be decomposed into two sub Problems:
1.Finding all sets of item (items) that have transaction support above minimum support . The support for an item set is the number of transaction that contains the itemsets. Itemsets with minimum support are called large itemsets, and all other small itemsets.
2.Use the large itemsets to generate the desired result.

Discovering Large Itemsets:

Algorithm for discovering large itemsets make multiple passes over the data. In the first pass, we count the support of individual items and determine which of them are large, i.e. have minimum support .In each subsequent pass, we start with a seed set of itemsets found to be large in the previous pass .we use this seed set for generating new potentially large itemsets, called candidate itemsets, and count the actual support for these candidate

itemsets during the pass over the data. At the end of the pass, we determine which of the candidate itemsets are actually large and they become the seed for the next pass. This process continues until no new large item set is found. The Apriori and Apriori Tid algorithm differ fundamentally from the AIS and SETM algorithms in terms of which candidate itemsets are in a pass and in the way that those candidates are generated. Specifically, after reading a transaction, it is determined which of the itemsets found large in the previous pass are present in the transaction. New candidate itemsets are generated by extending these large itemsets with other items in the transaction.
The Apriori and AprioriTid algorithms generate the candidate item set to be counted in a pass by using only the itemsets found large in the previous pass with out considering the transaction in the database. The basic intuition is that any subset of a large itemsets must be large. Therefore, the candidate itemsets having k items can be generated by joining large itemsets having k-1 items, and deleting those that contain any subset that is not large. This procedure results in generation of a much smaller number of candidate item set. The AprioriTid algorithm has the additional property that the database is not used at all for counting the support of the candidate item set after the first pass. Rather an encoding of the candidate itemsets used in the previous pass is employed for this purpose. In the later passes, the size of this encoding can become much smaller than the database, thus saving much reading effort.

Apriori algorithm is an influential algorithm for mining frequent itemsets for Boolean association rules. This algorithm contains a number of passes over the database. During pass k, the algorithm finds the set of frequent itemsets L_k of length k that satisfy the minimum support requirement. The algorithm terminates when L_k is empty. A pruning step eliminates any candidate, which has a smaller subset.

The pseudo code for Apriori Algorithm is following:

C_k: Set of candidate item set of size k (potentially large/ frequent itemset) . Each member of this set has two fields: - item set, support count
L_k: frequent item set of size k(those with minimum support). Each member of this set has two fields:- item set, support count

```
L_1 = {frequent items};
For (k=1; L_k != null; k++) do begin
    C_{k+1} = candidates generated from L_k;
    For each transaction t in database do
        Increment the count of all candidates in
        C_{k+1} that are contained in t
    L_{k+1} = candidates in C_{k+1} with min_support
    End
Return L_k;
```

3.5.1 Exploiting the apriori properties:

Exploiting the Apriori Property (1):

- If we know the frequent (K-1)- itemsets, which are candidates for being frequent K-itemsets?

a b c d e f g h j

Frequent K-itemsets?

a b c d e f g h

Frequent (K-1) –item set.

a b c d e f g j

Frequent (K-1)-itemsets

-If we know all frequent (K-1)- itemsets L_{K-1}, then we can construct a candidate set L_K for frequent K-itemsets by joining two frequent (K-1)-itemsets that differ by exactly 1 item:
Join step

- Only these itemsets can be frequent K-itemsets.

Assume that we know frequent itemsets of size k-1. Considering a k-item set we can immediately conclude that by dropping two different items we have two frequent (k-1) itemsets. From another perspective this can be seen as a possible way to construct k-itemsets. We take two (k-1) item sets, which differ only by one item and take their union. This step is called the join step and is used to construct potential frequent k-itemsets.

Algorithm for creating candidates

Suppose the items in the L_{K-1} are increasingly sorted in a list , Then C_K is created through the following SQL statement

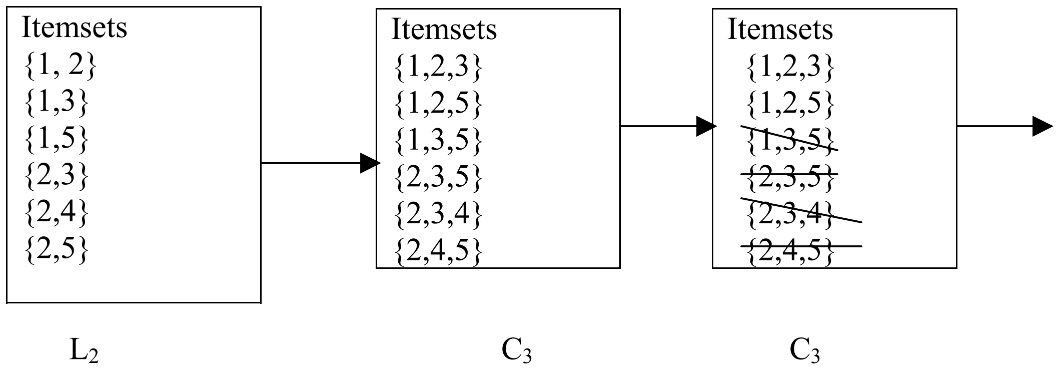
Insert into C_k
Select p.$item_1$, p.$item_2$,………,p.$item_{K-1}$,q.$item_{K-1}$
From L_{K-1} p , L_{K-1} q
Where p.$item_1$ = q.$item_1$,p.$item_2$=q.$item_2$,p.$item_3$= q.$item_3$…… , p.$item_{K-1}$ < q.$item_{K-1}$

$Item_2$	-	$Item_{k-2}$	$Item_{k-1}$
Item $_{2,1}$	-	$Item_{k-2,1}$	$Item_{k-1,1}$
Item $_{2,1}$	-	$Item_{k-2,1}$	$Item_{k-1,2}$
Item $_{2,1}$	-	$Item_{k-2,1}$	$Item_{k-1,2}$
Item $_{2,1}$	-	$Item_{k-2,1}$	$Item_{k-1,p1}$
Item $_{2,2}$	-	$Item_{k-2,2}$	$Item_{k-1,p1+1}$
Item $_{2,2}$	-	$Item_{k-2,2}$	$Item_{k-1,p1+2}$
Item $_{2,2}$	-	$Item_{k-2,2}$	$Item_{k-1,p2}$

We may express the step of creating all possible combinations of (k-1) itemsets by the SQL query shown above, assuming the itemsets are stored as relations with attributes $item_1$,…, $item_k$. The table illustrates of how these combinations are obtained. For each subset of items that share the first k-2 items, we construct all possible k-itemsets, by taking all-possible, ordered pairs from the last column.

Exploiting the Apriori Property (2)

- A candidate item set still not necessarily satisfies the apriori property



-After generating the candidate set C_K eliminate all the item set for which not all (K-1)-itemsets are elements of L_{K-1}, i.e. are frequent (K-1) itemsets: prune step.

-Only then count the remaining candidate K-itemsets in the database and eliminate those that are not frequent.

The k-itemsets constructed in the join step not necessarily are frequent k-itemsets. One possible reason is that they contain some subset of items, which is not frequent. These are eliminated in a prune step, by considering all itemsets of lower cardinality that have been constructed earlier.

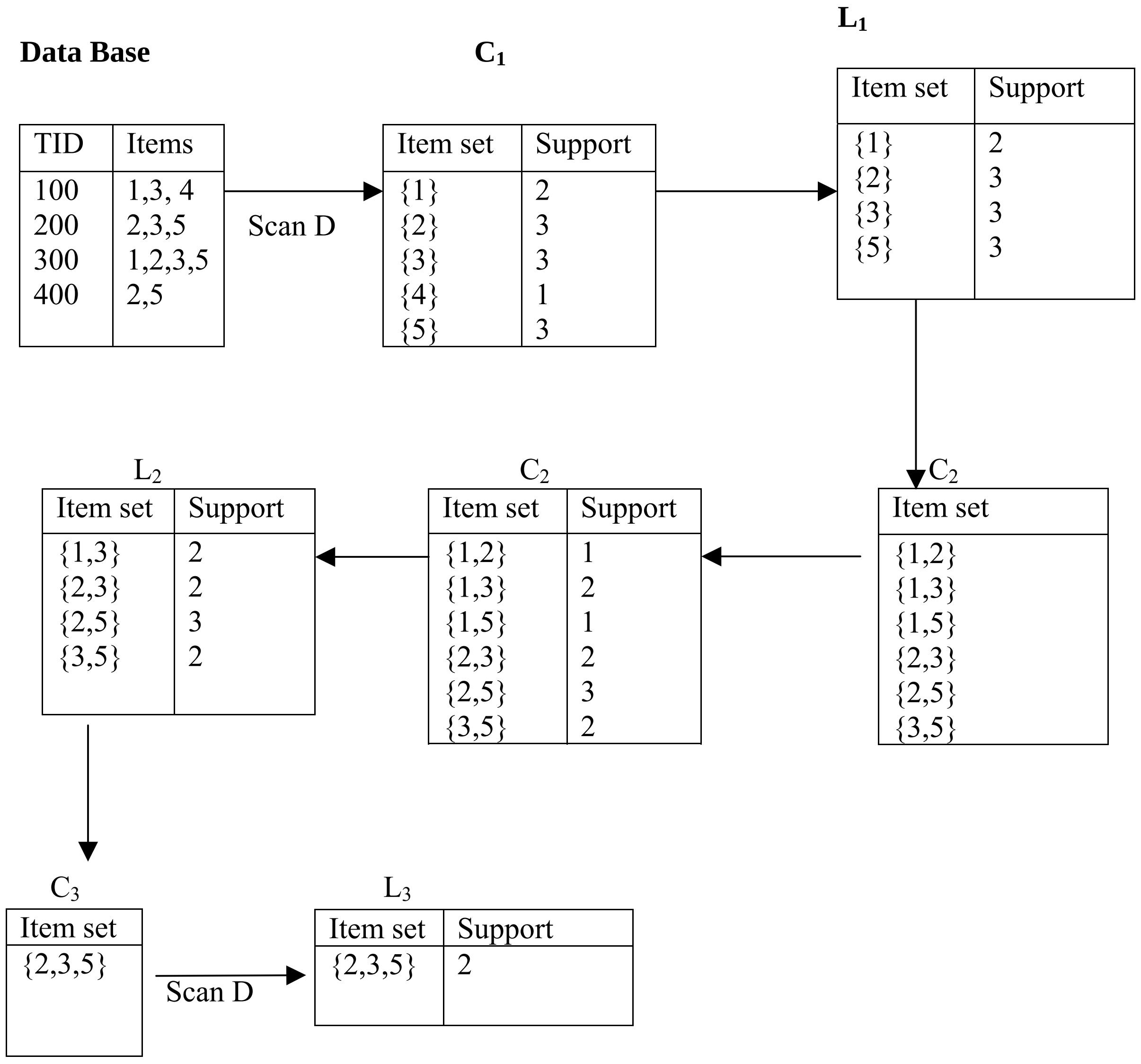
After that, it is still possible that among the remaining k-itemsets some are not frequent when determining their frequency in the database. The important point is that this last check, which is expensive, as it requires access to the complete database, needs to be performed for much fewer itemsets, since many possibilities have been eliminated in the join and prune step.

This is the complete apriori algorithm for determining frequent itemsets.

```
Generating frequent item sets : Apriori algorithm
K:=1; $L_k$:={frequent items in D};
While $L_k$ !=$\phi$
{
        $C_{k+1}$:= candidate generated from $L_k$ by joining and pruning

        For each transaction T in the database
        Increment the count of all candidate item sets in $C_{k+1}$
        that are contained in T;
        $L_{k+1}$ := candidate in  $C_{k+1}$ with minsup ;
        K : = k+1;
}
return $\cup_k L_k$ ;
```

Example of the Apriori algorithm:



Data Base

TID	Items
100	1,3, 4
200	2,3,5
300	1,2,3,5
400	2,5

C_1

Item set	Support
{1}	2
{2}	3
{3}	3
{4}	1
{5}	3

L_1

Item set	Support
{1}	2
{2}	3
{3}	3
{5}	3

C_2

Item set
{1,2}
{1,3}
{1,5}
{2,3}
{2,5}
{3,5}

C_2

Item set	Support
{1,2}	1
{1,3}	2
{1,5}	1
{2,3}	2
{2,5}	3
{3,5}	2

L_2

Item set	Support
{1,3}	2
{2,3}	2
{2,5}	3
{3,5}	2

C_3

Item set
{2,3,5}

L_3

Item set	Support
{2,3,5}	2

Notice in this example of how the scan steps (when determining the frequency with respect to the database) eliminate certain items. Notice that in this example pruning does not apply.

Generating Association Rules from Frequent Itemsets
For each frequent item set L generate all non-empty subsets S.
For every nonempty subset S output the rules $S \Rightarrow L / S$ if

$(sc(L))/(sc(S)) \geq min\_conf$.

sc = Support count
min_conf = minimum confidence
Confidence $(A \Rightarrow B) = P(B|A)$
Since

$$(sc(A \cup B)) / (sc(A)) \geq min\_conf$$

Once the frequent itemsets are found the derivation of association rules is straightforward: one checks for every frequent item set whether there exists a subset S that can occur as the head of a rule. For doing that, the support count, i.e. the frequency of the item set in the database, which was obtained during the execution of the apriori algorithm, is used to compute the confidence (as a conditional probability). Note that also L/S is a frequent item set, and therefore the support count is available for that set from the apriori algorithm

3.5.2 Improving Apriori Algorithm Efficiency

• Transaction reduction
 –A transaction that does not contain any frequent k-item set is useless in subsequent scans.
• Partitioning
 –Any item set that is potentially frequent in Database must be frequent in at least one of the partitions of Database.
• Sampling
 – Mining on a subset of given data, lower support threshold + a method to determine the completeness.
• Many further advanced techniques.

Though the basic apriori algorithm is designed to work efficiently for large datasets, there exist a number of possible improvements:

•Transactions in the database that turn out to contain no frequent k-itemsets can be omitted in subsequent database scans.
•One can try to identify first frequent itemsets in partitions of the database. This method is based on the assumption that if an item set is not frequent in one of the partitions at least (local frequent item set) then it will also not be frequent in the whole database.
•The sampling method selects samples from the database and searches for frequent Itemsets in the sampled database using a correspondingly lower threshold for the support.

4. New Proposed Algorithms:

4.1 Signature Algorithm:

Discovery of frequent occurring subset of items, called itemsets, is the core of many data mining methods. Most of the previous studies adopt Apriori –like algorithms, which iteratively generate candidate itemsets and check their occurrence frequencies in the database. These approaches suffer from serious cost of repeated passes over the analyzed database. To address this problem, we purpose a novel method; called signature method, for reducing database activity of frequent item set discovery algorithms. The idea of signature method consists of using signature table for pruning candidate itemsets. The proposed method requires fewer scans over the source database: The first scan creates signature, while the subsequent ones verify discovered itemsets.

The goodness of the signature algorithm is:
1) Lesser number of databases scans,
2) Faster pruning of the candidate itemsets.

Most of the previous studies on frequent itemsets adopt Apriori-like algorithms, which iteratively generate candidate itemsets and check their occurrence frequencies in the database. It has been shown that Apriori in its original form suffer from serious costs of repeated passes over the analyzed database and from the number of candidates that have to be checked, especially when the frequent itemsets to be discovered are long.
Signature method generates signature table from the original database and use them for pruning candidate itemsets in the iteration.
Apriori-like algorithms use full database scans for pruning candidate itemsets, which are below the support threshold. Signature method prunes candidates by using dynamically generated signature table thus reducing the number of database blocks read.

4.1.1 Signature Property:

A signature table used by Signature method is a set of signature generated for each database item set. The signature of a set X is an N-bit binary number created, by means of bit-wise OR operation from the signature of all data items contained in X.
The signature has the following property. For any two set X and Y, we have $X \subseteq Y$ if:

Signature (X) AND Signature (Y)=Signature (X)

Where AND is the bit-wise AND operator. The property is not reversible in general (when we find that the above formula evaluates to TRUE we still have to verify the result traditionally).

4.1.2 Algorithm:

```
Scan D to generate signature S and to find $L_1$;
For (K=2; $L_{K-1} \neq 0$ ; K++) do Begin
 $C_k$ = apriori_gen ($L_{k-1}$);
Begin
    Forall transactions $t \in D$ do begin
        $C_t$ = subset( $C_K$ ,t );
          Forall candidate $c \in C_t$ do c.count++;
     End
End
$L_k = \{ c \in C_k \mid c.count \geq minsup \}$;
End
Answer= $\cup_K L_K$;
Scan D to verify the answer;
```

Apriori_gen()

The apriori_gen() function works in two steps:

1. Join step
2. Prune step

First in the join step, large itemsets from L_{K-1} are joined with other large itemsets from L_{K-1} in the following SQL- like manner:

```
Insert into $C_k$
Select p.item1, p.item2,………,p.item$_{K-1}$,q.item$_{K-1}$
From  $L_{K-1}$ p , $L_{K-1}$ q
Where p.item1 = q.item$_1$
    And p.item2, q.item$_2$
    And p.item3, q.item$_3$
     .
     .
     .
     .
     .
    And p.itemK$_{-1}$, q.item$_{K-1}$
```

Next, in the prune step, each item set $c \in C_K$ such that some (K-1) – subset of c is not in L_{K-1} is deleted:

```
Forall itemsets $c \in C_K$ do
    Forall (K-1)-subsets s of c do
         If ($s \notin L_{K-1}$) then delete c from $C_K$;
```

The set of candidate K-itemsets C_K is then returned as a result of the function apriori_gen().

4.2 Database pruning Algorithm:

To determine large itemsets from a huge number of candidate large itemsets in early iterations is usually the dominating factor for the overall data mining performance. To address this issue we propose an effective algorithm for the candidate set generation. Explicitly, the number of candidate 2-itemsets generated by the proposed algorithm is, in order of magnitude, smaller than that by the previous method, thus resolving the performance bottleneck.

The initial candidate set generation, especially for the large 2-itemsets, is the key issue to improve the performance of the data mining.
Another performance related issue is on the amount of data that has to be scanned during large item set discovery. A straightforward implementation would require one pass over the database of all transactions for each iteration. As K increases, not only is there a smaller number of K –itemsets, but also there are fewer transactions containing any large K- itemsets. Reducing the number of transaction to be scanned can improve the data mining efficiency in later stages.

Generation of smaller candidate sets by the pruning method enables us to effectively trim the transaction database at much earlier stage of the iteration i.e. right after the generation of large 2-itemsets, therefore reducing the computational cost for later iteration significantly.

4.2.1 Problem Description:

Let $I=\{i_1,i_2,\ldots,i_m\}$ be a set of literals, called items. Let D be a set of transactions, where each transaction T is a set of items such that $T \subseteq I$.
The problem of mining association rules is composed of the following two steps:

1) Discover the large itemsets, i.e., all sets of itemsets that have transaction support above a predetermined minimum support s.
2) Use the large itemsets to generate the association rules for the database.

The overall performance of mining association rules is in fact determined by the first step. After the large itemsets are identified, the corresponding association rules can be derived in a straightforward manner.
To discover the set of large 2-itemsets, in view of the fact that any subset of a large item set must also have minimum support, Apriori uses L_1*L_1 to generate a candidate set of itemsets C_2 using the apriori candidate generation, where * is an operation for concatenation C_2 consists of $(^{|L1|}{}_2)$ 2-itemsets. When $|L_1|$ is large, $(^{|L1|}{}_2)$ becomes an extremely large number.

4.2.2 Major Features:

The algorithm proposed has two major features:
One is the efficient generation for large itemsets.
Second is effective reduction on transaction database size.

Pruning algorithm reduces the database size progressively by pruning the number of transactions in the database. As in mining association rules, any subset of a large item set must be a large item set by itself, that is, $\{B, C, D\} \in L_3$ implies $\{B, C\} \in L_2$, $\{B, D\} \in L_2$, and $\{C, D\} \in L_2$.
Here we add a number field with each database transaction. So when we check about the support of each item set in the candidate itemsets. Then if the candidate item set is the subset of that transaction then the number field is incremented by one each time.
After calculating the support of the candidate item set. Then we prune those transactions of the database that have:
Number < length of the transaction.
So if the transaction is ABC then the number of this transaction should be 3 or greater than 3.i.e. at least 3 candidate itemsets are subset of this transaction.
So each time the database is pruned on these criteria. So input output cost is reduced, as the scanning time is reduced.
Here we reduces the storage required by the candidate set item sets by only adding those itemsets in to the large itemsets which qualify the minimum support level.

Algorithm:

```
Scan D to generate signature $C_1$ and to find $L_1$;
For ($K=2$; $L_{K-1} \neq 0$ ; K++)do Begin
 $C_k$ = apriori_gen ($L_{k-1}$);
Begin
    Forall transactions $t \in D$ do begin
         $C_t$ = subset( $C_K$ ,t );
           Forall candidate $c \in C_t$ do c.count++;
               D.n++;
     End
   For all transactions $t \in D$ do  Begin
          If(n< transaction(length))
           delete(database, transaction);
    End
End
$L_k = \{ c \in C_k \mid c.count \geq minsup \}$;
End
Answer= $\cup_K L_K$;
Scan D to verify the answer;
```

5. Software requirement specifications

5.1 Introduction

5.1.1 Purpose

The purpose of this document is to describe the requirement of the evaluation of various data mining algorithms and their software implementation. The evaluation includes finding out in detail the factors affecting the algorithms performance.

5.1.2 Motivation

This project is rather like an E-Business application, which observe the customers purchasing patterns to predicate the association. It helps E-Business to predicate the association of sets of products/service, which they can provide to customer next time. For example, this time we observe a customer interests on certain products/service, then next time we can recommend the same products/service or give more information about that to the customer.
This project idea also applies to traditional store. Example, a store predicts an association between sets of products/services. Then the manager can put those associate sets of products/services together for increasing the number of sold.
As there are many new transaction records add to the database, they are important asset to find the association among each items. After mining the association, the store or company can provide more better and personal service to each customer.

5.1.3 Scope

This document is meant to be used by the developers and will be the basis of validating the final appraisal. The analyst will keep track of any change made to the requirements in the future and must spend considerable time examining components such as the various forms used in the system.
The output of the project is mainly used in the decision making in the industry and the Business from the transactions of the business.
The analyst can provide to the retailer in the business that how he should arrange the goods in his shop so that the over all sale of the shop get improved and on which items he should provide the discount, as that the things that are purchased together, so over all sale get incremented.

5.2 General Description

5.2.1 Project functional overview

The existing Apriori algorithm aims in finding the large itemsets for the given data sets depending on the user inputted support and confidence. The system tries to compare other algorithms with the Apriori algorithm on the basis of different datasets, support and confidence values and other parameters.
Some algorithms are compared on data sets of various sizes while other on the basis of various attributes like:

Time taken to find the large item set for the given support.
Number of database scans required for finding the large itemsets.

5.2.2 User Characteristics

The main users of the system are the analysts who will gather the results generated from simulation of the algorithm used .The data, thus collected determines the workability of the algorithm under different conditions.

5.2.3 General constraints

The system should run on the Pentium computers running Windows.

5.3 Specific Requirements

5.3.1 Input and outputs

Input file contains the list of all the transactions of one-month time provided to us from
“CHADHA PROVISION STORE ”
F-14/45,
MODEL TOWN-II
DELHI-10009

We have taken 52 items with the average size of the transaction 25.
The format of the file is as:

```
Transaction 1
Transaction 2
Transaction 3
          .
          .
          .
Transaction N
```

Output file contains the large item sets. That is, it contains the candidate itemsets and the large itemsets.
The format of this file is as:
I_1, I_2, I_3 …………I_n

From these item sets we made the association rules that will qualify the confidence specified by the analyst.

5.3.2 Functional Requirements

1) In determining the time taken by an algorithm to generate large itemsets such that the following conditions are satisfied:
    - The size of the array used must be enough to accommodate all the candidate and large itemsets produced.
    - The size of the input file must not be greater than 64 kilobytes.
2) The input in the file must be in the proper format.

5.3.3 Design and performance Constraints

5.3.3.1 Software Constraints

The program files will run under windows operating system. The software language used is Turbo C .

5.3.3.2 Hardware constraints

The programs should run on a Pentium processor with 128 Mb RAM running Windows.

6. Results:

Result from the Data from the Grocery shop

"CHADHA PROVISION STORE "
F-14/45,
MODEL TOWN-II
DELHI-10009

6.1 Association Rules for Real Time Data

Taking 52 items with the average size of the transaction 30.

Symbols used in the following tables are as:

A	Phenyl	B	Lyzole
C	Salt	D	Haldi
E	LalMirch	F	Dhania
G	Garam Masala	H	Jeera
I	Massor Dal	J	Mung Dal
K	Chana/Arhar Dal	L	Chocolate
M	Aatta	N	Red Label/Tata tea
O	Taj Mahal Tea	P	LifeBoy soap
Q	Pearse/Dettol soap	R	Dove
S	Nirma/Fena	T	Surf/Arial/Tide
U	Rice	V	Suger
W	Besen	X	Suji/Meda
Y	Ghee	Z	Refind/Oil
a	Cream/Powder/Shampoo	b	Hair Oil
c	Dry Fruit	d	Elaichi
e	Chips/Paped	f	Jharu
g	Sauce/Jam	h	Maggi
i	Coffee	j	Namkeen
k	Biskit	l	SilverFoil
m	Shaving material	n	MatchBox
o	Rasna	p	Roohafja
q	Mosquito Repellent	r	Colgate
s	ToothBrush	t	Chana Kala/safad
u	Rajma	v	Dalia Baked
w	Biurnvita Refil	x	Rin cake
y	VimBar/Nip	z	Butter

Part I Below is the table which is obtained when the user want the minimum support of 8%

Rules	Support Percentage	Confidence Percentage
C ->K	8.1	49.8
K->C	8.1	41.8
C->V	9.1	55.9
V->C	9.1	35.5
J->K	10.2	50.0
K->J	10.2	52.9
J->V	8.9	43.5
V->J	8.9	34.7
J->t	8.9	43.5
t->J	8.9	45.5
K->V	9.4	48.7
V->K	9.4	36.8
K->t	8.7	45.3
t->K	8.7	44.8
M->V	8.3	34.5
V->M	8.3	32.4
V->t	8.6	33.7
t->V	8.6	44.2
t->u	8.0	41.1
u->t	8.0	58.5

Rules formed when the minimum support is 8% and minimum confidence is 40%

Rules	Support Percentage	Confidence Percentage
C ->K	8.1	49.8
K->C	8.1	41.8
C->V	9.1	55.9
J->K	10.2	50.0
K->J	10.2	52.9
J->V	8.9	43.5
J->t	8.9	43.5
t->J	8.9	45.5
K->V	9.4	48.7
K->t	8.7	45.3
t->K	8.7	44.8
t->V	8.6	44.2
t->u	8.0	41.1
u->t	8.0	58.5

Part II Rules table when the minimum support is 9%

Rules	Support Percentage	Confidence Percentage
C->V	9.1	55.9
V->C	9.1	35.5
J->K	10.2	50.0
K->J	10.2	52.9
K->V	9.4	48.7
V->K	9.4	36.8

Rules table when the minimum support is 9% and minimum confidence is 40%

Rules	Support Percentage	Confidence Percentage
C->V	9.1	55.9
J->K	10.2	50.0
K->J	10.2	52.9
K->V	9.4	48.7

Part III Rules table when the minimum support is 10%

Rules	Support Percentage	Confidence Percentage
J->K	10.2	50.0
K->J	10.2	52.9

Rules table when the minimum support is 10% and minimum confidence is 40%

Rules	Support Percentage	Confidence Percentage
J->K	10.2	50.0
K->J	10.2	52.9

From the I part:

We got the following table:

Rules	Support Percentage	Confidence Percentage
C ->K	8.1	49.8
K->C	8.1	41.8
C->V	9.1	55.9
J->K	10.2	50.0
K->J	10.2	52.9
J->V	8.9	43.5
J->t	8.9	43.5
t->J	8.9	45.5
K->V	9.4	48.7
K->t	8.7	45.3
t->K	8.7	44.8
t->V	8.6	44.2
t->u	8.0	41.1
u->t	8.0	58.5

So we made the conclusion that by taking the concept of optimality our association rules with the support of 8 % and the confidence of 40 % are as :

1. C->K
2. K->C
3. C->V
4. J->K
5. K->J
6. J->V
7. J->t
8. t->J
9. K->V
10. K->t
11. t->K
12. t->V
13. t->u
14. u->t

So that
From rule 1 & 2 we should place **Salt** & **Chana/Arhar Dal** together.
From rule 3 we should place **Salt** & **Sugar** together.
From rule 4 & 5 we should place **Mung Dal** & **Chana/Arhar Dal** together.
From rule 6 we should place **Mung Dal** & **Sugar** together.
From rule 7 & 8 we should place **Mung Dal** & **Chana Kala/Safad** together.
From rule 9 we should place **Chana/Arhar Dal** & **Sugar** together.
From rule 10 & 11 we should place **Chana/Arhar Dal** & **Chana Kala/Safad** together.

From rule 12 we should place **Chana Kala/Safad** & **Sugar** together.
From rule 13 & 14 we should place **Chana Kala/Safad** & **Rajma** together.

These items have 8% support and the confidence greater than 40%.

From the II part:

We got the following table:

Rules	Support Percentage	Confidence Percentage
C->V	9.1	55.9
J->K	10.2	50.0
K->J	10.2	52.9
K->V	9.4	48.7

So we made the conclusion that by taking the concept of optimality our association rules are as:

1. C->V
2. J->K
3. K->J
4. K->V

So that
From the rule 1 we should place the **Salt** and **Sugar** together.
From the rule 2& 3we should place the **Mung dal** and the **Chana Dal/Arhar Dal** together.
From the rule 4 we should place the **Chana Dal /Arhar Dal** and the **Sugar** together.

These items have 9% support and the confidence greater than 45%.

From the III part :

We got the following table:

Rules	Support Percentage	Confidence Percentage
J->K	10.2	50.0
K->J	10.2	52.9

So we made the conclusion that by taking the concept of optimality our association rules are as:

1. J->K
2. K->J

So that
From the rule 1 & 2 we should place the **Mung dal** and the **Chana Dal/Arhar Dal** together.

These items have 10% support and the confidence greater than 50%.

We consider these items as per the retailer request he was more interested in finding the relations among the Masala items.
So here we consider only the five items to find the association rules for the Masala only ie.

D	Haldi
E	LalMirch
F	Dhania
G	Garam Masala
H	Jeera

I below is the table which is obtained when the user want the minimum **support** of 4**%**

Rules	Support Percentage	Confidence Percentage
D ->E	5.7	45.5
E->D	5.7	44.6
D->F	4.8	38.6
F->D	4.8	42.7
D->G	4.2	33.3
G->D	4.2	36.4
D->H	4.9	39.2
H->D	4.9	39.2
E->F	4.3	33.2
F->E	4.3	37.4
E->G	4.5	34.7
G->E	4.5	38.9
E->H	5.6	43.5
H->E	5.6	44.4
F->H	5.1	44.4
H->F	5.1	40.2
G->H	4.5	39.5
H->G	4.5	35.9

Rules formed when the minimum support is **4%** and minimum **confidence** is **40%**

Rules	Support Percentage	Confidence Percentage
D ->E	5.7	45.5
E->D	5.7	44.6
F->D	4.8	42.7
E->H	5.6	43.5
H->E	5.6	44.4
F->H	5.1	44.4
H->F	5.1	40.2

So we made the conclusion that by taking the concept of optimality our association rules are as :

1. D->E
2. E->D
3. F->D
4. E->H
5. H->E
6. F->H
7. H->F

So that
From the rule 1 & 2 we should place the **Haldi** and the **LalMirch** together.
From the rule 3 we should place the **Haldi** and the **Dhania** together.
From the rule 4& 5 we should place the **LalMirch** & the **Jeera** together.
From the rule 6& 7 we should place the **Dhania** and the **Jeera** together.

These items have **4% support** and the **confidence** greater than **40%.**

II Below is the table which is obtained when the user want the minimum **support** of **5%**

Rules	Support Percentage	Confidence Percentage
D ->E	5.7	45.5
E->D	5.7	44.6
E->H	5.6	43.5
H->E	5.6	44.4
F->H	5.1	44.4
H->F	5.1	40.2

Rules formed when the minimum **support** is **5%** and minimum **confidence** is **44%**

Rules	Support Percentage	Confidence Percentage
D ->E	5.7	45.5
E->D	5.7	44.6
H->E	5.6	44.4
F->H	5.1	44.4

So we made the conclusion that by taking the concept of optimality our association rules are as:

1. D->E
2. E->D
3. H->E
4. F->H

So that
From the rule 1 & 2 we should place the **Haldi** and the **LalMirch** together.
From the rule 3 we should place the **Jeera** & the **LalMirch** together.
From the rule 4 we should place the **Dhania** and the **Jeera** together.

As these items have **5% support** and the **confidence** greater than **44%.**
we suggest if the items are placed near to each other it would help in increasing the over sale of these goods.

Note: ---
But we also observe that if we take the **support** if **5%** and the **confidence** of **45%**
Then we get the only one rule that is as:

D->E
So we should place the **Haldi** and the **Lal Mirch** together to get the increment in the sell of these goods.

Criteria of performance evaluation: -

As we are comparing the efficiency of the association rule algorithms on two parameters. These parameters are:

1) Time taken by the algorithm in finding the large /frequents itemsets.

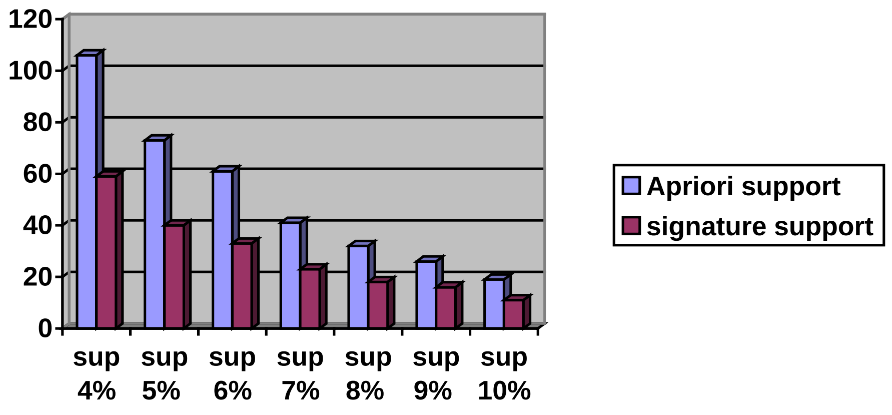
As time taken by the algorithm to find the large item set is crucial point in the analysis of data mining, as in case of Apriori algorithm maximum time is required during the database scan for finding the C_1 candidate item sets and in checking whether the item of the candidate C_k itemsets are subset of the L_{k-1} most frequent itemsets.

2) Number of times database is scanned.

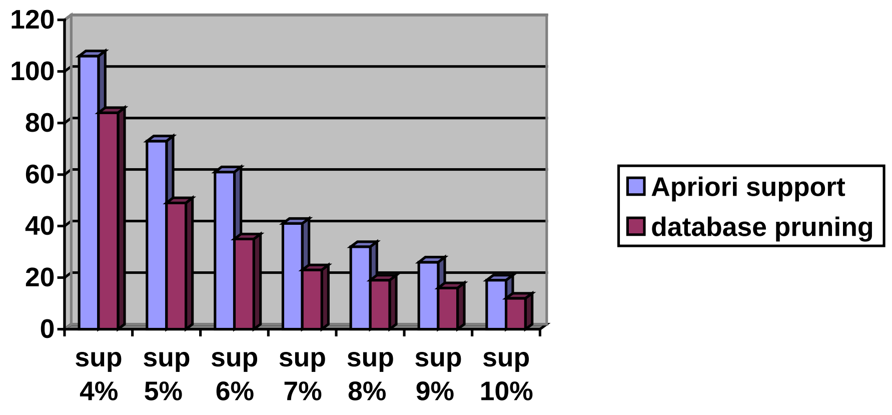
More the number of times database is scanned, the algorithm is lesser efficient. As the databases contain millions of entries, so if more number of database scans are required, it consumes more time, that make the algorithm less efficient so we try to design the algorithms that minimize the number of database scans.

6.2 Graphical representation of the time taken by the various algorithms.

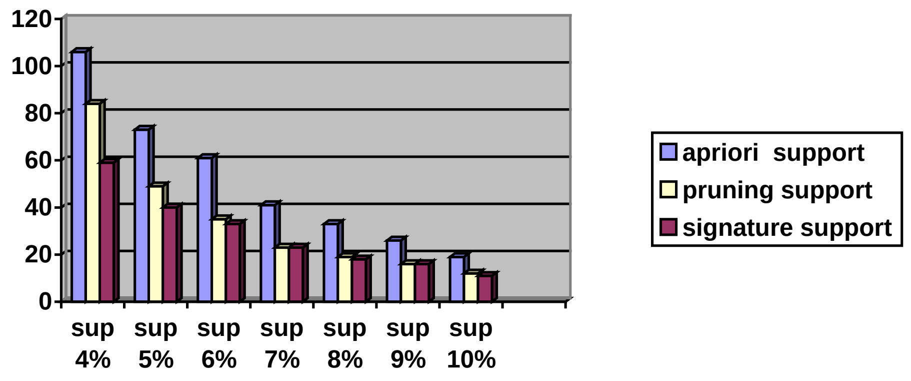
1) Chart showing relation of time taken by the basic Apriori algorithm and the Signature method of data mining



2) Chart showing relation of time taken by the basic Apriori algorithm and the Database method of data mining.



3) Chart showing relation of time taken by the basic Apriori algorithm, Database pruning method and Signature method of data mining



The number of database scans:

-In the Apriori algorithm the number of database scans are of K times where K is the largest value of K in L_K or of C_K.
-In the Signature method of number of database scans are only one i.e. while creating the signature table.

-In the Database Pruning algorithm the number of database scans are of K times where K is the largest value of K in L_K or of C_K. But the size of the database scanned is reduced in each iteration.

6.3 Strengths and Weaknesses of Association Rules Analysis

6.3.1 The strengths of association rule analysis

The strengths of association rule analysis are:

- It produces clear and understandable results.
- It support undirected data mining.
- It works on variable length data.
- The computations it does are simple to understand.

Results Are Clearly Understood.

The results of association rule analysis are association rules; these are readily expressed as English or as a statement in a query language such as SQL. The expression of patterns in the data as "if-then" rules makes the results easy to understand and facilitates turning the results into action. In some circumstances, merely the set of related items is of interest and rules do not even need to be produced.

Association rule Analysis Is Strong for Undirected Data Mining.

Undirected data mining is very important when approaching a large set of data and you do not know where to begin. Association rule analysis is an appropriate technique, when it can be applied, to analyze data and to get a start. Most data mining techniques are not primarily used for undirected data mining. Association rule analysis, on the other hand, is used in this case and provides clear results.

Association rule Analysis Works on Variable-Length Data.

Association rule analysis can handle variable-length data without the need for summarization. Other techniques tend to require records in a fixed format, which is not a natural way to represent items in a transaction. Association rule analysis can handle transactions without any loss of information.

Computationally Simple.

The computations needed to apply association rule analysis are rather simple, although the number of computations grows very quickly with the number of transactions and the number of different items in the analysis. Smaller problems can be set up on the desktop using a spreadsheet. This makes the technique more comfortable to use than complex techniques, like genetic algorithms or neural networks.

6.3.2 The weaknesses of association rule analysis

The weaknesses of association rule analysis are:

- It require exponentially more computation effort as the problem size grows.
- It has a limited support for attributes on the data.
- It is difficult to determine the right number of items
- It discount rare items

Exponential Growth as Problem Size Increases.

The computations required to generate association rules grow exponentially with the number of items and the complexity of the rules being considered. The solution is to reduce the number of items by generalizing them. However, more general items are usually less actionable. Methods to control the number of computations, such as minimum support pruning, may eliminate important rules from consideration.

Limited Support for Data Attributes.

Association rule analysis is a technique specialized for items in a transaction. Items are assumed to be identical except for one identifying characteristic, such as the product type. When applicable, association rule analysis is very powerful. However, not all problems fit this description. The use of item taxonomies and virtual items helps make rules more expressive.

Determining the Right Items.

Probably the most difficult problem when applying association rule analysis is determining the right set of items to use in the analysis. By generalizing items up their taxonomy, you can ensure that the frequencies of the items used in the analysis are about the same. Although this generalization process loses some information, virtual items can then be reinserted into the analysis to capture information that spans generalized items.

Association rule Analysis Has Trouble with Rare Items.

Association rule analysis works best when all items have approximately the same frequency in the data. Items that rarely occur are in very few transactions and will be pruned. Modifying minimum support thresh-old to take into account product value is one way to ensure that expensive items remain in consideration, even though they may be rare in the data. The use of item taxonomies can ensure that rare items are rolled up and included in the analysis in some form.

7.Conclusion:

The introduction of the Apriori algorithm marked a first peak in the development of data mining algorithms. In particular it motivated many variations of the algorithm, which addresses special situations. Even the most recent development is still to a large extent based on the Apriori algorithm.

But performance study shows Signature method is efficient than the Apriori algorithm for mining in terms of time as it reduces the number of and Database scans, and also the Database pruning is efficient than the Apriori algorithm for mining in terms of time, as it prunes the database in each iteration. The Signature method is roughly two times faster than the Apriori algorithm, as Signature method does not require more than one database scan, and Database pruning method prunes the database in each iteration. So the Database Pruning algorithm is slower in the earlier stages but faster in the later stages.

We have use the real time data form the grocery shop, “CHADHA PROVISION STORE ”F-14/45,MODEL TOWN-II DELHI-10009 and by the use of the rules suggested by us they got improvement in his sales.

8.Future Directions:

The major time consumed in the association rule mining is the time to find the frequent (large) itemsets, once these frequent itemsets are discovered the application association rules can be made in straightforward manner.
While working on this dissertation we feel that if we calculate the support of each candidate item set in C_2 while it is being formed from the L_1 and including those item set in the C_2 that qualify the support value. This way although the numbers of computations are same, but we can greatly reduce the size of the C_2 list, now as all the itemsets in the C_2 list have support greater than the support specified by the user so there is no need to calculate the L_2 list. As it is same as the C_2 list. So from C_2 onwards we have to maintain only one list. Which will take less space and we have to handle only one list and it is less time consuming also as it avoids the time to form the L_K list form the C_K list.

Further if by some other way we are able to reduce the database scan time then it also improves the time to find the frequent item sets. We can also develop the new techniques for the database pruning that further reduces time for finding the frequent itemsets.

Source code

/* Program for the Apriori Algorithm for association rules in data mining*/

```
//Header files included

#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<alloc.h>
#define max 30                    //average size of the  transactions
#define records 1500
#define n_size 6

//structure node for the candidate itemsets and the Large item set

struct node
{
char arr[n_size];         //It specify the number of items in the itemset
int data;
struct node*next;
};

struct node*list1=NULL,*list2=NULL;
char dbase[records][max];

//Prototypes of the function used

void show(struct node*);
void add(struct node**,char str[],int);
void del(struct node**,char str1[],int);
void findsup(struct node**p);
void removedup(struct node**);
void ctol_list(struct node* );
void ltoc_list(struct node*);
void freenode(struct node**);
int sub(char *s,int a,struct node*len_list);
int minsup=0;    // Minimum support required

//--To free the list nodes & assign it NULL value---//
```

```
void freenode(struct node**q)
{
struct node*temp,*r;
temp=*q;
while(temp!=NULL)
{
 r=temp;
 temp=temp->next;
 free(r);
}
*q=temp;
}

//--show function----//

void show(struct node*q)
{
printf("\n");
while(q!=NULL)
{
printf("%s %d\n",q->arr,q->data);
q=q->next;
}
}

//show the dbase

void showdbase()
{
int t=0;
for(t=0;t<records;t++)
{
printf("%s \n",dbase[t]);
}
printf("%d",t);
}

//----add function for the candidate list & the lenght list----//

void add(struct node**q,char str[],int a)
{
struct node *r=NULL,*temp=NULL;
temp=*q;
r=(struct node*)malloc(sizeof(struct node));
r->data=a;
strcpy(r->arr,str);
r->next=NULL;
if(*q==NULL)
   *q=r;
else
 {
  while((temp->next)!=NULL)
      temp=temp->next;
  temp->next=r;
  }
}

//--delete function---//

void del(struct node**q,char str[],int a)
{
struct node*old,*temp;
temp=*q;
while(temp!=NULL)
{
  if((temp->data==a)&&(!strcmp(temp->arr,str)))
   {
    if(*q==temp)
     {
    *q=temp->next;
    free(temp);
    return;
    }
    else
    {
      old->next=temp->next;
      free(temp);
      return;
    }
   }
   else
     {
    old=temp;
    temp=temp->next;
     }
}
}

//--remove the duplicates--//

void removedup(struct node**q)
```

```
{
 struct node*temp,*x1,*x2,*x3;
 temp=*q;
 x1=temp;
 x2=temp->next;
 while(x1!=NULL)
 {
   while(x2!=NULL)
   {
    if((x1->data==x2->data)&&(!strcmp(x1->arr,x2->arr)))
      {
          x3=x2;
          x2=x2->next;
          del(q,x3->arr,x3->data);
      }
    else
    x2=x2->next;
   }
   x1=x1->next;
   x2=x1->next;
 }
}

//--find the support of the candidate item sets--//

void findsup(struct node**p)
{
int i,k,count=0,j,num=0;
char str1[n_size],str2[max];
struct node*temp;
temp=*p;
while(temp!=NULL)
{
   strcpy(str1,temp->arr);
     for(i=0;i<records;i++)
     {
      strcpy(str2,dbase[i]);
       for(j=0;j<strlen(str2);j++)
       {
             for(k=0;k<strlen(str1);k++)
                     if(str2[j]==str1[k])
                       count++;
       }
       if(strlen(str1)==count)
              num++;
       count=0;
    }
    temp->data=num;
    num=0;
    temp=temp->next;
}
}

//--make lenght list from the candiadate list which staisfy the support--//

void ctol_list(struct node* p)
{
freenode(&list2);
list2=NULL;
while(p!=NULL)
{
 if((p->data)>=minsup)
 {
   add(&list2,p->arr,p->data);
 }
 p=p->next;
}
}

//It tells whether all the a subsets of string s are in the len_list or not

int sub(char *s,int a,struct node*len_list)
{
int i,j,k,l,m,n,c=0,count=0,num=0;   //count counts the number of subset items
struct node*x1=NULL;
struct node*list2=NULL,*x2=NULL;
char *p;
p=(char*)malloc((a+1)*sizeof(char));
switch(a)
{
  case 1 :
    { // To find the 1 item subset
        for(j=0;j<strlen(s);j++)
        {     count++;
                p[c++]=s[j];
                p[c]='\0';
                add(&list2,p,0);c=0;
        }
        break;
    }
  case 2 :
    { // To find the 2 item subset
```

```
        for(j=0;j<strlen(s);j++)
        {
          for(k=j+1;s[j]<s[k];k++)
          {
           count++;
          p[c++]=s[j];
          p[c++]=s[k];
          p[c]='\0';
          add(&list2,p,0);c=0;
          }
        }
        break;
   }
 case 3 :
  {   // To find the 3 item subset
        for(i=0;i<strlen(s);i++)
        {
         for(j=i+1;j<strlen(s);j++)
         {
          for(k=j+1;s[j]<s[k];k++)
          { count++;
          p[c++]=s[i];
          p[c++]=s[j];
          p[c++]=s[k];
          p[c]='\0';
          add(&list2,p,0);c=0;
          }
         }
        }
        break;
  }
  case 4 :
   {  // To find the 4 item subset
        for(i=0;i<strlen(s);i++)
        {
         for(j=i+1;j<strlen(s);j++)
         {
           for(k=j+1;k<strlen(s);k++)
            {
                  for(l=k+1;s[k]<s[l];l++)
                  { count++;
                  p[c++]=s[i];
                  p[c++]=s[j];
                  p[c++]=s[k];
                  p[c++]=s[l];
                  p[c]='\0';
```

```
                    add(&list2,p,0);c=0;
                    }
                }
            }
          }
    break;
  }
 case 5 :
   {  // To find the 5 item subset
          for(i=0;i<strlen(s);i++)
          {
           for(j=i+1;j<strlen(s);j++)
           {
             for(k=j+1;k<strlen(s);k++)
              {
                    for(l=k+1;s[k]<strlen(s);l++)
                     {
                            for(m=l+1;s[l]<s[m];m++)
                            { count++;
                            p[c++]=s[i];
                            p[c++]=s[j];
                            p[c++]=s[k];
                            p[c++]=s[l];
                            p[c++]=s[m];
                            p[c]='\0';
                            add(&list2,p,0);c=0;
                            }
                      }
              }
            }
          }
    break;
  }
case 6 :
   {  // To find the 6 item subset
          for(i=0;i<strlen(s);i++)
          {
           for(j=i+1;j<strlen(s);j++)
           {
             for(k=j+1;k<strlen(s);k++)
              {
                    for(l=k+1;s[k]<strlen(s);l++)
                     {
                     for(m=l+1;s[l]<strlen(s);m++)
                       {
                          for(n=m+1;s[m]<s[n];n++)
```

```
                              { count++;
                              p[c++]=s[i];
                              p[c++]=s[j];
                              p[c++]=s[k];
                              p[c++]=s[l];
                              p[c++]=s[m];
                              p[c++]=s[n];
                              p[c]='\0';
                              add(&list2,p,0);c=0;
                              }
                            }
                        }
                  }
              }
            }
      break;
    }
  }//end of switch
  free(p);
  x1=len_list;
  x2=list2;
   while(x2!=NULL)
    {
      while(x1!=NULL)
     {
      if(!strcmp(x2->arr,x1->arr)&&((x1->data)>=minsup))
             num++;

      x1=x1->next;
     }
    x2=x2->next;
    x1=len_list;
   }
   freenode(&list2);
   if(num==count)
    return(1);
   else
     return(0);
}

//--make the permutation of the lenght list and form the candidate list--//
//--and made the lenght list that satisfy the support--//

void ltoc_list(struct node*q)
{
static int k=2;
```

```
char str1[n_size],str2[n_size],str3[n_size];
struct node*x1=NULL,*x2=NULL;
int d[52]={0},i=0,j=0;
freenode(&list1);

x1=q;
x2=q->next;
while(x1!=NULL)
{
 while(x2!=NULL)
  {
    strcpy(str1,x1->arr);
    strcpy(str2,x2->arr);
          for(i=0;i<strlen(str1);i++)
           {
            if((str1[i]>='A') && (str1[i]<='Z'))
            d[str1[i]-'A']++;
            else
            d[str1[i]-'A'-6]++;
           }
          for(i=0;i<strlen(str2);i++)
            {
           if((str2[i]>='A')&&(str2[i]<='Z'))
             {
                 if(d[str2[i]-'A']==0)
                     d[str2[i]-'A']++;
             }
            else
            {
            if(d[str2[i]-'A'-6]==0)
                     d[str2[i]-'A'-6]++;
             }
           }
          for(i=0;i<52;i++)
          {
            if(d[i]>0)
            {
                if(i<26)
                 str3[j++]=(char)('A'+i);
                else
                 str3[j++]=(char)('A'+i+6);

                d[i]=0;
            }
          }
          str3[j]='\0';
```

```
            if(strlen(str3)==k)
            {
              if(sub(str3,k-1,q)) //It finds the k-1 subset of str3
              {
                add(&list1,str3,0);
              }
            }
           j=0;x2=x2->next;
  } //end of inner while
    x1=x1->next;
    x2=x1->next;
}//end of outer while
 removedup(&list1);
 findsup(&list1);
 k++;
}

// Main function

void main()
{
int i=0,j=0,k=0,num=2;
int d[52]={0};
int s[52]={0};
char abc[max];
char abcd[max];
char cht;
char m[2];
long int bios_time1,bios_time2;
FILE *fp;
clrscr();
fp=fopen("data1.txt","r");  //Input file that contains the transactions
if(fp==NULL)
printf("Unable to open the File:\n");
printf("enter the value of minsup :\n");
scanf("%d",&minsup);
bios_time1=biostime(0,0l);
while(fgets(abc,max,fp)!=NULL)
{
for(i=0;i<strlen(abc);i++)
{
          if(abc[i]>='A'&&abc[i]<='Z')
          {
          s[abc[i]-'A']++;
          if(d[abc[i]-'A']==0)
          d[abc[i]-'A']++;
```

```
               }
       else
               {
               s[abc[i]-'A'-6]++;
               if(d[abc[i]-'A'-6]==0)
               d[abc[i]-'A'-6]++;
               }
}
   for(i=0;i<52;i++)
     {
      if(d[i]>0)
       {
      if(i<26)
        {abcd[j++]=(char)(i+'A');}
      else
        {abcd[j++]=(char)(i+'A'+6);}
       }
      }
    abcd[j]='\0';

    strcpy(dbase[k++],abcd);
    strcpy(abcd," ");
    for(i=0;i<52;i++)
      d[i]=0;
   j=0;
}//end of while
//showdbase(dbase);
//getch();

for(i=0;i<52;i++)
{
 if(s[i]>0)
  {
  if(i<26)
   {
  m[0]=(char)(i+'A');
  m[1]='\0';
   }
  else
   {
   m[0]=(char)(i+'A'+6);
   m[1]='\0';
   }
  add(&list1,m,s[i]);
 }
}
```

```
printf("    Candidate(C1)     \n");
printf(" ITEMS   Support\n");
show(list1);// getch();
ctol_list(list1);
printf("\n   Lenght(L1)     \n");
printf(" ITEMS   Support\n");
show(list2);//getch();
ltoc_list(list2);
while((list1!=NULL)&&(list2!=NULL))
{
printf("    Candidate(C%d)     \n",num);
printf(" ITEMS            Support\n");
show(list1);//getch();
ctol_list(list1);
printf("\n   Lenght(L%d)     \n",num);
printf(" ITEMS            Support\n");
show(list2);//getch();
ltoc_list(list2);
num++;
}
freenode(&list1);
freenode(&list2);
fclose(fp);
bios_time2=biostime(0,0l);
printf("Time taken in the processing is as %ld",bios_time2-bios_time1);
printf("\nThanks  Have a nice day");
getch();

}
```

/* Program for the Signature method of Association rules in data mining */

```
//The header files used

#include<stdio.h>
#include<math.h>
#include<string.h>
#include<alloc.h>
#include<conio.h>

#define max 35                  //average size of the transaction
#define records 1500
//records represents the total number of records in the input file

//Structure node for the Candidate itemsets and the Large itemsets

struct node
{
int data;
char arr[max];          //maximum size of the item set
struct node*next;
};
struct node*llist=NULL,*clist=NULL;
int a[52][4];
int sign[records][4];
int numr=0;

// Prototypes of the functions used

void freenode(struct node**);
void show(struct node*);
void add(struct node**,char str[],int );
void del(struct node**,char str[],int );
void removedup(struct node**);
void ltoc_list(struct node*,int);
void ctol_list(struct node* );
void findsupport(struct node**p);
void value(char str[],int temp[]);
int subset(int *temp1,int *temp2);
int sub(char *s,int a,struct node*len_list);
int minsup=0;     //Minimum support required

//It frees all the nodes of the given list
```

```
void freenode(struct node**q)
{
struct node*temp,*r;
temp=*q;
while(temp!=NULL)
{
 r=temp;
 temp=temp->next;
 free(r);
}
*q=temp;
}

//--show function----//

void show(struct node*q)
{
printf("\n");
while(q!=NULL)
{
printf("%s  %d\n",q->arr,q->data);
q=q->next;
}
}

//----add function----//

void add(struct node**q,char str[],int a)
{
struct node *r=NULL,*temp=NULL;
temp=*q;
r=(struct node*)malloc(sizeof(struct node));
r->data=a;
strcpy(r->arr,str);
r->next=NULL;
if(*q==NULL)
    *q=r;
else
 {
  while((temp->next)!=NULL)
      temp=temp->next;
  temp->next=r;
  }
}
```

```
//--delete function---//

void del(struct node**q,char str[],int a)
{
struct node*old,*temp;
temp=*q;
while(temp!=NULL)
{
 if((temp->data==a)&&(!strcmp(temp->arr,str)))
  {
   if(*q==temp)
   {
   *q=temp->next;
   free(temp);
   return;
   }
   else
   {
    old->next=temp->next;
    free(temp);
    return;
   }
  }
  else
   {
   old=temp;
   temp=temp->next;
   }
}
}

//--remove the duplicates--//

void removedup(struct node**q)
{
 struct node*temp,*x1,*x2,*x3;
 temp=*q;
 x1=temp;
 x2=temp->next;
 while(x1!=NULL)
 {
  while(x2!=NULL)
  {
   if((x1->data==x2->data)&&(!strcmp(x1->arr,x2->arr)))
    {
            x3=x2;
            x2=x2->next;
            del(q,x3->arr,x3->data);
        }
      else
      x2=x2->next;
    }
    x1=x1->next;
    x2=x1->next;
 }
}

// It tell whether the a subset of the s are in the lenlist or not

int sub(char *s,int a,struct node*len_list)
{
int i,j,k,l,m,c=0,count=0,num=0;   //count counts the number of subset items
struct node*list2=NULL,*x2=NULL;
struct node*x1=NULL;
char *p;
p=(char*)malloc((a+1)*sizeof(char));
switch(a)
{
   case 1 :
      { // To find the 1 item subset
            for(j=0;j<strlen(s);j++)
            {       count++;
                        p[c++]=s[j];
                        p[c]='\0';
                        add(&list2,p,0);c=0;
            }
            break;
      }
   case 2 :
      { // To find the 2 item subset
            for(j=0;j<strlen(s);j++)
            {
              for(k=j+1;s[j]<s[k];k++)
              {
               count++;
              p[c++]=s[j];
              p[c++]=s[k];
              p[c]='\0';
              add(&list2,p,0);c=0;
              }
            }
            break;
      }
    case 3 :
     {   // To find the 3 item subset
            for(i=0;i<strlen(s);i++)
            {
              for(j=i+1;j<strlen(s);j++)
              {
              for(k=j+1;s[j]<s[k];k++)
              { count++;
              p[c++]=s[i];
              p[c++]=s[j];
              p[c++]=s[k];
              p[c]='\0';
              add(&list2,p,0);c=0;
              }
              }
            }
            break;
      }
      case 4 :
      {  // To find the 4 item subset
            for(i=0;i<strlen(s);i++)
            {
              for(j=i+1;j<strlen(s);j++)
              {
                 for(k=j+1;k<strlen(s);k++)
                  {
                           for(l=k+1;s[k]<s[l];l++)
                           { count++;
                           p[c++]=s[i];
                           p[c++]=s[j];
                           p[c++]=s[k];
                           p[c++]=s[l];
                           p[c]='\0';
                           add(&list2,p,0);c=0;
                           }
                   }
              }
            }
      break;
      }
      case 5 :
      {  // To find the 5 item subset
            for(i=0;i<strlen(s);i++)
            {
               for(j=i+1;j<strlen(s);j++)
               {
                  for(k=j+1;k<strlen(s);k++)
                   {
                         for(l=k+1;s[k]<strlen(s);l++)
                          {
                                 for(m=l+1;s[l]<s[m];m++)
                                 { count++;
                                 p[c++]=s[i];
                                 p[c++]=s[j];
                                 p[c++]=s[k];
                                 p[c++]=s[l];
                                 p[c++]=s[m];
                                 p[c]='\0';
                                 add(&list2,p,0);c=0;
                                 }
                           }
                   }
               }
              }
         break;
      }
    }//end of switch
    free(p);
    x1=len_list;
    x2=list2;
      while(x2!=NULL)
      {
         while(x1!=NULL)
         {
         if(!strcmp(x2->arr,x1->arr)&&((x1->data)>=minsup))
                 num++;
         x1=x1->next;
         }
        x2=x2->next;
        x1=len_list;
      }
      freenode(&list2);
      if(num==count)
      return(1);
      else
      return(0);
    }
```

```
//--make lenght list from the candiadate list which staisfy the support--//

void ctol_list(struct node* p)
{
freenode(&llist);
while(p!=NULL)
{
 if(p->data>=minsup)
 {
   add(&llist,p->arr,p->data);
 }
 p=p->next;
}
}

//--make thef permutation of the lenght list and form the candidate list--//
//--and made the lenght list that satisfy the support--//

void ltoc_list(struct node*q,int k)
{
char str1[max],str2[max],str3[max];
struct node*x1=NULL,*x2=NULL;
int d[52]={0},i,j=0;
freenode(&clist);
x1=q;
x2=q->next;
while(x1!=NULL)
{
 while(x2!=NULL)
 {
    strcpy(str1,x1->arr);
    strcpy(str2,x2->arr);
          for(i=0;i<strlen(str1);i++)
           {
            if(str1[i]>='A'&&str1[i]<='Z')
            d[str1[i]-'A']++;
            else
            d[str1[i]-'A'-6]++;
           }
          for(i=0;i<strlen(str2);i++)
            {
           if(str2[i]>='A'&&str2[i]<='Z')
             {
                if(d[str2[i]-'A']==0)
                     d[str2[i]-'A']++;
                }
               else
               {
               if(d[str2[i]-'A'-6]==0)
                         d[str2[i]-'A'-6]++;
                }
               }
             for(i=0;i<52;i++)
             {
               if(d[i]>0)
               {
                    if(i<26)
                     str3[j++]=(char)('A'+i);
                    else
                     str3[j++]=(char)('A'+i+6);

                    d[i]=0;
               }
             }
             str3[j]='\0';
             if(strlen(str3)==k)
             {
              if(sub(str3,k-1,q)) //It finds the k-1 subset of str3
               add(&clist,str3,0);
             }
            j=0;x2=x2->next;
  } //end of inner while
    x1=x1->next;
    x2=x1->next;
}//end of outer while
 removedup(&clist);
//k++;
}

//Put the signature of the string str in the temp array

void value(char str[],int temp[])
{
int v[4]={0};
int i;
for(i=0;i<strlen(str);i++)
{
  if(str[i]>='A'&&str[i]<='Z')
   {
    v[0]=v[0]|a[str[i]-'A'][0];
    v[1]=v[1]|a[str[i]-'A'][1];
    v[2]=v[2]|a[str[i]-'A'][2];
    v[3]=v[3]|a[str[i]-'A'][3];
   }
   else
   {
    v[0]=v[0]|a[str[i]-'A'-6][0];
    v[1]=v[1]|a[str[i]-'A'-6][1];
    v[2]=v[2]|a[str[i]-'A'-6][2];
    v[3]=v[3]|a[str[i]-'A'-6][3];
   }
}
for(i=0;i<4;i++)
 temp[i]=v[i];
}

//Tells whether firts string is subset of the second

int subset(int *temp1,int *temp2)
{
int i,j=0;
for(i=0;i<4;i++)
 if(temp1[i]==(temp1[i] & temp2[i]))
   j++;
if(j==4)
 return(1);
else
   return(0);
}

//To find the support via the signature

void findsupport(struct node**p)
{
struct node*temp=NULL;
char str[max];
int i,j=0;
int b[4]={0};
temp=*p;
while(temp!=NULL)
{
 strcpy(str,temp->arr);
 value(str,b);
 for(i=0;i<numr;i++)
  {
    if(subset(b,sign[i]))
            j++;
  }
 temp->data=j;
 temp=temp->next;
  for(i=0;i<4;i++)
   b[i]=0;
  j=0;
}
}

// The main program

void main()
{
int i=0,j=0;
static int temp=2;
int d[52]={0};
int s[52]={0};
char abc[max];  //This array contains the database initially
char abcd[max];
long int bios_time1,bios_time2;
char m[2],cht;
FILE *fp;
clrscr();
fp=fopen("data1.txt","r"); //Input file that conatins the transaction
if(fp==NULL)
printf("Unable to open the file\n");
printf("\nEnter the value of minimum support :");
scanf("%ld",&minsup);
bios_time1=biostime(0,0l);

for(i=0;i<records;i++)
 for(j=0;j<4;j++)
  sign[i][j]=0;

a[15][0]=1;a[15][1]=0;a[15][2]=0;a[15][3]=0;
for(i=14;i>=0;i--)
{
a[i][0]=a[i+1][0]<<1;a[i][1]=0;a[i][2]=0;a[i][3]=0;
}

a[31][0]=0;a[31][1]=1;a[31][2]=0;a[31][3]=0;
for(i=30;i>15;i--)
{
a[i][0]=0;a[i][1]=a[i+1][1]<<1;a[i][2]=0;a[i][3]=0;
}
```

```
a[47][0]=0;a[47][1]=0;a[47][2]=1;a[47][3]=0;
for(i=46;i>31;i--)
{
a[i][0]=0;a[i][1]=0;a[i][2]=a[i+1][2]<<1;a[i][3]=0;
}

a[51][0]=0;a[51][1]=0;a[51][2]=0;a[51][3]=0x1000;
for(i=50;i>47;i--)
{
a[i][0]=0;a[i][1]=0;a[i][2]=0;a[i][3]=a[i+1][3]<<1;
}

while(fgets(abc,max,fp)!=NULL)
{
            for(i=0;i<strlen(abc);i++)
            {
             if(abc[i]>='A'&&abc[i]<='Z')
             {
                s[abc[i]-'A']++;
             if(d[abc[i]-'A']==0)
                d[abc[i]-'A']++;
             }
             else
             {
               s[abc[i]-'A'-6]++;
             if(d[abc[i]-'A'-6]==0)
               d[abc[i]-'A'-6]++;
             }
            }
   for(i=0;i<52;i++)
     {
     if(d[i]>0)
      {
     if(i<26)
       {abcd[j++]=(char)(i+'A');}
     else
       {abcd[j++]=(char)(i+'A'+6);}
      }
     }
    abcd[j]='\0';
     value(abcd,sign[numr]);
    strcpy(abcd," ");
    for(i=0;i<52;i++)
      d[i]=0;
   j=0;numr++;
}//end of while
```

```
for(i=0;i<52;i++)
{
 if(s[i]>0)
 {
  if(i<26)
  {
  m[0]=(char)(i+'A');
  m[1]='\0';
  }
  else
  {
   m[0]=(char)(i+'A'+6);
   m[1]='\0';
  }
  add(&clist,m,s[i]);
  }
}

printf("\nThis is the C1 list \n");show(clist);
ctol_list(clist);
printf("%d",numr-1);
printf("\nThis is the L1 list \n");
show(llist);
ltoc_list(llist,temp);temp++;
findsupport(&clist);

j=2;
while(clist!=NULL&&llist!=NULL)
{
printf("\nThe C%d List is:\n",j);show(clist);
ctol_list(clist);
printf("This is the L%d list \n",j);show(llist);
ltoc_list(llist,temp);temp++;
findsupport(&clist);
j++;
}
//free all the memory
freenode(&llist);
freenode(&clist);
fclose(fp);

bios_time2=biostime(0,0l);
printf("Time taken in the processing is as %ld",bios_time2-bios_time1);
printf("\nThanks  Have a nice day");
getch();
```

```
}
```

/*Program for the Database pruning method of association rule mining */

```
 //These are the header files included.

#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<alloc.h>
#define max 30
#define n_size 4

//structure for the node in the candidate list and the large itemset list

struct node
{
char arr[n_size];        //It specify the number of items in the itemset
int data;
struct node*next;
};

//structure for the database node

struct dnode
{
char darr[max];
int n;
struct dnode*dnext;
};
struct node*list1=NULL,*list2=NULL;
struct dnode*dbase=NULL,*dbase1=NULL;

//prototypes of the function used

void show(struct node*);
void dshow();
void add(struct node**,char str[],int);
void dadd(struct dnode**,char str[]);
void del(struct node**,char str1[],int);
void ddel(struct dnode**,char str1[]);
void findsup(struct node**p,struct dnode**q);
void removedup(struct node**);
void ctol_list(struct node* );
void ltoc_list(struct node*);
void freenode(struct node**);
void dfreenode(struct dnode**);
int sub(char *s,int a,struct node*len_list);
int minsup=0; //specifies the minimum support

//--To free the list nodes & assign it NULL value---//

void freenode(struct node**q)
{
struct node*temp,*r;
temp=*q;
while(temp!=NULL)
{
 r=temp;
 temp=temp->next;
 free(r);
}
*q=temp;
}

//frees the databse nodes

void dfreenode(struct dnode**q)
{
struct dnode*temp,*r;
temp=*q;
while(temp!=NULL)
{
 r=temp;
 temp=temp->dnext;
 free(r);
}
*q=temp;
}

//--show function----//

void show(struct node*q)
{
printf("\n");
while(q!=NULL)
{
printf("%s %8d\n",q->arr,q->data);
q=q->next;
}
}
```

```
//show the dbase

void dshow()
{
struct dnode*q=NULL;
int i=0;
q=dbase;
printf("\n");
while(q!=NULL)
{
printf("%s %d\n",q->darr,q->n);i++;
q=q->dnext;
}
printf("%d",i);
}

//----add function for the candidate list & the lenght list----//

void add(struct node**q,char str[],int a)
{
struct node *r=NULL,*temp=NULL;
temp=*q;
r=(struct node*)malloc(sizeof(struct node));
r->data=a;
strcpy(r->arr,str);
r->next=NULL;
if(*q==NULL)
   *q=r;
else
 {
  while((temp->next)!=NULL)
      temp=temp->next;
  temp->next=r;
  }
}

//----add function for dbase  list----//

void dadd(struct dnode**q,char str[])
{
struct dnode *r=NULL,*temp=NULL;
temp=*q;
r=(struct dnode*)malloc(sizeof(struct dnode));
r->n=0;
strcpy(r->darr,str);
r->dnext=NULL;
if(*q==NULL)
   *q=r;
else
 {
  while((temp->dnext)!=NULL)
      temp=temp->dnext;
  temp->dnext=r;
  }
}

//--delete function---//

void del(struct node**q,char str[],int a)
{
struct node*old,*temp;
temp=*q;
while(temp!=NULL)
{
  if((temp->data==a)&&(!strcmp(temp->arr,str)))
   {
     if(*q==temp)
     {
     *q=temp->next;
     free(temp);
     return;
     }
     else
     {
       old->next=temp->next;
       free(temp);
       return;
     }
   }
   else
     {
     old=temp;
     temp=temp->next;
     }
}
}

//--delete the dnode from the dbase ---//

void ddel(struct dnode**q,char str[])
```

```
{
struct dnode*old,*temp;
temp=*q;
while(temp!=NULL)
{
  if(!strcmp(temp->darr,str))
   {
    if(*q==temp)
     {
    *q=temp->dnext;
    free(temp);
    return;
     }
    else
     {
      old->dnext=temp->dnext;
      free(temp);
      return;
     }
   }
   else
     {
    old=temp;
    temp=temp->dnext;
     }
}
}

//--remove the duplicates--//

void removedup(struct node**q)
{
 struct node*temp,*x1,*x2,*x3;
 temp=*q;
 x1=temp;
 x2=temp->next;
 while(x1!=NULL)
 {
   while(x2!=NULL)
   {
    if((x1->data==x2->data)&&(!strcmp(x1->arr,x2->arr)))
      {
          x3=x2;
          x2=x2->next;
          del(q,x3->arr,x3->data);
      }
    else
    x2=x2->next;
   }
  x1=x1->next;
  x2=x1->next;
 }
}

//--find the support of the candidate item sets--//

int ffindsup(char str[])
{
int i,k,count=0,j,num=0;
char str2[max];
struct dnode*x=NULL;
x=dbase;
while(x!=NULL)
     {
      count=0;
      strcpy(str2,x->darr);
       for(j=0;j<strlen(str2);j++)
       {
             for(k=0;k<strlen(str);k++)
                {
                     if(str2[j]==str[k])
                        count++;
                }
       }
       if(strlen(str)<=count)
             {num++;(x->n)++;}
       x=x->dnext;
     }//end of inner while
     return(num);
}

//--make lenght list from the candiadate list which staisfy the support--//

void ctol_list(struct node* p)
{
freenode(&list2);
list2=NULL;
while(p!=NULL)
{
  if((p->data)>=minsup)
  {
    add(&list2,p->arr,p->data);
  }
  p=p->next;
}
}

//It tells whether all the a subsets of string s are in the len_list or not

int sub(char *s,int a,struct node*len_list)
{
int i,j,k,l,m,n,c=0,count=0,num=0;   //count counts the number of subset items
struct node*x1=NULL;
struct node*list2=NULL,*x2=NULL;
char *p;
p=(char*)malloc((a+1)*sizeof(char));
switch(a)
{
  case 1 :
    { // To find the 1 item subset
        for(j=0;j<strlen(s);j++)
        {      count++;
                p[c++]=s[j];
                p[c]='\0';
                add(&list2,p,0);c=0;
        }
        break;
    }
  case 2 :
    { // To find the 2 item subset
        for(j=0;j<strlen(s);j++)
        {
          for(k=j+1;s[j]<s[k];k++)
          {
           count++;
          p[c++]=s[j];
          p[c++]=s[k];
          p[c]='\0';
          add(&list2,p,0);c=0;
          }
        }
        break;
    }
  case 3 :
   {  // To find the 3 item subset
        for(i=0;i<strlen(s);i++)
        {
         for(j=i+1;j<strlen(s);j++)
```

```
          {
           for(k=j+1;s[j]<s[k];k++)
           { count++;
          p[c++]=s[i];
          p[c++]=s[j];
          p[c++]=s[k];
          p[c]='\0';
          add(&list2,p,0);c=0;
           }
          }
        }
        break;
  }
 case 4 :
  {  // To find the 4 item subset
        for(i=0;i<strlen(s);i++)
        {
         for(j=i+1;j<strlen(s);j++)
         {
           for(k=j+1;k<strlen(s);k++)
            {
                   for(l=k+1;s[k]<s[l];l++)
                   { count++;
                   p[c++]=s[i];
                   p[c++]=s[j];
                   p[c++]=s[k];
                   p[c++]=s[l];
                   p[c]='\0';
                   add(&list2,p,0);c=0;
                   }
            }
         }
        }
   break;
 }
case 5 :
  {  // To find the 5 item subset
        for(i=0;i<strlen(s);i++)
        {
         for(j=i+1;j<strlen(s);j++)
         {
           for(k=j+1;k<strlen(s);k++)
            {
                   for(l=k+1;s[k]<strlen(s);l++)
                    {
                            for(m=l+1;s[l]<s[m];m++)
```

```
                              { count++;
                              p[c++]=s[i];
                              p[c++]=s[j];
                              p[c++]=s[k];
                              p[c++]=s[l];
                              p[c++]=s[m];
                              p[c]='\0';
                              add(&list2,p,0);c=0;
                              }
                         }
                 }
              }
           }
    break;
   }
 case 6 :
    {  // To find the 6 item subset
           for(i=0;i<strlen(s);i++)
           {
            for(j=i+1;j<strlen(s);j++)
            {
              for(k=j+1;k<strlen(s);k++)
               {
                      for(l=k+1;s[k]<strlen(s);l++)
                       {
                        for(m=l+1;s[l]<strlen(s);m++)
                         {
                           for(n=m+1;s[m]<s[n];n++)
                              { count++;
                              p[c++]=s[i];
                              p[c++]=s[j];
                              p[c++]=s[k];
                              p[c++]=s[l];
                              p[c++]=s[m];
                              p[c++]=s[n];
                              p[c]='\0';
                              add(&list2,p,0);c=0;
                              }
                           }
                       }
                 }
              }
           }
    break;
   }
 }//end of switch
 free(p);
 x1=len_list;
 x2=list2;
  while(x2!=NULL)
   {
     while(x1!=NULL)
     {
     if(!strcmp(x2->arr,x1->arr)&&((x1->data)>=minsup))
              num++;

     x1=x1->next;
     }
    x2=x2->next;
    x1=len_list;
  }
  freenode(&list2);
  if(num==count)
   return(1);
  else
    return(0);
}

//--make the permutation of the lenght list and form the candidate list--//
//--and made the lenght list that satisfy the support--//

void ltoc_list(struct node*q)
{
static int k=2;
int number=0;
char str1[n_size],str2[n_size],str3[n_size];
struct node*x1=NULL,*x2=NULL;
struct dnode*z=NULL;
int d[52]={0},i=0,j=0;

freenode(&list1);
x1=q;
x2=q->next;
while(x1!=NULL)
{
  while(x2!=NULL)
  {
     strcpy(str1,x1->arr);
     strcpy(str2,x2->arr);
            for(i=0;i<strlen(str1);i++)
             {
              if((str1[i]>='A') && (str1[i]<='Z'))
   d[str1[i]-'A']++;
   else
   d[str1[i]-'A'-6]++;
 }
for(i=0;i<strlen(str2);i++)
  {
  if((str2[i]>='A')&&(str2[i]<='Z'))
    {
        if(d[str2[i]-'A']==0)
             d[str2[i]-'A']++;
    }
   else
   {
   if(d[str2[i]-'A'-6]==0)
             d[str2[i]-'A'-6]++;
    }
  }
for(i=0;i<52;i++)
{
   if(d[i]>0)
   {
        if(i<26)
         str3[j++]=(char)('A'+i);
        else
         str3[j++]=(char)('A'+i+6);

        d[i]=0;
   }
}
str3[j]='\0';
if(strlen(str3)==k)
{
 if(k>2)
  {
  if(sub(str3,k-1,q)) //It finds the k-1 subset of str3
    {
    number=ffindsup(str3);
    if(number>=minsup)
    add(&list1,str3,number);
    }
  }
  else
  {
  number=ffindsup(str3);
  if(number>=minsup)
    add(&list1,str3,number);
```

```
                }
              }
            j=0;x2=x2->next;
  } //end of inner while
    x1=x1->next;
    x2=x1->next;
}//end of outer while
removedup(&list1);
//findsup(&list1,&dbase);
k++;
while(dbase!=NULL)
{
if((dbase->n)>=strlen(dbase->darr))
 {
   dadd(&dbase1,dbase->darr);
   z=dbase;
   dbase=dbase->dnext;
   free(z);
 }
 else
 {
   z=dbase;
   dbase=dbase->dnext;
   free(z);
 }
}
dbase=dbase1;
dbase1=NULL;
}

//The main function

void main()
{
int i=0,j=0,num=2;
int d[52]={0};
int s[52]={0};
char abc[max];
char ab[max];
char cht;
char m[2];
long int bios_time1,bios_time2;
FILE *fp;
clrscr();
fp=fopen("data1.txt","r");  //Name of the file that contains the transaction
if(fp==NULL)
```

```
printf("Unable to open the File:\n");
printf("enter the value of minsup :\n");
scanf("%d",&minsup);
bios_time1=biostime(0,0l);
while(fgets(abc,max,fp)!=NULL)
{
for(i=0;i<strlen(abc);i++)
{
            if(abc[i]>='A'&&abc[i]<='Z')
            {
            s[abc[i]-'A']++;
            if(d[abc[i]-'A']==0)
            d[abc[i]-'A']++;
            }
      else
            {
            s[abc[i]-'A'-6]++;
            if(d[abc[i]-'A'-6]==0)
            d[abc[i]-'A'-6]++;
            }
}
   for(i=0;i<52;i++)
     {
      if(d[i]>0)
      {
      if(i<26)
       {ab[j++]=(char)(i+'A');}
      else
       {ab[j++]=(char)(i+'A'+6);}
      }
     }
    ab[j]='\0';
    dadd(&dbase,ab);
    strcpy(ab,"");
    for(i=0;i<52;i++)
      d[i]=0;
   j=0;
}//end of while
//printf("INITIALLY DATABASE IS:\n");
//dshow();
//getch();

for(i=0;i<52;i++)
{
 if(s[i]>0)
 {
  if(i<26)
  {
  m[0]=(char)(i+'A');
  m[1]='\0';
  }
  else
  {
   m[0]=(char)(i+'A'+6);
   m[1]='\0';
  }
  add(&list1,m,s[i]);
 }
}
printf("    Candidate(C1)     \n");
printf(" ITEMS   Support\n");
show(list1);//getch();
ctol_list(list1);
printf("\n   Lenght(L1)     \n");
printf(" ITEMS   Support\n");
show(list2);//getch();
ltoc_list(list2);
//dshow();getch();
while((list1!=NULL)&&(list2!=NULL))
{
printf("    Candidate(C%d)     \n",num);
printf(" ITEMS            Support\n");
show(list1);//getch();
ctol_list(list1);
printf("\n   Lenght(L%d)     \n",num);
printf(" ITEMS            Support\n");
show(list2);//getch();
ltoc_list(list2);//dshow();getch();
num++;
}
dfreenode(&dbase);
freenode(&list1);
freenode(&list2);
fclose(fp);
bios_time2=biostime(0,0l);
printf("Time taken in the processing is as %ld",bios_time2-bios_time1);
printf("\nThanks  Have a nice day");
getch();
}
```

References

[1] M. S. Chen, J. Han, and P. S. Yu. Data mining: An overview from a database perspective. IEEE Trans. Knowledge and Data Engineering, 8:866-883, 1996.

[2] U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy. Advances in Knowledge Discovery and Data Mining. AAAI/MIT Press, 1996.

[3] W. J. Frawley, G. Piatetsky-Shapiro and C. J. Matheus, Knowledge Discovery in Databases: An Overview. In G. Piatetsky-Shapiro et al. (eds.), Knowledge Discovery in Databases. AAAI/MIT Press, 1991.

[4] J. Han and M. Kamber. Data Mining: Concepts and Techniques. Morgan Kaufmann, 2000.

[5] T. Imielinski and H. Mannila. A database perspective on knowledge discovery. Communications of ACM, 39:58-64, 1996.

[6] G. Piatetsky-Shapiro, U. M. Fayyad, and P. Smyth. From data mining to knowledge discovery: An overview. In U.M. Fayyad, et al. (eds.), Advances in Knowledge Discovery and Data Mining, 1-35. AAAI/MIT Press, 1996.

[7] G. Piatetsky-Shapiro and W. J. Frawley. Knowledge Discovery in Databases. AAAI/MIT Press, 1991.

[8] An Effective Hash-Base Algorithm for Mining Association Rules. Jong Soo Park,* Ming Chen and Philip S . Yu, IBM Thomas J. Watson Research Center ,New York 10598

[9] An Hash – Mine algorithm for discovery of frequent itemsets, Marek Wojciechowski , Maciej Zakrzewicz Institute of computer science, ul. Piotrowo 3a Poland.

[10] An Efficient Algorithm for mining Association rules in Large databases , Ashok Savasere,Edward Omiecinski, Shamkant Navathe ,college of computing , Georgia Institute of technology, Atlanta , GA 30332 .

[11] A Fast Apriori implementation, infomatics Laboratory, Computer and Automation Research institute, Hungarian academy of sciences.

[12] Mining Large Itemsets for Association Rules ,Charu C. Aggarwal ,IBM research Lab.

[13] Mining association rules between sets of items in large databases, Rakesh Agarawal,Tomasz Imielinski*,Arun swami,IBM research lab.

[14] Fast algorithm for mining association rules,Rakesh Agarwal Ramakrishna Srikannt*,IBM research labs 650 Harry Road , San Jose , CA 95120.

[15] J Han, Y. Cai , and N,Cercone . Knowledge Discovery in database: An attribute – oriented approach. Proceeding of the 18 th International Conference on very large data bases, Page 547-559,august 1992

[16] [Aggrawal1998c] Charu C. Aggarwal, and Philip S. Yu, A New Framework for Itemset Generation, *Principles of Database Systems (PODS)* 1998, Seattle, WA.

[17] Rakesh Agrawal and Ramakrishnan Srikant, Mining Sequential Patterns, *Proceedings of the 11th IEEE International Conference on Data Engineering*, Taipei,

Taiwan, March 1995, IEEE Computer Society Press.

[18] Roberto J. Bayardo Jr., Rakesh Agrawal, Dimitris Gunopulos, Constarint-Based Rule Mining in Large, Dense Databases, *Proceedings of the 15th International Conference on Data Engineering*, 23-26 March 1999, Sydney, Australia, pp.188-197

[19] [Brin1997a] Sergey Brin, Rajeev Motwani, Jeffrey D. Ullman, and Shalom Tsur, Dynamic Itemset Counting and Implication Rules for Market Basket Data, *Proceedings of the ACM SIGMOD Conference*, pp. 255-264, 1997

[20] [Fayyad1996] Usama M. Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth,From Data Mining to knowledge Discovery: An Overview, *Advances in Knowledge Discovery and Data Mining,* Edited by Usama M. Fayyad, Gregory Piatetsky-Shapiro, Padraic Smyth, and Ramasamy Uthurusamy, AAAI Press, 1996, pp 1-34.

[21] R. Feldman, Y. Aumann, A. Amir, and H. Mannila, Efficient Algorithms for Discovering Frequent Sets in Incremental Databases, *Proceedings of the 1997 SIGMOD Workshop on DMKD,* May 1997. Tucson, Arizona.

[22] [Han1995] Jiawei Han and Yongjian Fu, Discovery of Multiple-Level Association Rules from Large Databases, *Proceedings of the 21nd International Conference on Very Large Databases*, pp. 420-431, Zurich, Swizerland, 1995

[23] [Liu1998] Bing Liu, Wynne Hsu and Yiming Ma, Integrating Classification and Association Rule Mining, *American Association of Artificail Intelligence*, *KDD*1998, pp. 80-86.

[24] [Shintani1998] Takahiko Shintani and Masaru Kitsuregawa, Parallel Mining Algorithms for Generalized Association Rules with Classification Hierarchy, *Proceedings ACM SIGMOD International Conference on Management of Data, SIGMOD 1998,* June 2-4, 1998, Seattle, Washington, USA,

[25][Srikant1995] Ramakrishnan Srikant and Rakesh Agrawal, Mining Generalized Association Rules, *Proceedings of the 21nd International Conference on Very Large Databases*, pp. 407-419, Zurich, Swizerland, 1995