

# IMPLEMENTATION OF DISTRIBUTED ASSOCIATION RULE MINING ALGORITHM

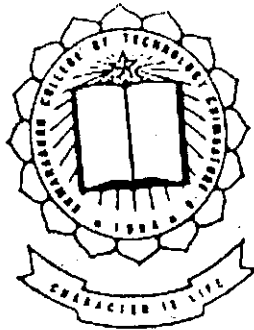
Thesis submitted in partial fulfillment of the requirements for the award of the Degree of  
**MASTER OF ENGINEERING IN COMPUTER SCIENCE AND ENGINEERING**  
OF BHARATHIAR UNIVERSITY

P-658

By  
**A.SAMPATH KUMAR**  
(Reg.No.0037K0008)



Under the Guidance of  
**Mr. K. R.BASKARAN B.E., M.S.**  
Asst. Professor  
Dept. of CS&E, KCT



DEPARTMENT OF COMPUTER SCIENCE ENGINEERING  
**KUMARAGURU COLLEGE OF TECHNOLOGY**

(Affiliated to Bharathiar University)

COIMBATORE – 641 006

2000 - 2001

# CERTIFICATE

Department of Computer Science and Engineering

Certified that this is a bonafide report

of

the thesis work done by

**A.SAMPATH KUMAR**

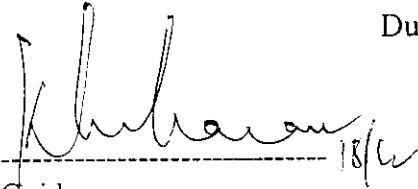
(Reg.No.0037K0008)

at

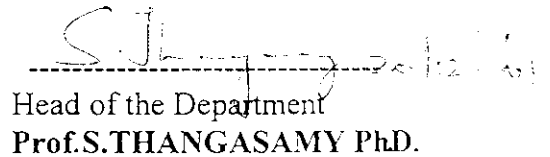
**KUMARAGURU COLLEGE OF TECHNOLOGY**

**COIMBATORE – 641 006**

During the year – 2000 – 2001



Guide  
**Mr.K.R.BASKARAN B.E., M.S.**  
Computer Science and Engineering Department  
K.C.T., Coimbatore.



Head of the Department  
**Prof.S.THANGASAMY Ph.D.**

Place : Coimbatore

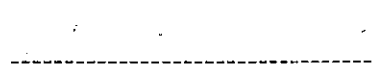
Date : 20-12-2001

Submitted for Viva – Voce examination held at

Kumaraguru College of technology on 22-12-2001



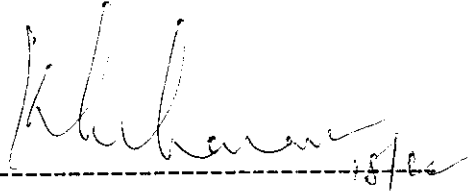
Internal examiner



External Examiner

## CERTIFICATE

This is to certify that this thesis work entitled “**IMPLEMENTATION OF DISTRIBUTED ASSOCIATION RULE MINING ALGORITHM**” being submitted by **A.SAMPATH KUMAR**, (Reg.No.0037K0008) for the award of degree of **MASTER OF ENGINEERING IN COMPUTER SCIENCE<sup>& ENGG</sup>**, is a bonafide work carried under my guidance. The results embodied in this thesis have not been submitted to any other university or institute for the award of any degree or diploma.



-----

**Mr.K.R.BASKARAN B.E., M.S.**

Assistant Professor

Department of Computer Science and Engineering

Kumaraguru College of Technology

Coimbatore.



Dedicated to  
My  
Beloved Parents



# Acknowledgement

---

## **ACKNOWLEDGEMENT**

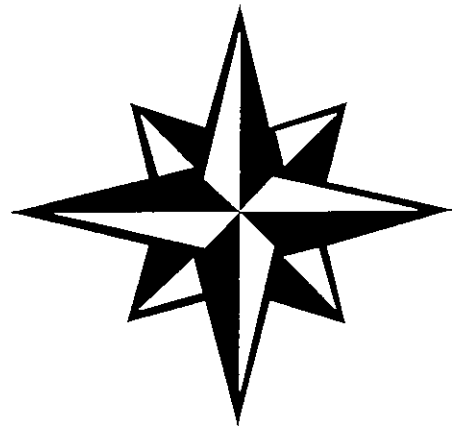
The author wishes to take this opportunity to offer a respectful note of thanks to **Dr. K.K.PADMANABAN, Ph.D.**, principal of the college, for the excellent facilities made available to accomplish this project work.

The author would like to deem it a privilege to record his sincere thanks to **Prof.S.THANGASAMY, Ph.D.**, Head of the Department of Computer Science Engineering for his valuable suggestions and motivations during the entire period of this course.

The author would like to express his heartfelt gratitude to his guide **Mr. K.R.BASKARAN B.E., M.S.** for his valuable guidance, suggestions, and consistent encouragement, which lead to the successful completion of the project.

The author would like to express his heartfelt gratitude to his co guide **Mr. A. MUTHUKUMAR M.Sc.,M.C.A.,M.Phil.** for his valuable guidance, suggestions, and consistent encouragement, which lead to the successful completion of the project.

**A.SAMPATH KUMAR**



# Synopsis

---

## SYNOPSIS

Data mining is the search for relationships and global patterns that exist in large databases but are hidden among the vast amount of data. These relationships represent valuable knowledge about the database and the objects in the database. It is a powerful new technology with vast potential to help companies focus on their data warehouses.

Each data mining application is supported by a set of algorithmic approaches used to extract the relevant relationships from the data. In the project I have implemented the Apriori algorithm for mining the database. The Apriori algorithm searches association rules in the database given a minimum support value. The user has to specify the minimum support value for finding the associative relationships.

I have implemented the project on distributed databases. A distributed database is a collection of data which belong logically to the same system but are spread over different sites in a network. This greatly reduces the time taken to mine large and voluminous data located in a single database. The project makes use of two database servers and one client machine. The client machine provides user control over the mining algorithms running in the two database servers. The mining algorithms have been implemented using Java which focuses on the remote method invocation concept.





# Contents

---

# CONTENTS

1. INTRODUCTION	1
1.1 THE CURRENT STATUS OF THE PROBLEM TAKEN UP	7
1.2 RELEVANCE AND IMPORTANCE OF THE TOPIC	9
2. LITERATURE SURVEY	10
3. PROPOSED LINE OF ATTACK	17
4. DETAILS OF THE PROPOSED METHODOLOGY	18
5. RESULTS OBTAINED	29
6. CONCLUSIONS AND FUTURE OUTLOOK	36
7. REFERENCES	37

APPENDICES



# Introduction

---

# 1. INTRODUCTION

## **Data Mining**

Data mining is a process of inferring, mining or extracting knowledge from huge amount of data. With an enormous amount of data stored in databases and data warehouses, it is increasingly important to develop powerful tools for analysis of such data and mining interesting knowledge from it. There are many other terms carrying similar or slightly different meanings to data mining such as knowledge mining from the database, knowledge extraction or data/pattern analysis. Data mining techniques are the result of a long process of research and product development.

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

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

Continuous innovations in computer processing power, disk storage, and statistical software are dramatically increasing the accuracy of analysis while driving down the cost.

## ARCHITECTURE OF A TYPICAL DATA MINING

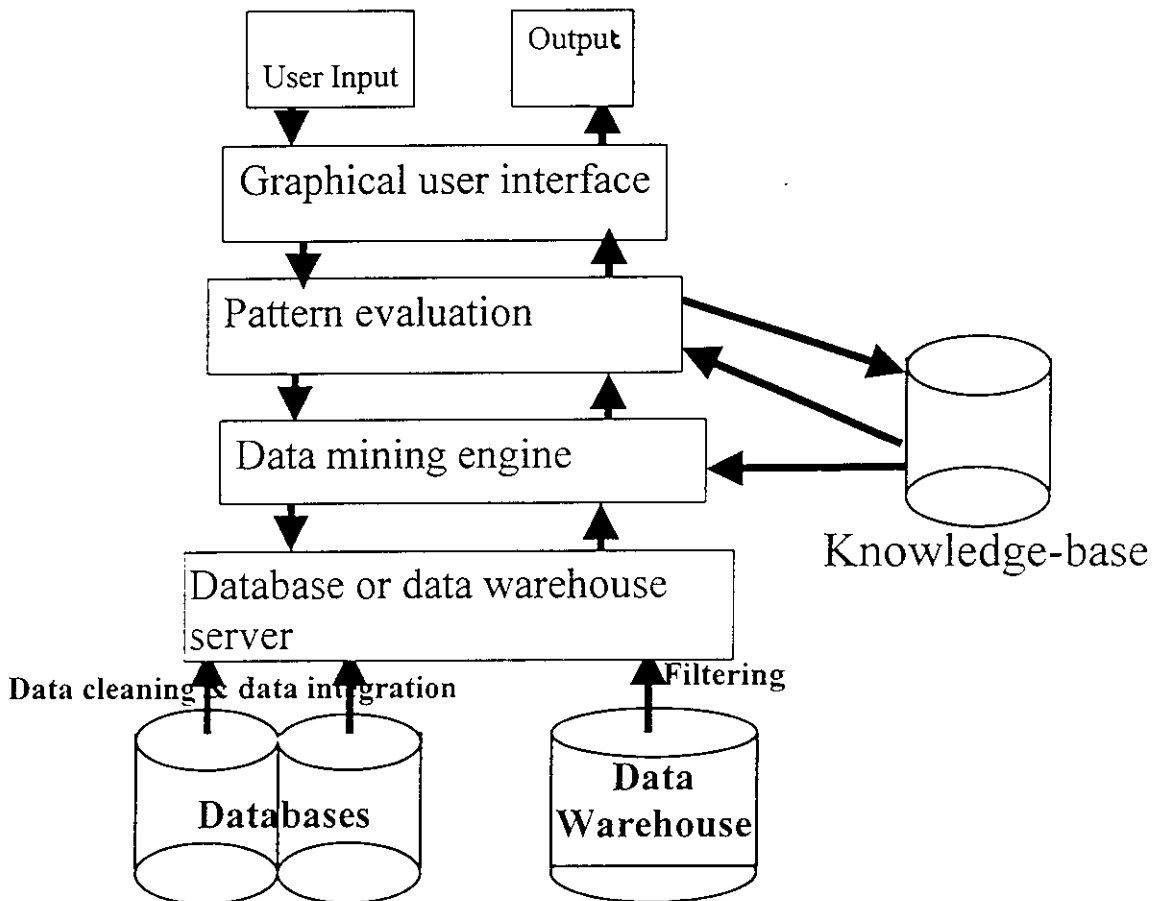


Figure: architecture of a typical data mining system

**Database, data warehouse, or other information repository:** This is one or a set of database, data warehouse, spreadsheets , or other kinds of information repositories. Data cleaning and data integration techniques may be performed on the data.

**Database or data warehouse server:** The database or data warehouse server is responsible for fetching the relevant data, based on the user's data mining request.

**Knowledge base:** This is the domain knowledge that is used to guide the search, or evaluate the interestingness of resulting patterns. Such knowledge can include concept hierarchies, used to organize attributes or attribute values into different levels of abstraction. Knowledge such as user beliefs, which can be used to assess a pattern's interestingness based on its unexpectedness, may also be included.

**Data mining engine:** This is essential to the data mining system and ideally consists of a set of functional modules for tasks such as characterization, association, classification, cluster analysis, and evolution and deviation analysis.

**Pattern evaluation module :** This component typically employs interestingness measures and interacts with the data mining modules so as to focus the search toward interesting patterns. It may use interestingness threshold to filter out discovered patterns.

**Graphical user interface:** This module communicates between users and the data mining system, allowing the user to interact with the system by specifying a data mining query or task, providing information to help focus the search, and performing of exploratory data mining based on the intermediate data mining result.

**Data mining consists of five major elements:**

- Extract, transform, and load transaction data onto the data warehouse system.
- Store and manage the data in a multidimensional database system.

- Provide data access to business analysts and information technology professionals.
- Analyze the data by application software.
- Present the data in a useful format, such as a graph or table.

### **Data Mining Approaches**

- **Artificial neural networks:** Non-linear predictive models that learn through training and resemble biological neural networks in structure.
- **Genetic algorithms:** Optimization techniques that use processes such as genetic combination, mutation, and natural selection in a design based on the concepts of natural evolution.
- **Decision trees:** Tree-shaped structures that represent sets of decisions. These decisions generate rules for the classification of a dataset. Specific decision tree methods include Classification and Regression Trees (CART) and Chi Square Automatic Interaction Detection (CHAID) . CART and CHAID are decision tree techniques used for classification of a dataset. They provide a set of rules that you can apply to a new (unclassified) dataset to predict which records will have a given outcome. CART segments a dataset by creating 2-way splits while CHAID segments using chi square tests to create multi-way splits.
- **Nearest neighbor method:** A technique that classifies each record in a dataset based on a combination of the classes of the  $k$  record(s) most similar to it in a historical dataset (where  $k \geq 1$ ). Sometimes called the  $k$ -nearest neighbor technique.
- **Rule induction:** The extraction of useful if-then rules from data based on statistical significance.

- **Data visualization:** The visual interpretation of complex relationships in multidimensional data. Graphics tools are used to illustrate data relationships.

## **DISTRIBUTED DATABASE MANAGEMENT SYSTEMS (DDBMSs)**

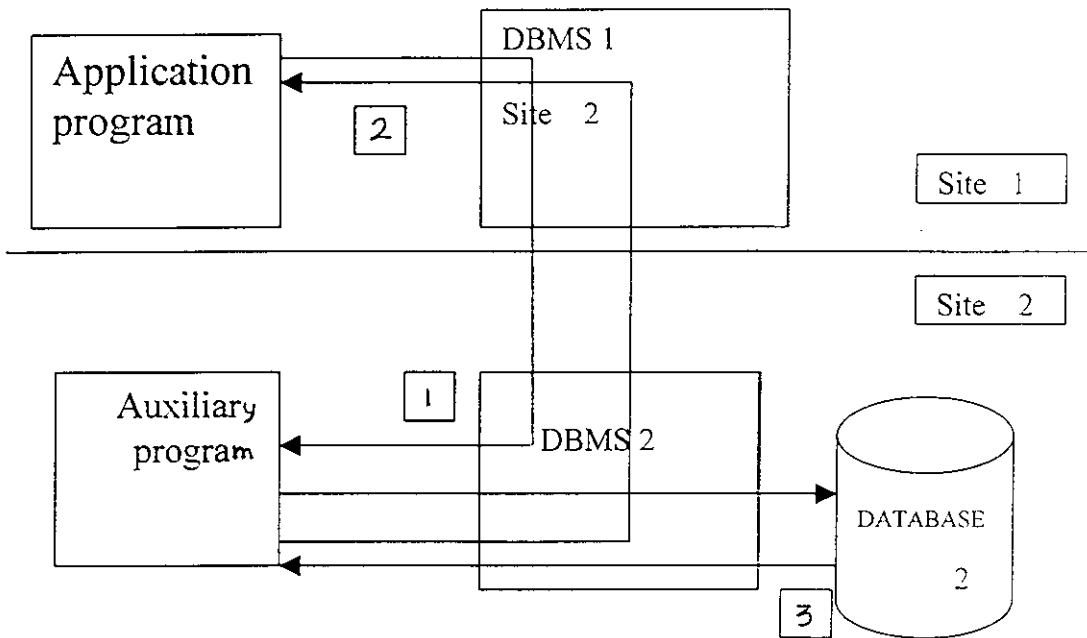
A distributed database is a collection of data which are distributed over different computer of a computer network. Each site of the network has autonomous processing capability and can perform local application. Each site also participates in the execution of at least one global application, which requires accessing data at retrieval sites using a communication subsystem.

A distributed database management system supports the creation and maintenance of distributed databases. The software components which are typically necessary for building a distributed database in this case are:

1. The database management component (DB)
2. The data communication component (DC)
3. The data dictionary (DD), which is extended to represent information about the distribution of data in the network.
4. The distributed database component (DDB)

The access to a remote database by an application can be performed by which the application requires the execution of an auxiliary program at a remote site. The auxiliary program, written by the auxiliary programmer, accesses the remote database and returns the result to the requesting application.





Remote access via a auxiliary program

**Fig: Type of access to a distributed database**

- 1-request for execution of auxiliary program
- 2-Global result
- 3-databaase access primitives and results

An important property of DDBMSs is whether they are homogeneous or heterogeneous. Homogeneous DDBMS refers to a DDBMS with the same DBMS at each site, even if the computers and the operating systems are not same. Heterogeneous uses different DBMS. Heterogeneous DDBMSs add the problem of translating between different data models of the different local DBMSS to the complexity of homogeneous DDBMSs .The problems of heterogeneous DDBMSs are very hard.

## 1.1 THE CURRENT STATUS OF THE PROBLEM TAKEN UP

### 1. Incomplete data

Some data may be missing (e.g., some fields may be left blank in a user profile, or perhaps the manufacturer has only very limited test data to report). The question is what to do about such situations. Sometimes the fact that data is missing is itself a valuable piece of information (e.g., surgical information for a patient who has never had surgery, disease information for a patient who has never been sick). At other times, the missing data constitutes a genuine problem (e.g., missing diagnostic information after a test has been performed).

### 2. Noisy data

The fields may contain incorrectly entered information. How does this affect the certainty factor or confidence level of the results?

### 3. Temporal data

Since databases grow rapidly, how can data be incrementally added to our results? Is current data "worth more" than data from, say, a year ago? Data is also subject to change. What effect should this have in the knowledge discovery process? Can results be "undone", or must the entire knowledge discovery process start from scratch to pick up changes?

### 4. An extremely large amount of data

Some datasets can grow significantly over time. How should such datasets be processed? One option is to perform parallel processing, whereby  $n$  processors each process approximately  $1/n$ 'th of the data in approximately  $1/n$ 'th of the time. Another option is to avoid processing the entire dataset, and simply sample the data. Even though this may result in a loss of information or in a reduced confidence level, perhaps the accuracy vs. efficiency trade-off warrants such an approach.

## 5. Non-textual data

There are many types of data that need to be manipulated, including image data, multimedia data (video, sound), spatial data in Geographic Information Systems, and user-defined data types.

## 6. Controversial data

There are privacy issues to be considered. Probing databases for personal information (especially medical information) may violate privacy laws. For example, using data mining techniques to create mailing lists of potential customers is controversial. Even probing government databases for instances of fraud or criminal intent has privacy implications. Similarly, probing medical databases "in the interest of science" while trying to isolate common characteristics among affected individuals (for a cure to a disease) can be controversial.

## 1.2 RELEVANCE AND IMPORTANCE OF THE TOPIC

### Data Mining Rules

Data mining Rules include the following-

- 1) Classification rule
- 2) Association rule
- 3) Sequential Analysis

The classification-rule learning involves finding rules that partition given data into predefined classes. In the data mining domain where millions of records and a large number of attributes are involved, the execution time of existing algorithms can become prohibitive, particularly in interactive applications.

An association rule is a rule which implies certain association relationships among a set of objects in a database. In this process we discover a set of association rules at multiple levels of abstraction from the relevant set(s) of data in a database.

In sequential Analysis, we seek to discover patterns that occur in sequence. The input data is a set of sequences, called data-sequences. Each data sequence is a ordered list of transactions (or item sets), where each transaction is a sets of items (literals). Typically there is a transaction-time associated with each transaction. A sequential pattern also consists of a list of sets of items. The problem is to find all sequential patterns with a user-specified minimum support, where the support of a sequential pattern is the percentage of data sequences that contain the pattern.

## POTENTIAL APPLICATIONS

Data mining has many and varied fields of application some of which are listed below.

- **Retail/Marketing**

1. Identify buying patterns from customers
2. Find associations among customer demographic characteristics
3. Predict response to mailing campaigns
4. Market basket analysis

- **Banking**

1. Detect patterns of fraudulent credit card use
2. Identify 'loyal' customers
3. Predict customers likely to change their credit card affiliation



# Literature Survey

---

## 2. LITERATURE SURVEY

---

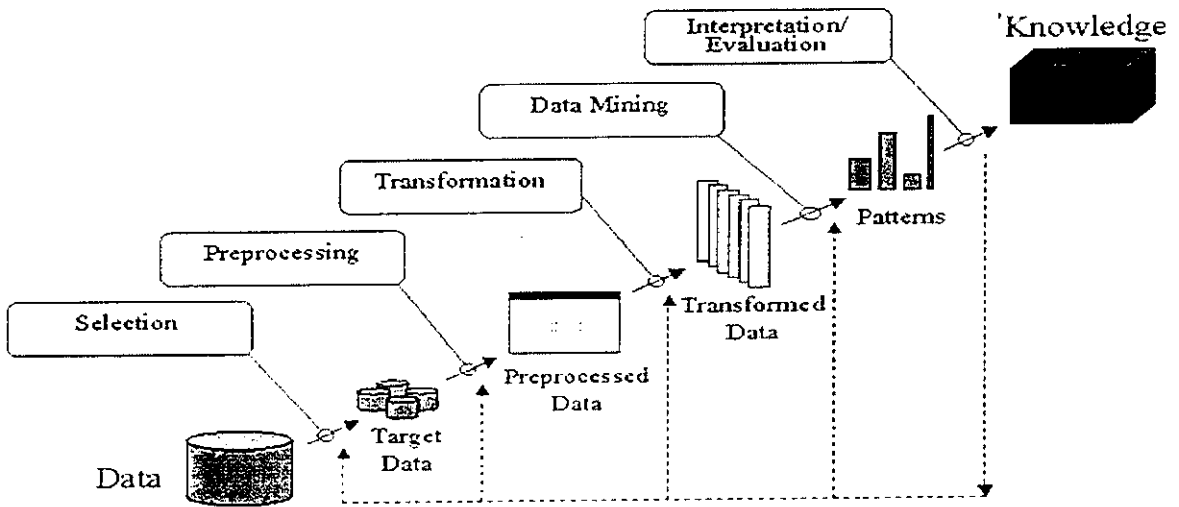
### **KDD PROCESS (from 7.2)**

The term *Knowledge Discovery in Databases* or KDD for short, refers to the broad process of finding knowledge in data, and emphasizes the "high-level" application of particular data mining methods. It is of interest to researchers in machine learning, pattern recognition, databases, statistics, artificial intelligence, knowledge acquisition for expert systems, and data visualization.

### **GOAL OF KDD PROCESS**

The unifying goal of the KDD process is to extract knowledge from data in the context of large databases. It does this by using **data mining methods** (algorithms) to extract (identify) what is deemed knowledge, according to the specifications of measures and thresholds, using a database along with any required preprocessing, sub sampling, and transformations of that database.

## AN OUTLINE OF THE STEPS OF THE KDD PROCESS



1. Developing an understanding of
  1. the application domain
  2. the relevant prior knowledge
  3. the goals of the end-user
2. Creating a target data set: selecting a data set, or focusing on a subset of variables, or data samples, on which discovery is to be performed.
3. Data cleaning and preprocessing.
  1. Removal of noise or outliers.
  2. Collecting necessary information to model or account for noise.
  3. Strategies for handling missing data fields.



#### 4. Accounting for time sequence information

4.Data reduction and projection.

5.Choosing the data mining task.

6.Choosing the data mining algorithm(s).

7.Data mining.

8.Interpreting mined patterns.

9.Consolidating discovered knowledge.

### **WHY DISTRIBUTED DATA BASE (from 7.1.2)**

There are several reasons why distributed databases were developed.

#### **Organizational and economic reasons:**

A distributed database approach naturally fits more to the structure of the problem. The organizational and economic motivations are probably the most important reason for the development for distributed database.

#### **Interconnection of existing database:**

For performing global applications from the existing database, the distributed database is created bottom-up from the existing database.

Incremental growth: for adding new relatively autonomous organizational units (new warehouses),the distributed database supports a smooth

incremental growth with a minimum degree of impact on the already existing exists.

### **Reduced communication overhead**

In geographically distributed database, many applications are local; clearly reduce the communication overhead with respect to a centralized database. The maximization of locality of applications is one of the primary objectives in distributed database.

### **Performance considerations**

Distributed databases have the advantage of decomposing data reflects application dependent criteria which maximize application locality; the load is shared between different processors in a multi-processor environment is minimized. The load is shared between different processors, and critical bottlenecks, such as the communication network itself or common services of the whole system are avoided.

### **Reliability and Availability**

The distributed database approach, especially with redundant data, can be used also in order to obtain higher reliability and availability.

## **DATA MINING VERSUS QUERY TOOLS(from 7.1.1)**

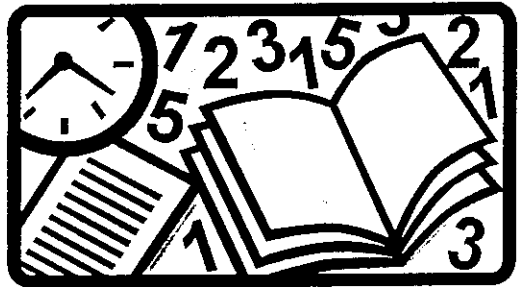
Firstly it is important to realize that query tools and data mining tools are complementary. A data mining tool does not replace a query tool, but it

does give the user a lot of additional possibilities. Suppose that you have a large file containing millions of records that describe your customer's purchase over the last ten years. There is a wealth of potentially useful knowledge in such a file, most of which can be found by firing normal queries at the database, such as 'who bought which product on what date?', 'what is the average turnover in a certain sales region in July?' and so on. There is, however, knowledge hidden in your database that is much harder to find using SQL. Examples would be the answers to questions such as 'what is an optimal segmentation of my clients?' (that is how do I find the most important different customer profiles?), or 'What are the most important trends in customer behavior?'. Of course, these questions could be answered using SQL. You could try to guess for yourself some defining criteria for customer profiles and query the database to see whether they work or not. In a process of trial and error, one could gradually develop intuitions about what the important distinguishing attributes are. Processing in such a way, it could take days or months to find an optimal segmentation for a large database, while a machine-learning algorithm like a neural network or a genetic algorithm could find the answer automatically in a much shorter time, sometimes even in minutes or a couple of hours. Once the data mining tool has found a segmentation, you can use your query environment again to query and analyze the profiles found.

One could say that if you know exactly what you are looking for, use SQL; but if you know only vaguely what you are looking for, turn to data mining. Generally there are far more occasions when your initial approach is vague than times when you know precisely what you are looking for, it is this that has motivated the recent surge of interest in data mining.

It is clear that KDD is not an activity that stands on its own: a good foundation in terms of data warehouse is a necessary condition of its

effective implementation .Noisy and incomplete data, and legal and privacy issues, constitute important problems. One must pay attention to the process of data cleaning- remove duplicate records, correct typographical errors in strings, and missing information, and so on. In KDD too, the old 'garbage in, garbage out' rule still holds. To implement KDD in an organization is to start a process of permanent refinement and dealing of data. The real aim should be ultimately to create of data. The real aim should be ultimately to create a self-learning organization.



Proposed  
Line of  
Attack

---

### 3. PROPOSED LINE OF ATTACK

---

The project implements an important associative form of data mining technique. It employs the basic Apriori algorithm in mining a given database. The Apriori algorithm searches for association patterns or relationships in the database. The project also aims at implementing the above data mining methodology on a distributed database.

The problem of mining association rules in the project has been proposed to be reduced to two sub problems. They are,

- (1) Finding all large itemsets for a given minimum support threshold from a given database, and
- (2) Generating the association rules from the large itemsets found.

The project is demonstrated using a client machine and two servers with the client machine as the user interface and the server as the database. The data in the database is stored as a flat file and retrieved from them using the Apriori algorithm for extraction of associative rules.



Details  
of  
Proposed  
Methodology

---

## 4. DETAILS OF PROPOSED METHODOLOGY

---

### Algorithm for Mining Association Rules

Let  $I = \{i_1, i_2, \dots, i_n\}$  be a set of items. Let DB be a database of transactions, where each transaction  $T$  consists of a set of items such that  $T \subseteq I$ . Given an itemset  $X \subseteq I$ , a transaction  $T$  contains  $X$  if and only if  $X \subseteq T$ . An association rule is an implication of the form  $X \Rightarrow Y$ , where  $X \subseteq I$ ,  $Y \subseteq I$  and  $X \cap Y = \emptyset$ . The association rule  $X \Rightarrow Y$  holds in DB with confidence  $c$  if the probability of a transaction in DB which contains  $X$  also contains  $Y$  is  $c$ . The association rule  $X \Rightarrow Y$  has support  $s$  in DB if the probability of a transaction in DB contains both  $X$  and  $Y$  is  $s$ . The task of mining association rules is to find all the association rules whose support is larger than a minimum support threshold and whose confidence is larger than a minimum confidence threshold.

For an itemset  $X$ , its support is the percentage of transactions in DB which contains  $X$ , and its support count, denoted by  $X:\text{sup}$ , is the number of transactions in DB containing  $X$ . An itemset  $X$  is large (or more precisely, frequently occurring) if its support is no less than the minimum support threshold. An itemset of size  $k$  is called a  $k$ -itemset. It has been shown that the problem of mining association rules can be reduced to two subproblems :

- (1) find all large itemsets for a given minimum support threshold, and
- (2) generate the association rules from the large itemsets found.



## Distributed Algorithm for Mining Association Rules

We examine the mining of association rules in a distributed environment. Let  $DB$  be a database with  $D$  transactions. Assume that there are  $n$  sites  $S_1; S_2; \dots; S_n$  in a distributed system and the database  $DB$  is partitioned over the  $n$  sites into  $\{DB_1; DB_2; \dots; DB_n\}$ , respectively. Let the size of the partitions  $DB_i$  be  $D_i$ , for  $i = 1; \dots; n$ . Let  $X:\text{sup}$  and  $X:\text{sup}_i$  be the support counts of an itemset  $X$  in  $DB$  and  $DB_i$ , respectively.  $X:\text{sup}$  is called the global support count, and  $X:\text{sup}_i$  the local support count of  $X$  at site  $S_i$ . For a given minimum support threshold  $s$ ,  $X$  is globally large if  $X:\text{sup} \geq s \cdot D$ ; correspondingly,  $X$  is locally large at site  $S_i$ , if  $X:\text{sup}_i \geq s \cdot D_i$ . In the following,  $L$  denotes the globally large itemsets in  $DB$ , and  $L(k)$  the globally large  $k$ -itemsets in  $L$ . The essential task of a distributed association rule mining algorithm is to find the globally large itemsets  $L$ .

The algorithm is an adaptation of the Apriori algorithm in the distributed case. At each iteration, it generates the candidate sets at every site by applying the Apriori gen function on the set of large itemsets found at the previous iteration. Every site then computes the local support counts of all these candidate sets and broadcasts them to all the other sites. Subsequently, all the sites can find the globally large itemsets for that iteration, and then proceed to the next iteration.

## Techniques for Distributed Data Mining

### Generation of Candidate Sets

It is important to observe some interesting properties related to large itemsets in distributed environments since such properties may substantially reduce the number of messages to be passed across network at mining association rules. There is an important relationship between large itemsets and the sites in a distributed database: every globally large itemsets must be locally large at some site(s). If an itemset  $X$  is both globally large and locally large at a site  $S_i$ ,  $X$  is called *gl-large* at site  $S_i$ . The set of *gl-large* itemsets at a site will form a basis for the site to generate its own candidate sets.

Two monotonic properties can be easily observed from the locally large and *gl-large* itemsets. First, if an itemset  $X$  is locally large at a site  $S_i$ , then all of its subsets are also locally large at site  $S_i$ . Secondly, if an itemset  $X$  is *gl-large* at a site  $S_i$ , then all of its subsets are also *gl-large* at site  $S_i$ . Notice that a similar relationship exists among the large itemsets in the centralized case. At each iteration (the  $k$ -th iteration), the *gl-large*  $k$ -itemsets can be computed at each site  $S_i$  according to the following procedure.

**1. Candidate sets generation:** Generate the candidate sets  $CG_i(k)$  based on the *gl-large* itemsets found at site  $S_i$  at the  $(k - 1)$ -st iteration using the formula,  $CG_i(k) = \text{Apriori gen}(GL_i(k-1))$ .

**2. Local pruning:** For each  $X \in CG_i(k)$ , scan the partition  $DB_i$  to compute the local support count  $X:\text{sup}_i$ . If  $X$  is not locally large at site  $S_i$ , it is excluded from the candidate sets  $LL_i(k)$ . (Note: This pruning only

removes  $X$  from the candidate set at site  $S_i$ .  $X$  could still be a candidate set at some other site.)

**3. Support count exchange:** Broadcast the candidate sets in  $LL_i(k)$  to other sites to collect support counts. Compute their global support counts and find all the gl-large  $k$ -itemsets in site  $S_i$ .

**4. Broadcast mining results:** Broadcast the computed gl-large  $k$ -itemsets to all the other sites.

For clarity, the notations used so far are listed in the Table .

$D$	number of transactions in DB
$s$	support threshold minsup
$L(k)$	globally large $k$ -itemsets
$CA(k)$	candidate sets generated from $L(k)$
$X:sup$	global support count of $X$
$D_i$	number of transactions in $DB_i$
$GL_i(k)$	globally large $k$ -itemsets at $S_i$
$CG_i(k)$	candidate sets generated from $GL_i(k)$
$LL_i(k)$	locally large $k$ -itemsets in $CG_i(k)$
$X:sup_i$	local support count of $X$ at $S_i$

Table : Notation Table.

### Global Pruning of Candidate Sets

The local pruning at a site  $S_i$  uses only the local support counts found in  $DB_i$  to prune a candidate set. In fact, the local support counts from other sites can also be used for pruning. A global pruning technique is developed to facilitate such pruning and is outlined as follows.

At the end of each iteration, all the local support and global support counts of a candidate set  $X$  are available. These local support counts can be broadcasted together with the global support counts after a candidate set is found to be globally large. Using this information, some global pruning can be performed on the candidate sets at the subsequent iteration. Assume that the local support count of every candidate itemset is broadcasted to all the sites after it is found to be globally large at the end of an iteration. Suppose  $X$  is a size- $k$  candidate itemset at the  $k$ -th iteration. Therefore, the local support counts of all the size- $(k-1)$  subsets of  $X$  are available at every site.

### **Count Polling**

The local support count of every candidate itemset is broadcasted from every site to every other site. Therefore, the number of messages required for count exchange for each candidate itemset is  $O(n^2)$ , where  $n$  is the number of partitions. In our method, if a candidate itemset  $X$  is locally large at a site  $S_i$ ,  $S_i$  needs  $O(n)$  messages to collect all the support counts for  $X$ . In general, few candidate itemsets are locally large at all the sites. To ensure that DM requires only  $O(n)$  messages for every candidate itemset in all the cases, a count polling technique is introduced. At the  $k$ -th iteration, after the pruning phase, (both local and global pruning), has been completed, DM uses the following procedure at each site  $S_i$  to do the count polling.

1. Send candidate sets to polling sites: At site  $S_i$ , for every polling site  $S_j$ , find all the candidate itemsets in  $LL_i(k)$  whose polling site is  $S_j$  and store them in  $LL_{i;j}(k)$  (i.e., candidates are being put into groups by their polling sites). The local support counts of the candidate itemsets are also stored in the corresponding set  $LL_{i;j}(k)$ . Send each  $LL_{i;j}(k)$  to the corresponding polling site  $S_j$ .

2. Poll and collect support counts: If  $S_i$  is a polling site,  $S_i$  receives all  $L_{j,i}(k)$  sent to it from the other sites. For every candidate itemset  $X$  received,  $S_i$  finds the list of originating sites from which  $X$  is being sent.  $S_i$  then broadcasts the polling requests to the other sites not on the list to collect the support counts.

3. Compute gl-large itemsets:  $S_i$  receives the support counts from the other sites, computes the global support counts for its candidates, and finds the gl-large itemsets. Eventually,  $S_i$  broadcasts the gl-large itemsets together with their global support counts to all the sites.

3. Remote site: return support counts to polling site .When  $S_i$  receives polling requests from the other sites, it acts as a remote site. For each candidate set  $Y$  it receives from a polling site, it retrieves  $Y$ :supi from the hash tree  $T_i(k)$  and returns it to the polling site.

4. Polling site: receive support counts and find large itemsets . As a polling site,  $S_i$  receives the local support counts for the candidate sets in  $L_{P_i}(k)$ . Following that, it computes the global support counts of all these candidate sets and find out the globally large itemsets among them. These globally large  $k$ -itemsets are stored in the set  $G_i(k)$ . Finally,  $S_i$  broadcasts theset  $G_i(k)$  to all the other sites.

5.Home site: receive large itemsets . As a home" site,  $S_i$  receives the sets of globally large  $k$ -itemsets  $G_i(k)$  from all the polling sites. By taking the union of  $G_i(k)$ , ( $i = 1; : : n$ ),  $S_i$  finds out the set  $L_k$  of all the size- $k$  large itemsets. Further,  $S_i$  finds out from  $L_k$  the set  $GL_i(k)$  of gl-large itemsets for each site by using the site list in  $X$ :large sites. The sets  $GL_i(k)$  will be used for candidate set generation at the next iteration.

## Apriori Algorithm: Basic Apriori Algorithm

.Apriori algorithm generate the candidate itemsets to be counted in the pass by using only the itemsets found large in the previous pass-without the transaction in the database. Apriori beats AIS and SETM by more than an order of magnitude for datasets.

The key idea of Apriori algorithm lies in the “*downward-closed*” property of support which means if an itemset has minimum support, then all its subsets also have minimum support is called frequent itemset having k item can be generated by joining frequent itemsets having k-1 items, and deleting those that contain any subset that is not frequent.

Apriori is an influential algorithm for mining frequent itemset for Boolean association rules. The name of the algorithm is based on the fact that the algorithm uses *prior knowledge* of frequent itemset properties, as we shall see below. Apriori employs an interactive approach known as a *level-wise* search, here k-itemset are used to explore(k+1)-itemset. Starting by finding all frequent 1-itemsets(items with 1 item) denoted as  $L_1$ , we then consider 2-itemsets say  $L_2$ , and so forth. The finding of each  $L_k$  requires one full scan of the database. So during each iteration only candidates found to be frequent in the previous iteration are used to generate a new candidate set during the next iteration. The algorithm terminates when there are no frequent k-itemsets.

To improve the efficiency of the level-wise generation of frequent in itemsets, an important property called the *Apriori property*, presented below, is used to reduce the search space. In order to use the *Apriori property*, all nonempty subsets of a frequent itemset must also be frequent. This property is based on the following observation. By definition ,if an itemset I does not satisfy the minimum support threshold , $min\_sup$ , then I is not frequent, that is,  $P(I) < min\_sup$ . If an item A is added to the itemset

I, then resulting itemset  $I \cup A$  cannot occur more frequently than I. Therefore,  $I \cup A$  is not frequent either, that is,  $P(I \cup A) < \text{min\_sup}$ .

This property belongs to a special category of properties called *anti-monotone* in the sense that if a set cannot pass a test, all of its subsets will fail the same test as well. It is called anti-monotone because the property is monotonic in the context of failing a test.

Notation is given below

k-itemset	An itemset having k items
$L_k$	Set of frequent k-itemset(those with minimum support)
$C_k$	Set of candidate k-itemset(potentially frequent itemset)

**Algorithm: Apriori.** Find frequent itemsets using an interactive level-wise approach base on candidate generation

**Input:** Database D, of transaction; minimum support threshold, min\_sup.

**Output:** L, frequent itemsets in D.

**Method:**

$L_1 = \text{find\_frequent\_1-itemset}(D);$

**for**( $k=2; L_{k-1} \neq \emptyset; k++$ )

{

```

Ck=apriori_gen(Lk-1,min_sup);
for each transaction t∈D
{
// scan D for counts
Ct=subset(Ck,t);// get the subset of t that are candidates
for each candidate c∈Ct
    c.count++;
}
Lk={c∈ Ck | c.count ≥min_sup}
}
return L = ∪k Lk;

```

**procedure apriori\_gen**(L<sub>k-1</sub>:frequent(k-1)- itemsets; *min\_sup*: minimum support threshold)

```

for each itemsets l1∈ Lk-1
    for each itemset l2∈ Lk-1
    {
        if(l1[1]=l1[1] ∧ l1[2]=l1[2] ∧.....∧ l1[k-2]= l1[k-2] ∧ l1 [k-1]=
l1 [k-1]) then
            c= l1 ∪ l2; // join step: generate candidates
            if has_infrequent_subset(c, Lk-1)then
                delete c; //prune step: remove unfruitful
candidate
            else add c to Ck;
    }
return Ck;

```

**procedure has\_infrequent\_subset**(c:candidate k-itemset; L<sub>k-1</sub>:frequent(k-1)-itemset);  
//has prior knowledge

**for each** (k-1)-subset s of c



```

    if  $s \in L_{k-1}$  then
        return TRUE;
return FALSE;

```

**Apriori-gen** function takes as argument  $L_{k-1}$  and returns a superset of the set of all frequent k-itemset. It consists join step and prune step.

### The Join step

To find  $L_k$ , a set of candidate k-itemset is generated by joining  $L_{k-1}$  with itself. This set of candidates is denoted  $C_k$ . Let  $l_1$  and  $l_2$  be itemsets in  $L_{k-1}$ . The notation  $l_i[j]$  refers to the jth item in  $l_i$  (e.g.,  $l_1[k-2]$  refers to the second to the last item in  $l_1$ ). By convention, Apriori assumes that items within a transaction or itemset are sorted in lexicographic order. The join,  $L_{k-1} \bowtie L_{k-1}$ , is performed, where members of  $L_{k-1}$  are joinable if their first(k-2) items are in common. That is, member  $l_1$  and  $l_2$  of  $L_{k-1}$  are joined if  $(l_1[1]=l_2[1]) \wedge (l_1[2]=l_2[2]) \wedge \dots \wedge (l_1[k-2]=l_2[k-2]) \wedge (l_1[k-1]=l_2[k-1])$ . The condition simply ensures that no duplicated are generated. The resulting itemset formed by joining  $l_1$  and  $l_2$  is  $l_1[1] l_1[2] \dots l_1[k-2] l_1[k-1]$

### The Prune step

$C_k$  is a superset of  $L_k$ , that is its member may or may not be frequent but all of the frequent k itemset are included in  $C_k$ . A scan of the database to determine the count of each candidate in  $C_k$  would result in the determination of  $L_k$ . To reduce the size of  $C_k$ , the Apriori properties is used as follows. Any k-1 itemset that is not frequent cannot be a subset of frequent k itemset. Hence if any k-1 subset of candidate k itemset is not in  $L_{k-1}$ . Then the candidate cannot be frequent either and so can be removed from  $C_k$ . This subset testing can be done by maintaining a hash tree of a frequent patterns.

## The Apriori Algorithm — Example

Database D

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

$C_1$

itemset	sup.
{1}	2
{2}	3
{3}	3
{4}	1
{5}	3

$L_1$

itemset	sup.
{1}	2
{2}	3
{3}	3
{5}	3

Scan D →

→

$C_2$

itemset	sup.
{1 2}	1
{1 3}	2
{1 5}	1
{2 3}	2
{2 5}	3
{3 5}	2

$C$

itemset
{1 2}
{1 3}
{1 5}
{2 3}
{2 5}
{3 5}

Scan D →

←

$L_2$

itemset	sup.
{1 3}	2
{2 3}	2
{2 5}	3
{3 5}	2

$C_3$

{2 3 5}
{1 2 3}
{1 3 5}

Scan D →

itemset	sup.
{2 3 5}	2





# Results

---

MS JAVA



Auto



```
C:\jdk1.2.2\bin\server2>java Mainmenu  
Server started
```

```
Algorithm apriori starting now.....
```

```
Input configuration: 6 items, 7 transactions,C1(1-candidate-itemset): [1, 2, 3,  
4, 5, 6]
```

```
Now reading transactions, increment counters of itemset
```

```
Traverse 1-candidate hashtree ...
```

```
In main
```

```
6 5
```

```
5 3
```

```
4 4
```

```
3 2
```

```
2 5
```

```
1 3
```

```
Candidate sets prepared and has been sent to Others
```

```
Algorithm apriori starting now.....
```

```
Input configuration: 6 items, 7 transactions,C2(2-candidate-itemset): []
```

```
Config File : C:\jdk1.2.2\bin\server2\config.txt
Transaction File : C:\jdk1.2.2\bin\server2\transa.txt
Server Started
Algorithm apriori starting now....
C1(1-candidate-itemset): [1, 2, 3, 4, 5, 6]
Now reading transactions, increment counters of itemset
Traverse 1-candidate hashtree ...
In main
1
2
3
4
5
6
Candidate sets prepared and has been sent to Others
Algorithm apriori starting now....
C2(2-candidate-itemset): []
```

Data mining Server



Apriori Process Help

- Set configfile
- Set transactionfile
- Run
- PrintReport
- Exit

```
MS JAVA
Auto
Input configuration: 5 items, 7 transactions,C2(2-candidate-itemset): []
F:\jdk1.4\bin\server1>java Mainmenu
Server started
Algorithm apriori starting now.....

Input configuration: 5 items, 7 transactions,C1(1-candidate-itemset): [1, 2, 3,
4, 5]
Now reading transactions, increment counters of itemset
Traverse 1-candidate hashtree ...
In main
5 3
4 4
3 2
2 5
1 3
Candidate sets prepared and has been sent to Others
Algorithm apriori starting now.....

Input configuration: 5 items, 7 transactions,C2(2-candidate-itemset): []
```

Data mining Server

Apriori Process Help

Set configfile
Set transactionfile
Run
PrintReport
Exit



```
Config File : F:\jdk1.4\bin\server1\config.txt
Transaction File : F:\jdk1.4\bin\server1\transa.txt
Server Started
Algorithm apriori starting now....
C1(1-candidate-itemset): [1, 2, 3, 4, 5]
Now reading transactions, increment counters of itemset
Traverse 1-candidate hashtree ...
In main
5 3
4 4
3 2
2 5
1 3
Candidate sets prepared and has been sent to Others
Algorithm apriori starting now....
C2(2-candidate-itemset): []
```



```
MS-JAVA
Auto
Both candidate1 and candidate2 are empty
F:\jdk1.4\bin\menu1>
F:\jdk1.4\bin\menu1>exec
F:\jdk1.4\bin\menu1>java Clientmenu 192.168.1.1 192.168.1.2
rmi://192.168.1.1/largeServer
rmi://192.168.1.2/largeServer1
after
after getting remote objects
k = 1
In apriori Test befor call CreatCandidate
In apriori Test
6 5
5 3
4 6
3 2
2 5
1 3
k = 2
In apriori Test befor call CreatCandidate
In apriori Test
Both candidate1 and candidate2 are empty
```

Data mining Client

Apriori Process Help

Set Minimum Suport

PrintReport

Exit

Minimum Support

Enter Minimum Support Value:

```
URL1 = rmi://192.168.1.1/largeServer
URL2 = rmi://192.168.1.2/largeServer1
after getting remote objects
K = 1
In apriori Test
Candidate11
1 3
5 3
4 4
6 3
2 5
Candidate12
5 6
4 4
3 3
2 3
1 3
K = 2
In apriori Test
Candidate21
```



# Conclusion

---

## 6. CONCLUSION

---

Here, we proposed and implemented an efficient and effective distributed algorithm DM for mining association rules. Some interesting properties between locally and globally large itemsets are observed, which leads to an effective technique for the reduction of candidate sets in the discovery of large itemsets. Two powerful pruning techniques, local and global pruning, are proposed. Furthermore, the optimization of the communications among the participating sites is performed in DM using the polling sites. The result shows the high performance of DM at mining association rules.

Recently, there have been interesting studies on the mining of generalized association rules, multiple level association rules, quantitative association rules , etc. Extensions of our method to the mining of these kinds of rules in a distributed or parallel system are interesting issues for future research. Also, parallel and distributed data mining of other kinds of rules, such as characteristic rules , classification rules, clustering , etc. is an important direction for future studies.



# References

---

## 7. REFERENCES

---

### 7.1 Reference book

7.1.1 Jiawei Han, Micheline Kamber. "*Data Mining: Concepts and Techniques*"; Harcourt India private limited, 2001.

7.1.2 Stefano Ceri, Giuseppe Pelagatti "*Distributed Databases principles and systems*"

### 7.2 Journals

7.2.1 R. Agrawal, T. Imielinski, A. Swami: "Database Mining: A Performance Perspective", IEEE Transactions on Knowledge and Data Engineering, Special issue on Learning and Discovery in Knowledge-Based Databases, Vol. 5, No. 6, December 1993, 914-925.

7.2.2 Rakesh Agrawal and Ramakrishnan Srikant. "*Fast algorithms for mining association rules in large databases*." In Proc. of the VLDB Conference, Santiago, Chile, September 1994.

### 7.3 Websites

7.3.1 IBM Corp., (1995) "*Data mining - an IBM overview*", <http://ibm.com>. IBM's view of data mining - explains the data mining techniques in some detail, June 25, 2001

7.3.2 *Analysis of Data Mining Algorithms* by Karuna Pande Joshi, [www.more.net/~karuna](http://www.more.net/~karuna). July 5, 2001

7.3.3 The Parallel Computer Centre, nor of The Queen's University of Belfast. "*What is data mining* ", [www.qub.ac.uk](http://www.qub.ac.uk) July 10, 2001



# Appendix

---

## CLIENT PROGRAM

```
import java.io.*;
import java.util.*;
import java.rmi.*;
import java.rmi.server.*;
import java.lang.*;
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

public class aprioriTest extends JDialog
{
    public Vector l,ul;
    public Hashtable c;
    public Hashtable cand1,cand2,cand3;
    public largeIntf lint1;
    public largeIntf1 lint2;

    RCanvas rc ;
    String s = "";

    int minsup = 2;

    public aprioriTest(String args[],int mins)
    {
        super();
        minsup = mins;
        setSize(500,500);
        setVisible(true);
        getContentPane().setLayout(new BorderLayout());

        rc = new RCanvas();

        JScrollPane js;
        js = new
        JScrollPane(rc,JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,JScrollPane.
        HORIZONTAL_SCROLLBAR_ALWAYS);
        getContentPane().add(js,BorderLayout.CENTER);

        //getContentPane().add(rc);
        try
        {
```



```

        DataInputStream ds = new DataInputStream(System.in);

        l = new Vector();
        ul = new Vector();

        cand1 =new Hashtable();
        cand2 =new Hashtable();

        cand1.put("k1",new Integer(0));
        cand2.put("k2",new Integer(1));

        String url1 = "rmi://" +args[0]+"/largeServer";
        String url2 = "rmi://" +args[1]+"/largeServer1";

        rc.printStr.addElement("URL1 = "+url1);
        rc.printStr.addElement("URL2 = "+url2);
        rc.repaint();

        System.out.println(url1);
        System.out.println(url2);

        lint1 = (largeIntf)Naming.lookup(url1);
        System.out.println("after");
        lint2 = (largeIntf1)Naming.lookup(url2);

        System.out.println("after getting remote objects");
        rc.printStr.addElement("after getting remote objects");
        rc.repaint();

    }
    catch(Exception e)
    {
        System.err.println("Exception occured in binding"+e);
        //System.exit(0);
    }
    /*    String url = "rmi://" +args[2]+"/largeServer3";
        lint3 =(largeIntf)Naming.lookup(url);*/

}

public boolean FindCompleteCand()
{
// Compute C from cand1,cand2,cand3;

```

```

Enumeration enumr1,enumr2;
String key1,key2;
Integer value1,value2;
Hashtable temp;

    if(cand1.isEmpty() && cand2.isEmpty())
        return false;

//if (cand1.isEmpty()&& cand1==null)
//return false;

    enumr1 = cand1.keys();

//    enumr3 = cand3.keys();
//    temp = new Hashtable();

    while(enumr1.hasMoreElements())
    {
        key1 = (String)enumr1.nextElement();
        value1 = (Integer)cand1.get(key1);

enumr2 = cand2.keys();

        while(enumr2.hasMoreElements())
        {
            key2 = (String)enumr2.nextElement();
            value2 = (Integer)cand2.get(key2);
//            System.out.println("k1 = "+key1+"k2 = " +key2);
            if(key1.equals(key2))
                {
//                    System.out.println("key1 = "+key1+"key2 = "+key2);
//
System.out.println("k"+key2+"v1 "+value1+"v2="+value2);
                    temp.put(key2,new
Integer(value1.intValue()+value2.intValue()));
                    break;
                }
        }
    }

enumr1 = cand1.keys();
while(enumr1.hasMoreElements())
{
    key1 = (String)enumr1.nextElement();
    value1 =(Integer)cand1.get(key1);
    if(!temp.containsKey(key1))
        temp.put(key1,value1);
}

```

```

    }
    enumer2 = cand2.keys();
    while(enumer2.hasMoreElements())
    {
        key2 = (String)enumer2.nextElement();
        value2 = (Integer)cand2.get(key2);
        if(!temp.containsKey(key2))
            temp.put(key2,value2);
    }

/*    enumer1 = temp.keys();
    enumer2 = cand3.keys();

    while(enumer1.hasMoreElements())
    {
        key1 = (String)enumer1.nextElement();
        value1 = temp.get(key1);

        while(enumer2.hasMoreElements())
        {
            key2 = (String)enumer2.nextElement();
            value2 = cand3.get(key2);

                if(enumer1.containsKey(key2))
                    {
                        c.put(key2,value1+value2);
                    }
                else
                    {
                        c.put(key1,value1);
                        c.put(key2,value2);
                    }
            }
        }
    }*/

```

```

c = temp;
// Display the whole candidate set
//    System.out.println("The whole candidate set");
rc.printStr.addElement("The whole candidate set");
rc.repaint();

```

```

Enumeration e = c.keys();

    while(e.hasMoreElements())
    {
        key1 = (String)e.nextElement();
    }

```

```

        value1 =(Integer) c.get(key1);
//      System.out.println(key1+" "+value1);
        rc.repaint();

    }

/*-----
// For temporary use
// -----
//-----*/

        return true;

}
public boolean GetCandidateSets(int k)
{
String key1;
Integer value1;

Enumeration e;

try
{
    System.out.println("In apriori Test befor call CreatCandidate");

//      if(!cand1.isEmpty())
cand1 = lint1.CreateCandidateSet(k,l,ul);
//      if(!cand2.isEmpty())
cand2 = lint2.CreateCandidateSet(k,l,ul);
    System.out.println("In apriori Test");
    rc.printStr.addElement("In apriori Test");
    rc.repaint();
// listing of candidate sets in cand2
//      System.out.println("Candidate"+k+"1");
rc.printStr.addElement("Candidate"+k+"1");
rc.repaint();
    if((cand1.isEmpty()|| cand1==new Hashtable(0)) && (cand2.isEmpty()||
cand2==new Hashtable(0)))
    {
        System.out.println("Both candidatel and candidate2 are empty");
rc.printStr.addElement("Both candidatel and candidate2 are empty");
    }
}

```

```

        rc.repaint();
        return false;
    }
    if(cand1.isEmpty()|| cand1==new Hashtable(0))
    {
        System.out.println("Empty candidate set1");
    }
    else
    {
        e = cand1.keys();

        while(e.hasMoreElements())
        {
            key1 = (String)e.nextElement();
            value1 =(Integer) cand1.get(key1);
// System.out.println(key1+" "+value1);
            s = key1+" "+value1;
            rc.printStr.addElement(s);
            rc.repaint();
        }
    }
}

```

```

// listing of candidate sets in cand2
//System.out.println("Candidate"+k+"2");
s = "Candidate"+k+"2";
rc.printStr.addElement(s);
rc.repaint();

    if(cand2.isEmpty()|| cand2==new Hashtable(0))
    {
        System.out.println("Empty candidate set2");
    }
    else
    {
        e = cand2.keys();

        while(e.hasMoreElements())
        {
            key1 = (String)e.nextElement();
            value1 =(Integer) cand2.get(key1);
            System.out.println(key1+" "+value1);
            s = key1+" "+value1;
            rc.printStr.addElement(s);
            rc.repaint();
        }
    }
}

```

```

}
catch(Exception ex)
{
System.out.println("Exception occurred in GetCandidateSet"+ex);
System.exit(0);
}
//      cand3 = lint3.CreateCandidateSet(k,l);*/
return true;
}
public void FindLargeItemSet()
{
// if candidate items in c exceeds the minimum support, and pruning test is
// completed,prepare l
Enumeration e = c.keys();
String str=new String();
l=new Vector();

//l.clear();
while(e.hasMoreElements())
{
    str = (String)e.nextElement();
    Integer it = (Integer) c.get(str);
    if((it.intValue()) >= minsup)
    {
        l.addElement(str);
        ul.addElement(str);
    }
}
System.out.println(" The large Union set is"+ul);
s = "The large item set is : "+l;
rc.printStr.addElement(s);
rc.repaint();

System.out.println(" The large Union set is"+ul);
s = "The large Union set is : "+ul;
rc.printStr.addElement(s);
rc.repaint();

}
public void printLargeItemset()
{
System.out.println("Printing large item set...");
for(int i=0;i<l.size();i++)
    System.out.println(ul.elementAt(i));
}
public void aprioriProcess()

```

```

{
int k=0;
if(lint1 != null && lint2 != null)
{
while(true)
{

    k++;
    System.out.println("k = "+k);
    s = "k = "+k;
    rc.printStr.addElement(s);
    rc.repaint();
    if(GetCandidateSets(k))
    {
        // Create the complete candidate set from cand1,cand2 and cand3
        if(FindCompleteCand())
        {
            rc.printStr.addElement("printing large & union item sets");
            rc.repaint();
            FindLargeItemSet();
        }
    }
    else
        break;

}
}
else
{
System.err.println("null in main");
//System.exit(0);
}

}
}

```

## SERVER PROGRAM

```
import java.io.*;
import java.rmi.*;
import java.rmi.server.*;
import java.lang.*;
import java.util.*;

public class largeImpl1 extends UnicastRemoteObject implements largeIntfl
{

    private final int HT=1; // state of tree node (hash table or
    private final int IL=2; // itemset list)
    int N; // total item #
    int M; // total transaction #

    RCanvas rc = new RCanvas();
    String tempstring = new String();

    // Vector largeitemset=new Vector();
    Vector candidate=new Vector();
    // int minsup;
    String fullitemset;
    String configfile="config.txt";
    String transafile="transa.txt";
    Hashtable cand1 = new Hashtable();
        Hashtable cand2=new Hashtable();

    public largeImpl1(String sconfig,String stransa) throws RemoteException
    {
        configfile=sconfig;
        transafile=stransa;
        tempstring = "Config File : "+configfile;
        rc.printStr.addElement(tempstring);
        rc.repaint();
        tempstring = "Transaction File : "+transafile;
        rc.printStr.addElement(tempstring);
        rc.repaint();
    }
    //-----
    // Class Name : candidateelement
    // Purpose : object that will be stored in Vector candidate
    // : include 2 item
    // : a hash tree and a candidate list
```



```
//-----  
class candidateelement {  
    hashtreenode hroot;  
    Vector candlist;  
}
```

```
//-----  
// Class Name : hashtreenode  
// Purpose   : node of hash tree  
//-----
```

```
class hashtreenode {  
    int nodeattr; // IL or HT  
    int depth;  
    Hashtable ht;  
    Vector itemsetlist;
```

```
    public void hashtreenode() {  
        nodeattr=HT;  
        ht=new Hashtable();  
        itemsetlist=new Vector();  
        depth=0;  
    }
```

```
    public void hashtreenode(int i) {  
        nodeattr=i;  
        ht=new Hashtable();  
        itemsetlist=new Vector();  
        depth=0;  
    }  
}
```

```
//-----  
// Class Name : itemsetnode  
// Purpose   : node of itemset (value,counter pair)  
//-----
```

```
class itemsetnode {  
    String itemset;  
    int counter;  
  
    public itemsetnode(String s1,int i1) {  
        itemset=new String(s1);  
        counter=i1;  
    }  
}
```

```

public itemsetnode() {
    itemset=new String();
    counter=0;
}

public String toString() {
    String tmp=new String();
    tmp=tmp.concat("<\"");
    tmp=tmp.concat(itemset);
    tmp=tmp.concat("\",");
    tmp=tmp.concat(Integer.toString(counter));
    tmp=tmp.concat(">");
    return tmp;
}
}

//-----
// Method Name: getconfig
// Purpose   : open file config.txt
//           : get the total number of items of transaction file
//           : and the total number of transactions
//           : and minsup
//-----
public void getconfig() throws IOException {

    FileInputStream file_in;
    DataInputStream data_in;
    BufferedReader br =new BufferedReader(new InputStreamReader(System.in));
    String oneline=new String();
    int i=0;

    try {
        file_in = new FileInputStream(configfile);
        data_in = new DataInputStream(file_in);

        oneline=data_in.readLine();
        N=Integer.valueOf(oneline).intValue();
        oneline=data_in.readLine();
        M=Integer.valueOf(oneline).intValue();
        oneline=data_in.readLine();
//    minsup=Integer.valueOf(oneline).intValue();
        System.out.print("\nInput configuration: "+N+" items, "+M+" transactions,");
//    System.out.println("minsup = "+minsup+"%");
//    System.out.println();
    } catch (IOException e) {
        System.out.println(e);
    }
}

```

```

}
}

//-----
// Method Name: getitemat
// Purpose   : get an item from an itemset
//           : get the total number of items of transaction file
// Parameter : int i : i-th item ; itemset : string itemset
// Return    : int : the item at i-th in the itemset
//-----
public int getitemat(int i,String itemset) {

    String str1=new String(itemset);
    StringTokenizer st=new StringTokenizer(itemset);
    int j;

    if (i > st.countTokens())
        System.out.println("eRRor! in getitemat, !!!!");

    for (j=1;j<=i;j++)
        str1=st.nextToken();

    return(Integer.valueOf(str1).intValue());
}
//-----
// Method Name: itesetsize
// Purpose   : get item number of an itemset
// Parameter : itemset : string itemset
// Return    : int : the number of item of the itemset
//-----
public int itemsetsize(String itemset) {
    StringTokenizer st=new StringTokenizer(itemset);
    return st.countTokens();
}

//-----
// Method Name: gensubset
// Purpose   : generate all subset given an itemset
// Parameter : itemset
// Return    : a string contains all subset delimited by ","
//           : e.g. "1 2,1 3,2 3" is subset of "1 2 3"
//-----
public String gensubset(String itemset) {

```

```

int len=itemsetSize(itemset);
int i,j;
String str1;
String str2=new String();
String str3=new String();

if (len==1)
    return null;
for (i=1;i<=len;i++) {
    StringTokenizer st=new StringTokenizer(itemset);
    str1=new String();
    for (j=1;j<i;j++) {
        str1=str1.concat(st.nextToken());
        str1=str1.concat(" ");
    }
    str2=st.nextToken();
    for (j=i+1;j<=len;j++) {
        str1=str1.concat(st.nextToken());
        str1=str1.concat(" ");
    }
    if (i!=1)
        str3=str3.concat(",");
    str3=str3.concat(str1.trim());
}

return str3;

} //end public String gensubset(String itemset)

```

```

//-----
// Method Name: createcandidate
// Purpose   : generate candidate n-itemset
// Parameter : int n : n-itemset
// Return    : Vector : candidate is stored in a Vector
//-----
public Vector createcandidate(int n,Vector l) {

    Vector tempcandlist=new Vector();
    Vector ln_1=new Vector();
    int i,j,length1;
    String cand1=new String();
    String cand2=new String();
    String newcand=new String();

```

```

//System.out.println("Generating "+n+"-candidate item set ....");

```

```

if(n==1)
    for (i=1;i<=N;i++)
        tempcandlist.addElement((new Integer(i)).toString());
else {
//    ln_1=(Vector)largeitemset.elementAt(n-2);

//System.out.println("the value of l "+l);
    ln_1 = l;
    length1=ln_1.size();
    for (i=0;i<length1;i++) {
        cand1=(String)ln_1.elementAt(i);
        for (j=i+1;j<length1;j++) {
            cand2=(String)ln_1.elementAt(j);
            newcand=new String();
            if (n==2) {
                newcand=cand1.concat(" ");
                newcand=newcand.concat(cand2);
                tempcandlist.addElement(newcand.trim());
            }
            else {
                int c,i1,i2;
                boolean same=true;

                for (c=1;c<=n-2;c++) {
                    i1=getitemat(c,cand1);
                    i2=getitemat(c,cand2);
                    if ( i1!=i2 ) {
                        same=false;
                        break;
                    }
                }
                else {
                    newcand=newcand.concat(" ");
                    newcand=newcand.concat(Integer.toString(i1));
                }
            }
        }
        if (same) {
            i1=getitemat(n-1,cand1);
            i2=getitemat(n-1,cand2);
            newcand=newcand.concat(" ");
            newcand=newcand.concat(Integer.toString(i1));
            newcand=newcand.concat(" ");
            newcand=newcand.concat(Integer.toString(i2));
            tempcandlist.addElement(newcand.trim());
        }
    } //end if n==2 else
} //end for j

```

```

    } //end for i
  } //end if n==1 else

  if (n<=2)
    return tempcandlist;

  Vector newcandlist=new Vector();
  for (int c=0; c<tempcandlist.size(); c++) {
    String c1=(String)tempcandlist.elementAt(c);
    String subset=gensubset(c1);
    StringTokenizer stsubset=new StringTokenizer(subset,",");
    boolean fake=false;
    while (stsubset.hasMoreTokens())
      if (!ln_1.contains(stsubset.nextToken())) {
        fake=true;
        break;
      }
    if (!fake)
      newcandlist.addElement(c1);
  }
  return newcandlist;

} //end public createcandidate(int n)

//-----
// Method Name: createcandidatehashtre
// Purpose   : generate candidate hash tree
// Parameter : int n : n-itemset
// Return    : hashtreenode : root of the hashtree
//-----
public hashtreenode createcandidatehashtree(int n) {

  int i,len1;
  hashtreenode htn=new hashtreenode();

  //System.out.println("Generating candidate "+n+"-itemset hashtree ....");
  if (n==1)
    htn.nodeattr=IL;
  else
    htn.nodeattr=HT;

  len1=((candidateelement)candidate.elementAt(n-1)).candlist.size();
  for (i=1;i<=len1;i++) {
    String cand1=new String();
    cand1=(String)((candidateelement)candidate.elementAt(n-
1)).candlist.elementAt(i-1);

```

```
    genhash(1,htn.cand1);
}
```

```
return htn;
```

```
} //end public createcandidatehashtree(int n)
```

```
//-----  
// Method Name: genhash  
// Purpose : called by createcandidatehashtree  
// : recursively generate hash tree node  
// Parameter : htnf is a hashtreenode (when other method call this method,it  
//is the root)  
// : cand : candidate itemset string  
// : int i : recursive depth,from i-th item, recursive  
// Return :  
//-----
```

```
public void genhash(int i, hashtreenode htnf, String cand) {
```

```
    int n=itemsetsize(cand);
```

```
    if (i==n) {
```

```
        htnf.nodeattr=IL;
```

```
        htnf.depth=n;
```

```
        itemsetnode isn=new itemsetnode(cand,0);
```

```
        if (htnf.itemsetlist==null)
```

```
            htnf.itemsetlist=new Vector();
```

```
        htnf.itemsetlist.addElement(isn);
```

```
    }
```

```
    else {
```

```
        if (htnf.ht==null)
```

```
            htnf.ht=new Hashtable(HT);
```

```
        if (htnf.ht.containsKey(Integer.toString(getitemat(i,cand)))) {
```

```
            htnf=(hashtreenode)htnf.ht.get(Integer.toString(getitemat(i,cand)));
```

```
            genhash(i+1,htnf,cand);
```

```
        }
```

```
    } else {
```

```
        hashtreenode htn=new hashtreenode();
```

```
        htnf.ht.put(Integer.toString(getitemat(i,cand)),htn);
```

```
        if (i==n-1) {
```

```
            htn.nodeattr=IL;
```

```
            //Vector isl=new Vector();
```

```
            //htn.itemsetlist=isl;
```

```
            genhash(i+1,htn,cand);
```

```
        }
```

```
    } else {
```

```

        htn.nodeattr=HT;
        //Hashtable ht=new Hashtable();
        //htn.ht=ht;
        genhash(i+1,htn,cand);
    }
}
}
} //end public void genhash(int i, hashtreenode htnf, String cand)

```

```

//-----
// Method Name: transatraverse
// Purpose   : read each transaction, traverse hashtree,
//             incrmnt appropriate itemset counter.
// Parameter : int n : n-itemset
// Return    :
//-----
public void transatraverse(int n,Hashtable cand1) {

    FileInputStream file_in;
    DataInputStream data_in;
    String oneline=new String();
    int i=0,j=0,len=0;
    String transa;
    hashtreenode htn=new hashtreenode();
    StringTokenizer st;
    String str0;
    int numRead=0;
//   Hashtable cand1 =new Hashtable();

    System.out.println("Traverse "+n+"-candidate hashtree ... ");
    tempstring = "Traverse "+n+"-candidate hashtree ... ";
    rc.printStr.addElement(tempstring);
    rc.repaint();

    htn=((candidateelement)candidate.elementAt(n-1)).htroot;
    try {
        file_in = new FileInputStream(transafile);
        data_in = new DataInputStream(file_in);

        while ( true ) {
//           if(cand1.isEmpty())
//               System.out.println("Empty cand1 in while");
//           cand1.clear();
            transa=new String();

```



```

oneline=data_in.readLine();
    numRead++;
if ((oneline==null)|| (numRead > M))
    break;
st=new StringTokenizer(oneline.trim());
    j=0;
while ((st.hasMoreTokens()) && j < N) {
    j++;
    str0=st.nextToken();
    i=Integer.valueOf(str0).intValue();
    if (i!=0) {
        transa=transa.concat(" ");
        transa=transa.concat(Integer.toString(j));
        len++;
    }
}
transa=transa.trim();
//    System.out.println("Before Transatrahash");

    transatrahash(0,htn,transa,cand1);
//    System.out.println("After Transatrahash");
    //printhashtree(htn,transa,0);
}
} catch (IOException e) {
    System.out.println("Exception in Transatraverse : "+e.getMessage());
}
//    System.out.println("Before returning");
}

```

```

//-----
// Method Name: transatrahash
// Purpose   : called by transatraverse
//           : recursively traverse hash tree
// Parameter : htnf is a hashtreenode (when other method call this method,it
//is the root)
//           : cand : candidate itemset string
//           : int i : recursive depth,from i-th item, recursive
// Return   :
//-----
public void transatrahash(int i,hashtreenode htnf,String transa,Hashtable cand1)
{
    String stris=new String();
    Vector itemsetlist=new Vector();

```

```

int j,lastpos,len,d;
itemsetnode tmpnode=new itemsetnode();
if (htnf.nodeattr==IL) {
    itemsetlist=(Vector)htnf.itemsetlist;

    len=itemsetlist.size();
    //System.out.println("Item List");
    for (j=0;j<len;j++) {
        tmpnode=(itemsetnode)itemsetlist.elementAt(j);
        d=getitemat(htnf.depth,tmpnode.itemset);
        lastpos=transa.indexOf(Integer.toString(d));
        if (lastpos!=-1)
            ((itemsetnode)(itemsetlist.elementAt(j))).counter++;
    }

    for(j=0;j<len;j++)
    {
        Integer value = new
Integer(((itemsetnode)itemsetlist.elementAt(j)).counter);
        String key = ((itemsetnode)itemsetlist.elementAt(j)).itemset;

        cand1.put(key,value);
    }
}
else //HT
    for (int b=i+1;b<=itemsetsize(transa);b++)
        if (htnf.ht.containsKey(Integer.toString(getitemat(b,transa))))

transatrahash(i,(hashtreenode)htnf.ht.get(Integer.toString(getitemat(b,transa)))
,transa,cand1);

} // public transatrahash(int ii,hashtreenode htnf,String transa)

```

```

//-----
// Method Name: CreateCandidateSet()
// Purpose   : main processing method
// Parameters :
// Return    :Hashtable
//-----
public Hashtable CreateCandidateSet(int k,Vector l,Vector ul) throws
RemoteException,IOException
{
    candidateelement cande;
    Hashtable candidate1 = new Hashtable();

```

```

System.out.println();
System.out.println("Algorithm apriori starting now.....");
System.out.println();
if(l.isEmpty())
{
rc.printStr.addElement("Algorithm apriori starting now.....");
rc.repaint();
}

getConfig();

fullitemset=new String();
fullitemset=fullitemset.concat("1");
for (int i=2;i<=N;i++) {
fullitemset=fullitemset.concat(" ");
fullitemset=fullitemset.concat((new Integer(i)).toString());
}

tempstring = "large Union item set";
rc.printStr.addElement(tempstring);
rc.repaint();

if(!ul.isEmpty())
{
tempstring ="Item ";
rc.printStr.addElement(tempstring);
rc.repaint();
tempstring ="----- ";
rc.printStr.addElement(tempstring);
rc.repaint();

for(int it=0;it<ul.size();it++)
{
tempstring =(String)ul.elementAt(it);
rc.printStr.addElement(tempstring);
rc.repaint();
}

}
else
{
rc.printStr.addElement("Empty");
rc.repaint();
}
}

```

```

cande=new candidateelement();
cande.candlist=createcandidate(k,l);

/* System.out.println("C"+k+"("+k+"-candidate-itemset): "+cande.candlist);

tempstring = "C"+k+"("+k+"-candidate-itemset): "+cande.candlist;

rc.printStr.addElement(tempstring);
rc.repaint();*/

if (cande.candlist.isEmpty())
    return (new Hashtable(0));

cande.htroot=null;
candidate.addElement(cande);

((candidateelement)candidate.elementAt(k-
1)).htroot=createcandidatehashtree(k);

System.out.println("Now reading transactions, increment counters of
itemset");

rc.printStr.addElement("Now reading transactions, increment counters of
itemset");
rc.repaint();

transatraverse(k,candl);
System.out.println("In main");
rc.printStr.addElement("In main");
rc.repaint();

if(candl.isEmpty())
{
System.out.println("Empty");
rc.printStr.addElement("Empty");
rc.repaint();
}

else
{
Enumeration e;
tempstring ="Item      "+ "Frequency";
rc.printStr.addElement(tempstring);
rc.repaint();
}

```

```

        tempstring = "-----";
        rc.printStr.addElement(tempstring);
        rc.repaint();

        rc.printStr.addElement("");
        rc.repaint();

        e = cand1.keys();
        while(e.hasMoreElements())
        {
            String str = (String)e.nextElement();

            Integer value1 = (Integer)cand1.get(str);

            candidate1.put(str,value1);
            System.out.println(str+" "+ value1);
            tempstring =str+" "+ value1;
            rc.printStr.addElement(tempstring);
            rc.repaint();

            rc.printStr.addElement("");
            rc.repaint();
        }
    }
    System.out.println("Candidate sets prepared and has been sent to Others");
    rc.printStr.addElement("Candidate sets prepared and has been sent to
Others");
    rc.repaint();

    cand1.clear();
    return candidate1;
}
}
import java.io.*;
import java.rmi.*;
import java.rmi.server.*;
import java.lang.*;
import java.util.*;

public class largeImpl1 extends UnicastRemoteObject implements largeIntfl
{

    private final int HT=1; // state of tree node (hash table or
private final int IL=2; // itemset list)

```

```

int N; // total item #
int M; // total transaction #

RCanvas rc = new RCanvas();
String tempstring = new String();

// Vector largeitemset=new Vector();
Vector candidate=new Vector();
// int minsup;
String fullitemset;
String configfile="config.txt";
String transafile="transa.txt";
Hashtable cand1 = new Hashtable();
    Hashtable cand2=new Hashtable();

public largeImpl1(String sconfig,String stransa) throws RemoteException
{
configfile=sconfig;
transafile=stransa;
tempstring = "Config File : "+configfile;
rc.printStr.addElement(tempstring);
rc.repaint();
tempstring = "Transaction File : "+transafile;
rc.printStr.addElement(tempstring);
rc.repaint();
}
//-----
// Class Name : candidateelement
// Purpose   : object that will be stored in Vector candidate
//           : include 2 item
//           : a hash tree and a candidate list
//-----
class candidateelement {
    hashtreenode htroot;
    Vector candlist;
}

//-----
// Class Name : hashtreenode
// Purpose   : node of hash tree
//-----
class hashtreenode {
    int nodeattr; // IL or HT
    int depth;
    Hashtable

```

```
Vector itemsetlist;
```

```
public void hashtreenode() {  
    nodeattr=HT;  
    ht=new Hashtable();  
    itemsetlist=new Vector();  
    depth=0;  
}
```

```
public void hashtreenode(int i) {  
    nodeattr=i;  
    ht=new Hashtable();  
    itemsetlist=new Vector();  
    depth=0;  
}  
}
```

```
//-----  
// Class Name : itemsetnode  
// Purpose   : node of itemset (value,counter pair)  
//-----
```

```
class itemsetnode {  
    String itemset;  
    int counter;  
  
    public itemsetnode(String s1,int i1) {  
        itemset=new String(s1);  
        counter=i1;  
    }  
  
    public itemsetnode() {  
        itemset=new String();  
        counter=0;  
    }  
  
    public String toString() {  
        String tmp=new String();  
        tmp=tmp.concat("<");  
        tmp=tmp.concat(itemset);  
        tmp=tmp.concat("\",");  
        tmp=tmp.concat(Integer.toString(counter));  
        tmp=tmp.concat(">");  
        return tmp;  
    }  
}
```

```

//-----
// Method Name: getconfig
// Purpose   : open file config.txt
//           : get the total number of items of transaction file
//           : and the total number of transactions
//           : and minsup
//-----
public void getconfig() throws IOException {

    FileInputStream file_in;
    DataInputStream data_in;
    BufferedReader br =new BufferedReader(new InputStreamReader(System.in));
    String oneline=new String();
    int i=0;

    try {
        file_in = new FileInputStream(configfile);
        data_in = new DataInputStream(file_in);

        oneline=data_in.readLine();
        N=Integer.valueOf(oneline).intValue();
        oneline=data_in.readLine();
        M=Integer.valueOf(oneline).intValue();
        oneline=data_in.readLine();
        // minsup=Integer.valueOf(oneline).intValue();
        System.out.print("\nInput configuration: "+N+" items, "+M+" transactions,");
        // System.out.println("minsup = "+minsup+"%");
        // System.out.println();
    } catch (IOException e) {
        System.out.println(e);
    }
}

```

```

//-----
// Method Name: getitemat
// Purpose   : get an item from an itemset
//           : get the total number of items of transaction file
// Parameter : int i : i-th item ; itemset : string itemset
// Return    : int : the item at i-th in the itemset
//-----
public int getitemat(int i,String itemset) {

```

```

    String str1=new String(itemset);

```



```

StringTokenizer st=new StringTokenizer(itemset);
int j;

if (i > st.countTokens())
    System.out.println("eRRor! in getitemat, !!!!");

for (j=1;j<=i;j++)
    str1=st.nextToken();

return(Integer.valueOf(str1).intValue());
}
//-----
// Method Name: itesetsize
// Purpose   : get item number of an itemset
// Parameter : itemset : string itemset
// Return    : int : the number of item of the itemset
//-----
public int itesetsize(String itemset) {
    StringTokenizer st=new StringTokenizer(itemset);
    return st.countTokens();
}

//-----
// Method Name: gensubset
// Purpose   : generate all subset given an itemset
// Parameter : itemset
// Return    : a string contains all subset delimited by ", "
//           : e.g. "1 2,1 3,2 3" is subset of "1 2 3"
//-----
public String gensubset(String itemset) {

    int len=itesetsize(itemset);
    int i,j;
    String str1;
    String str2=new String();
    String str3=new String();

    if (len==1)
        return null;
    for (i=1;i<=len;i++) {
        StringTokenizer st=new StringTokenizer(itemset);
        str1=new String();
        for (j=1;j<i;j++) {
            str1=str1.concat(st.nextToken());
            str1=str1.concat(" ");
        }
    }
}

```

```

str2=st.nextToken();
for (j=i+1;j<=len;j++) {
    str1=str1.concat(st.nextToken());
    str1=str1.concat(" ");
}
if (i!=1)
    str3=str3.concat(",");
str3=str3.concat(str1.trim());
}

```

```
return str3;
```

```
} //end public String gensubset(String itemset)
```

```

//-----
// Method Name: createcandidate
// Purpose   : generate candidate n-itemset
// Parameter : int n : n-itemset
// Return    : Vector : candidate is stored in a Vector
//-----
public Vector createcandidate(int n,Vector l) {

```

```

    Vector tempcandlist=new Vector();
    Vector ln_1=new Vector();
    int i,j,length1;
    String cand1=new String();
    String cand2=new String();
    String newcand=new String();

```

```

//System.out.println("Generating "+n+"-candidate item set ....");
if (n==1)
    for (i=1;i<=N;i++)
        tempcandlist.addElement((new Integer(i)).toString());
else {
//    ln_1=(Vector)largeitemset.elementAt(n-2);

```

```

//System.out.println("the value of l "+l);
    ln_1 = l;
    length1=ln_1.size();
    for (i=0;i<length1;i++) {
        cand1=(String)ln_1.elementAt(i);
        for (j=i+1;j<length1;j++) {
            cand2=(String)ln_1.elementAt(j);
            newcand=new String();
            if (n==2) {

```

```

    newcand=cand1.concat(" ");
    newcand=newcand.concat(cand2);
    tempcandlist.addElement(newcand.trim());
}
else {
    int c,i1,i2;
    boolean same=true;

    for (c=1;c<=n-2;c++) {
        i1=getitemat(c,cand1);
        i2=getitemat(c,cand2);
        if ( i1!=i2 ) {
            same=false;
            break;
        }
        else {
            newcand=newcand.concat(" ");
            newcand=newcand.concat(Integer.toString(i1));
        }
    }
    if (same) {
        i1=getitemat(n-1,cand1);
        i2=getitemat(n-1,cand2);
        newcand=newcand.concat(" ");
        newcand=newcand.concat(Integer.toString(i1));
        newcand=newcand.concat(" ");
        newcand=newcand.concat(Integer.toString(i2));
        tempcandlist.addElement(newcand.trim());
    }
} //end if n==2 else
} //end for j
} //end for i
} //end if n==1 else

if (n<=2)
    return tempcandlist;

Vector newcandlist=new Vector();
for (int c=0; c<tempcandlist.size(); c++) {
    String c1=(String)tempcandlist.elementAt(c);
    String subset=gensubset(c1);
    StringTokenizer stsubset=new StringTokenizer(subset,",");
    boolean fake=false;
    while (stsubset.hasMoreTokens())
        if (!ln_1.contains(stsubset.nextToken())) {
            fake=true;

```

```

        break;
    }
    if (!fake)
        newcandlist.addElement(c1);
    }
    return newcandlist;

} //end public createcandidate(int n)

//-----
// Method Name: createcandidatehashtre
// Purpose   : generate candidate hash tree
// Parameter : int n : n-itemset
// Return    : hashtreenode : root of the hashtree
//-----
public hashtreenode createcandidatehashtree(int n) {

    int i,len1;
    hashtreenode htn=new hashtreenode();

//System.out.println("Generating candidate "+n+"-itemset hashtree ....");
    if (n==1)
        htn.nodeattr=IL;
    else
        htn.nodeattr=HT;

    len1=((candidateelement)candidate.elementAt(n-1)).candlist.size();
    for (i=1;i<=len1;i++) {
        String cand1=new String();
        cand1=(String)((candidateelement)candidate.elementAt(n-
1)).candlist.elementAt(i-1);
        genhash(1,htn,cand1);
    }

    return htn;

} //end public createcandidatehashtree(int n)

//-----
// Method Name: genhash
// Purpose   : called by createcandidatehashtree
//           : recursively generate hash tree node
// Parameter : htnf is a hashtreenode (when other method call this method,it
//is the root)
//           : cand : candidate itemset string

```

```

//      : int i : recursive depth,from i-th item, recursive
// Return :
//-----
public void genhash(int i, hashtreenode htnf, String cand) {

    int n=itemsetsize(cand);
    if (i==n) {
        htnf.nodeattr=IL;
        htnf.depth=n;
        itemsetnode isn=new itemsetnode(cand,0);
        if (htnf.itemsetlist==null)
            htnf.itemsetlist=new Vector();
        htnf.itemsetlist.addElement(isn);
    }
    else {
        if (htnf.ht==null)
            htnf.ht=new Hashtable(HT);
        if (htnf.ht.containsKey(Integer.toString(getitemat(i,cand)))) {
            htnf=(hashtreenode)htnf.ht.get(Integer.toString(getitemat(i,cand)));
            genhash(i+1,htnf,cand);
        }
        else {
            hashtreenode htn=new hashtreenode();
            htnf.ht.put(Integer.toString(getitemat(i,cand)),htn);
            if (i==n-1) {
                htn.nodeattr=IL;
                //Vector isl=new Vector();
                //htn.itemsetlist=isl;
                genhash(i+1,htn,cand);
            }
            else {
                htn.nodeattr=HT;
                //Hashtable ht=new Hashtable();
                //htn.ht=ht;
                genhash(i+1,htn,cand);
            }
        }
    }
} //end public void genhash(int i, hashtreenode htnf, String cand)

```

```

//-----
// Method Name: transatraverse
// Purpose : read each transaction, traverse hashtree,
//          : incrmnt appropriate itemset counter.

```

```

// Parameter : int n : n-itemset
// Return   :
//-----
public void transatraverse(int n,Hashtable cand1) {

    FileInputStream file_in;
    DataInputStream data_in;
    String oneline=new String();
    int i=0,j=0,len=0;
    String transa;
    hashtreenode htn=new hashtreenode();
    StringTokenizer st;
    String str0;
    int numRead=0;
//  Hashtable cand1 =new Hashtable();

    System.out.println("Traverse "+n+"-candidate hashtree ... ");
    tempstring = "Traverse "+n+"-candidate hashtree ... ";
    rc.printStr.addElement(tempstring);
    rc.repaint();

    htn=((candidateelement)candidate.elementAt(n-1)).htroot;
    try {
        file_in = new FileInputStream(transafile);
        data_in = new DataInputStream(file_in);

        while ( true ) {
//          if(cand1.isEmpty())
//              System.out.println("Empty cand1 in while");
//          cand1.clear();
            transa=new String();
            oneline=data_in.readLine();
            numRead++;
            if ((oneline==null)|| (numRead > M))
                break;
            st=new StringTokenizer(oneline.trim());
            j=0;
            while ((st.hasMoreTokens()) && j < N) {
                j++;
                str0=st.nextToken();
                i=Integer.valueOf(str0).intValue();
                if (i!=0) {
                    transa=transa.concat(" ");
                    transa=transa.concat(Integer.toString(j));
                    len++;
                }
            }
        }
    }
}

```

```
cande.htroot=null;
candidate.addElement(cande);
```

```
((candidateelement)candidate.elementAt(k-
1)).htroot=createcandidatehashtree(k);
```

```
System.out.println("Now reading transactions, increment counters of
itemset");
```

```
rc.printStr.addElement("Now reading transactions, increment counters of
itemset");
rc.repaint();
```

```
transatraverse(k,cand1);
System.out.println("In main");
rc.printStr.addElement("In main");
rc.repaint();
```

```
if(cand1.isEmpty())
{
System.out.println("Empty");
rc.printStr.addElement("Empty");
rc.repaint();
}
```

```
else
{
Enumeration e;
tempstring="Item      "+ "Frequency";
rc.printStr.addElement(tempstring);
rc.repaint();
tempstring="-----      -----";
rc.printStr.addElement(tempstring);
rc.repaint();

rc.printStr.addElement("");
rc.repaint();
```

```
e = cand1.keys();
while(e.hasMoreElements())
{
String str = (String)e.nextElement();

Integer value1 = (Integer)cand1.get(str);
```

```

        for(j=0;j<len;j++)
        {
            Integer value = new
Integer(((itemsetnode)itemsetlist.elementAt(j)).counter);
            String key = ((itemsetnode)itemsetlist.elementAt(j)).itemset;

            candl.put(key,value);

        }
    }
    else //HT
        for (int b=i+1;b<=itemsetsize(transa);b++)
            if (htnf.ht.containsKey(Integer.toString(getitemat(b,transa))))
transatrahash(i,(hashtreenode)htnf.ht.get(Integer.toString(getitemat(b,transa)))
,transa,candl);

} // public transatrahash(int ii,hashtreenode htnf,String transa)

```

```

//-----
// Method Name: CreateCandidateSet()
// Purpose : main processing method
// Parameters :
// Return :Hashtable
//-----
public Hashtable CreateCandidateSet(int k,Vector l,Vector ul) throws
RemoteException,IOException
{
    candidateelement cande;
    Hashtable candidate1 = new Hashtable();

    System.out.println();
    System.out.println("Algorithm apriori starting now.....");
    System.out.println();
    if(l.isEmpty())
    {
        rc.printStr.addElement("Algorithm apriori starting now.....");
        rc.repaint();
    }

    getconfig();

```

```

fullitemset=new String();
fullitemset=fullitemset.concat("1");
for (int i=2;i<=N;i++) {

```



```

fullitemset=fullitemset.concat(" ");
fullitemset=fullitemset.concat((new Integer(i)).toString());
}

tempstring = "large Union item set";
rc.printStr.addElement(tempstring);
rc.repaint();

if(!ul.isEmpty())
{
    tempstring ="Item ";
    rc.printStr.addElement(tempstring);
    rc.repaint();
    tempstring ="----- ";
    rc.printStr.addElement(tempstring);
    rc.repaint();

    for(int it=0;it<ul.size();it++)
    {
        tempstring =(String)ul.elementAt(it);
        rc.printStr.addElement(tempstring);
        rc.repaint();
    }
}
else
{
    rc.printStr.addElement("Empty");
    rc.repaint();
}

cande=new candidateelement();
cande.candlist=createcandidate(k,l);

/* System.out.println("C"+k+"("+k+"-candidate-itemset): "+cande.candlist);

tempstring = "C"+k+"("+k+"-candidate-itemset): "+cande.candlist;

rc.printStr.addElement(tempstring);
rc.repaint();*/

if (cande.candlist.isEmpty())
    return (new Hashtable(0));

```

```
candidate1.put(str,value1);
System.out.println(str+"          "+ value1);
    tempstring =str+"          "+ value1;
    rc.printStr.addElement(tempstring);
    rc.repaint();

    rc.printStr.addElement("");
    rc.repaint();
}

}
System.out.println("Candidate sets prepared and has been sent to Others");
rc.printStr.addElement("Candidate sets prepared and has been sent to
Others");
rc.repaint();

cand1.clear();
return candidate1;
}
}
```