36733

A Study on Association Rule Mining Algorithms in Data Mining

A thesis submitted in partial fulfilment of the requirements
for the degree of Doctor of Philosophy

Anjan Das

Regn No: 029 of 1995

School of Science and Technology

Department of Computer Science and Information Technology

Tezpur University

April, 2006

Abstract

Different domains such as financial investment, health care, manufacturing and production, telecommunication network, scientific domains, etc. have collected huge amount of data due to advancements of database technologies. It has been observed that these databases contain various types of hidden patterns and knowledge, which can be used for different purposes. KDD(Knowledge Discovery in Databases) is the subject which deals with discovery of hidden patterns in large databases. KDD is a long process and data mining is just a part of whole KDD process. More formally, data mining is defined as non-trivial extraction of implicit, previously unknown and potentially useful information from large databases. Data mining is not a single subject. It is a result of confluence of many inter-disciplinary subjects such as statistics, machine learning, neural networks, information retrieval, database technology, etc. Data mining is not a simple task either; it is a very challenging task. Some of the main challenges of data mining are large data set and high dimension, user interaction and prior knowledge, over-fitting and assessing the statistical significance, understanding the patterns, non-standard and incomplete data, mixed media data, management of changing data and knowledge, integration, etc. As far as applications are concerned, data mining has wide area of applications - loan repayment prediction, crime detection, risk analysis, target marketing, banking, etc.

Association rule mining is one type of data mining techniques and also known as market-basket problem. Association rules were first discussed by Agarwal et al. in 1993 and find the influence of one set of items/attributes over another set of items/attributes. One example of an association rule may be "When people buy bread, they also buy butter 70% of the time". Association rules have wide range of applications. Some of the domains where association rules have been used successfully are business, engineering, medicine, telecommunications, etc. Some more applications of association rules are: market basket analysis, financial services, fraud detection, partial classifications, understanding customers' buying patterns, etc. The important keywords used in the context of association rules are: *itemset, support, candidate itemset, frequent / large itemset, confidence, minimum support, etc.* Itemset is a non-empty set of items/attributes. Support of an itemset X is the percentage of records /transactions or number of records/transactions in the database containing the itemset. Frequent/large itemset is the itemset with support greater than a minimum threshold value. Generally, association rule mining is a two step process - first step finds frequent itemsets and second step finds association rules among frequent itemsets. Between them, finding frequent itemsets is more challenging and interesting task. That is why, most of the research works concentrate only on the first step i.e. developing fast and efficient algorithms to find frequent itemsets in a large database. For these reasons, the thesis also concentrates mostly on frequent itemsets finding algorithms in large databases.

There exist algorithms to find frequent itemsets. However, most of the algorithms are not efficient and scalable. Among the existing algorithms, *Apriori* is one of the popular and robust algorithms to find frequent itemsets in static databases. However, the algorithm generates too many unnecessary candidate sets, which is responsible for exponential execution time of the algorithm. Another very recent popular algorithm is *Bit_AssocRule* algorithm, which uses the same technique as *Apriori* and bitmaps of the attributes to find support counts of the candidate itemsets. This algorithm also generates too many unnecessary candidate sets. The thesis has reported modified versions of these two algorithms (*Modified_Apriori* and *Modified_Bit_AssocRule*) which generate less number of unnecessary candidate sets in lesser time. These modified algorithms have used Boole's inequality to generate candidate sets with high probability to become frequent itemsets. Experimental results have shown that the modified algorithms generate much less number of candidate sets in comparison to their parent algorithms.

Another popular technique to find frequent itemsets from large databases is horizontal partitioning algorithm. The partitioning algorithm partitions the database horizontally and finds frequent itemsets in each partition. Then, these frequent itemsets are merged together to find the final frequent itemsets. However, it has been observed that this technique is not effective for larger dimensions. In addition to that, execution time increases exponentially with the increase of number of partitions. The thesis has reported one algorithm using vertical partitions. Experimental results have shown that this technique is more effective in databases with larger dimensions and execution time decreases with the number of partitions.

Nowadays, most of the databases are dynamic. Finding frequent itemsets in dynamic databases is a challenging task. There exist some algorithms to find frequent itemsets in dynamic databases. One of the popular and robust algorithms to find frequent itemsets in dynamic databases is *Borders* algorithm. The algorithm uses border sets to find frequent itemsets in updated database. However, the algorithm suffers from the drawback of having to scan the old database very frequently, which in turn increases the execution time of the algorithm. To solve this problem, the thesis has reported a modified version of the *Borders* algorithm (*Modified_Borders*), which uses two border sets instead of one border set. It has been observed from the experimental results that the modified algorithm does not scan the whole database frequently. Hence, execution time also significantly lesser than that of the original *Borders.*

So far, there has been a little work, to the best of our knowledge, to find frequent itemsets in distributed dynamic databases. Existing algorithms to find frequent itemsets in dynamic databases cannot be used directly in distributed environment. So, the thesis has reported a distributed algorithm (*Distributed_Borders*) to find frequent itemsets for distributed dynamic databases. The scalability experiments showed that execution time increases linearly with the increase in size of the databases and speed-up experiments showed that algorithm achieves sub-linear speedup.

KDD process and machine learning techniques take too much time when applied on databases with irrelevant features. So, relevant feature selection is an important task in machine learning techniques. There exist some algorithms to select relevant features. These algorithms use different criteria to decide whether a feature is relevant or not. Moreover, it has been observed that there has been very little work done to select relevant features using frequency (support) count of the features. The thesis has reported one algorithm, called *FFC* (Feature selection using Frequency Count) to select relevant features using frequency (support) count of the features. Experimental results showed that the algorithm is equally efficient with its counterparts such as *Branch & Bound, Relief, Focus*, etc. The main advantage of the algorithm is that it is very easy to implement in comparison to its counterparts.

View selection is an important technique in a data warehouse system. Here, the problem is to select an optimum views for materialization so that the query response time is minimized. The existing algorithms have used very complex techniques to select the views. As a result, these algorithms take too much time to select the views. The thesis has reported a very simple algorithm, called *DVMAFC* (Density-based View Materialization Algorithm using Frequency Count), to select the views. The algorithm selects the views based on the benefit of the views, access frequency of views, frequency/support count of sub-views, size of the views, frequency of updated (insert, edit, delete) operations on each view and number of rows affected by the update operations. The important concept used in the algorithm is the use of concept of density to form clusters of the views and then select the views from the clusters. Another concept is the use of supports of the sub-views to select the views because it has been observed that the support of the sub-views help select better views for materialization. The algorithm was compared with one of the popular algorithms called *PVMA* (Progressive View Materialization Algorithm). It was observed from the experimental results that *DVMAFC* selects better views than that of *PVMA*. The main advantage of *DVMAFC* over *PVMA* is the execution time complexity which is only *O(nlogn)* in comparison to $O(n^2)$ in case of *PVMA*, where n is the total number of views.

The thesis has concentrated mainly on the first step of the association mining problem, i.e. frequent itemset finding because, the second step i.e. rule generation task is quite trivial. It has analyzed important algorithms and reported improved versions of some of the important algorithms. The thesis has also reported two new algorithms - one for feature selection and the other for view selection. Association rule mining being a vast area of research, it is not possible to explore every aspect of it in a stipulated period of time. So, there are still ample scopes for future works. Some of the future works are as follows.

1. To extend the existing developments to enable to work over spatial, temporal and high dimensional data such as gene expression data, protein synthesis data, etc.

2. To explore the possibility of developing better and robust association mining algorithms using soft computing approach to discover more comprehensive and interesting patterns.

3. To develop better and robust dynamic rule mining algorithms over huge market-basket data as well as other huge data source such as quantitative, temporal and spatial data.

Declaration

I hereby declare that the thesis entitled *A Study on Association Rule Mining Algorithms in Data Mining* submitted to the Department of Computer Science and Information Technology. Tezpur University has been carried out by me and was not submitted to any other institution for awards of any other degree.

[illegible]

Date: April 5, 2006 Anjan Das

Place: Shillong

TEZPUR UNIVERSITY

This is to certify that the thesis entitled *A Study on Association Rule Mining Algorithms in Data Mining* submitted to Tezpur University in the Department of Computer Science and Information Technology under the School of Science and Technology in partial fulfillment for the award of the Degree of Doctor of Philosophy in *Computer Science* is a record of research work carried out by *Mr. Anjan Das* under my personal supervision and guidance.

All helps received by him from various sources have been duly acknowledged.

No part of this thesis has been reproduced elsewhere for award of any other degree.

[illegible]

Date: 25.4.2006

Place: Tezpur

Signature of Principal Supervisor
Designation: Professor
School: Science and Technology
Department: Computer Science and Information Technology

TEZPUR UNIVERSITY

This is to certify that the thesis entitled *A Study on Association Rule Mining Algorithms in Data Mining* submitted by *Mr. Anjan Das* to Tezpur University in the Department of Computer Science and Information Technology under the School of Science and Technology in partial fulfillment of the requirement for the award of the Degree of Doctor of Philosophy in *Computer Science* has been examined by us on _____________ and found to be satisfactory.

The Committee recommends for the award of the degree of Doctor of Philosophy.

Signature of:

Principal Supervisor External examiner

Associate Supervisor

Co-Supervisor

Date____________

List of Tables

List of Figures

List of Algorithms

Acknowledgment

First of all, I would like to express my profound sense of gratitude and sincere thanks to my supervisor Prof. Dhruba Kr. Bhattacharyya, Department of Computer Science and Information Technology, Tezpur University for his suggestion to undertake the study of this challenging topic and his able guidance in completing the work. His patience in this regard is praiseworthy.

I acknowledge with a deep sense of gratitude and appreciation the exchange of ideas with Prof. Dilip Kr. Saikia, Prof. Malayananda Dutta, Dr. Rajib Kr. Das, Dr. Smriti Kr. Sinha, Mr. Nityananda Sarma, Mr. Sarat Saharia and all other faculty members of the Department of Computer Science and Information Technology, Tezpur University.

I am also thankful to Mr. Bhogeshwar Borah of Tezpur University who equipped me with necessary information and study materials for this venture. I also thank him for giving me various suggestions from time to time.

My special thanks are also due to Dr. Shyamanta Moni Hazarika of Tezpur University who had been a constant source of encouragement and inspiration from the time of preparation of the thesis till its completion. I also thank him for going through the thesis carefully and his valuable comments.

I would also like to thank those who took pain to review all my papers that were submitted in various conferences or published in different journals.

I would also like to thank Dr. N. C. Bharali of St. Anthony's College, Shillong for his willingness to go through the thesis carefully and fine tuning it. I also thank him for his valuable suggestions in this regard.

I feel obliged to Fr. Joseph Nellanatt, Head, Dept. of Computer Science, St. Anthony's College for his constant encouragement. He deserves my sincere thanks for being flexible which enabled me to go to Tezpur University whenever situation demanded. I am also thankful to him for allowing me to do some experiments in the labs as well as use the resources in the department.

I am thankful to Fr. Stephen Mavely, former Principal of St. Anthony's College. Shillong and Fr. I. Warpakma, Principal, St. Anthony's College, Shillong who had given me necessary permission to undertake this work.

I would also like to acknowledge the help and constructive criticism offered by my friends and colleagues during the course of the study. In this regard, I am indebted to Miss Aiusha V Hujon, Lecturer, St. Anthony's College, Shillong for helping me draw certain difficult figures, Mr Basav Roy Choudhury, Lecturer, St. Anthony's College, Shillong for his generous help, suggestions and encouragement in difficult times and Mr. Shantu Saikia, Lecturer, St. Anthony's College, Shillong for helping me in all possible ways.

Finally, my thanks are particularly due to my parents, brothers, sisters and other family members for their patience, help and encouragement during the course of the study which enabled me to complete the work with full devotion.

[illegible]

Anjan Das

Contents

List of Symbols

I	:	Set of all items/attributes in a database.
D	:	A database.
t	:	A transaction in D.
$t.set\_of\_items$	:	Set of itemsets contained in t.
X, Y, Z	:	Non-empty set of items/attributes.
A, B, E, G, H, J, K, Q	:	Item/attribute.
$R, S, A_1, A_2, A_3, A_4, A_5$	:	Item/attribute.
X_k	:	A non-empty set of k items.
$Sup(X)$	:	Support of an itemset X.
$k - itemset$	:	An itemset with k items.
s	:	Support.
$minsup$	:	Minimum support.
$conf$	:	Confidence.
$minconf$	:	Minimum confidence.
$X \Rightarrow Y$	:	Association rule between X and Y.
$\rho(X \Rightarrow Y)$	:	Confidence of $X \Rightarrow Y$.
C	:	Set of all candidate itemsets.
L	:	Set of large itemsets.
a, b, l_1, l_2	:	Frequent/large itemsets.
c	:	Candidate itemset.
$c[i]$, $a[i]$, $b[i]$	:	i^{th} item in the itemset.
$c.count$	:	Support count of the candidate itemset c.
$C(t)$	:	Candidates contained in transaction t.
C_k	:	Set of candidate k-itemset.
L_k	:	Set of large $k - itemsets$.
TID	:	Transaction ID.
$t.TID$	:	TID of the transaction t.
C'_k	:	A set, where each member is of the form $< TID, \{X_k\} >$.
$P(A)$	:	Probability of the event/item/feature/view.
$P(A \cap B)$	:	Probability of occurring events A and B together.
$P[i]$	:	Probability of i^{th} item/attribute/feature.
PA	:	Array of probabilities.

β	:	Real number.
$\|T\|$	:	Average size of a transaction.
$\|ML\|$	:	Mean size of a potentially large itemset.
P	:	Set of all partitions in D.
p, q	:	A partition in D.
np	:	Number of partitions.
p_i	:	i^{th} partition.
c_k^p	:	A local candidate k-itemset in partition p.
l_k^p	:	A local large k-itemset in a partition p.
$tidlist$	:	Array of TIDs.
$c.tidlist$	:	$tidlist$ of c.
C_k^p	:	Set of local candidate k-itemset in a partition p.
L_k^p	:	Set of local large k-itemset in a partition p.
C_k^G	:	Set of global candidate k-itemset.
C^G	:	Set of all global candidate itemset.
L_k^G	:	Set of global large k-itemset.
L^i	:	The set of large itemsets in the partition i.
L^G	:	The set of global large itemsets.
S_i	:	The site i.
n_s	:	Number of sites.
β'	:	Positive real number($< minsup$).
T_{old}^i	:	Old database at the site i.
T_{new}^i	:	Incremental database at site i.
T_{old}	:	The old database($\bigcup T_{old}^i$).
T_{new}	:	The newly added records/transactions($\bigcup T_{new}^i$).
T_{del}	:	Records/transactions to be deleted.
T_{whole}^i	:	$T_{old}^i \cup T_{new}^i$.
T_{whole}	:	The whole database i.e. $T_{old} \cup T_{new} - T_{del}$. or $\bigcup T_{whole}^i (T_{old} \cup T_{new})$.
L_{old}	:	Set of frequent itemsets in T_{old}(with local support).
B_{old}	:	Set of border itemsets in T_{old}(with local support).
B'_{old}	:	First border set in T_{old}.
B''_{old}	:	Second border set in T_{old} .
B'_{whole}	:	First border set in T_{whole}.

B''_{whole}	:	Second border set in T_{whole} .
L^i_{whole}	:	Frequent itemsets in T_{whole} at the site i.
L_{whole}	:	Set of frequent itemsets in T_{whole}.
B_{whole}	:	Set of border itemsets in T_{whole}.
PB^i	:	Promoted border itemsets at the site i.
PB	:	Set of all promoted border set ($\bigcup PB^i$).
PB', PB''	:	First and second promoted border set.
F^i	:	Frequent itemsets in the updated database at the site i.
F	:	Frequent itemsets in the updated database($\bigcup F^i$).
$L_{whole}(i)$	:	$\{x \mid x \in L_{whole}, \|x\| = i\}$.
B^i_{whole}	:	Border itemsets in T_{whole} at the site i.
$PB(i)$	:	$\{x \mid x \in PB, \|x\| = i\}$.
$Sup(X)_y$	:	Support of the itemset X in the database y.
$Sup(X)^i_y$	:	Support of X at the site i for the database y.
F	:	Set of all features.
M	:	Number of features to be selected.
$NoSample$	:	Sample size.
$Threshold$	:	Lower limit of a feature's weight.
N	:	Total number of features.
W_j	:	Weight of j-th feature.
$Maxtries$	:	Number of iterations.
$ucon$	:	Upper level of inconsistency.
f	:	A feature.
f'	:	Non-occurrence of the feature f.
f^i	:	i^{th} feature.
C_l	:	Class attribute.
$incr$	:	The increment to the minimum support.
$minf$	:	Minimum number of selected features.
L'_1	:	Set of features/items/attribute whose non-occurrence is frequent.
S_f	:	Set of selected relevant features.
$f(x)$	:	A criterion function.
u, v, w, v_i	:	Views.

$u \rightarrow$ v	:	u is the parent of v in data cube lattice.
$R(u)$	:	Size(number of rows) of u.
V	:	Set of all the views.
S	:	Set of selected views.
NR	:	Set of views with negative profit.
T_{rba}	:	Time for random block access of the storage device.
bf	:	Blocking factor of the storage device.
$NMPV(v)$	:	Nearest Materialized Parent View of v.
$child(v)$	:	Child views of the view v in a lattice of views.
$benefit_k(v)$	:	Benefit of v at iteration k.
$profit(v)$	:	Profit of v.
$cost(v)$	:	Cost of the view v.
SI, SU, SD	:	Set of insert, delete and update operations respectively.
N_i, N_d, N_u	:	Number of rows affected by insert, delete and update operations respectively.
f_i, f_d, f_u	:	Frequencies of insert, delete and update operations respectively.
f_v	:	Frequency of v.
$N(v)$	:	Neighborhood of v.
$MinBen$	:	Minimum benefit.
Fv	:	Set of frequent views.
$Sup(v)$	:	Support of the view v.
Cl	:	A cluster.
$clid$	:	Cluster id.
$MaxD$	:	Maximum difference between two views.
$seeds$	:	List of possible core views.
P_{view}	:	View with maximum profit.

List of Abbreviations

AP	:	*Apriori* algorithm.
AT	:	*AprioriTid* algorithm.
AH	:	*AprioriHybrid* algorithm.
BA	:	*Bit_AssocRule* algorithm.
DVMAFC	:	Density-based View Materialization Algorithm using Frequency Count.
FFC	:	Feature Selection using Frequency Count.
HP	:	Horizontal Partition algorithm .
KDD	:	Knowledge Discovery in Databases.
MA	:	*Modified_Apriori* algorithm.
MBA	:	*Modified_Bit_AssocRule* algorithm.
PVMA	:	Progressive View Materialization Algorithm.
VP	:	Vertical Partition algorithm.

Chapter 1

Introduction

Huge amount of data have been collected through the advances of database technologies and data collection techniques. Some of the domains, where large volume of data are stored are Financial Investment, Health Care, Manufacturing and Production, Telecommunication Network, Scientific Domain, etc. These databases are full of hidden patterns and knowledge, which can be used for different purposes. The subject which deals with hidden patterns in a large database and finds knowledge from a large database is known as Knowledge Discovery in Database(KDD). Data Mining can be defined as extracting knowledge from huge amount of data. The Following subsections will clarify the concept of data mining more clearly.

1.1 What is Data Mining

In the simplest form, Data Mining can be defined as extraction of knowledge from huge amount of data. More formally, data mining can be defined as the non-trivial extraction of implicit, previously unknown and potentially useful information from database. So, it can be compared with gold mining, diamond mining, etc. Some other terms with similar meaning are *knowledge mining*, *knowledge extraction*, *pattern analysis*, *data dredging*, etc. Some people consider data mining as a part of the whole KDD (Knowledge Discovery in Database) (*Figure* 1.1 on page 3 [HK01]) process, which can be defined as the non-trivial process of identifying *valid*, *novel*, *potentially useful* and *ultimately understandable* patterns. Again, many people treat data mining as synonym for KDD.

As shown in the *Figure* 1.1 on the next page, KDD consists of the following steps.

1. Cleaning and Integration: Noise and inconsistent data are removed. Multiple data sources are combined.
2. Selection and Transformation : Required data are selected and transformed into forms appropriate for mining using different data mining techniques.
3. Data Mining : Different techniques, algorithms are used to extract knowledge from the data.
4. Evaluation and Presentation: Interesting patterns are found depending on some criteria. Patterns are represented using different types of GUI.

There are different architectures of a data mining system. However, the three-tier architecture (*Figure* 1.2 on page 4 [HK01]) is more popular. In this architecture, the major components are *Database and Data Warehouse, Database or Data warehouse server, Data mining engine, Knowledge base, Pattern evaluation* and *Graphical user interface.* A brief description of these components is given below.

- Database, data warehouse: This refers to set of databases, data warehouses, spreadsheets and other sources of data. Data cleaning and integration may be required.
- Database or data warehouse server: This is required to store and fetch relevant data.
- Knowledge base: This refers to knowledge repository, which is required to find interesting patterns. It may include *concept hierarchy, meta data, user beliefs, some threshold*, etc.

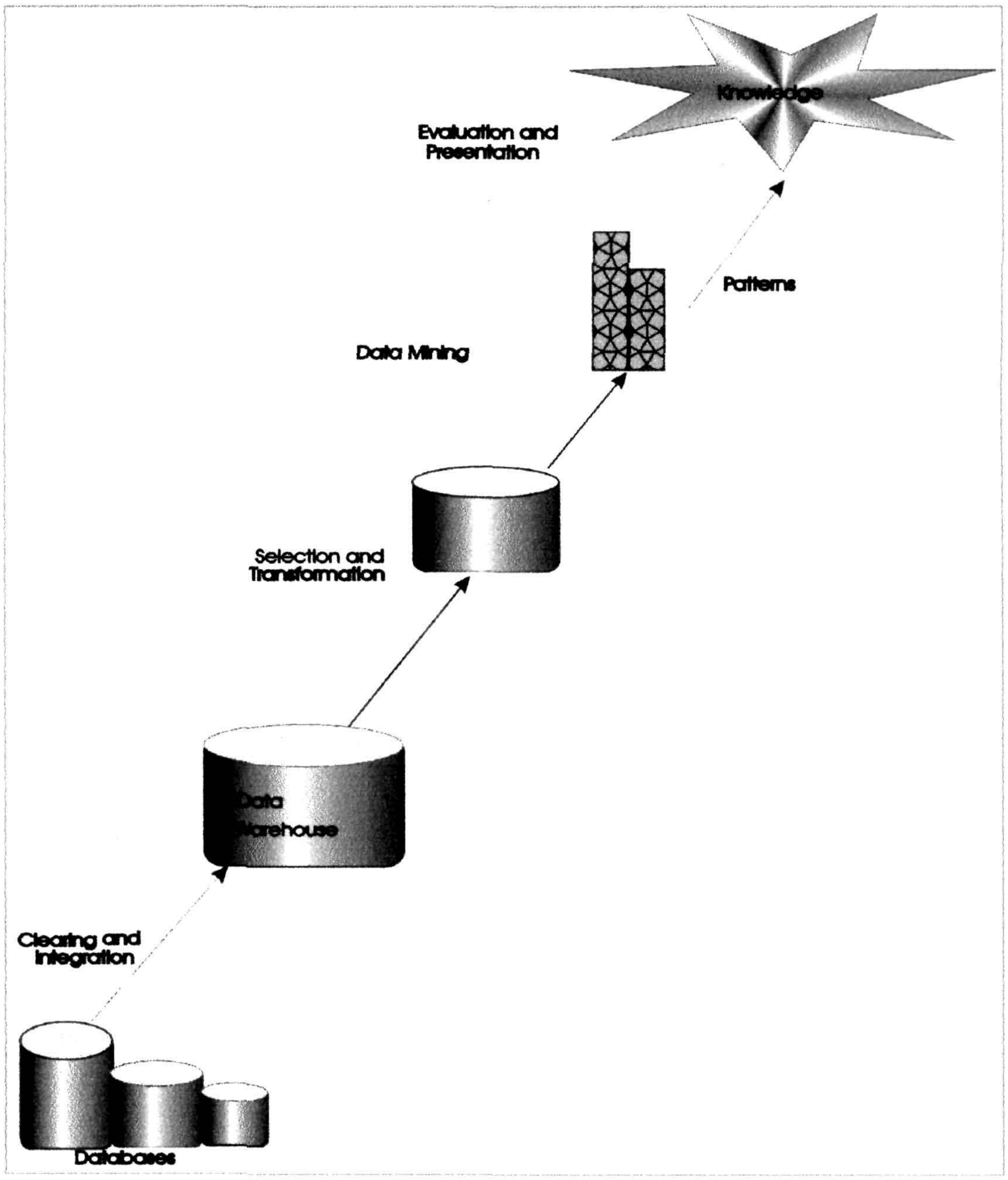


Figure 1.1: KDD Process

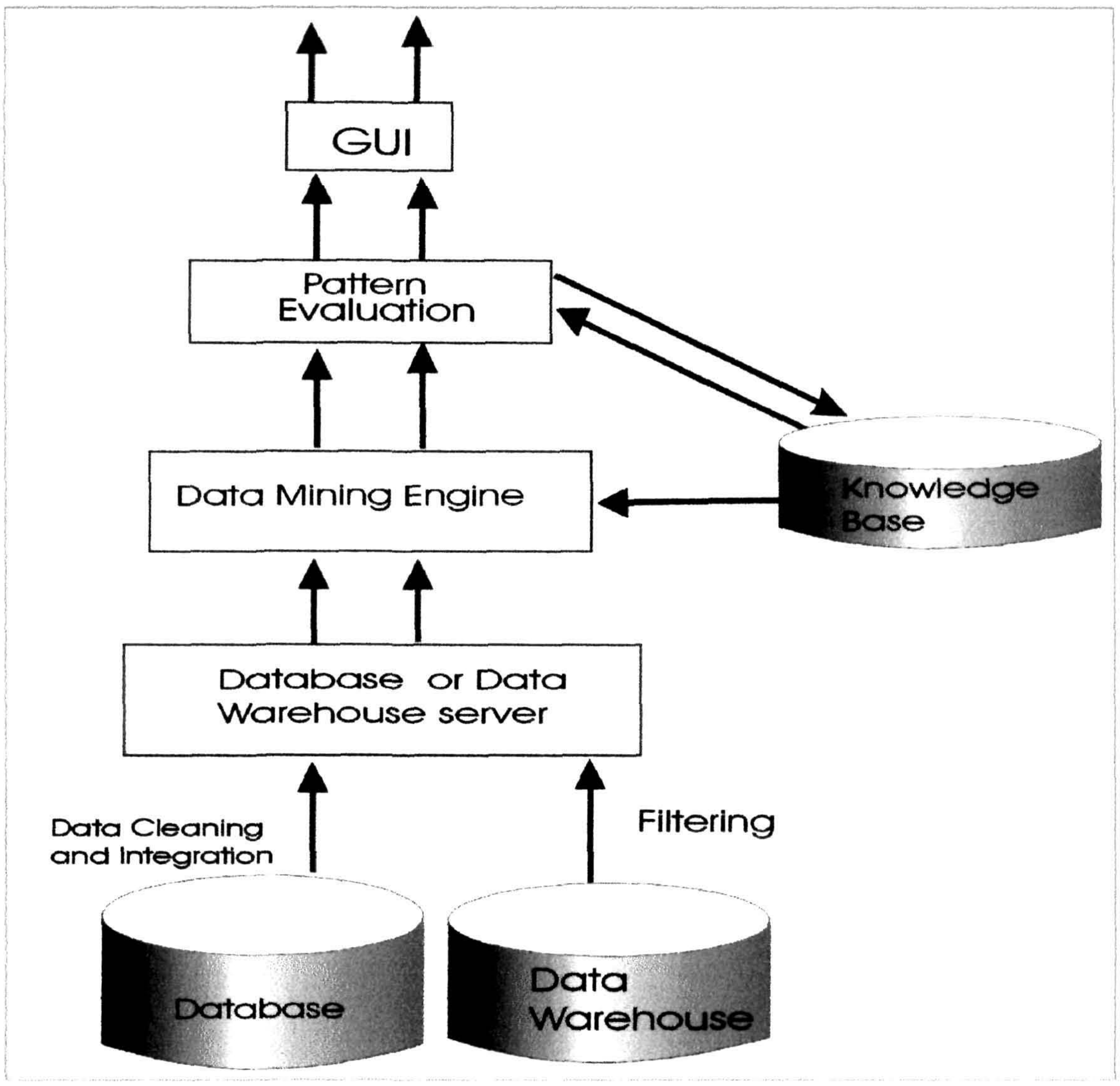


Figure 1.2: Three-tier Architecture of Data Mining

- Data mining engine: This is the main module which performs tasks such as association, classification, clustering, evaluation, etc.
- Pattern evaluation: It determines whether a pattern is interesting or not. To find the interestingness of a pattern, it interacts with data mining engine, knowledge base, etc.
- Graphical user interface: This module is responsible to interact with users. The major task of this module is to take the user's query and other parameters. Then it presents the results of the queries in some understandable formats using the available GUI tools.

1.1.1 Definitions

The main purpose of data mining is to find hidden patterns from large databases. However, data mining has been defined in many ways by different authors. Some of the definitions are given below [Puj01].

1. *Data mining or knowledge-discovery in databases, as it is also known, is the non-trivial extraction of implicit, previously unknown and potentially useful information from the data. This encompasses a number of technical approaches, such as clustering, data summarization, classification, finding dependency networks, analyzing changes and detecting anomalies.* By *non-trivial*, it means that information should not be easily retrievable. As for example, calculating *age* from *date of birth*, which is stored in a database, is not non-trivial, but finding average age of employees, who suffer from a particular disease and work in a particular department, may be non-trivial. Another term used in the definition is *implicit.* It means that information retrieved should not be stored in database explicitly. However, it could be derived from the existing data. Again, information or pattern should be previously unknown and unexpected. As for example "80% people buy bread and butter together" is not unknown. However, "2% people buy bread and spoon together" may be unexpected. Information should be useful to users. In other words, information should be presented in the user understandable format so that they can be used in decision support systems, etc.

2. *Data mining is the search for the relationships and global patterns that exist in large databases but are hidden among vast amount of data, such as the relationship between patient data and their medical diagnostics. This relationship represents valuable knowledge about database, and the objects in the database, if the database is faithful mirror of the real world registered by the database.* This definition gives importance on the relationships among the objects in a database. Suppose, there is a database which stores customers' data and sells data in a super market. Finding relationships between customers' age and items bought by them may be interesting. As for example, "customers in the age group of 10 to 20 years prefer food items like maggi, cake, etc." may be useful to the super market owner.

3. *Data mining refers to using a variety of techniques to identify nuggets of information or decision-making knowledge in the database and extracting these in such a way that they can be put to use in areas such as decision support, prediction, forecasting and estimation. The data is often voluminous, but it has low value and no direct use can be made of it. It is the hidden information in the data that is useful.* Huge volume of data is not useful by itself. Dàta mining techniques find value from this huge volume of data, which can be used by decision makers.

4. *Discovering relations that connect variables in a database is the subject of data mining. The data mining system self-learns from the previous history of the investigated system, formulating and testing hypothesis about rules which systems obey. When concise and valuable knowledge about the system of interest is discovered, it can and should be interpreted into some decision support system, which helps the manager to make wise and informed business decision.* Here, a data mining system has been considered as a learning system, which learns from the existing data. So, it can be compared with machine learning systems.

5. *Data mining is a process of discovering meaningful, new correlation patterns and trends by shifting through large amount of data stored in repositories, using pattern recognition techniques as well as statistical and mathematical techniques.* This definition says that data mining is meant to handle large amount of data, which makes it different from other data ana-

lyzing tools. While dealing with large amount of data, it also uses existing statistical and mathematical tools.

1.1.2 True Data Mining

A true data mining system should be able to handle large volume of data and uses advanced techniques to understand the data. So, it can be considered as the advanced stage of OLAP (OnLine Analytical Processing). It is often confused with OLAP. OLAP is generally involved with aggregate-style analytical processing. However, data mining uses advanced techniques to find the patterns in the data in different forms. There are some commercial systems which are used for information retrieval, answering queries, finding aggregate values, statistical analysis, etc. These systems are not true data mining systems and should not be confused with data mining systems.

1.2 Data Mining as a Multi-disciplinary Subject

Data mining is not a single subject. It is the result of confluence of many interdisciplinary subjects(*Figure* 1.3 on the next page [Puj01]). It uses techniques from various subjects such as machine learning, statistics, neural networks, information retrieval, spatial data analysis, database technology, etc. These subjects are established by themselves and have contributed a lot in developing different data mining algorithms and enhancing their performance.

Let us consider the subject of statistics. Statistics is one important subject from data mining point of view and a theory-rich method for data analysis. It provides theoretical foundations and generates results which is difficult to interpret. However, statistics is the foundation which data mining is based on. There exist statistical tools to find patterns from data, which can be understood by people with strong statistical background. Moreover, these tools deal with small amount of data.

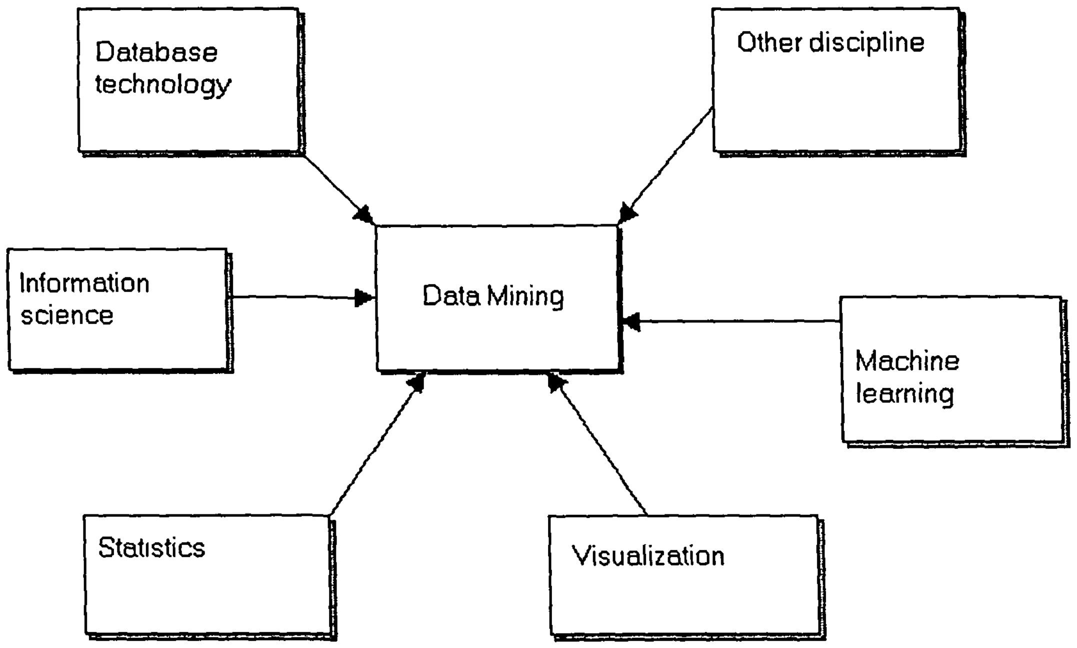


Figure 1.3: Data Mining as a Multi-Disciplinary Subject

Machine learning also has contributed in the development of data mining. Machine learning is the automation of learning process which includes learning from examples, reinforcement learning, learning with a teacher, etc. Again, machine learning can be of two types - supervised learning and unsupervised learning. In case of supervised learning, the system uses some training set to find description of each class. This description is used to place an unknown object in the appropriate class. On the other hand, unsupervised learning system does not use any training set and prior knowledge. It generates class descriptions from observations and discovery.

Data visualization also is an important subject in the context of data mining.

Data visualization helps analysts get deeper understanding of data. It gives analysts visual representations such as map, charts, etc. for a large volume of data. It also helps analysts concentrate on certain patterns and trends, which are represented by different colors.

Database technology also helps data mining systems in different ways. However, database systems and data mining systems are not same. In most of the cases, data mining systems use database systems as a simple repository of data. which data mining algorithms are based on. Database systems use some powerful and established techniques such as SQL, query optimization to retrieve data efficiently. These techniques are used to develop similar techniques for data mining systems. Some database systems integrate some data mining tools within themselves. In that case, data mining system is highly coupled with database systems and both the systems use same memory and disk space. So, it can be seen that database technologies have contributed a lot in development of data mining.

Other disciplines such as neural network, genetic algorithms, fuzzy sets, information science, etc. also have been used to develop efficient data mining algorithms. So, it can be concluded that data mining is not an isolated subject. It is the confluence of multi-disciplinary subjects.

1.3 Data Warehouse and Data Mining

Data warehouse is considered to be a pre-processing step for data mining. According to W H Inmon, "A data warehouse is *subject-oriented*, *integrated*, *time variant* and *non-volatile* collection of data in support of management's decision making process" [Inm96]. According to this definition, main characteristics of a data warehouse are

- It generally deals with broad subjects like customer, sales, etc. It does not deal with day-to-day activities.

- Data warehouse integrates many heterogeneous sources of data such as relational databases, spreadsheets, flat files, etc.

- Data in the data warehouse are attached with some time element because it stores the historical data.
- Data in data warehouse is permanent. It does not require recovery, concurrency control, etc. Data are just accessed for decision making.

Data warehouse provides a platform on which data mining techniques are based on. It also provides various OLAP tools which can be integrated with data mining techniques. So, a clear understanding of data warehouse is must to understand data mining techniques.

1.3.1 Data Cube, Cuboid and View

Multidimensional data model is the basic data structure on which OLAP and data warehouse tools are based on. This model views data in the form of *data cube.* A data cube allows data to be modeled and view in multiple dimensions and it consists of cuboids/views. In SQL terminology, cuboids/views are nothing but group-bys. As for an example, suppose, an organization keeps sales data with respect to *time(t), location(l)* and *branch(b).* Here, the data cube consists of eight possible group-bys: *tlb, tl, tb, bl, t,l,b* and *none.* Each individual group by is called *sub-cube* or cuboid or *view.*

DSS queries find the answers from the data cube. It may take long time due to huge size of data warehouse and the complexity of the query itself, which is not acceptable in DSS environment. The requirement of the query execution time is in the order of few seconds. Different techniques like query optimization and query evaluation techniques [CS94, GHQ95, YL95] are being used to deal with this problem. Different indexing techniques like bit-map index, join index are also used to reduce the query response time to a great extent. In data warehouse, the query response time largely depends on the efficient computation of data cube. However, creating data cube on the fly is very much time and space consuming.

One very useful technique used in data warehouse systems is partial materialization (pre-compute), which refers to materialization of some cuboids of a data cube, so that OLAP queries can be answered from these cuboids. However, the big question is "Which cuboids/views should be materialized ?" Partial materialization should *i*) select the cuboids to be materialized *ii*) use materialized views to answer the queries and *iii*) efficiently update materialized cuboids when data warehouse is updated. Selection of cuboids is not easy. Many factors are to be considered to select the cuboids. Among them, access frequencies of the queries, accessing cost of the cuboids, storage requirements, physical database design, etc. are important.

1.4 Challenges of Data Mining

There are many challenges which can be found to be bottlenecks in the development of data mining techniques. Among them, some of the main challenges are given below.

- *Large data set and high dimension*: Data mining algorithms have to deal with huge amount of data - both in terms of size and dimension. That is why, faster and efficient algorithms are required to handle these huge data. Some possible solutions are sampling, partitioning, parallel processing, etc.
- *User interaction and prior knowledge*: Data mining is an interactive and iterative process. Here, user's interactions at various stages are required. Domain knowledge may be used either in the form of high level specification of the model or at the more detailed level
- *Over-fitting and assessing the statistical significance*: Data sets used for data mining are collected from various sources resulting in the spurious data sets. Therefore, some kinds of regularization methods and sampling techniques are used to design the models for data mining.
- *Understanding the patterns*: Discoveries should be made understandable to the human. The frequently used techniques are rule structuring, natural language processing, visualization of data, etc.
- *Non-standard and incomplete data*: The data can be missing and/or noisy.
- *Mixed media data*: Learning from data is represented by a combination of various media - numeric, symbolic, images and text, etc.

- *Management of changing data and knowledge*: Data are often added, modified and deleted from database. The algorithms should take care of the changing patterns of the database.
- *Integration*: Data mining being a part of the entire decision making process. it is required to integrate with database and final decision making process.

Researchers have tried and are trying to develop techniques to overcome these challenges. However, there are still ample scopes for exploring best possible solutions.

1.5 Applications of Data Mining

Data mining has wide area of applications. Some of them are highlighted below.

- *Loan Prepayment Prediction*: The financial return of loans that a financial institution recovers, depends mainly on life-span of the loan. Data mining techniques help financial institutions predict number of loan repayments in a year as a function of interest rates, borrowers' characteristics, account data, etc. These information can be used to fix the parameters such as interest rate, fees, etc. to maximize profits.
- *Crime Detection*: Data mining techniques can be used to solve cases which do not have obvious leads. Suppose crime data are recorded in a database. Then clustering techniques can be used to cluster the similar crimes based on modus operandi and other parameters. If some suspects can be connected to some cases of a cluster, all other crimes of the cluster might have been done by the same suspects. This way, it will be possible to clear up old cases and determine patterns of behavior.
- *Risk Analysis*: Insurance companies can use data mining techniques to form clusters of customers depending on various risk factors so that when a new customer comes, he/she can be placed in one of the risk groups.
- *Target Marketing*: It will be useless to send the information for a new product to all the customers. Companies can use data mining techniques

to find potential customers, who will respond to the new product or the mailing campaign. Companies can use data mining techniques to find buying patterns of the customers to promote their products.

- *Banking*: Banks can use data mining techniques in various ways to increase their profits and manage their business properly. Banks can use data mining techniques to detect withdrawal patterns of the customers, patterns of credit card use, identifying loyal customers, determine credit card spending patterns by the customer groups, etc.

1.6 Different Types of Data Mining Techniques

Databases contain different types data. So, different types of patterns exist in these databases. Data mining systems try to find these patterns depending on the database types and the users' needs. Basically there are two types of techniques - *descriptive* and *predictive*. Descriptive techniques find patterns which describe or characterize the data and the predictive techniques find patterns which are used to make prediction. Some important data mining techniques are described below.

- *Classification*: It refers to the classification of a data item into one of several predefined categorical classes.
- *Regression*: It refers to the mapping of a data item to a real-valued prediction variable.
- *Clustering*: It refers to the mapping of a data item into one of several clusters, where clusters are the natural groupings of data items based on similarity metrics or probability density function.
- *Association rule mining*: It describes association relationship among the attributes.
- *Summarization*: It provides a compact description of a subset of data.
- *Dependency modeling*: It describes the dependencies among variables.

- *Sequence analysis*: It models sequential patterns like time series analysis. The goal is to model the states of the process generating the sequence or extract and report deviation of trends over time.

All of the above techniques are useful in deriving various kinds of interesting patterns from databases and they have got different applications in different domains. However, only association rule mining techniques will be studied in the thesis because of wide area of applications of association rules in various domains. Association rule mining technique and some of its applications have been discussed in this chapter.

1.6.1 All Patterns Are Not Interesting

Data mining systems generate huge number of patterns. Obviously, all patterns are not interesting. Interestingness of a pattern depends on many factors such as *understandability, validity* or *usefulness* of the pattern. Many a time users give some thresholds to measure the interestingness of patterns. As for example, users may supply different support and confidence values to find interesting association rules. A pattern may also be interesting, if it validates some hypothesis given by user. Similarly, a pattern may be interesting depending on user's belief - an unexpected pattern may be interesting to certain kind of users. As for example, the pattern "80% people buy bread and butter together" is not interesting because it is expected, but "1% people buy bread and spoon together" may be interesting. Two important terms are generally used in relation to interestingness - *completeness* and *optimization*. *Completeness* refers to developing of data mining algorithms which can find all patterns and *Optimization* refers to developing of data mining algorithms which can find only interesting patterns. The following section discusses association rule mining and its applications.

1.7 Association Rule Mining

Association rules were first discussed by Agarwal et al. in 1993. It is often referred to as *market-basket* problem. Association rules find influence of one set of items/attributes over another set of items/attributes in a database of transactions. One example may be "When people buy diapers, they also buy beer 60% of the time". Here, meaning of items and transaction depend on applications. Formally, it can be defined as a rule in the form $A_1, A_2, ...A_m \rightarrow B_1, B_2, ..., B_n$, where A_i's and B_j's are predicates or items. The rule can also be interpreted as conditional probability of occurring of B_j's in a transaction is very high, given that A_i's have already occurred in the transaction. Following are some examples of association rules.

1. Product(X, bread) $\rightarrow$ Product(X, butter). Here, items are the things bought by the customers and a transaction is the items bought together. The meaning of the rule is that customers generally buy bread and butter together.

2. age(X, 20-30), income(X, 10000 - 15000) $\rightarrow$ product(X, mobile). Here, items are values of the dimensions from a data warehouse with three dimensions - age, income and product. The rule says that customers with age between 20 and 30 years and income between Rs. 10,000 and 15,000 tend to buy a mobile phone.

Association rules are evaluated by the measures such as *support count*, *confidence*, *interestingness*, etc. So, it can be viewed as multi-objective problem. However, it is viewed as single-objective problem in most of the applications. As far as applications are concerned, association rules have got numerous applications in various domains. The important applications are highlighted in the following subsection.

1.7.1 Some Applications

Association rules originated from market basket data. However, it is also widely being used in other databases and different problem domains. Some of the problem domains, where association rules have been used successfully, are business, engineering, medicine, telecommunication, etc. Association rules are also used for other data mining tasks such as prediction, modeling, decision support, etc. Some of the important applications of association rules are given below.

- *Market Basket Analysis*: Nowadays, it is very essential for the retailers to know the buying preferences and buying patterns of the customers. If a retailer knows the buying patterns of the customers of a region, he can formulate strategies to attract the customers by giving appropriate gifts with different products. Buying patterns also help a retailer organize the products in the shelves so that customers find the related products together. Thus, association rules can help retailers get the buying patterns, preferences of the customers, which in turn will increase sales.

- *Financial Services*: Association rule mining plays a big role in financial sector. Financial experts use association rules to develop investment models, risk models in stock markets, etc. Associating rule mining systems have been used successfully in stock selection, claims processing by insurance companies, currency trading, etc.

- *Fraud Detection*: With the increase of the use of electronic money like debit cards, credit cards, etc., fraud detection has been one of the prime tasks of the crime branches. Association rule mining can find using patterns of cards by card holders, and crime branches can use these patterns to detect the frauds.

- *Partial Classification*: Conventional classifiers may not be effective in a database, where most of the values of the attributes are missing. Association rules can be used to solve these kind of problems. Association rules can be used to see if one set of attributes are related to another set of attributes. In that case, one set of attributes can be replaced by another set of attributes with most of values are present. As for example, if result of one complex and costly medical test can be predicted from a set of simple and cheap medical tests, doctors can prescribe the simple medical tests instead of complex medical tests.

1.8 Motivation

Huge amount of data are collected by regional sale system, telecommunication system, World Wide Web and other data collecting tools. These databases contain many useful patterns and knowledge, which can be used by decision makers and analysts to take appropriate decisions. There exists some data analyzing tools. These tools generally use statistical approaches and cannot deal with large data. Data mining techniques overcome these disadvantages because data mining techniques are meant to deal with large amount of data. There are different data mining techniques meant to find different patterns from large databases. All the techniques are useful in their respective domains. However, association rule mining is more interesting and challenging because of its wide areas of applications.

Generally, association rule mining is two step process [AMS+96]. First step finds frequent (or large) itemsets and second step finds association rules among the frequent itemsets. Between them, the first step is more challenging and interesting. That is why, most of the research works concentrate on the first step i.e. finding frequent itemsets from a large databases. For these reasons, mostly frequent itemsets finding techniques have been studied and analyzed in this thesis.

Finding frequent itemsets is even more complex in dynamic databases. An itemset which is frequent in the existing database, may not be frequent when some more records are added to the database, some records are deleted from the database or some records are updated. So, special algorithms are required to deal with such situations. The situation becomes more complex in distributed environment, where data is distributed in different geographic locations/sites. Here also special algorithms are required to deal with distributed environment. The thesis has reported a distributed algorithm to find frequent itemsets in distributed dynamic database.

Most of the databases contain lots of unnecessary or redundant features. These redundant features degrade the performance of machine learning algorithms. So. removing redundant features from a database is considered to be a major preprocessing step in many machine learning algorithms. The concept of frequent itemsets can help to remove redundant features to a great extent. This idea has been explored to remove redundant features from a database.

There exists some algorithms to deal with the above problems. However, the algorithms are not efficient and scalable. This has motivated us to enhance some existing algorithms and to develop some new algorithms to address the above issues. Main motivations are listed below.

- *Unnecessary candidates*: It has been found that algorithms generates too many unnecessary candidate sets, which is the main reason for exponential execution time. It has been tried to reduce the unnecessary candidates.

  Contribution: *Apriori* and *Bit_AssocRule* algorithms have been modified and probability has been used to reduce candidate sets and execution time.

- *Existing horizontal partition based algorithms are not effective*: Partition algorithm partitions a large database horizontally and then find the frequent itemsets. It has been observed that this technique is not effective in databases with larger dimensions and execution time increases with the increase in the number of partitions.

  Contribution: An algorithm has been developed by using vertical partition. It has been found that this technique is more effective for databases with larger dimensions and execution time decreases with the increase of number of partitions. Vertical partitioning technique also has been used with *FP-growth* [HPY00] algorithm to increase the performance of *FP-growth* for databases with large dimensions.

- *Feature selection is an important step in KDD and machine learning techniques*: KDD process and machine learning techniques take too much time when applied on databases with irrelevant features. So, relevant feature selection is one of the important tasks in machine learning techniques. There exists some algorithms to select relevant features and they use different criteria to decide whether a feature is relevant. However, the existing algorithms are not good enough in terms of both feature selection and execution time. In addition to that, it has been found that there was very little work done to select relevant features using frequency count (support count) of the features.

  Contributions: One algorithm has been developed to select relevant features by using frequency counts (support count) of the features.

- *Lack of efficient algorithm to find frequent itemsets in dynamic database*: Finding frequent itemsets in dynamic database is a challenging task. There exist some algorithms to find frequent itemsets in dynamic databases. However, they suffer from the drawback of having to scan the whole database repeatedly, which in turn, increases the execution time of the algorithms. So an economic solution is required to solve this problem.

Contribution: One algorithm has been proposed, which uses two levels of border sets. The main advantage of the algorithm is that it does not have to scan the whole database repeatedly.

- *Lack of algorithm to find frequent itemsets in distributed dynamic databases*: Nowadays, most of the databases are distributed. However, there has been a little work to find frequent itemsets in distributed dynamic databases. Again, existing algorithms to find frequent itemsets in dynamic databases cannot be used directly in distributed environment. So, some algorithms are required, which can find frequent itemsets from distributed dynamic databases.

  Contribution: A distributed algorithm has been developed to find frequent itemsets for distributed dynamic database.

- Selection of useful cuboids/views to be materialized in data warehouse systems in minimum possible time is very important. However, there are some algorithms to select views to be materialized. These algorithms have used very complex techniques to select the views. So, these algorithms take too much time to select the views.

  Contribution : A very simple algorithm has been proposed to select the views. The algorithm selects views based on the concept of density. The algorithm also uses the frequency(support) of the sub-views to calculate the benefits of the views.

1.9 Scope of The Thesis

The thesis embodies an exhaustive experimental study on some of the popular existing frequent itemsets finding algorithms and feature selection algorithms in the light of real and artificial datasets. It also includes some of the enhanced versions of the algorithms such as *Apriori*, *Bit_AssocRule*, *Borders*, etc. A detailed comparative study of those enhanced versions with their respective counterparts are also included to establish the superiority of the algorithms. Concentration has been given on the following areas.

- Partitioning is one of the important techniques used in data mining. A vertical partition based frequent itemset generation algorithm has been developed and a detailed comparative-study of both-these approaches are given.

- Feature selection is one of the important aspects in machine learning techniques. It has been shown how frequency count (support) of the attributes can be used to select relevant features in a database.

- Nowadays, most of the databases are distributed and dynamic. So, distributed algorithms also have been analyzed and studied in detail. One distributed algorithm has been proposed for distributed dynamic databases.

1.10 Organization of The Thesis

The thesis is organized as follows. *Chapter 2* reports the related works. *Chapter 3* reports some of the popular existing association mining algorithms. It also reports a modified and faster version of an existing robust association mining algorithm. *Chapter 4* discusses the partitioning approach in the frequent itemset generation. It also introduces a vertical partitioning approach for frequent itemset generation. A comparative study between both these approaches is also reported in this chapter. *Chapter 5* covers a crucial issue of association rule mining i.e. rule mining in dynamic databases. It reports experimental analysis of a popular dynamic association mining algorithm and also it reports an enhanced and distributed version of the algorithm. *Chapter 6* is an attempt to report some useful and popular algorithms for feature selection. It also reports a novel feature selection algorithm. A detailed comparative study is also reported in the chapter. *Chapter 7* is dedicated to a potential application of association mining techniques. In *Chapter 8*, concluding remarks and future works are given.

Chapter 2

Review

Today, association rule mining has been considered to be one of the important data mining techniques. It was introduced by Agarwal et al. in 1993. This is also referred to as *market-basket* problem because originally it was formulated for sales data. In simple term, association rule finds the measure of influence of one set of *items* on another set of *items*. The meaning of *items* varies from application to application. As for example, one association rule may be of the form "80% of the customers who buy bread also buy butter". Here, the rule finds the influence of bread on butter. Association rules have got numerous applications such as decision support, telecommunication alarm diagnosis, prediction, catalogue design, add-on sales, store layout, customer segmentation based on buying pattern, etc.

The problem of association rule mining can be divided into two subproblems.

1. Find all the frequent (or large) itemsets in a given database.

2. Find the association rules using the large itemsets found in the first step.

Out of these two steps, the first step is important and difficult one. That's why, most of the algorithms concentrate on finding the large itemsets from a large database in minimum possible time and using minimum resources. Once the large itemsets are known, finding association rules are straightforward. There are many algorithms to find frequent itemsets. All these algorithms target different types of database.

Concept of data mining can be traced back to induction of classification approach [BFO+83, FWD93, HCC92]. The closest work in the machine learning literature is the *KID3* algorithm presented in [Pia91]. Related work in the database literature is the work on inferring functional dependencies from data [Bit92, MR87]. Frequent itemset finding algorithms can be basically divided into two categories : for static databases and for dynamic databases. Next successive sections discuss some well known algorithms to find frequent itemsets in different domains.

2.1 Finding Frequent Itemsets in Static Databases

This section discusses some well known frequent itemsets finding algorithms for static databases. Static databases mean that databases do not grow. Its size, number of attributes remain static.

One of the earliest algorithms is *SETM* [HS93], which was proposed by M Houtsma and A Swami. The algorithm uses SQL to compute large itemsets. It generates candidates on-the-fly based on transactions and remembers the TIDs for generating transactions with the candidate itemsets.

Another popular and robust frequent itemset finding algorithm is *Apriori* [AMS+94], proposed by Agarwal et al. This algorithm is based on the fact that all the subsets of a large itemset are also large. The algorithm consists of multiple passes over the database. The first pass counts the number of occurrences of each item in the database to find the large 1-item sets. These 1-itemsets are used to generate the candidate 2-itemsets. Then, large 2-itemset are found by making a pass over the whole database. This process continues till there is at least one candidate itemset. This algorithm is simple and easy to implement. The algorithm is robust enough to find all the large itemsets in a database. The algorithm uses bottom-up and breadth-first approach. Another point to be observed is that number of database passes is equal to the length of the longest frequent itemset.

The main disadvantage of the algorithm is that it passes over the database several times which is responsible to increase the execution time for a large database. The same paper also proposed *AprioriTid* and *AprioriHybrid*. *AprioriTid* is a little improvement over the *Apriori*. This algorithm uses *TID*, a unique number used to represent a transaction, and a different data structure . So, each transaction is identified by a *TID*. The main advantage of this algorithm is that the database is used to count the support of the candidate set only once - for the 1-itemsets. 2-itemsets onward, a different data structure is used to count the support of the itemsets. The size of this data structure gets reduced with the increase of number of iterations . Thus, it takes much less time than that of *Apriori*. *AprioriHybrid* is just a combination of *Apriori* and *AprioriTid*. It uses *Apriori* in the initial passes and switches to *AprioriTid* for the remaining passes. Thus, it gets benefits from both the algorithms and can be found to be better than the other two in terms of execution time.

Another important algorithm is *Pincer-Search* [LK98]. This algorithm uses bi-directional approach i.e. top-down and bottom-up. It finds frequent itemsets in bottom-up manner and at the same time it maintains a list of maximal frequent itemsets. Maximum benefit is obtained when maximum frequent itemsets is found in the very early passes of the algorithm.

Park et al. proposed *DHP* (Direct Hashing and Pruning) [PCY95a]. It has two major features such as efficient generation of large itemsets using hashing technique and effective reduction on transaction database size. *DHP* is useful for generation of candidate large itemsets, particularly large 2-itemsets. However, this algorithm does not work properly for dense databases.

Some important algorithms can be found in [Zak00, ZPO+97, ZH99]. Among them, *CHARM* [ZH99] is an important algorithm. The algorithm introduced the concept of *closed* frequent itemsets, which is much smaller than the set of all frequent itemsets. With this concept, it is not necessary to generate all possible frequent itemsets and rules. The paper has shown that any rule is equivalent to some rules between *closed* frequent itemsets, resulting in reduction in redundant frequent itemsets and association rules.

Most of the algorithms mentioned above and other algorithms of its kind generate candidate sets and pass over the whole database to count the support of the candidate sets. Generating the candidate sets and repeated passing over the database is a time consuming and tedious task. Moreover, it takes lot of space in the memory to store the candidate sets. To overcome these problems there are some algorithms which find the frequent itemsets without generating candidate sets. One such algorithm is *FP-growth* algorithm [HPY00, Puj01]. The algorithm consists of two phases. In the first phase, it constructs the FP-tree with respect to a given minimum support and in the second phase, it finds the frequent itemsets from the FP-tree. The algorithm first makes one pass over the database to find the frequent 1-itemsets. Then it removes the non-frequent items from the transactions and rearrange the items in the transactions in the descending order of their frequency. Then the algorithm makes one pass over the whole database to construct the FP-tree. In order to find the frequency of different combinations, the algorithm computes the conditional FP-tree. Obviously the *FP-growth* algorithm has the advantage of not having to generate the candidate sets. The algorithm finds all the frequent itemsets and works very fast. However, this algorithm also has shortcomings [HPY00, Bor] such as i) it takes lot of time to construct the FP-tree for high dimensional dense large databases ii) its performance degrades with increase of minimum support.

There have been some attempts to develop frequent itemsets finding algorithms using bitmap techniques [BAG99, Gra94, JDO99, Joh98, MZ98, NG95]. The latest one being the *Bit_AssocRule* [HLL03]. This algorithm uses bitmaps of the items and applies the basic bit operations like AND, OR, etc. to find the support of the candidates. So, the algorithm does not require to scan the database more than once and works much faster than the other algorithms mentioned above.

Other approaches such as sequential patterns [AS95], generalized association rules [SA95], multilevel association rules [HF95], quantitative association [SA96] rules are worth mentioning.

Partitional, *parallel* and *distributed* methods also have been studied to find the frequent itemsets. In the partitioning approach [SON95], database is partitioned and the rule mining is carried out for each partition. Finally, frequent itemsets computed for each partition are merged to generate the frequent itemsets for the whole database. The main shortcoming of the algorithm is the choosing of number of partitions. Two aspects are taken into consideration while choosing the number of partitions- available buffer space and available memory. In this approach, the number of candidate sets and execution time are reduced to a great extent. Further, it provides scope for parallelization of the rule mining task.

2.1.1 Distributed and Parallel Algorithms

There have been some works on parallel and distributed algorithms. Main motivations behind parallel and distributed algorithms are as follows

- Mining databases containing huge amount of data needs more processing power.
- Most of the databases are distributed in nature.
- The algorithms also can be used in centralized databases by partitioning the database and placing the portions in different sites.

[AMS+94] proposed two parallel versions of *Apriori* called *Count Distribution(CD) and Data Distribution(DD). CD* algorithm scales linearly and speedup of the algorithm is also good with respect to the number of transactions. The drawback of the algorithm is that it does not parallelize building of the hash tree. *DD* algorithm partitions the candidate sets and assigns each partition to a processor. However, the algorithm takes maximum time in data movement among the processors and most of the time processors remain idle due to poor interaction scheme among the processors. Parallel version of *DHP* algorithm, called *PDM*, was proposed in [PCY95b]. The main disadvantage of the algorithm is that $O(n^2)$ messages are required for support count exchange for each candidate set. Cheung et al. [CHN+96a] proposed one efficient distributed algorithm called *Fast Distributed Mining* of association rules(*FDM*). The algorithm is advantageous due to following reasons.

- It uses some relationships between locally large and globally large itemsets to reduce the candidate sets and in turn number of messages to be passed is reduced.

- It uses local and global pruning techniques to prune away the candidates in the sites.
- It requires only O(n) messages for support count exchange, where n is the number of sites.

[CHN$^+$96a] introduced three versions of *FDM* i.e. *FDM-LP, FDM-LUP* and *FDM-LPP.* In [CNF$^+$96], distributed version is proposed called *DMA* (Distributed Mining Association Rules). This algorithm also needs O(n^2) messages for support count for each candidate set, where n is the number of sites.

In [HKK00], two new parallel algorithms called *Intelligent Data Distribution (IDD)* and *Hybrid Distribution (HD)* can be found. *IDD* is improvement over *DD.* It reduces communication time and processor idle time. *HD* combines the advantages of *CD* and *IDD.* It groups the processors and partitions the candidate sets to maintain load balance.

2.1.2 Multilevel Association Rules Mining

In many real-life scenario, data items exist in the hierarchy of concept level. So, it is difficult to find strong association rules among data items at low levels of abstraction due to the sparsity of data in multi-dimensional space. Association rules at high concept level generally represent a common pattern. As for example, "bread and butter are bought together" may not be an interesting pattern, but "honey, bread and butter are bought together" may be an interesting pattern. Therefore, data mining systems should provide capabilities to mine association rules at multiple levels of abstraction (concept hierarchies) and traverse easily among different abstraction spaces. Concept hierarchies may be specified by the users familiar with the data or may be specified implicitly in the data itself. As for example, there may not exist any association rule between IBM laptop computer and Philips b/w printer, but there may exist one association rule between IBM computer and Philips printer. Rules generated from association rule mining with concept hierarchies are called multiple-level or multi-level association rules. [HF95, SA95] have discussed some issues of multiple-level association rule mining.

Different approaches may be used for multilevel association rule mining. In general a top-down approach is used. To find frequent itemsets at each level, general algorithms like *Apriori* can be used. One main difficulty in multilevel association rule mining with reduced support is applying the search strategy thorough the concept hierarchy. Some of the widely used search strategies are *Level-by-level independent*, *Level-cross filtering by single item*, *Level-cross filtering by k-itemsets*, etc. As far as algorithms are concerned, some algorithms to find association rules in multi-level databases can be found in [HF95, SA95].

2.1.3 Multidimensional Association Rule Mining

Multidimensional association rule mining refers to the mining of rules involving more than one predicate or dimension. One rule "bread $\Rightarrow$ butter" can be written as "*buys*(X, bread) $\Rightarrow$ *buys*(X, butter)". Here, the rule consists of only one predicate *buys*. So, this is an example of single-dimensional or intra-dimensional association rule. In reality, the databases and warehouses store many other related information in addition to only transactional information. As for example, one database may store the sales transactions of a supermarket along with the customers' age, address, income,occupation, etc. So, in this case, it may be interesting to find the association rules containing more than one predicate/dimension such as *age*(X, "20...25") $\wedge$ *occupation* (X, researcher) $\Rightarrow$ *buys*(X, laptop). This association rule contains three predicates - *age, occupation* and *buys*. This kind of multidimensional association rule without any repetition of predicates is called *inter-dimensional* association rule, otherwise it is called *intra-dimensional* association rule.

Techniques for mining multidimensional association rules are categorized depending on the treatment of the quantitative attributes.

- The first category is called *multidimensional association rules using static discretization of quantitative attributes*, where a predefined concept hierarchy is used to replace the original numeric values of the quantitative attribute.

- In the second approach, quantitative attributes are discretized into *beans* based on the distribution of the data. These beans may be further merged during mining process. This process is dynamic and established so as to satisfy some mining criteria such as the maximizing the confidence of the rules mined. This method is also called quantitative association rules.

- The third approach discretize the quantitative attributes to capture the semantic meaning of the interval data. It considers the distance between two points. So, it is also called *distance-based* association rules.

Mining Multidimensional Association Rules Using Static Discretization of Quantitative Attributes

[KHC97] has given the detail account of the discretization of quantitative attributes and data cubes. In this technique, quantitative attributes are discretized prior to mining using predicate concept hierarchies and categorical attributes may also be generalized to the higher conceptual level. Classical algorithm like *Apriori* may be modified to find the frequent predicate sets instead of frequent itemsets. Other techniques such as sampling, hashing, partitioning may also be applied. Data cubes will be suitable for mining multidimensional association rule mining. Data cubes are the lattice of cuboids which are multidimensional. If the warehouse under study already contains some data cube, then it can be used to find the frequent predicates. Otherwise, some data cube will be required to be created.

Quantitative Association Rules Mining

This is the type of association rules in which the numeric attributes are dynamically discretized during mining process to satisfy some mining criteria. A quantitative association rule with n quantitative attributes in the antecedent is called n-dimensional quantitative association rule. As for example, the association rule $income(X,$ "10000...30000"$) \wedge age(X,$ "20...30"$) \Rightarrow buys(X,$ "high resolution monitor"$)$ is a 2-D quantitative association rules.

Reference [LCK98] has given an approach to find the quantitative association rules called ARCS (Association Rule Clustering System). This approach finds the association rules for two quantitative attributes - one in the antecedent and one categorical attribute in the consequent. In this approach, the pair of quantitative attributes are mapped onto a 2-D grid for tuples satisfying a given categorical attribute condition. Then the grid is searched for clusters of points from which the association rules are generated.

Srikant and Agarwal [SA96] proposed a non-grid based technique for mining quantitative association rules which uses a measure of partial completeness. Other techniques such as mining quantitative rules based on rectilinear regions was proposed by Fukuda et al. [FMM+96] and Yoda et al. [YFM+97].

Distance-Based Association Rules Mining

If the quantitative attributes are discretized with the previous two methods, it may not capture the semantics of intervals since they do not consider relative distance between the points or intervals. As for example, equidepth partition may find interval 20000..50000, which is quite wide. The distance based partitioning seems to be most intuitive, since it groups values that are close together within the same interval. So, the distance based intervals produce more meaningful discretization. Intervals for each quantitative attribute can be established by clustering the values for the attributes. Another advantage of the distance-based association rule is the that it gives the scope of closeness or approximation in the predicates. As for example, in the association rule "$is(X$, cosmetics) $\wedge$ $make(X$,foreign) $\Rightarrow$ $price(X$, 300)", the *price* predicate is fixed at 300. The distance-based association rules allows the scope to give the range of values instead of a fixed value such as this.

The algorithms for distance-based association rule mining can employ two phase technique. The first phase finds the intervals or clusters using some clustering algorithm and the second phase finds the distance-based association rule by searching for groups of clusters that occur frequently together. [MY97] has proposed an approach to find the distance-based association rules by employing the above two phase techniques.

2.1.4 Spatial Association Rule Mining

With the wide application of remote sensing technology and automatic data collection tools, huge amount of spatial data have been collected in the large spatial databases. In other words, a spatial database stores a large amount of space-related data, such as maps, pre-processed remote sensing or medical imaging data. The extraction of the knowledge discovery in the large spatial pose great challenges to the currently available spatial database technologies. Spatial databases have many features that distinguishes them from relational databases. They carry the topological or distance information, usually organized by sophisticated, multidimensional spatial indexing structures that are accessed by spatial data access method and often require spatial reasoning, geometric computation and spatial knowledge representation techniques. Spatial data mining refers to the extraction of implicit knowledge, spatial relations or other patterns not explicitly stored in the spatial databases [KAH96].

Spatial data mining can be categorized based on the kinds of rules to be discovered in spatial databases. A spatial characteristic rule is a general description of a set of spatial-related data. For example, the description of a general weather pattern in set of geographic regions is a spatial characteristic rule. A spatial discriminant rule is the general description of the contrasting or discriminating features of class of spatial-related data from other classes. For example, the comparison of weather patterns in two geographic regions is spatial discriminant rule. There have been some interesting studies in spatial characteristic rules and spatial discriminant rules [NH94].

Statistical spatial analysis tools have been used extensively for analyzing spatial data [FR94]. Statistical tools are good for numerical data, but statistical techniques usually require the assumptions regarding to statistical independence and of spatially distributed data. Such assumptions do not apply in the real world situation because spatial objects are often influenced by the neighboring objects/regions. Again, predicate rules cannot be described using standard methods of statistical spatial analysis. It requires a lot of domain and statistical knowledge. So, only the persons who are experts in statistics can handle it. These arguments suggest that statistical techniques alone can not be used for spatial data mining. Another major approach in data mining is to apply generalization techniques to spatial and non-spatial data to generalize detailed spatial data to certain level and study the general characteristics and data distribution at this level.

Although the concept of spatial association rule is same as that of association rules in a relational database, the definitions are required to be redefined to meet the requirement of spatial association rules. As for example, a spatial association rule may look like

is_a(A, large_town), *intersects*(A, B), adjacent_to(A, C) $\rightarrow$ is_a(B, motorway), is_a(C, sea). (30%,80%).

This rule states that "30% of large towns intersects a motorway and are adjacent to the sea". This rule also states that "If a large towns intersects a spatial object B and is adjacent to C then B is a motorway and C is a sea in 80% cases".

Mining spatial association rules is more complex task than mining transactional association rules. The degree of complexity are due to the implicit definition of association relations and the granularity of spatial objects. The spatial relations may be topological [Ege91] such as intersect; overlap; disjoint; distance such as close_to, far_way, etc. and direction such as left, right, etc. Therefore, complex data transformation processes are required to make spatial relations explicit.

Reference [KH95] has proposed a new algorithm for mining association rules in Geographic Information Databases. The algorithm specifies an SQL-like spatial data mining query interface, which is based on *Spatial-SQL* [EH94], for an experimental spatial data mining system protocol *GeoMiner*. Many variations of the above algorithm can be explored to enhance the power and performance of spatial association rule mining.

ILP methods also have been extensively used in spatial data mining. Most of the mining algorithm requires the reduction of multi-relational database to the single format. The strength of ILP method is the common background with deductive relational database (DDB) which can be exploited to implement the notion of inductive database [Man97] as pointed out by Flach [Fla98]. In recent times, a database(DB) approach to multi-relational data mining has been presented [KBJ+99]. It explodes the semantic information in the database schema to prune the search space and define the database primitives to ensure efficiency.

Some more works can be found in [Pop98]. The paper has presented a general purpose ILP system: *INGENS* [MEL+00], which is an inductive graphic information system with learning capabilities that currently support the classification task. There is another ILP system called *SPADA* (Spatial Pattern Discovery Algorithm) [ML01] which operates on DDB set up by an initial step of feature extraction from a spatial database. The basic idea in this ILP approach is that a spatial database can be boiled down to a DDB once that reference objects and task- relevant objects, their spatial properties and the spatial relationship among them have been extracted according to predefined semantics. As for topological relations, the algorithm has adopted the 9-intersection [EH94] model. The *SPADA* can tackle applications which cannot be handled by either Geo-Associator [HKS97] or WARMR [DT99].

2.1.5 Constraint-based Association Rule Mining

As the name suggests, constraint-based association rule mining allows [SMO+94] users to specify some constraints. Thus, association rules become more useful and interesting to the users. A simple way is to find all the association rules and then filter out the rules which do not satisfy the users' constraints. However, it may generate a lot of redundant rules. So, it is required to incorporate the constraints into the steps of rule generations. Constraints may be of different types.

1. *Knowledge type constraints*: This refers to the type of knowledge to be mined such as association rules.
2. *Data constraints*: This specify set of task-relevant data.
3. Dimension/level constraint: This refers to the number of dimension and levels of concept hierarchy to be used.
4. *Interestingness constraints*: This refers to the interestingness measurement such as support, confidence, etc.
5. *Rule constraints*: This refer to the form of rules to be mined. As for example, user may specify number of predicates in the antecedent and consequents of the rules, attributes values, aggregate values, etc.

Ng. et al. [NLH+98] carried out some work on constraint-based association rule mining. They proposed *CAP* algorithm for constraint-based association rule mining. Some works on meta-rule guided constraint-based mining can be found in [KHC97]. Meta-rules are generally based on users' experience, expectation, intuition, etc. Again, rule constraints can be classified into five categories with respect to frequent itemset mining.

1. *Anti-monotone*: These constraints are generally applied to iterative algorithms like *Apriori* so that number of iterations are reduced and at the same time preserves the completeness. One example may be "sum(price)$\leq$500". So, any itemsets whose total price is greater than Rs. 500 can be rejected because all of its supersets will be having price more than Rs. 500.

2. *Monotone*: These constraints are opposite to anti-monotone. One example is "sum(price) $>$ 500". Here, superset of any itemset, which satisfies this constraint, will also satisfy the constraint. So, it is not required to check the constraint for the supersets.

3. *Succinct*: By this constraint, one can find all the itemsets which are guaranteed to satisfy the constraint. One example may be "max(prie)$\geq$500". This constraint can be tested before the support count starts, which in turn reduces the execution time.

4. *Convertible*: These constraints can be converted to anti-monotone or monotone by rearranging the items in the itemset and transactions. One example may be "avg(prices) $\leq$ 1000". Here, if the items in a transaction are added to an itemset in the ascending order of process, superset of an itemset, which violates this constraint, will also violate the constraint.

5. *Inconvertible*: These constraints are tough and cannot be converted to previous constraints. One example may be "sum(itemset) $\leq$ 1000 and value of each item in the itemsets is any real number".

Some useful concepts of the predicate constraints can be found in [AK93, LHC97, SVA97]. [AMS+94] also gives an efficient method for mining constrained correlated sets.

2.2 Finding Frequent Itemset in Dynamic Databases

One general assumption in all the above algorithms is that database is static. However, in practice, no database is static. The itemsets which are frequent may not be frequent when the database is updated and the itemsets which are not frequent may become frequent when the database is updated. So, some algorithms are required to update the set of frequent itemsets when the database is updated. Moreover, new database may contain some new interesting rules which were not present in the old database.

One obvious technique to find frequent itemsets in dynamic databases is re-running the algorithms for the updated database, which is not desirable. The main thrust is to use the already existing frequent itemsets. [CHN+96b] has given *FUP* (Fast Update Algorithm). This algorithm has been found to be superior to re-running of the *Apriori* algorithm over the updated database by a factor of 2 to 16. The algorithm works in the similar way as the *Apriori*. It also generates the candidate-sets based on the large itemsets in the previous pass. Followings are the main features of the algorithm, which distinguishes it from *Apriori*:

- In each iteration, the support of large itemsets are updated against the incremental database to filter out itemsets that are no longer large in the updated database. Only the incremental database is scanned to do the filtering.
- While scanning the increment, a set of candidate sets is extracted from the transactions in the incremental database, together with their supports. The support of these itemsets are then updated against the the old database to find the new large itemsets.
- Many itemsets are pruned by a simple check on their supports in the incremental database.
- the size of the updated database is reduced at each iteration by pruning some items from some transactions in the updated database.

Another version of *FUP* is *FUP2* [CLK97], which addresses the maintenance problem for association rule mining. The algorithm has taken care of the deletion of transactions also. The algorithm works like *Apriori*. The difference is that it divides the candidate itemsets into two subsets - one subsets keeps the candidate sets which were large in the old database and the other keeps the new candidate sets. The algorithm scans the old database, if there are some new itemsets in the updated database.

Thomas et al. [TBA+97] has discussed one very efficient algorithm for the incremental updating of association rules. This algorithm has used the concept of negative border sets. The negative border consists of all itemsets that were candidates of level-wise method which did not have enough support i.e. an itemset which is not large, but all its subsets are large. This algorithm has been found to be superior to the *FUP* algorithm both in terms of execution time and number of candidate generation.

Incremental algorithms were also considered in [FAA+97]. The algorithm *DELI* [LCK98] has used a sampling technique to find the amount of changes of new association rules. It has used the concept of upper and lower bound to determine if the maintenance is required or not.

Feldman et al. [FAL+99, Puj01] proposed one very efficient algorithm which uses the concept of *border set* and *promoted border set*. The algorithm is called *Borders* algorithm. An itemset X is called a border set if X is not frequent, but all its subsets are frequent. An itemset that was a border set before update and has become frequent set after update is called a *promoted border set*. The *Borders* algorithm maintains support counts for all the frequent sets as well as for all the border sets. The main advantage of the algorithm is that it uses the existing frequent itemsets to find the new frequent itemsets in the updated database. However, the disadvantage of the algorithm is that it has to scan the whole database frequently if there is even one promoted border set.

The algorithm *MAAP* [ZE01] also efficiently generates the incremental association rules in the updated database by applying the *Apriori* property. The algorithm first computes the high level large itemsets. Then it starts by generating all lower level large itemsets. This algorithm takes care of the small itemsets in the old database also. Incremental algorithms were also considered in [FAA+97].

Another algorithm can be found in [ES02], which has used *FP-tree* to update the association rules in an incremental database.

2.3 Interestingness of Association Rules

It is obvious that all the strong rules are not interesting. To support this idea some work on quantifying the *usefulness* and *interestingness* of the generated rules can be found in [PM94]. Several metrics such as confidence and support [AIS93], variance and chi-squared value [NM, Mor98], gain [FMM+96], entropy gain [MFM+98], gini [MFM+98], laplace [CB91, Web95], conviction [BMU+97], etc. are used to measure interestingness of a rule. There are several algorithms that efficiently find best rule according to some of these metrics [FMM+96, NM, RS02, BA99]. Among them, the technique given in [BA99] is worth mentioning. [BA99] has defined an optimized rule mining problem using partial order. It has also shown that solving the optimized problem with respect to a particular partial order is guaranteed to identify most interesting rule according to other interesting metrics mentioned above. Ultimately, it is the users who decide if a rule is interesting or not.

Sometimes, the rules of the form $X \Rightarrow Y$ may be misleading because algorithms may find strong rules among the items/attributes which are negatively correlated. So, some alternative framework is required to measure the interestingness of a rule. [BMS97] has given one such framework of correlation between the itemsets, which can be used to find if a rule is interesting or not. One strong rule can be interesting, if the itemsets in the rule is positively correlated. In addition to that, χ^2 statistic can be used to see if the correlation is statistically significant or not.

2.4 Multi-Objective Rule Mining

Association rules are evaluated by metrics such as *support, comprehensibility, interestingness*, etc. These metrics can be thought of as different objectives of association rule mining. So, association rule mining is a multi-objective problem instead of single-objective problem. Multi-objective rule mining has been discussed in [GN04], which has used *Pareto* based genetic algorithm to extract useful and interesting rules.

2.5 Feature Selection

Relevant feature selection is important in all kinds of databases including static and dynamic databases, because of the fact that all the attributes/features in a database are not important. So, it is required to find the relevant features/attributes in a database to better represent the domain. There have been some works in the field of relevant feature selection in a dataset. One of the earliest works is *Branch and Bound* [NF77]. The algorithm attempts to find the best set of features according to some monotonic function. *Relief* [KR92] is a weight-based algorithm. It uses random sample and is based on the concept of *NearHit* and *NearMiss*. The algorithm calculates weights of the features in each iteration and selects the features with highest weights. However, The algorithm is suitable for noisy, correlated features and binary classes. Another algorithm *Focus* [AD91] selects features based on consistency measures. It works well with noise-free data. [LS96] also uses consistency measures to select subset of features. It first selects a random sample of features and then applies consistency measures on the sample. The algorithm uses one inconsistency threshold, which can be tuned according to requirement. The good things about the algorithm are that it is very simple to implement and guaranteed to find optimal subset. The algorithm *MDLM* [SDN90] is based on the concept that if the features in a subset X can be expressed as a fixed non-class-dependent function of the features in another subset Y, then once the values in the features in the subset X are known, the features in the subset Y are useless. *Minimum Description Length Criterion(MDLC)* is used for this purpose. The algorithm exclusively searches all the possible subsets and returns the subset satisfying *MDLC*. This method can find all the useful features for *Gaussian* cases.

2.6 Discussion

As can be experienced from the related works reported so far that substantial works have been carried out in various dimensions of association rule mining techniques. Based on the survey, it can be observed that

1. Generation of unnecessary candidate itemsets by most of the algorithms such as *Apriori* is the main reason behind the degraded performance in terms of execution time. Even the performance of efficient algorithm with reduced candidate sets such as *FP-growth* also can be found to degrade with the increase of minimum support.
2. Partitioning is an effective approach of finding frequent itemsets over large databases. However, with the increase of dimensionality, performance of horizontal partitioning also degrades.
3. *Borders* algorithm suffers from the drawback of having to scan the entire database frequently.
4. There does not exist, to the best of our knowledge, any distributed version of *Borders* algorithm for distributed dynamic databases.

Feature selection plays an important role in machine learning problems. Based on the existing survey, it has been found that

- The existing algorithms cannot select relevant features in all kind of databases and they are very much time consuming.
- The drawbacks can be overcome by the use of frequent features.

Both static and dynamic association rule mining techniques, mostly fundamental association mining problems such as frequent itemsets generation problems, have been studied and analyzed thoroughly in this thesis.

Chapter 3

Frequent Itemsets in Static Databases

Association rule was introduced by Agarwal et al. in 1993 [AIS93]. Basically, association rule finds influence of one set of items/objects on another set of items/objects. One example of association rule may be of the form "80% of the customers who buy bread also buy butter". Association rules have found numerous applications in real world such as decision support, understanding customer behavior, telecommunication alarm diagnosis, prediction, etc. Departmental stores also can use association rules in many fields such as catalog design. add-on sales, store layout, etc. The terms used in connection with association rule mining are *itemset, frequent/large itemset, support, confidence*, etc.

This chapter reports some of the existing popular frequent itemsets finding algorithms and presents a detailed analysis of these algorithms in terms of their efficiency in execution time and storage utilization. It also presents two enhanced versions of *Apriori* [AMS$^+$94] , which overcomes (*i*) too many candidate generations and (*ii*) execution time bottleneck due to huge repository of market-basket databases. The following section gives some basic concepts of association rule mining techniques.

3.1 Basic Concepts

Basic concept of association rule is best explained in [AMS+96]. Let I be a set of items in a super market and D be a database of customers' transactions, where each transaction t is a set of items such that $t \subset I$. I also can be considered as a set of attributes over the binary domain $\{1, 0\}$. In that case, one transaction will be a string of 0's and 1's, where 1 represents that corresponding item has been bought and 0 represents that corresponding item has not been bought. A set of items $X \subset I$ is called an itemset. *Support* of an itemset X, denoted by $Sup(X)$, is defined as the percentage of transactions in D, that contain X. An itemset with support greater than a pre-defined value(called *minimum support*) is called *frequent* or *large* itemset. An association rule between two frequent itemsets X, Y (denoted by $X \Rightarrow Y$) may exist with *support* s and *confidence* $conf$, if $X \cap Y = \phi$. Support of $X \Rightarrow Y$ is the $Sup(X \cup Y)$. The confidence of $X \Rightarrow Y$ is defined as the percentage of transactions in D that contain X, also contain Y, and is calculated as $Sup(X \cup Y)/Sup(X)$.

Example 3.1

Let us consider a database D (*Table* 3.1) with 5 items (A, B, Q, R, S) and 5 transactions. There are 31 possible itemsets. Some of them are $\{A\}$, $\{B\}$, $\{Q\}$, $\{AB\}$, $\{ABQ\}$, etc. Support of $\{A\}$ is 60% because it has occured in 3 transactions. Similarly, support of $\{B\}$, $\{Q\}$, $\{AB\}$, $\{ABQ\}$ are 80%, 60%, 60%, and 40% respectively.

TID	A	B	Q	R	S
100	1	1	1	0	1
200	1	1	1	0	1
300	0	0	1	1	0
400	1	1	0	0	1
500	0	1	0	0	1

Table 3.1: A Sample Database - I

Suppose, minimum support is 30%. Then, all the above mentioned itemsets will be frequent, because support of all of them are at least 30%. It can be observed that AB has occured in 3 transactions and Q has occured in 2 transactions out of those 3 transactions. So, association rule $AB \Rightarrow Q$ exists with confidence 66.6% and support 40%. Similarly, $A \Rightarrow B$ exists with support 60% and confidence 100%.

Given a database D, the problem of mining association rules is to generate the association rules that have certain pre-defined minimum support and confidence. The problem of association rule mining can be divided into two subproblems.

1. Find all the frequent itemsets in the given database.
2. Find the association rules using the large itemsets found in the first step. As for example, suppose $ABQR$ and AB are large itemsets. Then the confidence of the rule $AB \Rightarrow QR$ is calculated as $conf = Sup(ABQR)/Sup(AB)$. If $conf \geq minconf$, then the rule $AB \Rightarrow QR$ is said to exist with confidence $conf$.

Out of these two steps, the first step is important and difficult one. That's why, most of the algorithms concentrate on finding the frequent itemsets from a large database in minimum possible time and using minimum resources. Once large itemsets are known, finding association rules are straightforward. Next section discusses some popular algorithms to find frequent itemsets.

3.2 Some Existing Algorithms

There are many algorithms to find frequent itemsets. However, different algorithms target different types of database. Here, three basic algorithms for frequent itemset generations i.e. *Apriori*, *AprioriTid* and *AprioriHybrid* [AMS+94] have been chosen and their performance have been analyzed.

3.2.1 *Apriori*

Among the popular algorithms to find the large itemsets, this algorithm stands at the top because of its simplicity and effectiveness. The algorithm is based on the fact that all subsets of a frequent itemset are also frequent. The algorithm first makes one pass over the database and finds the large items. Then, the algorithm makes many passes over the data. Each pass starts with the seed set of large itemsets which are used to generate new potentially large itemsets called candidate itemsets. Then, support of each candidate itemset is found during the pass over the data and the actual large itemsets are determined. These large itemsets become the seed for the next pass. This process continues till large itemsets can be found.

Let $c[1], c[2], c[3], ...c[k]$ be a k-itemset stored in the lexicographic order i.e. $c[i] < c[i+1]$ (i=1,.., k-1). Let L_k be a set of large k-itemsets with two fields : itemset and the support count; C_k be a set of candidate k-itemsets with two fields : itemset and the support count; D be the database of transactions; *minsup* and *minconf* be the minimum support and minimum confidence. With these symbols, the algorithm is given in *Algorithm* 3.1.

```
Input : $D$, minsup
Output : All frequent/large itemsets

1. $L_1$ = {Large 1-itemsets};
2. For($k$=2; $L_{k-1} \neq \phi$; $k$++) do {
3.     $C_k$ =Apriori-gen($L_{k-1}$); // New candidates generation
4.      For all transactions $t \in D$ do {
5.          $C(t)$ = Subset($C_k, t$); // Candidates contained in $t$
6.          For all candidates $c$ in $C(t)$ do
7.          c.count++; }
8.     $L_k = \{c \in C_k | c.count \geq minsup\};$ }
9. Return all large itemsets = $\bigcup L_k$;
```

Algorithm 3.1: *Apriori*

Candidate Generation and Pruning

The algorithm uses the function *Apriori-gen* to generate candidate sets. The function takes set of all large k-1 itemsets (L_{k-1}) as input and produces the candidate k-itemsets (C_k). The function generates a candidate k-itemset from two k-1 frequent itemsets with the same first k-2 items. If a and b are two large k-1 itemsets such that first k-2 items of the itemsets are same, then the algorithm of the function can be described as follows.

1. Insert $a \cup b$ into C_k.

2. Select $a[1], a[2], ...a[k-1], b[k-1]$ from a and b where $a[i] = b[i] (i = 1,..,$ k-2) and $a[k-1] < b[k-1]$ i.e., $a \cup b$ will be inserted into C_k if a and b have first k-2 items in common and $a[k-1] < b[k-1]$.

As for example, let $\{ABQR\}$ and $\{ABQS\}$ be two frequent 4-itemsets such that first 3 items of the itemsets are same. These two itemsets can be combined to form a candidate 5-itemset $\{ABQRS\}$ (i.e. $\{ABQ\} \cup \{R\} \cup \{S\}$). However, $\{ABQ\}$ and $\{ARS\}$ cannot be combined, because only first items are same.

Pruning is one important step in the algorithm because the algorithm may generate a lot of redundant candidate sets. The basic principle of *Apriori* is that if an itemset is frequent, then all the subsets will also be frequent. So, any candidate itemset, with a subset which is not frequent, is pruned away. Thus, it reduces number of candidate itemsets to a great extent.

Example 3.2

Let us again consider the *Table* 3.1 on page 40 and take minimum support as 60% (3 transactions). If *Apriori* is run, results are obtained as given in *Figure* 3.1 on the next page. The item R is not included in L_1 because $Sup(R)$ is less than minimum support(3).

L_1

Itemset	Support
A	3
B	4
Q	3
S	4

C_2

Itemset	Support
AB	3
AQ	2
AS	3
BQ	2
BS	4
QS	2

L_2

Itemset	Support
AB	3
AS	3
BS	4

C_3

Itemset	Support
ABS	3

L_3

Itemset	Support
ABS	3

Figure 3.1: Example of *Apriori* Algorithm

3.2.2 *AprioriTid*

This algorithm is a modification of the *Apriori* algorithm. This algorithm also generates the candidates using the same generating function *Apriori-gen* as in the *Apriori*. The main feature of the algorithm is that the original database is not used after the first pass. Instead of that, a data structure C'_k is used. Each member of the set C'_k is of the form $< TID, \{X_k\} >$, where X_k is a potentially large k-itemset present in the transaction with the identifier TID. For k=1, C'_k is the database itself with each item i is replaced by itemset $\{i\}$. For k >1, the member of C'_k corresponding to a transaction t is $< t.TID, \{c \in C_k | c$ contained in $t >\}$. If a transaction does not contain any candidate set, then C'_k will not have any entry for that transaction. So, number of entries in C'_k gets reduced in the successive passes resulting in fewer transactions to be scanned in each subsequent passes. One shortcoming of the algorithm is the creation and updating of C'_k, which takes considerable amount of execution time. The algorithm is given in

Algorithm 3.2.

Example 3.3

Let us apply the algorithm on the same sample database of *Table* 3.1 on page 40 with minimum support 60%(3 transactions). The result is given in *Figure* 3.2.

```
Input: $D$, $minsup$
Output: All frequent/large itemsets.

1. $L_1$={large 1-itemsets };
2. $C_1'$= database $D$;
3. For $(k = 2; L_{k-1} \neq \phi; k++)$ do begin
4.     $C_k$=$Apriori-gen(L_{k-1})$; //New candidates
5.     $C_{k'} = \phi$;
6.     For all entries $t \in C_{k-1'}$ do begin
7.         //determine candidates contained in the transaction $t.TID$
       $C(t) = \{c \in C_k | (c[1].c[2]...c[k-1]) \in t.set\_of\_itemsets \wedge (c[1].c[2]...c[k-2].c[k]) \in t.set\_of\_itemsets\}$;
8.         For all candidates $c \in C(t)$ do
9.             $c.count{+}{+}$;
10.        If $(C(t) \neq \phi)$ then $C_{k'} += < t.TID, C(t) >$;
11.    End
12.    $L_k$=$\{c \in C_k | c.count \geq minsup\}$;
13. End
14. Answer=$\bigcup_k L_k$;
```

Algorithm 3.2: *AprioriTid*

Transaction with TID 300 has been dropped from C_2' on-wards because the transaction does not contain any more frequent itemset. Thus, number of transactions are reduced which results in reduction in execution time.

C_1'

TID	Set of itemsets
100	$\{\{A\},\{B\},\{Q\},\{S\}\}$
200	$\{\{A\},\{B\},\{Q\},\{S\}\}$
300	$\{\{Q\},\{R\}\}$
400	$\{\{A\},\{B\},\{S\}\}$
500	$\{\{B\},\{S\}\}$

L_1

Itemset	Support
$\{A\}$	3
$\{B\}$	4
$\{Q\}$	3
$\{S\}$	4

C_2

Itemset	Support
$\{AB\}$	3
$\{AQ\}$	2
$\{AS\}$	3
$\{BQ\}$	2
$\{BS\}$	4
$\{QS\}$	2

C_2'

TID	Set of itemsets
100	$\{\{AB\},\{AQ\},\{AS\},\{BQ\},\{BS\},\{QS\}\}$
200	$\{\{AB\},\{AQ\},\{AS\},\{BQ\},\{BS\},\{QS\}\}$
400	$\{\{AB\},\{AS\},\{BS\}\}$
500	$\{\{BS\}\}$

L_2

Itemset	Support
$\{AB\}$	3
$\{AS\}$	3
$\{BS\}$	4

C_3

Itemset	Support
$\{ABS\}$	3

C_3'

TID	Set of itemsets
100	$\{\{ABS\}\}$
200	$\{\{ABS\}\}$
400	$\{\{ABS\}\}$

L_3

Itemset	Support
$\{ABS\}$	3

Figure 3.2: Example of *AprioriTid* Algorithm

3.2.3 *AprioriHybrid*

The algorithm is basically a fusion of *Apriori* and *AprioriTid* . It uses *Apriori* for the first few passes and *AprioriTid* for the remaining passes based on some threshold values i.e. when it is found that candidates can be stored in memory, it uses *AprioriTid.* It uses good characteristics of both the algorithms and superior to both the algorithms. It has been found that *AprioriHybrid* is better than *Apriori* by 30% and *AprioriTid* by 60% [AMS+96].

3.2.4 Experimental Results

Experiments were carried out to compare the performance of *Apriori*, *AprioriTid* and *AprioriHybrid.* Experiments were carried out on a Pentium IV PC with 256 MB RAM. The experimental results are given in *Figures* 3.3 on the following page and 3.4 on page 49.

Data Sources: Two synthetic databases (T20.I4.D100K and T20.I6.D100K) and one real database *Connect4*, publicly available in UCI machine learning repository (http://www.ics. uci.edu / mlearn/mlrepository.html) have been chosen for the experiments. The synthetic databases were generated using the technique given in [AMS+96] with the value of parameters as given in *Table* 3.2 on the following page and with dimensionality 500. Here, $|T|$, $|ML|$ and $|D|$ represent the average size of a transaction, mean size of a potentially large itemset and database size (number of transactions) respectively. Each of the synthetic datasets contains 100K records. *Connect-4* database contains around 65K instances with 43 attributes. Each attribute can have one of three values. Each distinct value of the attributes has been considered as one item, resulting in total 129 items.

Name	$\lvert T\rvert$	$\lvert ML\rvert$	$\lvert D\rvert$
T20.I4.D100K	20	4	100K
T20.I6.D100K	20	6	100K

Table 3.2: Parameters for Synthetic Databases - I

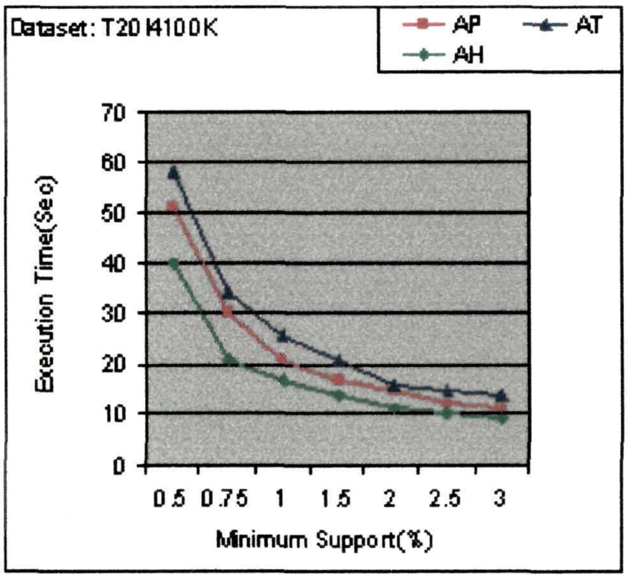
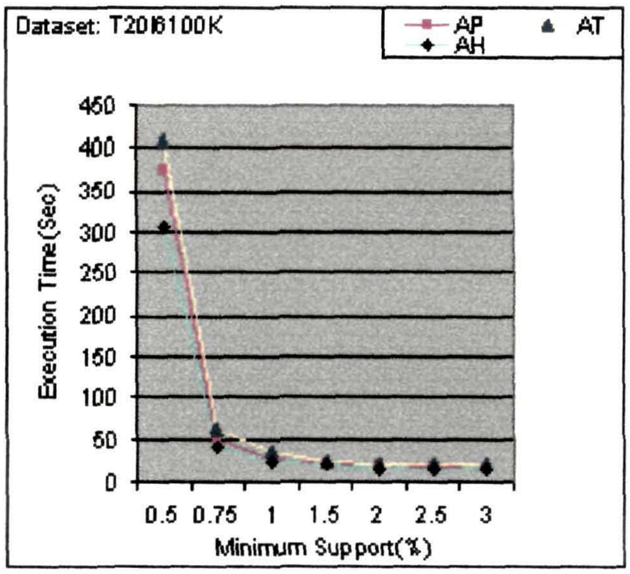


Figure 3.3: Experimental Results of *Apriori,AprioriTid* & *AprioriHybrid* - I



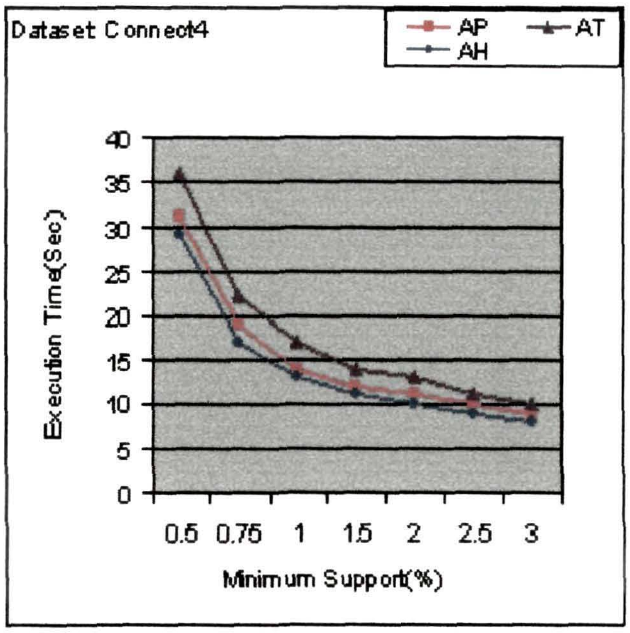


Figure 3.4: Experimental Results of *Apriori*,*AprioriTid* & *AprioriHybrid* - II

3.2.5 Discussion

The algorithm *Apriori* is very easy to implement and finds all possible frequent itemsets. However, it suffers from some serious drawbacks, which increase the execution time.

- It makes several passes over the databases, which can be found to be very expensive for large databases.
- Though it prunes away some candidates, yet it generates a lot of unnecessary candidates. So, for a database with large number of items, it can be found too expensive from storage as well as execution time point of view.

AprioriTid is also robust enough to find all the frequent itemsets. It differs from *Apriori* in the sense that it scans the database once and uses a better data structure for the rest of iterations. The size of the data structure is smaller than that of original database, which is the main advantage of the algorithm. However, it also suffers from the problems of *Apriori*. In addition to that, some extra memory and extra disk space are required for the data structure. Some extra time also spent to maintain the data structure.

AprioriHybrid uses the benefits of both the algorithms. That is why, it is the best algorithm among all the three algorithms. However, it also suffers from the problems of both the algorithms.

The basic problem of all these algorithms is the ratio of number of candidate sets to frequent sets. Based on exhaustive experiments in the light of several real and synthetic datasets, it has been observed that, on the average, this ratio is as high as 50:1. So, some new techniques are required to address this problem to bring down this ratio to as low as possible.

It can be observed from the experimental results that *AprioriHybrid* is the best algorithm in terms of execution time among the three algorithms. Poor performance of *AprioriTid* can be attributed to two possible reasons: one is that it could not reduce the database size and the other is that it took maximum time to maintain the data structure. *Apriori* performed better than *AprioriTid* and worse than *AprioriHybrid*. The reason is that it has to make a pass over the database in every iteration and it does not employ any database reduction technique.

As mentioned above, main disadvantage of all these algorithms is the generation of too many candidate sets. Some possible solutions of this problem are given below.

- *Using hash based techniques* : Hash based techniques has been found to be useful in reducing the redundant candidate sets. Some hash-based algorithms can be found in [PCY95a]. These techniques work well, when the hash tables are small and can be kept in memory. However, these algorithms suffer from the drawbacks of having to create hash tables and maintaining them.
- *Using sample*: Sampling is also an useful technique to reduce the database size and number of candidate sets. Use of sampling technique to find frequent itemsets can be found in [LCK98]. The idea is to pick a random sample, use it to determine all association rules in the sample. However, there is a trade off between accuracy and efficiency.
- *Using efficient data structure*: Performance of the algorithms for frequent itemsets generation can be improved to a great extent using efficient data structures. One such efficient data structure is FP-tree [HPY00], which has been used in *FP-growth* algorithm [HPY00].
- *Using probability*: Probability is one useful technique, which can be used to predict whether an itemset will be frequent or not. If the probability of a candidate set is high enough, then it can be kept for calculating its frequency count. Otherwise, it can be rejected with least risk.

Next section presents two enhanced versions of *Apriori* to address the problem of too many candidate generations and for faster implementation.

3.3 The *Modified_Apriori* Algorithm

It has been observed that *Apriori* algorithm is robust enough to find all frequent itemsets from a database. However, it suffers from two major drawbacks.

1. It is not scalable.
2. It generates huge number of candidate sets, out of which only a few are actually large.

In *Modified_Apriori*, *Boole's inequality* [GK84] has been used to reduce the number of candidate itemsets. The main part of the algorithm is same as *Apriori*. The point by which it differs from the *Apriori* algorithm is the use of the Boole's inequality in generating the candidate itemsets. The heart of the algorithm is the function *Comp_Apriori_gen*, which computes the candidate sets reasonably by exploiting the inequality (reported in the next subsection).

3.3.1 Background

The main objective of the *Modified_Apriori* is to reduce the number of unnecessary candidate sets in the frequent itemset generation by exploiting the Boole's inequality.

Boole's inequality : For k events A_i(i=1,2,3,...k),

$$P(\bigcap_{i=1}^{k} A_i) \geq \sum_{i=1}^{k} P(A_i) - (k-1)$$

The above inequality can be found very useful in the generation of candidate sets reasonably. As for example, let us consider an itemset $X = \{A, B, Q\}$. The support of X can be calculated as the probability of occurrence of the items in X together in the database. So, support of X = (Number of records containing X / Total number of records). Using Boole's inequality, minimum support of X (in %) can be estimated as $P(A)$+$P(B)$+$P(Q)$ - 2. Obviously, this is not the actual support. It just gives an representative value of the support of the itemset. Here, probabilities of individual items i.e. $P(A), P(B)$ and $P(Q)$ are required, which can be calculated in the first pass of the *Apriori*.

Using Boole's inequality, representative values of minimum supports of the candidate itemsets can be calculated easily. Candidate itemsets with representative values greater than some threshold, will have high probability of being frequent , which can be used as an additional pruning step. However, calculating the threshold is not an easy task. The optimum threshold is the minimum support itself. However, it is too high. Very few candidate itemsets will cross that threshold. So, the threshold is calculated as

$$k * minsup - (k-1) + \frac{(k-1)(1-minsup)}{\beta}$$

where k is the size of the itemset and $\beta > 1$.

3.3.2 The Algorithm

The algorithm is given in *Algorithm* 3.3 on the next page. Most of the symbols used in the algorithm are same as those of *Apriori*. One additional symbol used here is PA, which is an array to store the probabilities of occurrences of each large item in a transaction. Although the algorithm is similar to *Apriori*, yet the first line needs a little explanation. The first line of the algorithm finds the large 1-itemsets and calculates the probability of occurrences of each large item i, i.e. $PA[i]$, which is calculated as

$$PA[i] = (\text{Number of records containing } i) \ / |D|$$

3.3.3 Candidate Generation

Candidates are generated by the function *Comp_Apriori_gen*, which uses the Boole's inequality to generate candidate sets. The function *Comp_Apriori_gen* is given in *Function* 3.1 on page 55, which takes in L_{k-1} and returns C_k. The union of itemsets $a, b \in L_{k-1}$ (i.e. $a \cup b$) is inserted in C_k if they share their first k-2 items and the value of the Boole's inequality of the items in $a \cup b$ is at least

$$k * minsup - (k-1) + \frac{(k-1)(1-minsup)}{\beta}$$

The rationale behind calculating the Boole's inequality is that if the value is very low, there is little chance that the itemset will be frequent. So, these itemsets can

```
Input: $D$, $minsup$, $\beta$
Output: Frequent/large itemsets

1. $L_1$ = Large 1-item sets and calculate $PA[i]$;
2. For ($k = 2$; $L_{k-1} \neq \phi$ ; $k$++) do {
3.     $C_k$ = $Comp\_Apriori\_gen(L_{k-1})$;
4.     For all transaction $t \in D$ do {
5.         $C(t)$ =$subset(C_k, t)$;
6.         For all candidates $c \in C(t)$ do
7.             $c.count$++; }
8.     $L_k = \{c \in C_k | c.count \geq minsup\}$; }
9. Return the set of all large itemsets = $\bigcup_k L_k$;
```

Algorithm 3.3: *Modified_Apriori*

easily be rejected without much overhead, resulting in smaller size of candidate itemsets. The algorithm generates candidate itemsets with the probability of being frequent very high. Thus, it reduces the difference between number of candidate itemsets and that of actual large itemsets to a great extent, resulting in significant reduction of the I/O overhead.

Selecting β : Here is a very simple technique to choose the value of β. The algorithm is run with a small random sample of the database and an initial value of β. If the result is not satisfactory, β is tuned according to requirement and the algorithm is rerun . The process is continued several times to obtain optimum results. When optimum results are obtained, the corresponding value of β can be used for the entire database. As a guideline to choose the values of β, it has been observed that values of β within the range of 3 to 6 give better result.

```
Function $Comp\_Apriori\_gen(L_{k-1})$
 1. $C_k=\phi$;
 2. $x = k * minsup - (k-1) + \frac{(k-1)(1-minsup)}{\beta}$
 3. For all $a, b \in L_{k-1}$
 4. Set $Temp$ to 0;
 5. For ( $i$ =1 ; $i$=$k$-2; $i$++) do
 6.      $Temp = Temp + PA[a[i]]$ ;
 7. $Temp = Temp + PA[a[k-1]] + PA[b[k-1]]$;
 8. Insert $a \cup b$ into $C_k$, if $a[i] = b[i]$ ($i$=1,2, ...$k$-2) and $a[k-1] < b[k-1]$ and
    $Temp \geq x$;
 9. End for
10. Return $C_k$;
```

Function 3.1: *Comp_Apriori_gen*

3.3.4 Experimental Evaluation

Experiments were carried out for performance comparison of the proposed *Modified_Apriori* with *Apriori*, *AprioriTid* and *AprioriHybrid*. Experiments were carried out on a Pentium IV PC with 256MB RAM. The method discussed above has been used to find value of β. Average results of the experiments are reported in *Figures* 3.5 through 3.13.

Data set : Two synthetic databases (T20.I4.D100K and T20.I6.D100K) and one real database *Connect4* (http://www.ics. uci.edu/ mlearn/mlrepository.html) were used for the experiments. The synthetic databases were generated using the technique given in [AMS$^+$96] with the parameters given in *Table* 3.3 on the next page and with dimensionality 500. Here, $|T|$, $|ML|$ and $|D|$ represent the average size of a transaction, mean size of a potentially large itemset and number of transactions respectively. Each of the synthetic datasets contains 100K records. *Connect-4* database contains around 65K instances with 43 attributes. Each attribute can have one of three values. Here also, each distinct value of the attributes has been considered as one item, resulting in total 129 items.

Observations : It can be seen from the results (*Figures* 3.6 on the following page, 3.9 on page 59 and 3.12 on page 60) that *Apriori*, *AprioriTid* and *AprioriHybrid* generate huge volume of candidate sets, which increases the I/O overhead and execution time. On the other hand, *Modified_Apriori*, as can be seen in the results, generates much less number of candidates. As far as frequent sets (*Figures* 3.7 on page 58, 3.10 on page 59 and 3.13 on page 61) are concerned, *Modified_Apriori* finds almost same number of frequent itemsets as that of other algorithms. *Figures* 3.5 on the following page, 3.8 on page 58 and 3.11 on page 60 show the execution time comparison of the four algorithms. It can be observed that *Modified_Apriori* has taken much less time than that of other algorithms. This can be attributed to the fact that *Modified_Apriori* generates and has to process less number of candidates than that of other algorithms. Another point to be observed is that frequent itemsets in *Modified_Apriori* is a subset of those of *Apriori*, *AprioriTid* and *AprioriHybrid*.

Name	$\|T\|$	$\|ML\|$	$\|D\|$
T20.I4.D100K	20	4	100K
T20.I6.D100K	20	6	100K

Table 3.3: Parameters for Synthetic Databases - II

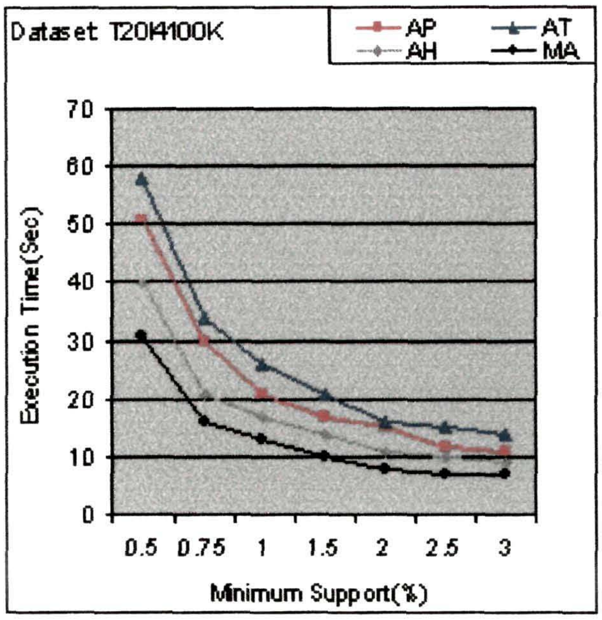


Figure 3.5: Experimental Results of *Modified_Apriori* - I

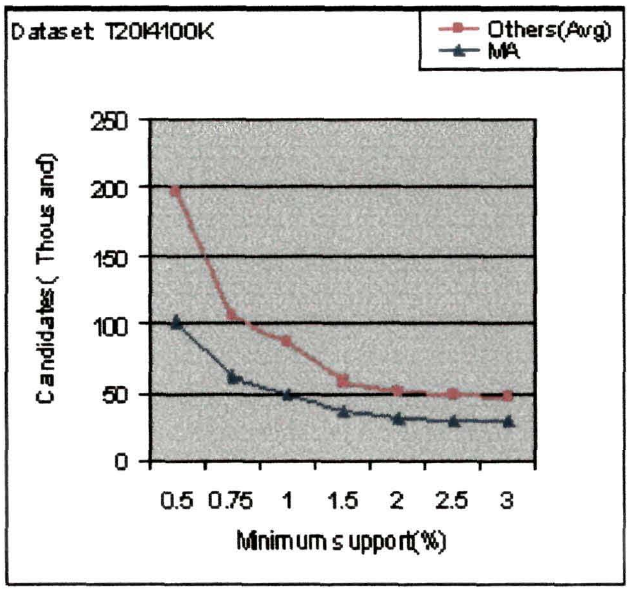


Figure 3.6: Experimental Results of *Modified_Apriori* - II

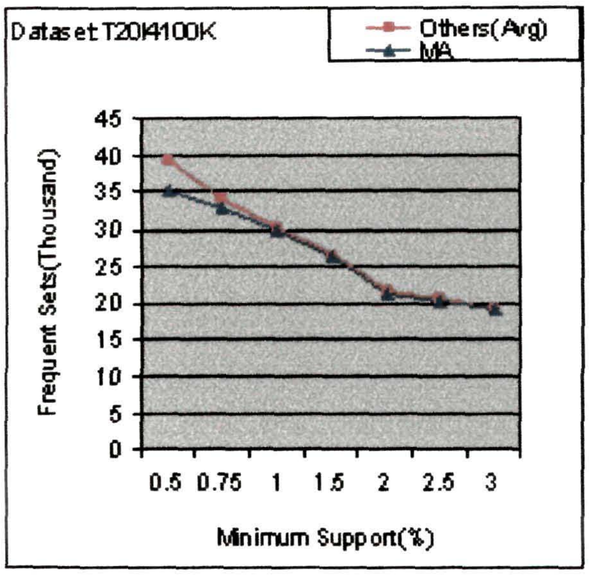


Figure 3.7: Experimental Results of *Modified_Apriori* - III

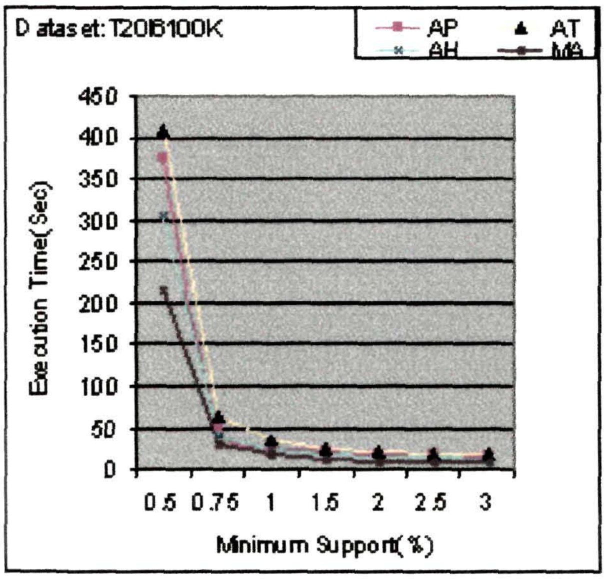


Figure 3.8: Experimental Results of *Modified_Apriori* - IV

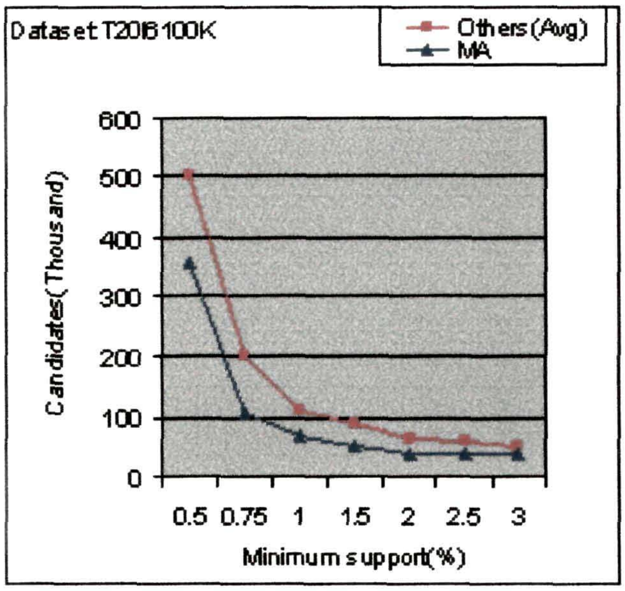


Figure 3.9: Experimental Results of *Modified_Apriori* - V

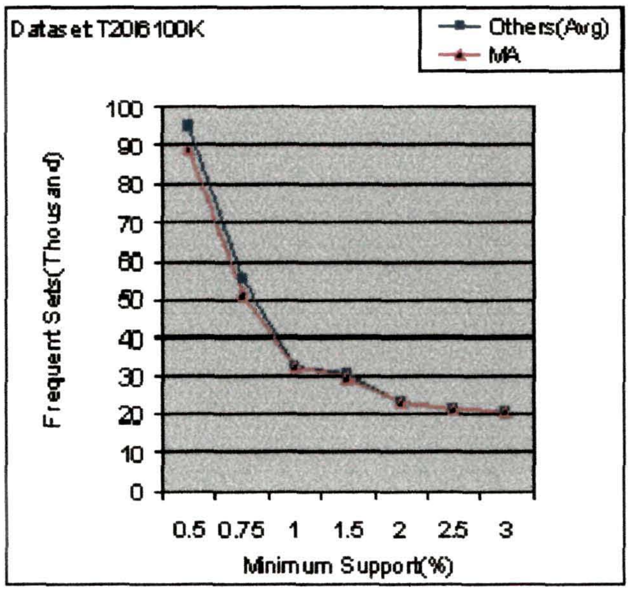


Figure 3.10: Experimental Results of *Modified_Apriori* - VI

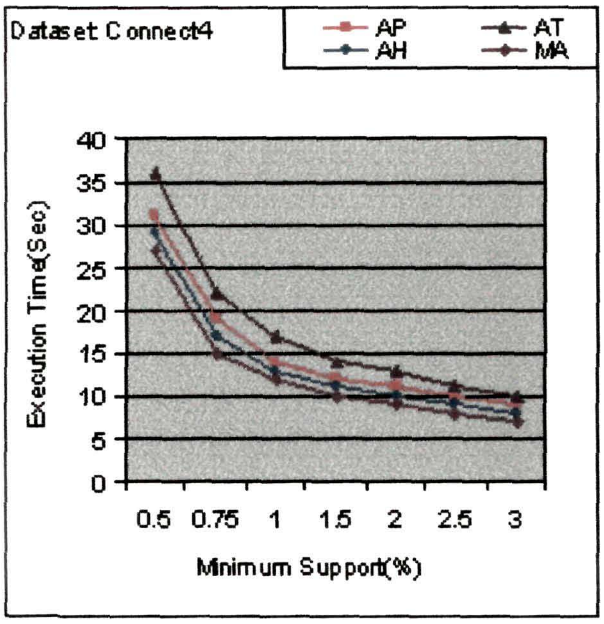


Figure 3.11: Experimental Results of *Modified_Apriori* - VII

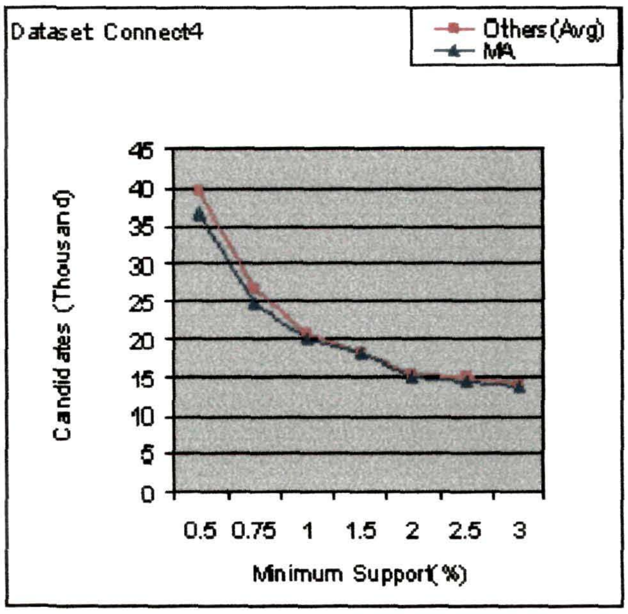


Figure 3.12: Experimental Results of *Modified_Apriori* - VIII

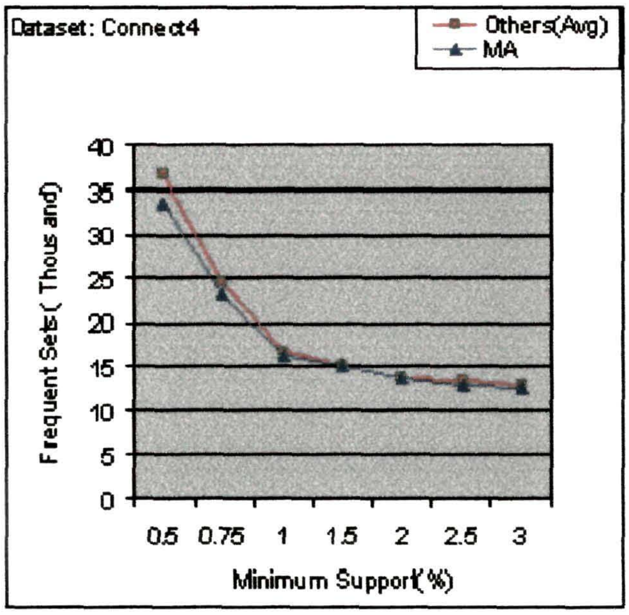


Figure 3.13: Experimental Results of *Modified_Apriori* - IX

3.3.5 Using Multiplication Law of Probability to Generate Candidate Sets

Multiplication law of probability was also used to reduce unnecessary candidate itemsets in *Apriori* [DB04]. The multiplication law of probability states that the probability of occurring independent events together is equal to the product of the probabilities of occurring of the events individually [GK84]. Support of an itemset is the probability of occurrence of the itemset in the database. If it is assumed that items are independent in the database, estimated support of a candidate itemset can be calculated as the product of the probabilities (supports) of the items in the itemset. If this probability(support) is greater than some threshold, the candidate can be kept for further processing, otherwise it can be pruned away. Thus, unnecessary candidates are reduced to a great extent in *Apriori*. However, this approach can be found to be suitable in those cases, where faster frequent itemsets is essential and all the frequent itemsets may not be required to be generated.

3.4 Using Bitmaps to Find Frequent Itemsets

Bitmap techniques have been applied in various application domains [BAG99, Gra94, JDO99, Joh98, MZ98, NG95]. Bitmap indexing method is popular in OLAP products because it allows quick searching in data cubes. In the bitmap index for a given attribute, there is a distinct bit vector for each value in the domain of the attribute. If the domain of the attribute consist of n values, then n bits are needed for each entry in the bitmap index. If the attribute has the value V for a given row in the table, then the bit representing the value is set to 1 in the corresponding row of the bitmap index. All the other bits for that row are set to 0. As for example, let us consider the database (*Table* 3.4). The corresponding bitmap entries of the table are shown in *Table* 3.5.

Item	City
A	V
B	V
A	V
B	V

Table 3.4: A Sample Database - II

A	B
1	0
0	1
1	0
0	1

Table 3.5: Bitmaps for The *Item* Attribute Values

Bitmap indexing is very useful because bit arithmetic is very fast. Bitmap indexing leads to significant reduction in space and I/O since a string character can be represented by a single bit. However, a problem with using bitmap index for a column with high cardinality is its high storage cost and potentially high expression evaluation cost. That is why compression of bitmap is required. The form of compression is most crucial aspect since it must be designed to save disk space and perform the basic operations AND, OR, NOT, etc. A considerable amount of work has been done in the study of bitmap index compression. The use of bitmap compression has several potential performance advantages such as less storage requirement, faster accession as well as easy caching in the memory, etc. There are several good representation and compressions techniques such as *Verbatim, Run Length Encoding, Gzip, Expgol ,BBC,* etc. [AYJ00]. Bitmap technique also has been successfully utilized in association rule mining[LL00, MZ98]. *Bit_AssocRule* [HLL03] is one of the promising association rule mining techniques designed using similar concept. Next section reports the algorithm.

3.4.1 *Bit_AssocRule* Algorithm

A detailed discussion of the algorithm can be found in [HLL03]. The algorithm uses bitmaps of the attribute values to find the frequent itemsets. The algorithm works as follows. The algorithm starts with the list L_1, which contains the large attribute values and bitmap of large 1-itemset. The k-candidates consist of k attribute values from k attributes. The candidate is large itemset if the bit count(i.e. the number of 1's) of the intersection of all the bitmaps in the candidate is equal to or greater than the minimum count. During each cycle. combination of length k i.e. k-candidates are generated. At the end of the cycle if any k-candidate becomes large, new candidate of length $k+1$ is generated in the next cycle. The cycle stops when no k candidates are found to be large itemsets or if no new k+1 candidates can be generated. k- candidate is generated by joining a $k-1$ itemset with 1-itemset in L_1 that has an attribute index greater than all attribute index of elements in that $k-1$ itemset. The algorithm is given in *Algorithm* 3.4 on the next page.

```
Input : $D$, $minsup$, bitmaps of the items
Output : All frequent/large itemsets

1. $L_1$= large 1-itemsets with their bitmaps;
2. For ($k$=2; $L_{k-1} \neq \phi$;$k$++)
3.     Remove those 1-itemsets in $L_1$ which are not included in any itemset
       of $L_{k-1}$ ;
4.     $C_k$= { Join the 1-itemset in $L_1$ that is larger than any element in the
       $k$-1 large itemsets with the itemsets in $L_{k-1}$};
5.     $L_k$= {$c \in C_k$| bitmap count of c $\geq$ $minsup$ };
6. End for
7. Answer= $\bigcup L_k$;
```

Algorithm 3.4: *Bit_AssocRule*

Example 3.4

Let us consider the database D (*Table* 3.1 on page 40) and take minimum support as 60% (3 transactions). If *Bit_AssocRule* is run in the database, the result is obtained as given in *Figure* 3.14 on the next page.

Discussion

The algorithm works much faster than *Apriori, AprioriTid* and *AprioriHybrid* to find the frequent itemsets because it finds the support count of the candidate sets by intersection of the bitmaps of the items. However, experiments show that it generates a huge number of unnecessary candidate sets. The number of candidate sets can be reduced to a great extent by using Boole's inequality in the same way as has been used in *Modified_Apriori.* The following algorithm is a modification of *Bit_AssocRule* algorithm, which uses Boole's inequality to generate the candidate sets.

L_1				C_2					
A	B	Q	S	AB	AQ	AS	BQ	BS	QS
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1
0	0	1	0	0	0	0	0	0	0
1	1	0	1	1	0	1	0	1	0
0	1	0	1	0	0	0	0	1	0

L_2

AB	AS	BS
1	1	1
1	1	1
0	0	0
1	1	1
0	0	1

L_1			C_3	L_3
A	B	S	ABS	ABS
1	1	1	1	1
1	1	1	1	1
0	0	0	0	0
1	1	1	1	1
0	1	1	0	0

Answer= $\{A, B, Q, S, AB, AS, BS, ABS\}$

Figure 3.14: Example of *Bit_AssocRule* Algorithm

3.4.2 *Modified_Bit_AssocRule* Algorithm

The main purpose of *Modified_Bit_AssocRule* is to reduce the number of candidate sets by using Boole's inequality in the same way as has been used in *Modified_Apriori* algorithm. The algorithm uses an additional array PA to store the probability of each attribute value. $PA[i]$ is calculated as (bitmap count of i^{th} item)/$|D|$. The algorithm is given in *Algorithm* 3.5. Candidates are generated by the function *Bit-gen*, which takes L_1, L_{k-1} as input and returns C_k as output.

```
Input: $D$, $minsup$, bitmaps.
Output: Frequent/large itemsets.

1. $L_1$= Large 1-itemsets with their bitmaps;
2. Calculate $PA[i]$(i=1,2,3...m);
3. For ($k$=2; $L_{k-1} \neq \phi$;$k$++)
4.     Remove those 1-itemsets in $L_1$ which are not included in any itemset
       of $L_{k-1}$;
5.     $C_k$= Bit-gen($L_{k-1}, L_1$);
6.     $L_k$= $\{c \in C_k |$ bitmap count of $c \geq$ minsup $\}$;
7. Endfor
8. Answer= $\bigcup L_k$;
```

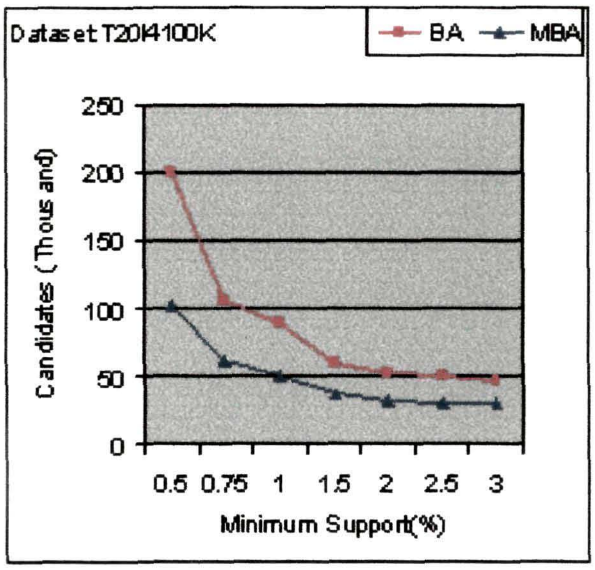
Algorithm 3.5: *Modified_Bit_AssocRule*

```
Function Bit-gen($L_{k-1}, L_1$)
  1. Select $a \in L_{k-1}$ and $b \in L_1$ such that $b$ is larger than all the elements in $a$.;
  2. $x = k * minsup - (k-1) + \frac{(k-1)(1-minsup)}{\beta}$;
  3. $Temp = 0$;
  4. for ( $i$ =1 ; $i$=$k$-1; $i$++)
  5.     $Temp = Temp + PA[a[i]]$;
  6. $Temp$=$Temp$+ $PA[b]$;
  7. Insert $a \cup b$ into $C_k$ if $Temp \geq x$;
  8. Return $C_k$;
```

Function 3.2: *Bit-gen*

Experimental Results

To evaluate the performance of *Modified_Bit_AssocRule* over *Bit_AssocRule*, experiments were carried out on the same environment as *Modified_Apriori* algorithm. Here also, same datasets (*T20.I4.D100K*, *T20.I6.D100K* and *Connect4*) were used. Other parameters were also kept same as those of *Modified_Apriori*. The experimental results are given in *Figures* 3.15 through 3.19.



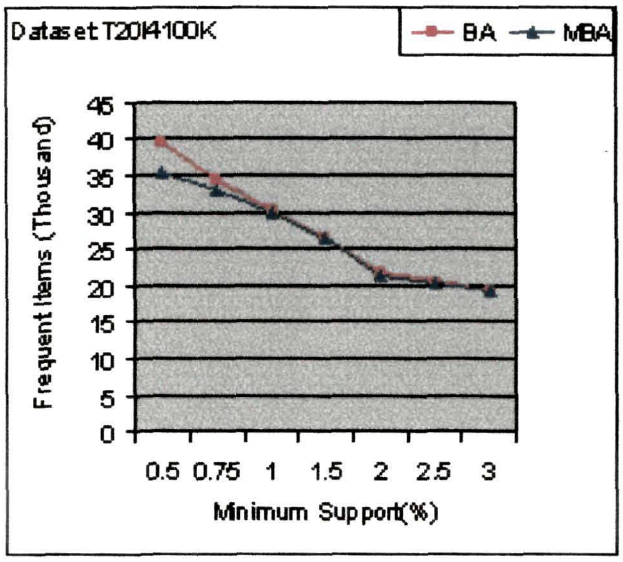
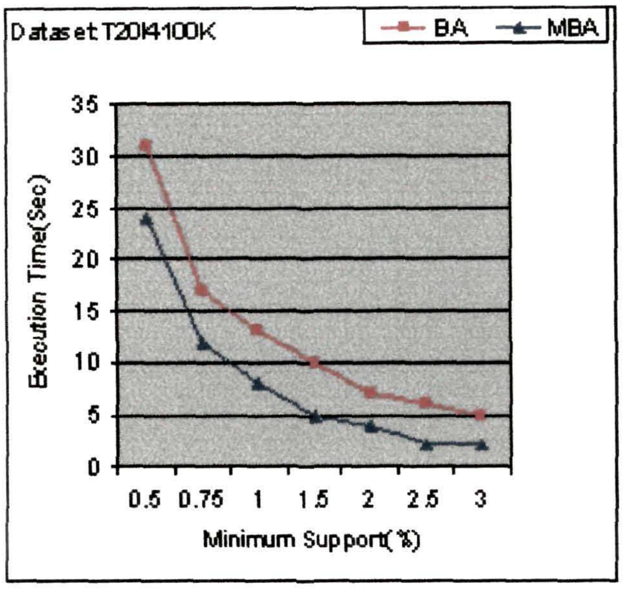


Figure 3.15: Experimental Results of *Modified_Bit_AssocRule* - I



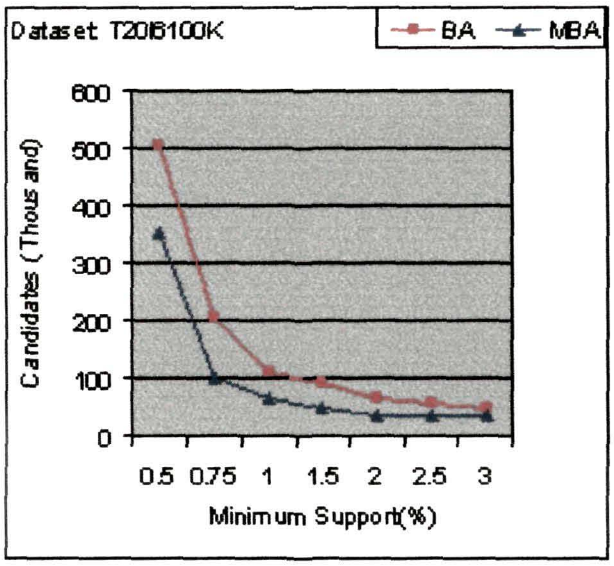
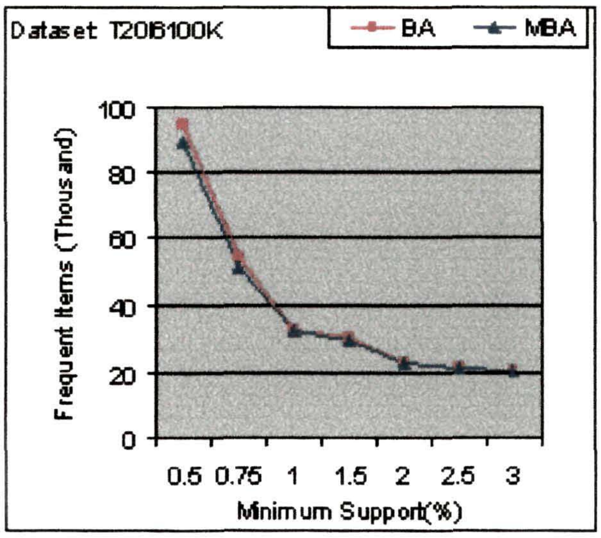


Figure 3.16: Experimental Results of *Modified_Bit_AssocRule* - II



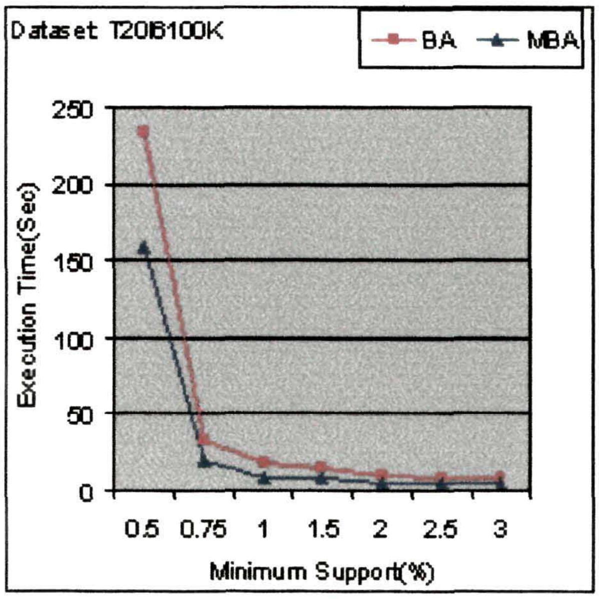
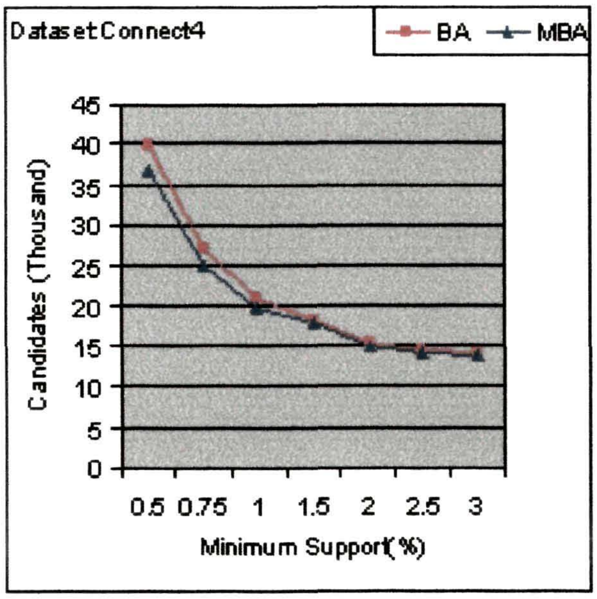


Figure 3.17: Experimental Results of *Modified_Bit_AssocRule* - III



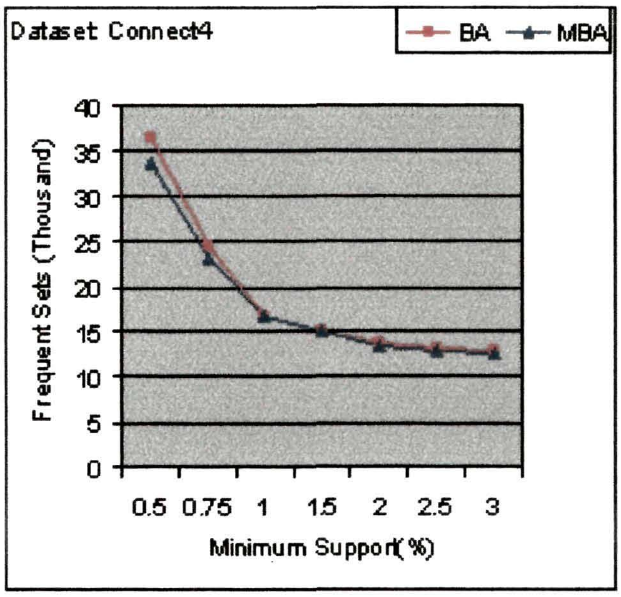


Figure 3.18: Experimental Results of *Modified_Bit_AssocRule* - IV

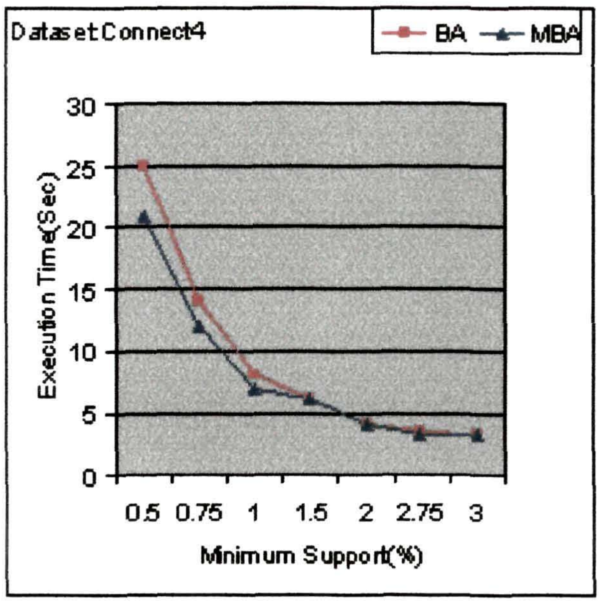


Figure 3.19: Experimental Results of *Modified_Bit_AssocRule* - V

Observations : It can be observed from the results that *Bit_AssocRule* generates large number of candidate sets, which increases the I/O overhead and execution time. On the other hand, *Modified_Bit_AssocRule* generates less candidate sets and almost same frequent sets.

3.5 Discussion

Based on the possible solutions as reported in the subsection 3.2.5, this chapter has reported two modified versions of *Apriori* algorithm for fast frequent itemset generation. The performance of the algorithm were evaluated by comparing with its other counterparts from [AMS+94] based on two synthetic data sets and one real dataset. The algorithm outperformed *Apriori, AprioriTid* and *AprioriHybrid* in terms of both execution time and memory utilization.

FP-growth [HPY00] is one of the efficient algorithms to find frequent itemsets from databases without candidate generations. However, it suffers from two major problems: the algorithm takes much time to construct the FP-tree, specially for higher dimensions and the performance of the algorithm degrades with the increase of minimum support [HPY00]. These problems can be solved to a great extent by using vertical partitioning approach, which is discussed in the next chapter.

Partitioning is another useful approach used in frequent itemset generation problem. Next chapter reports an existing partitioning approach and analyzed its performance in the light of several real and synthetic datasets. It also reports an enhanced partitioning algorithm for faster frequent itemset generation.

Chapter 4

Frequent Itemsets Using Partitioning Approach

Most of the algorithms to find frequent itemsets need multiple passes over the databases. This means that the disk-resident database has to be read several times. So, these algorithms spend a lot of time to perform disk I/O, resulting in maximum burden on the I/O subsystem. Moreover, running these algorithms on OLTP systems, where thousands of transactions are made per second, will increase the query response time to a great extent. If the algorithms are run on network, it may cause network congestion also. One feasible solution is to partition the database and apply the algorithms in the partitions individually. Another advantage of partitioning the database is that it helps parallelize the algorithms.

This chapter presents a useful frequent itemsets finding algorithm using vertical partitioning approach. It also includes an experimental analysis of the horizontal partitioning based frequent itemset finding approach. Then, it has made a comparison between both the approaches : horizontal partitioning and vertical partitioning.

4.1 *Partition* Algorithm

Partition algorithm [SON95] overcomes difficulties mentioned above to a great extent for the following reasons.

1. It reads the database only twice to generate the itemsets.
2. CPU overhead is much lower than the other algorithms.
3. The algorithm can generate results with no false negative.
4. The algorithm can be parallelized with minimal communication and synchronization among the processing nodes.

The algorithm uses the concept of partition, local support, local large itemsets, global support and global large itemsets. A partition is defined as any subset of transactions contained in the database D and partitions are non-overlapping. Local support of an itemset X in a partition is defined as the fraction of transactions in the partition, which contains the itemset. If the local support of an itemset X is at least the user-defined minimum support, the itemset is called local large itemset. Global support, global large itemsets, etc. also are defined in the same way, but in the context of global database.

Partition algorithm works in two phases. In phase I, it logically divides the database into some non-overlapping partitions. Next, each partition is considered individually and the large itemsets for each partition are generated. At the end of phase I, large itemsets found in each partition are merged to generate global candidate sets. In phase II, the actual support for these itemsets are computed by scanning the whole database and large itemsets are found.

4.1.1 The Algorithm

The algorithm is reported in *Algorithm* 4.1 on page 77. The algorithm assumes that transactions are in the form $(TID, i_j, i_k, ...)$ and items are kept in lexicographic order. TID, called transaction id, is an unique number associated with each transaction. The algorithm first partitions the database into some non-overlapping logical partitions. Then it reads one partition at a time and finds the local large itemsets of the partition using one function *Gen_large_itemsets* (*Function* 4.1 on page 78). These local large itemsets are potential global large itemsets. At the end of first phase, these local large itemsets of all the partitions are merged together to form global candidate sets. In the second phase, the algorithm scans the database once to find the actual support of global candidate sets. Global candidate sets whose support is greater than the minimum support. becomes global large itemsets. The main strength of the algorithm can be summarized as follows.

- A global large itemset must be locally large in at least one partition.
- It finds all the global large itemsets because global candidate set is the union of all local large itemsets.

4.1.2 Generation of Local Large Itemsets and Global Large Itemsets

Local large itemsets of a partition p are generated by the function *Gen_large_itemsets* (*Function* 4.1 on page 78). The function takes one partition p and returns large itemsets in the partitions. The function works in the same way as *Apriori* works. The function maintains *tidlist* of the items. *tidlist* of an itemset X is nothing, but an array of transaction ids(TID) of the transactions in the database containing X. Support count of an itemset is found by intersection of the *tidlists* of the items in the itemset. One important step in the function is the pruning step (line 8). A candidate c is pruned away if one of its subsets is not large because any superset of it also cannot be large.

Union of all local large itemsets form the global candidate sets. This global candidate set is the superset of the set of all possible global large itemsets because any global large itemset must be locally large in at least one partition. Then, the algorithm reads one partition at a time and calls the procedure *Gen_final_count*(*Procedure* 4.1 on page 79) to find support count of all the global candidate sets in the partition. Support count of all the partitions are added to find global support count of the itemsets. Itemsets with support count greater than minimum support are the global large itemsets.

```
Input : $D$, $minsup$, $np$
Output : All large itemsets $L^G$.

 1. $P = Partition\_database(D)$;
    //Phase I...................
 2. For $i$=1 to $np$ do begin
 3.       $Read\_in\_partition\ (p_i \in P)$;
 4.       $L^i = Gen\_large\_itemsets\ (p_i)$;
 5. End
    // Merge phase.............
 6. For ($i$=1; $L_i \neq \phi$ ; $j$=1,2.....$np$; $i$++) do begin
 7.       $C_i^G = \bigcup_{j=1,2.....np} L_i^j$;
 8. End
 9. $C^G = \cup C_i^G$
    ; //Phase II..................
10. For $i$=1 to $np$ do begin
11.       $Read\_in\_partition\ (p_i \in P)$;
12.             $Gen\_final\_count(C^G, p_i)$;
13. End
14. $L^G = \{c \in C^G | c.count \geq minsup\}$;
15. Answer= $L^G$;
```

Algorithm 4.1: *Partition*

```
Function Gen_large_itemsets(p : a partition)
 1. $L_1^p$=large 1-itemsets along with their tidlists;
 2. For $(k = 2; L_{k-1}^p \neq \phi; k++)$
 3.      For all itemsets $l_1 \in L_{k-1}^p$
 4.         $L_k^p = \phi$;
 5.        For all itemsets $l_2 \in L_{k-1}^p$;
 6.           If $l_1[i] = l_2[i] (i=1,2,3...k-2)$ and $l_1[k-1] < l_2[k-1]$ then
 7.              $c= l_1[1].l_1[2]...l_1[k_1].l_2[k-1]$;
 8.              If $c$ cannot be pruned then
 9.                 $c.tidlist = l_1.tidlist \cap l_2.tidlist$;
10.                 If $|c.tidlist| \geq minsup$ then
11.                    $L_k^p= L_k^p \cup \{c\}$;
12.                 Endif
13.              Endif
14.           Endif
15.        Endfor
16.     Endfor
17. Endfor
18. Return $\cup_k L_k^p$;
```

Function 4.1: *Gen_large_itemsets*

```
Procedure Gen_final_count($C^G$ : Set of all global candidate sets, $p$ : database partition)

1. For ($k$=1; $C_k^G \neq \phi$; $k$++)
2.     Forall $k$-itemset $c \in C_k^G$
3.         templist = c[1].tidlist ∪c[2].tidlist ∪c[3].tidlist...∪c[k].tidlist;
4.         c.count=c.count+|templist|;
5.     Endfor
6. Endfor
```

Procedure 4.1: *Gen_final_count*

4.1.3 An Example

Let us consider a sample database with five items(A, B, Q, R, S) and ten transactions as given in *Table* 4.1 on the next page. Two logical partitions p_1 and p_2 are created with five transactions in each partition. Partition p_1 contains first five transactions and partition p_2 contains next five transactions. Minimum support is assumed to be 40%. Now, partition p_1 is read and the function *Gen_large_itemset* is called to find the local large itemsets in p_1. Local large itemsets for p_1 is L_1=$\{A, B, Q, S, AB, AQ, AS, BQ, BS, ABQ, ABS\}$. Similarly, local large itemsets for p_2 are obtained as L_2=$\{A, B, Q, S, AB, AQ, AS, BQ, ABQ\}$. After merging. global candidate set is obtained as C^G = $\{A, B, Q, S, AB, AQ, AS, BQ, BS, ABQ, ABS\}$. Now, p_1 is read and support count of all the itemsets $c \in C^G$ in p_1 are calculated, which are 5, 5, 2, 4, 5, 2, 4, 2 , 4, 2 and 4 respectively . Similarly, p_2 is read and support count of all the itemsets $c \in C^G$ in p_2 are calculated, which are 4, 3, 3, 2, 3, 2, 2, 2, 1, 2 and 1 respectively. Global support count of the itemsets $c \in C^G$ are 9, 8, 5, 6, 8, 4, 6, 4, 5 ,4 and 5 respectively. The itemsets $c \in C^G$ with support count greater than minimum support count(i.e. 4 transactions) constitute L^G, set of global large itemsets. So, L^G = $\{A, B, Q, S, AB, AQ, AS, BQ, BS, ABQ, ABS\}$.

A	B	Q	R	S
1	1	0	0	1
1	1	1	0	1
1	1	0	0	1
1	1	1	0	0
1	1	0	0	1
1	1	1	0	0
1	1	1	0	0
1	1	0	0	1
1	0	0	0	1
0	0	1	0	0

Table 4.1: A Sample Database - III

4.1.4 Experimental Results

Experiments were carried out to show the execution time of *Partition* algorithm . All the experiments were carried out using *Pentium IV* machine with 256 MB RAM and 40GB disk space. Minimum support was taken as 1% for all the experiments.

Data set : Two synthetic data sets and one real data set were used for the experiments. Synthetic data were generated based on the method given in [AMS$^+$96] and the real database (*Connect-4*) was downloaded from UCI machine learning repository (http://www.ics.uci.edu). The *Table* 4.2 on the following page shows the parameters used in the synthetic data generation, where $|D|, |T|$ and $[ML]$ denote the number of transactions. the average size of transactions and the mean size of a potentially large itemset. For synthetic datasets, number of items, potential large itemsets and number of records were taken as 500, 500 and 100K respectively. *Connect-4* database contains around 65K instances with 43 attributes. Each attribute can have one of three values. Here also, each distinct value of the attributes was considered as one item, resulting in total 129 items.

Experiments were carried out to find execution time for different number of partitions. In case of synthetic data sets, number of partitions was increased from 1 to 15 and in case of *Connect 4* number of partitions was increased from

1 to 10. Experimental results are given in *Figures* 4.1 and 4.2 on the following page.

Data Set	$\|T\|$	$\|ML\|$	$\|D\|$
T20I4100K	20	4	100K
T20I6100K	20	6	100K

Table 4.2: Parameters for Synthetic Databases - III

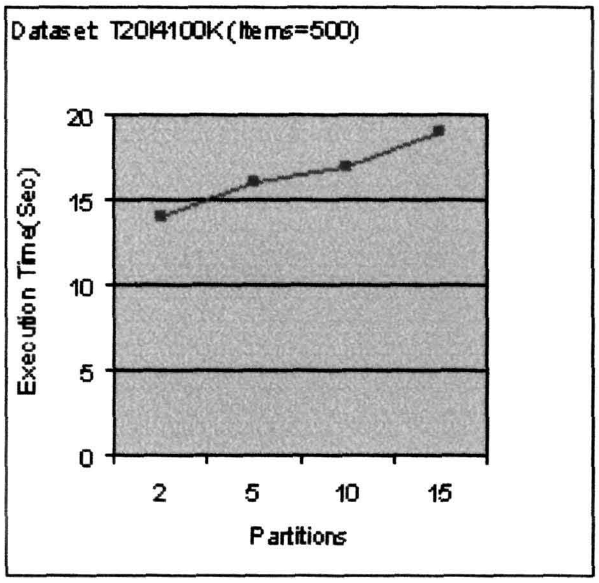


Figure 4.1: Experimental Results of *Horizontal Partition* - I

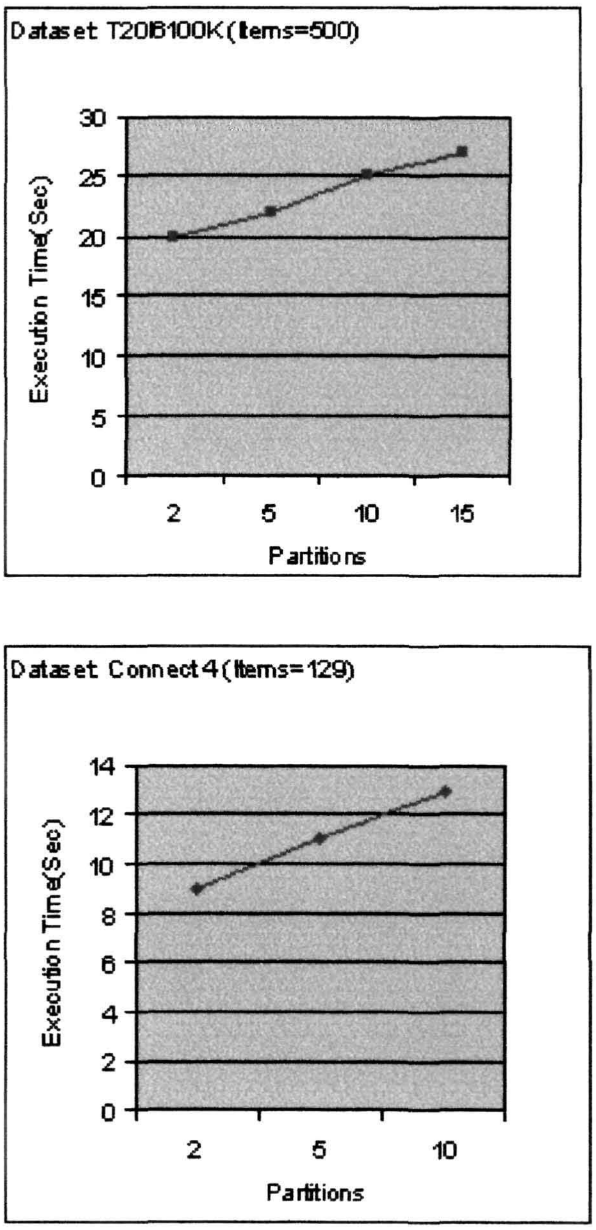


Figure 4.2: Experimental Results of *Horizontal Partition* - II

4.1.5 Discussion

The partition algorithm, which has been discussed above, is basically a horizontal partition algorithm because the database is partitioned horizontally. The algorithm is robust enough to find all the large itemsets in the whole database. It works much faster than *Apriori*, as has been demonstrated in [SON95] because it needs only two scans of the database. Proper choice of the ***number of partitions*** and the ***partition sizes*** are crucial for the success of the algorithm. Partition sizes are so chosen that each partition can be accommodated in the main memory and the partitions are read only once in each phase. However, the algorithm is not scalable with number of dimensions and number of partitions as has been shown by the experimental results.

4.2 *Vertical Partition* Algorithm

As mentioned above, one major factor which greatly affects the performance of the partition algorithm, is the size of the global candidate sets. As the size of the global candidate sets increases, the execution time also increases. Although there are many factors, which affect the size of the global candidate set, one of the main factors is the *dimensionality* of the records/transactions. It has been observed that with the increase in dimensionality, the size of the global candidate set as well as the execution time also increases. *Vertical Partition* algorithm, which partitions the database vertically instead of horizontally, overcomes this problem to a great extent.

4.2.1 The Algorithm

Let us consider a customer transaction database *D*, which contains transactions of items. It is assumed that each transaction/record is of the form *<TID,1,0,1,1,0...>* and the items are in lexicographic order. Here, *TID* is unique transaction identification number; 1 represents that the corresponding item is bought and 0 represents that the item is not bought in the transaction. It is also assumed that the *TID*s are kept in monotonically ascending order and the database is kept in secondary storage. Considering the assumptions, the algorithm works as follows. The algorithm works in two phases. In the first phase, the database is partitioned vertically into some logical partitions and large itemsets for each partition are determined. In the second phase, large itemsets found in each of the partitions are combined to form the global candidate sets. Then, supports of the candidates are also calculated. It should be noted that local large itemsets of a partition are also global large itemsets. The support of the global candidate sets are calculated by intersecting the *tidlists* of the items of the itemsets. Obviously, the global candidate set is the superset of the actual large itemsets. So, the algorithm is sure to find all possible large itemsets in the database. The algorithm is presented in *Algorithm* 4.2 on the next page.

4.2.2 Generating Large Itemsets in a Partition

Large itemsets in a partition are generated using the function *Gen_large_itemsets* (*Function* 4.2 on page 86). The function takes a partition as input and returns the set of large itemsets in that partition. The function works in the same way as *Apriori* works. It finds the support count of an itemset by intersection of the *tidlists* of the items in the itemset.

4.2.3 Combining Local Large Itemsets

The algorithm uses the function *Combine_local_large* (*Function* 4.3 on page 87) to find the global candidate itemsets. The function takes two sets of large itemsets; concatenates a large itemset of one set with a large itemset of another set and returns the concatenated sets. Combining the large itemsets in all the partitions at a time will generate too many candidate sets. So, they are combined incrementally. At first, sets of large itemsets of first two partitions are combined to find candidate itemsets. Then, support count of the candidates are found by intersections of *tidlists* of the items. The itemsets with support count greater than minimum support becomes actual large itemsets . Then, the set of resulting large itemsets is combined with the set of large itemsets of the next partition and the process continues so on. As for example, let the large itemsets in three partitions be { AB, A, B }; { Q, R } and { S }. At first, partitions 1 and 2 will

```
Input : $D$, $minsup$, $np$
Output : All large itemsets $L^G$.

 1. $P$= $Vertical\_partition\_database(D)$;
 2. For $i$=1 to $np$ do begin
 3.       $Read\_in\_partition(p_i \in P)$. create the tidlists for each of the items;
 4.       $L^i = Gen\_large\_itemsets(p_i \in P)$;
 5. End
 6. $L^G = L^1$;
 7. For $i = 2$ to $np$ do begin
 8.       $C^G = Combine\_local\_large(L^G, L^i)$;
 9.       For all candidate $c \in C^G$ $gen\_count(c)$;
10.       $L^G = L^G \cup L^i \cup \{c \in C^G | c.count \geq minsup\}$;
11. End
12. Answer= $L^G$;
```

Algorithm 4.2: *Vertical Partition*

```
Function Gen_large_itemsets(p : a partition)
 1. $L_1^p$ = large 1-itemsets from the tidlist;
 2. For ($k = 2; L_{k-1}^p \neq \phi; k++$)
 3.      $L_k^p = \phi$;
 4.      For all itemsets $l_1 \in L_{k-1}^p$
 5.        For all itemsets $l_2 \in L_{k-1}^p$
 6.           If $l_1[i] = l_2[i]$($i$=1,2,3...$k$-2) and $l_1[k-1] < l_2[k-1]$ then
 7.              $c = l_1[1].l_1[2]...l_1[k_1].l_2[k-1]$;
 8.              If c cannot be pruned then
 9.                 $c.tidlist = l_1.tidlist \cap l_2.tidlist$;
10.                 If $|c.tidlist| \geq minsup$ then
11.                    $L_k^p = L_k^p \cup \{c\}$;
12.                 Endif
13.              Endif
14.           Endif
15.        Endfor
16.     Endfor
17. Endfor
18. Return $\cup_k L_k^p$;
```

Function 4.2: *Gen_large_itemsets*

be combined to find the potential large itemsets { ABQ, ABR, AQ, AR, BQ, BR }. Suppose AQ and BQ are large among them. Then {AQ, BQ} will be combined with the partition 3 i.e. {S} to generate {AQS,BQS }.

```
Function Combine_local_large($X, Y$: Sets of large itemsets)

1. $Z=\phi$ ;
2. For all $a \in X$
3. For all $b \in Y$
4.         $c$=concatenate $a$ and $b$ ;
5.         $Z=Z \cup c$ ;
6. Endfor
7. Endfor
8. Return $Z$ ;
```

Function 4.3: *Combine_local_large*

4.2.4 Support Count

The supports of the global candidate sets are obtained by the procedure *Gen_count* (*Procedure* 4.2). The procedure takes one candidate itemset as input and calculates support count of the itemsets as output. The support count is calculated by intersecting the *tidlists* of the items.

```
Procedure Gen_count($c$ : itemset)
1. $k=|c|$;
2. $templist= c[1].tidlist \cap c[2].tidlist \cap ... \cap c[k].tidlist$;
3. c.count=|templist|;
```

Procedure 4.2: *Gen_count*

4.2.5 An Example

Let us try to understand the *Vertical Partition* algorithm with an example. Let us consider a Database D with three partitions as given in *Table* 4.3 and the minimum support be 40%(i.e. 2 transactions)

TID	Partition I			Partition -II			Partition-III		
	A	B	E	G	H	J	K	R	S
1	1	1	0	0	0	0	0	1	0
2	0	1	1	0	0	1	0	1	0
3	0	0	0	0	1	0	1	0	1
4	1	1	0	1	0	0	0	1	0
5	0	0	0	0	1	0	0	0	1

Table 4.3: A Sample Database - IV

The Frequent itemsets in the partition I: L^1= {A, B, AB}
The Frequent itemsets in the partition II : L^2= {H}
The Frequent itemset in the partition III: L^3 = {R. S}
Candidate sets generated by combining L^1 and L^2 are C= { AH, BH, ABH}.
The Frequent itemsets in C are $L' = \phi$
L' is recalculated as $L' = L' \cup L^1 \cup L^2$ = {A, B, AB, H}
Candidate sets generated by combining L' and L^3 are C = { AR, AS, BR, BS. ABR, ABS, HR. HS }
The Frequent sets in C are L'' = { AR, BR, ABR, HS }
Therefore $L^G = L'' \cup L' \cup L^3$ = {A, B, AB, H, R, S, AR, BR, ABR, HS}.

4.2.6 Discussion

The *Vertical Partition* algorithm finds all possible large itemsets and needs one scan of the database. Another advantage is that it generates less number of global candidate sets because of the fact that an itemset, which is large in a vertical partition, is also large in the whole database. So, it is not required to test the itemset again for whole database. However, the algorithm requires to maintain the *tidlist* of the items. If they cannot be kept in memory, they can be kept in secondary storage and accessed whenever required. As in the *Horizontal Partition* algorithm, determining number of partitions poses a major consideration. Generally, the size of partition should be chosen in such a way that it fits in the main memory. However, other factors such as the type of database. minimum support, etc. should also be taken care of accordingly.

4.2.7 Experimental Results

Experiments were carried out to show comparison of execution time between *Horizontal Partition*(HP) algorithm and *Vertical Partition*(VP) algorithm for different number of partitions and dimensions. *Horizontal Partition* algorithm was implemented using the same approach as it was discussed in [SON95] and *Vertical Partition* was implemented using the same approach as was discussed above. All the experiments were carried out using *Pentium IV* machine with 256 MB RAM and 40GB disk space.

Data set : Two synthetic databases and one real database were used for the experiments. Synthetic databases were generated based on the method given in [AMS+96] and the real database(*Connect-4*) was downloaded from UCI machine learning repository(http://www.ics.uci.edu). The *Table* 4.4 on the following page shows the parameters used in the synthetic data generation, where $|D|$, $|T|$ and $[ML]$ denote the number of transactions, the average size of transactions and the mean size of a potentially large itemset respectively. For all datasets, number of items, potential large itemsets and number of records were taken as 500, 500 and 100K respectively. *Connect-4* database contains around 65K instances with 43 attributes. Each attribute can have one of three values. Here also, each distinct value of the attributes has been considered as one item, resulting in total 129 items. For all the experiments minimum support was taken as 1%.

Experiments were carried out to compare execution times for different number of items and different number of partitions. For synthetic data sets, numbers of items were 100, 300 and 500; for *Connect 4* numbers of items were 50. 100 and 129. In case of synthetic data sets, number of partitions was increased from 1 to 15 and in case of *Connect 4* number of partitions was increased from 1 to 10. Experimental results are given in *Figures* 4.3 through 4.7.

Data Set	$\|T\|$	$\|ML\|$	$\|D\|$
T20I4100K	20	4	100K
T20I6100K	20	6	100K

Table 4.4: Parameters for Synthetic Databases - IV

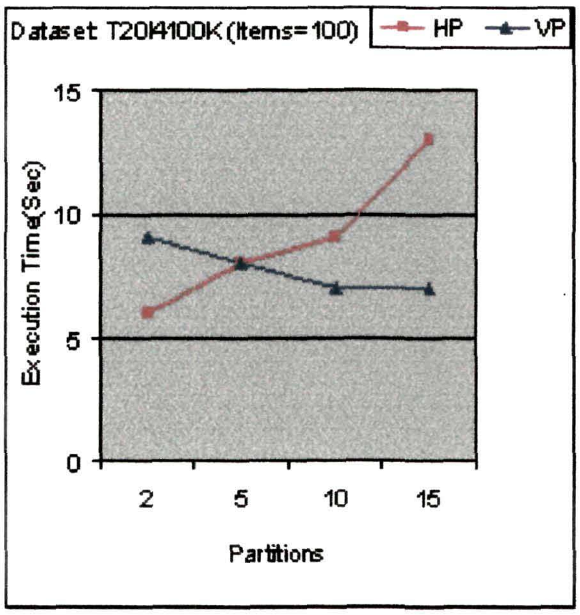
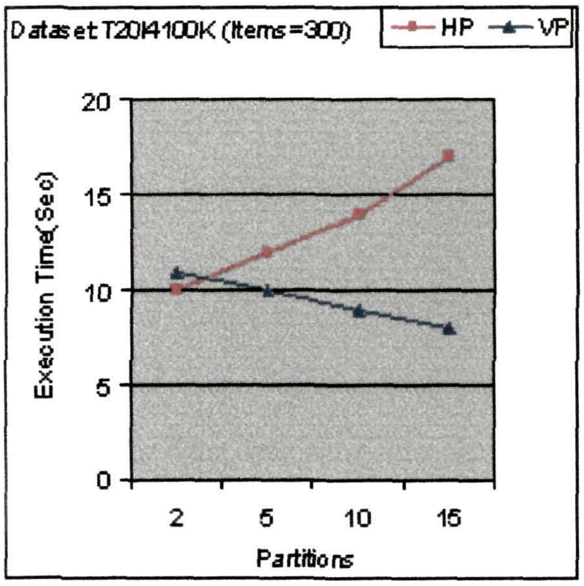


Figure 4.3: Experimental Results of *Horizontal Partition* and *Vertical Partition* - I



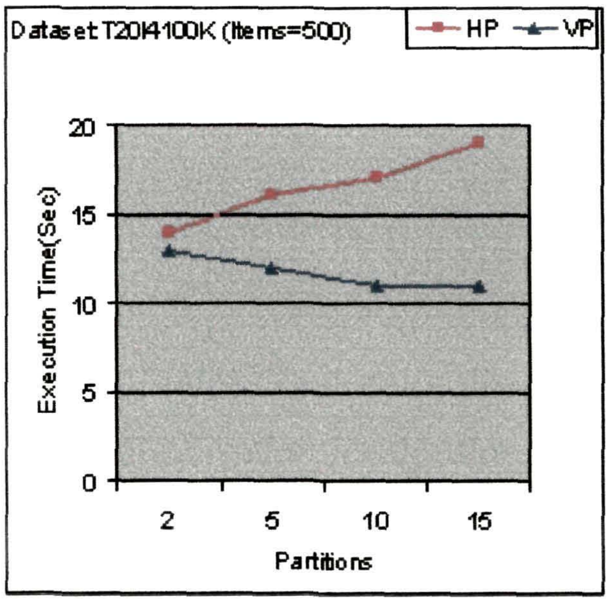
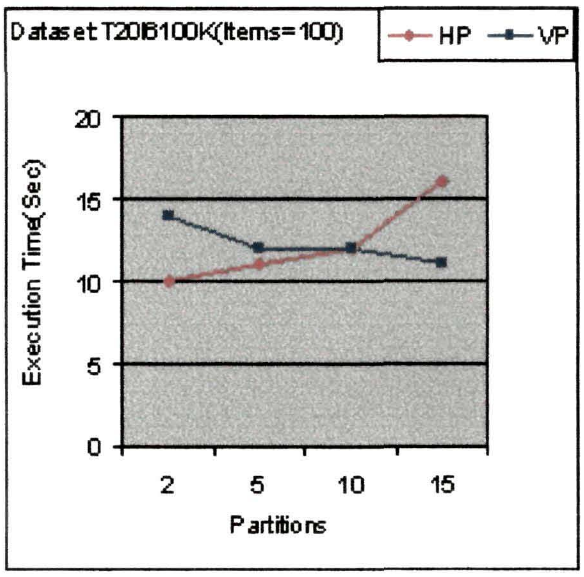


Figure 4.4: Experimental Results of *Horizontal Partition* and *Vertical Partition* - II



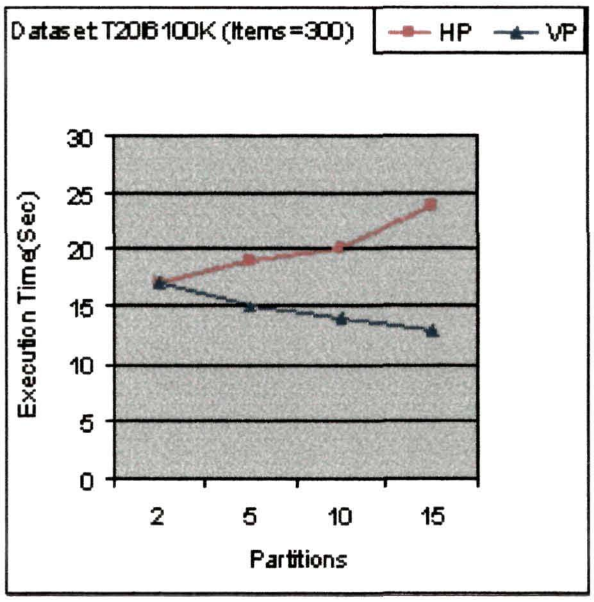
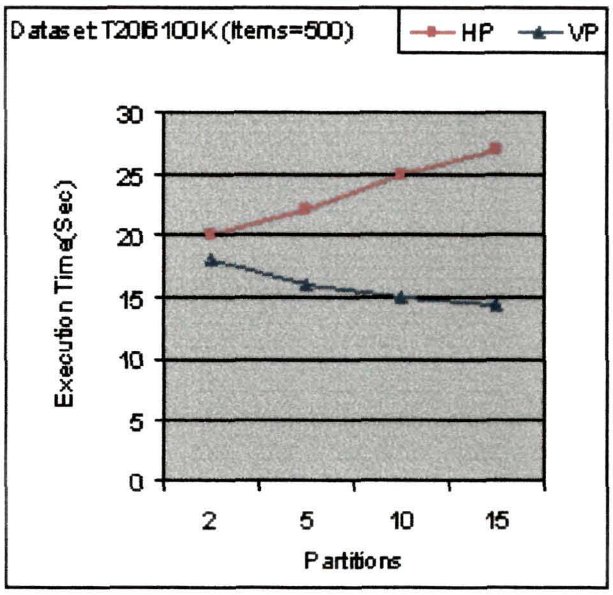


Figure 4.5: Experimental Results of *Horizontal Partition* and *Vertical Partition* - III



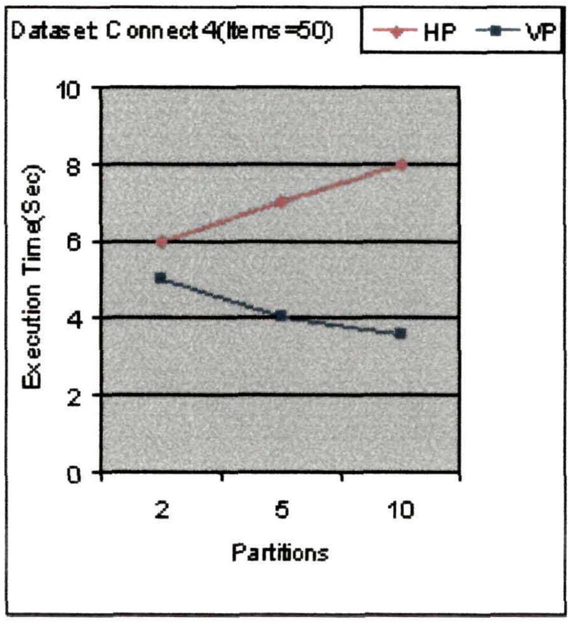
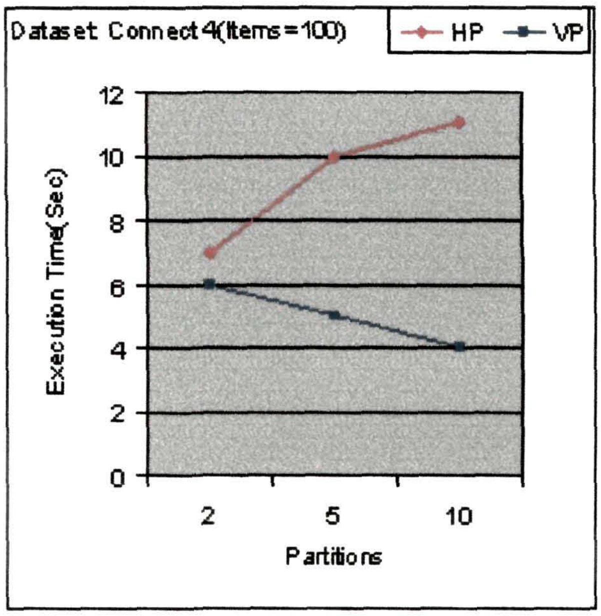


Figure 4.6: Experimental Results of *Horizontal Partition* and *Vertical Partition* - IV



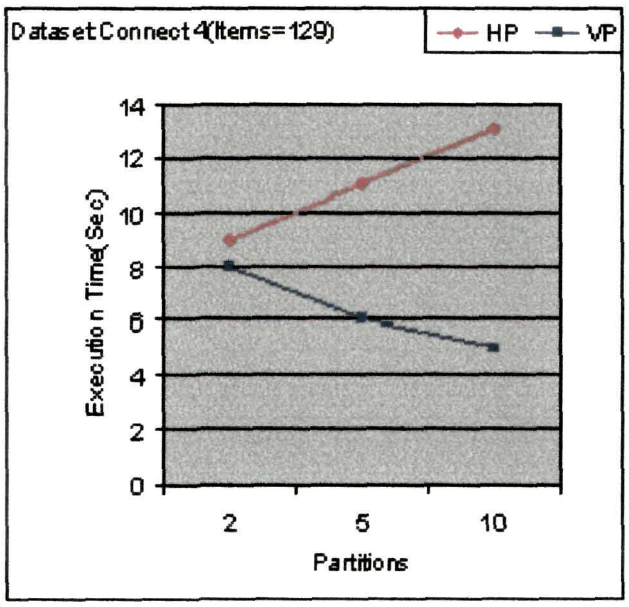


Figure 4.7: Experimental Results of *Horizontal Partition* and *Vertical Partition* - V

4.3 *FP-growth* and Vertical Partition

FP-growth algorithm [HPY00] finds frequent itemsets without candidate generation. The algorithm is based on a special data structure called FP-tree, which is basically a prefix tree of the transactions of the database such that each path represents a set of transactions that share the same prefix. The algorithm works as follows. The algorithm first scans the database once to find the frequent items in the database. Infrequent items are removed from the database and items in the transactions are rearranged in the descending order of the frequencies of items. Then, all the transactions containing the least frequent item are selected and the item is removed from the transactions, resulting a reduced (projected) database. This projected database is processed to find frequent itemsets. Obviously, the removed item will be a prefix of all the frequent itemsets. Then the item is removed from the database and the above process is repeated with the next least frequent item. It is to be noted that FP-tree contains all the information about the transactions and the frequent itemsets. So, to find any information about the transactions and frequent itemsets, just the tree is to be searched.

FP-growth algorithm is one of the efficient algorithms to find frequent itemsets from large databases. However, it suffers from following two problems.

- The algorithm takes much time to construct the FP-tree, specially for higher dimensions.
- The performance of the algorithm degrades with the increase of minimum support [HPY00].

Vertical partitioning approach helps solve these problems to a great extent by using *FP-growth* in each partition to find local frequent itemsets. The steps can be summarized as follows.

```
1. Partition the database into some vertical partitions;
2. For each partition do
3.     Use FP-growth to find frequent itemsets;
```

4. Generate global candidate sets from frequent itemsets;
5. Find support of the global candidate sets;
6. Return global frequent itemsets.

4.3.1 Experimental Results

Experiments were carried out to evaluate the performance of original *FP-growth* and, that of *FP-growth* with vertical partitions. The *FP-growth* program was downloaded from http://fuzzy.cs.uni-magdeburg.de/ borgelt/software.html. Experiments were carried out on a Pentium IV PC with 256 MB RAM and two synthetic databases (T20.I4.D100K and T20.I6.D100K) were taken for the experiments. The synthetic data were generated using the technique given in [AMS+96] with the values of parameters as given in *Table* 4.5 and with dimensionality 500. Here, $|T|$, $|ML|$ and $|D|$ represent the average size of a transaction, mean size of a potentially large itemset and database size (number of transactions) respectively. Each of the datasets contained 100K records. For *FP-growth* with vertical partitions, each database was divided into five logical vertical partitions with 100 items in each partitions. Experimental results are given in the *Figures* 4.8 on the following page & 4.9 on the next page.

Name	$\|T\|$	$\|ML\|$	$\|D\|$
T20.I4.D100K	20	4	100K
T20.I6.D100K	20	6	100K

Table 4.5: Parameters for Synthetic Databases - V

4.4 Discussion

This chapter presents a novel association rule mining algorithm based on vertical partitioning. It also reports a comparative study of the proposed algorithm with the *Partition* algorithm [SON95].

It has been observed from the experimental results that

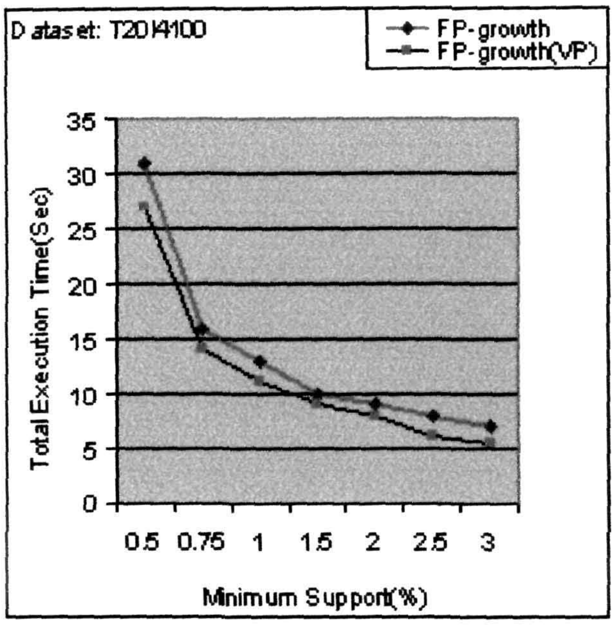


Figure 4.8: Experimental Results of *FP-growth*(Database: T20I4D100K)

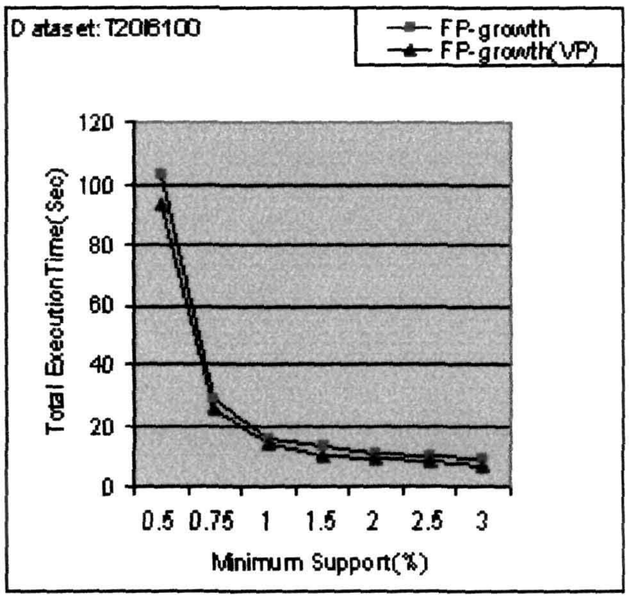


Figure 4.9: Experimental Results of *FP-growth*(Database: T20I6D100K)

- In both the algorithms, the execution time increases with the increase in number of items. It can be seen from the *Figures* that *Horizontal Partitions* (HP) performs better when number of items is small. However, with the increase in number of items, the performance of the *Vertical Partitions*(VP) improves over *Horizontal Partitions*.
- In case of *Horizontal Partitions* , the execution time increases with the increase of number of partitions. Whereas, in case of *Vertical Partitions* . the execution time decreases with the increase of number of partitions and becomes almost constant at the end.

Chapter 5

Frequent Itemsets in Dynamic Databases

Most of the databases are dynamic and are updated frequently i.e. new records are added , old records are deleted and existing records are modified very frequently. So, the itemsets which are frequent (or large) may not be frequent when the database is updated and the itemsets which are not frequent may become frequent when the database is updated. So, some algorithms are required to update the set of frequent itemsets when the database is updated. Moreover. new database may contain some new interesting rules which were not present in the old database. One obvious technique is re-running association rule mining algorithms in the updated database to find the frequent itemsets in the updated database. However, this is not optimal solution because of the following reasons.

1. It will require to run the algorithms on adding, deleting or updating a small number of records.
2. It will take too much time because it will scan the same database every time.
3. It will generate most of the itemsets repeatedly .

Some of the popular algorithms to find frequent itemsets in dynamic databases are *FUP* [CHN+96b], *FUP2* [CLK97], *DELI* [LCK98] and *MAAP* [ES02]. Some more algorithms can be found in [TBA+97, FAA+97]. The main requirements of a dynamic association rule mining algorithm are:

1. It should be able to use the already discovered frequent itemsets to discover new frequent itemsets.

2. It should not have to scan the old records/transactions.

3. It should scan the new records/transactions as few number of times as possible.

The most popular and important algorithm, which follows the above criteria, to find frequent itemsets in dynamic database is the *Borders* algorithm [FAL+99, Puj01]. This algorithm has used the concept of *border* and *promoted border* set to update the frequent itemsets. However, the algorithm suffers from scalability problem and cannot be used in distributed environment.

As mentioned above, *Borders* algorithm suffers from scalability problem and cannot be used in distributed environment directly. To address these problems, this chapter presents one modified version of *Borders* algorithm, which takes less time than that of *Borders* algorithm. This chapter also presents *Distributed_Borders* algorithm, which is meant for distributed dynamic databases.

5.1 *Borders* Algorithm

As mentioned above, it is not efficient to rerun frequent itemsets finding algorithms reported so far to find frequent itemsets in dynamic/incremental databases. So, it is desirable to have incremental algorithms, which can generate frequent itemsets in incremental manner by processing only the new transactions/records. *Borders* is one such algorithm, which satisfies this requirement by using the concept of *border set* and *promoted border set*.

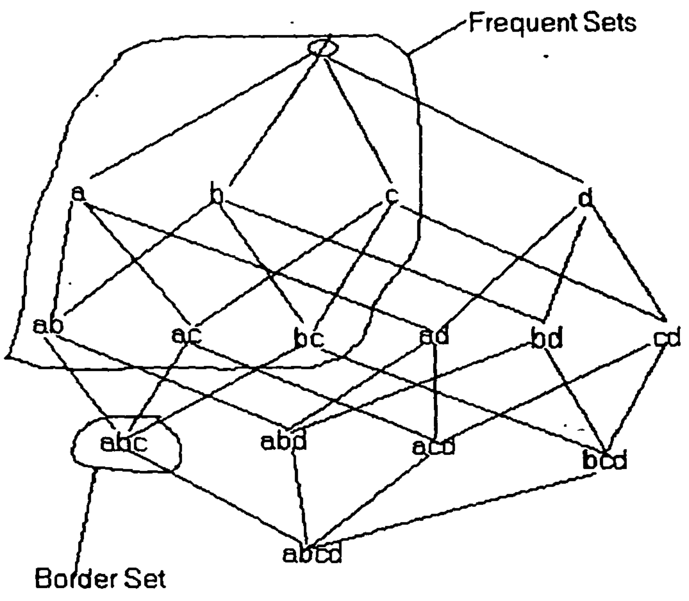


Figure 5.1: Border Set

Initially the concept of border set was given by Manila and Toivonenn [Man97]. An itemset X is called a *border* set if X is not frequent, but all its proper subsets are frequent. Collection of *border* sets forms the border line between the frequent sets and non-frequent sets. An itemset that was a border set before the database was updated and has become frequent after the database has been updated is called a *promoted border* set.

Borders algorithm uses the same concept of *border* and *promoted border*, and maintains support counts for all the frequent sets as well as for all the border sets. There are three variations of *Borders* algorithm : addition of transactions/records. addition and deletion of transactions/records and changing of minimum support threshold. The main characteristics of the algorithm are as follows.

1. Entire database is scanned only when new candidate sets are generated.

2. There are only few candidates for which support is counted even if entire database is scanned.

The algorithm is based on the observation that a set is required to be considered as a candidate set only if it has a subset that is a promoted border. The following lemma proves this observation. In addition to *Lemma* 5.1, the proof of correctness of *Borders* algorithm also can be found in [FAL+99].

Lemma 5.1

If X is an itemset which is frequent in T_{whole} and not frequent in T_{old}, then there exists a subset $Y \subseteq X$ such that Y is promoted border.

Proof: Let Y be a minimal cardinality subset of X which is frequent in T_{whole}. but not in T_{old}. So, all proper subsets of Y are frequent in T_{whole}. However, by minimality of Y. none of these subsets is a new frequent set in T_{whole}. So, Y is a promoted border. □

Given L_{old} and B_{old}, the task of the *Borders* algorithm is to find L_{whole} and B_{whole}. The algorithm(addition) is presented in *Algorithm* 5.1 on the next page. The algorithm assumes that L_{old} and B_{old} are known with their respective supports. L_{old} and B_{old} can be found by any association rule mining algorithms like *Apriori*. etc. The algorithm starts by making one pass over the new database T_{new} and counts the supports of the itemsets of $L_{old} \cup B_{old}$ for T_{new} . During this pass the algorithm calculates PB and L_{whole}. If PB is null then L_{whole} contains all the frequent sets of T_{whole}. However, if there is at least one promoted border (i.e. PB is not null), then the algorithm generates new candidate sets(steps 8-10) and scans over the entire database to count the support of them (steps 11-13). Then it finds the large itemsets L_{i+1} based on the support count and updates L_{whole} and B_{whole}. This process continues as long as new candidates can be generated. So, the algorithm scans the whole database, if there is some promoted border set. Otherwise, it does not require to scan the whole database.

```
Input : $T_{new}$, $T_{old}$, $minsup$, $L_{old}$ and $B_{old}$
Output : $L_{whole}$ and $B_{whole}$
 1. Scan $T_{new}$ and increment the support count of $X \in (L_{old} \cup B_{old})$;
 2. $PB := \{X | X \in B_{old}$ and $Sup(X)_{T_{whole}} \geq minsup\}$;
 3. $L_{whole} := PB \cup \{X | X \in L_{old}$ and $Sup(X)_{T_{whole}} \geq minsup\}$;
 4. $B_{whole} = \{X | \forall x \in X, X - \{x\} \in L_{whole}\}$;
 5. $m = \max\{i | PB(i) \neq \phi\}$;
    Steps for Candidate-generation:
 6. $L_0 = \phi$, $i=1$;
 7. While ($L_i \neq \phi$ or $i \leq m$) do
 8.     $C_{i+1} = \{X = S_1 \cup S_2 |$ (i) $|X| = i+1$,
 9.                        (ii) $\exists x \in S_1, S_1 - \{x\} \in PB(i) \cup L_i$,
10.                        (iii) $\forall x \in S_2, S_2 - \{x\} \in L_{whole}(i) \cup L_i\}$
11.     Scan $T_{whole}$ and obtain $Sup(X)_{T_{whole}}$ for all $X \in C_{i+1}$;
12.     $L_{i+1} = \{X | X \in C_{i+1}$ and $Sup(X)_{T_{whole}} \geq minsup\}$;
13.     $L_{whole} = L_{whole} \cup L_{i+1}$;
14.     $B_{whole} = B_{whole} \cup (C_{i+1} - L_{i+1})$;
15.     $i=i+1$;
16. Enddo
```

Algorithm 5.1: *Borders* (Addition)

5.1.1 An Example

Let us take T_{old} and T_{new} as given in *Table* 5.1 and *Table* 5.2 respectively and the minimum support be 40%.

A1	A2	A3	A4	A5
1	0	1	0	1
1	0	0	0	1
1	0	1	0	1
0	1	0	0	0
0	1	0	1	0

Table 5.1: T_{old} - I

A1	A2	A3	A4	A5
0	0	1	1	1
0	0	0	1	0
1	0	0	0	1
0	0	0	0	0
0	1	0	1	0

Table 5.2: T_{new} - I

Then L_{old} is {(A1,A2,A3,A5),(A1A3).(A1A5),(A3A5),(A1A3A5) }
B_{old} is {(A4),(A1A2).(A2A3),(A2A5) }
Next, T_{new} is added with T_{old}. Now, T_{new} is scanned, and PB, L_{whole} and B_{whole} are calculated as follows.
PB = {(A4)}
L_{whole}= {(A4),(A1),(A5),(A1A5)}
B_{whole} = $\{(A1A4), (A4A5), (A1A5)\}$

Here, only one promoted border $\{(A4)\}$ has been found. So, new candidate itemsets are to be generated. New Candidates are generated in level-wise fashion. Candidate 2-itemsets are C_2= {(A1A4), (A4A5) }
Now, T_{whole} is to be scanned to update L_{whole} and B_{whole}.

5.1.2 Deletion

In some situations, records/transactions may have to be deleted from the database. So, it will affect the already discovered frequent itemsets. Deletion will have following effects.

1. Some frequent sets may not be frequent any more.
2. Some new frequent sets may emerge because absolute frequency count will decrease.

Borders algorithm has been modified to handle the deletion of records. The algorithm is given in *Algorithm* 5.2 on the following page.

5.1.3 Changing of Threshold

In some situations, users may have to change the threshold of minimum support. If the new minimum support is greater than the previous minimum support, then new frequent sets and border sets can be found easily by excluding the frequent sets with minimum support less than the new minimum support. However. if the new minimum support is less than the previous minimum support, then new frequent sets may emerge. *Borders* handles this as a case of deletion.

5.1.4 Discussion

The *Borders* algorithm is robust enough to find the frequent itemsets in a dynamic database. However. from the experimental study it has been observed that

- with the increase in the volume of T_{old} and T_{new}, the cost of scanning of T_{whole} in the every iteration becomes too expensive.
- it suffers from scalability problem.

```
Input : $T_{new}$, $T_{old}$, $T_{del}$, $minsup$, $L_{old}$ and $B_{old}$
Output : $L_{whole}$ and $B_{whole}$

 1. Scan $T_{new}$ and update the support count of $X \in L_{old} \cup B_{old}$;
 2. Scan $T_{del}$ and update the support count of $X \in L_{old} \cup B_{old}$;
 3. $PB := \{X | X \in B_{old}$ and $Sup(X)_{T_{whole}} \geq minsup\}$;
 4. $L_{whole} := PB \cup \{X | X \in L_{old}$ and $Sup(X)_{T_{whole}} \geq minsup\}$;
 5. $B_{whole} = \{X | \forall x \in X, X - \{x\} \in L_{whole}\}$;
 6. $m = \max\{i | PB(i) \neq \phi\}$;
    *Steps for Candidate-generation:*
 7. $L_0 = \phi$, $i$=1;
 8. While ($L_i \neq \phi$ or $i \leq m$) do
 9.     $C_{i+1} = \{X = S_1 \cup S_2 |$ (i) $|X| = i + 1$,
10.                         (ii) $\exists x \in S_1, S_1 - \{x\} \in PB(i) \cup L_i$,
11.                         (iii) $\forall x \in S_2, S_2 - \{x\} \in L_{whole}(i) \cup L_i\}$;
12.     Scan $T_{whole}$ and obtain $Sup(X)_{T_{whole}}$ for all $X \in C_{i+1}$;
13.     $L_{i+1} = \{X | X \in C_{i+1}$ and $Sup(X)_{T_{whole}} \geq minsup\}$;
14.     $L_{whole} = L_{whole} \cup L_{i+1}$;
15.     $B_{whole} = B_{whole} \cup (C_{i+1} - L_{i+1})$;
16.     $i = i + 1$;
17. Enddo
```

Algorithm 5.2: *Borders* (Addition and Deletion)

Some possible solutions to these problems are

- Some other data structure like FP-tree can be used to find the frequent itemsets faster.

- Probability can be used to predict whether itemsets will be frequent on adding new transactions. The itemsets whose probabilities are very less, can be removed from border sets and itemsets with higher probability can be retained.
- New candidate itemsets C_{i+1} can be generated from C_i instead of from L_i because $|C_i|$ is generally very small. After the candidates are generated, T_{whole} can be scanned to find the large and border sets. So, maximum of one scan of entire database will be required. However, if C_i is large, then this method will not give optimum result.
- New candidate itemsets C_{i+1} can be generated from C_i instead of from L_i as long as $| \cup C_i|$ is less than some threshold value. When $| \cup C_i|$ becomes greater than the threshold value, T_{whole} can be scanned to find frequent and border sets. Then, the next iteration begins.
- More than one border sets can be used.
- Distributed approach also can be used to improve the scalability performance of the algorithm. In case of distributed approach, data will be processed in different sites/nodes resulting in improvement in execution time.

This chapter proposes two enhanced versions of the present *Borders* (addition) algorithm: *Modified_Borders* and *Distributed_Borders*. *Modified_Borders* has used two border sets to reduce scanning of entire database. On the other hand, *Distributed_Borders* is the extension of *Borders* in distributed environment.

5.2 *Modified_Borders* Algorithm

As it is mentioned above, *Borders* has to scan entire database when there are some promoted borders. However, scanning entire database is very expensive, particularly, when the database is very large. Scanning entire database can be avoided if new candidates are not generated frequently. New candidates are generated, if there is even one promoted borders because borders are not included to generate candidate in the old database. So, if borders are included to generate candidates in the old database, then there will be no new candidates. However, it will be infeasible to include all the borders to generate candidates in the old database. This concept led us to include some of the borders, which are likely to become promoted border, to generate candidate sets in the old database so that if those borders become promoted, no new candidates will be generated. New candidates will be generated only when some borders, which were not included to generate candidates in the old database, become promoted.

Based on the above discussion, *Borders* has been modified by including two border sets. The first border set is B'_{old} and the second border set is B''_{old}. B'_{old} is calculated as $\{X|\forall x \in X, X - \{x\} \in L_{old} \cup B'_{old}; Sup(X)_{T_{old}} \geq \beta'$ and $Sup(X)_{T_{old}} < minsup\}$. B''_{old} is calculated as $\{X|\forall x \in X, X - \{x\} \in L_{old} \cup B'_{old}; Sup(X)_{T_{old}} < \beta'\}$. B'_{old} and L_{old} take part in candidate generation, whereas the elements of B''_{old} are not used in candidate generation. Another requirement in the algorithm is that all the subsets of an itemset $X \in B'_{old} \cup B''_{old}$ must be $\in L_{old} \cup B'_{old}$. Obviously, B'_{old} contains the itemsets with higher probability of becoming promoted when new transactions are added. New candidate sets will be generated only when any of the elements of the B''_{old} becomes promoted. If new candidate itemsets are generated, one scan over the whole database is required to find supports of the new candidate itemsets.

5.2.1 The Algorithm

The algorithm works as follows. L_{old}, B'_{old} and B''_{old} are assumed to be known with their respective support counts. The algorithm starts by making one pass over the new database T_{new} and updates supports of the elements of $L_{old} \cup B'_{old} \cup B''_{old}$. During the pass, the algorithm generates four categories of itemsets - PB', PB'', B''' and B''''. If PB'' is null, then no new candidate set is required to be generated. If PB'' contains at least one itemset, then new candidate sets are required to be generated. If new candidate sets are generated, the algorithm makes one pass over the entire database to count the support of the new candidate sets. At the end, the algorithm generates L_{whole} , B'_{whole} and B''_{whole}, which are

```
Input: $T_{new}$, $T_{old}$, $minsup$, $\beta'$, $L_{old}$ and $B'_{old}$, $B''_{old}$ Output. $L_{whole}$ and $B'_{whole}$, $B''_{whole}$

1.  Scan $T_{new}$ and increment the support count of $X \in L_{old} \cup B'_{old} \cup B''_{old}$;
2   $PB' = \{X|X \in B'_{old}$ and $Sup(X)_{T_{whole}} \geq minsup\}$; $PB'' = \{X|X \in B''_{old}$ and $Sup(X)_{T_{whole}} \geq minsup\}$,
3   $L_{whole} = PB' \cup PB'' \cup \{X|X \in L_{old}$ and $Sup(X)_{T_{whole}} \geq minsup\}$;
4.  $B''' = \{X|X \in B''_{old}, \forall x \in X, X - \{x\} \in L_{whole}, Sup(X)_{T_{whole}} \geq \beta'$ and $Sup(X)_{T_{whole}} < minsup\}$;
5.  $B'''' = \{X|X \in B'_{old} \cup L_{old}, \forall x \in X, X - \{x\} \in L_{whole}, Sup(X)_{T_{whole}} \geq \beta', Sup(X)_{T_{whole}} < minsup\}$;
6.  $B'_{whole} = B''' \cup B''''$; $B''_{whole} = \{X|\forall x \in X, X - \{x\} \in L_{whole}, Sup(X)_{T_{whole}} < \beta'\}$;
7.  If $PB'' \neq \phi$ then
8.  $m = \max\{i|PB''(i) \neq \phi\}$;
    Steps for Candidate-generation:
9.  $L_0 = \phi$, $B_0 = \phi$, $k=2$,
10  While ($L_{k-1} \neq \phi$ or $B_{k-1} \neq \phi$ or $k \leq m+1$) do
11.         $C_k = \phi$;
12.         $L = PB''(k-1) \cup L_{k-1} \cup B'''(k-1) \cup B_{k-1}$; $M = L_{k-1} \cup L_{whole}(k-1) \cup B'_{whole}(k-1)$ ;
13.         For all itemsets in $l_1 \in L$ do begin
14.         For all itemsets in $l_2 \in M$ do begin
15.                 If $l_1[i] = l_2[i]$ $(1 \leq i \leq k-2)$ and $l_1[k-1] < l_2[k-1]$ then
16.                         $C = \{l_1[1], l_1[2], \cdots l_1[k-2], l_1[k-1], l_2[k-1]\}$;
17.                         $C_k = C_k \cup C$;
18          End for
19.         End for
20.         Prune $C_k$ : All the subsets of $C_k$ of size $k$-1 must be present in $M$ ;
21.         Scan $T_{whole}$ and obtain support $Sup(X)_{T_{whole}}$ for all $X \in C_k$;
22.         $L_k = \{X|X \in C_k$ and $Sup(X)_{T_{whole}} \geq minsup\}$;
23.         $L_{whole} = L_{whole} \cup L_k$;
24.         $B_k = \{X|X \in (C_k - L_k), \forall x \in X, X - \{x\} \in L_{whole}, Sup(X)_{T_{whole}} \geq \beta'$ and $Sup(X)_{T_{whole}} < minsup\}$;
25.         $B'_{whole} = B'_{whole} \cup B_k$;
26.         $B''_{whole} = B''_{whole} \cup \{X|X \in (C_k - L_k), \forall x \in X, X - \{x\} \in L_{whole}, Sup(X)_{T_{whole}} < \beta'\}$;
27.         $k = k + 1$ ;
28. Enddo
29. endif
```

Algorithm 5.3: *Modified_Borders*

counterparts of the L_{old} , B'_{old} and B''_{old} respectively, for the whole database T_{whole}. The algorithm'is presented in the *Algorithm* 5.3 on the previous page.

5.2.2 An Example

Let us consider T_{old} as given in *Table* 5.3 and assume $minsup = 40\%$ & $\beta' = 30\%$;

A_1	A_2	A_3	A_4	A_5
1	1	0	0	1
0	0	1	0	0
0	0	0	1	0
0	0	0	1	0
0	1	0	1	0
1	1	1	0	0
1	0	0	0	0
0	0	0	0	0
1	1	0	1	0
1	1	1	1	1

Table 5.3: T_{old} - II

Now, L_{old} , B'_{old}, B''_{old} are obtained as given in *Table* 5.4.

L_{old}	=	$\{(A_1, A_2, A_4), (A_1A_2)\}$
B'_{old}	=	$\{(A_3)\}$
B''_{old}	=	$\{(A_5), (A_1A_4, A_2A_4, A_1A_3, A_2A_3, A_3A_4)\}$

Table 5.4: Results - I

Now, suppose T_{new}(*Table* 5.5 on the following page) added to the T_{old}(*Table* 5.3).

A_1	A_2	A_3	A_4	A_5
0	0	1	1	1
1	1	0	0	1
1	1	1	0	1
0	0	1	0	1

Table 5.5: T_{new} - II

When T_{new} is scanned , PB' ,PB'', B''' and B'''', L_{whole},B'_{whole} and B''_{whole} are obtained as given in *Table* 5.6.

PB'	=	$\{A_3\}$
PB''	=	$\{A_5\}$
B'''	=	ϕ
B''''	=	ϕ
L_{whole}	=	$\{(A_1, A_2, A_4), (A_3), (A_5), (A_1A_2)\}$

Table 5.6: Results - II

Since PB'' is not null, new candidate itemsets will be generated. The new candidate itemsets after pruning will be $\{(A_1A_5, A_2A_5, A_3A_5, A_4A_5),(A_1A_2A_5)\}$. Then, entire database is scanned to find the support count of the new candidates.

5.2.3 Experimental Results

Modified_Borders was compared with *Borders* algorithm using two synthetic databases and one real database . Both the algorithms were implemented on a Intel PIV based WS (with 256 MB RAM). L_{old}, B_{old} , B'_{old} and B''_{old} had been computed separately.

Test Data : Two synthetic databases, which were generated using the technique given in [AMS+96], and the *Connect-4* dataset, which was downloaded from UCI machine learning repository (www.ics.uci.edu), were used for the experiments. All the synthetic datasets contain 100K records and dimensionality of each record is 500. Other parameters for synthetic databases are shown in the *Table* 5.7. For•all the experiments, initial sizes of the synthetic databases and *Connect-4* were taken as 80K records and 47557 records respectively.

Data Set	$\|T\|$	[ML]	$\|D\|$
T20I4100K	20	4	100K
T20I6100K	20	6	100K

Table 5.7: Parameters for Synthetic Databases - VI

Each of the experiments were executed several times. *Tables* 5.8 through 5.10 show average number of full database scan required in *Borders* and *Modified_Borders* as the value of β' increased from 3.5% to 4.5% and size of incremental database is increased from 5K records to 20K records. The value of *minsup* was taken as 5% for all the experiments. Average execuiton times over all increments are given in *Figures* 5.2 on the following page and 5.3 on page 114.

Increment	*Borders*	*Modified_Borders* (β'=3.5%)	*Modified_Borders* (β'=4%)	*Modified_Borders* (β'=4.5%)
5K	1	0	0	1
10K	3	0	1	2
15K	2	0	0	2
20K	3	1	1	3

Table 5.8: Comparison on Whole Database Scan for Database T20I4100K

Increment	*Borders*	*Modified_Borders* (β'=3.5%)	*Modified_Borders* (β'=4%)	*Modified_Borders* (β'=4.5%)
5K	0	0	0	0
10K	3	0	0	2
15K	2	1	1	1
20K	3	0	1	2

Table 5.9: Comparison on Whole Database Scan for Database T20I6100K

Increment	*Borders*	*Modified_Borders* (β'=3.5%)	*Modified_Borders* (β'=4%)	*Modified_Borders* (β'=4.5%)
5K	0	0	0	0
10K	1	0	1	1
15K	1	0	1	1
20K	2	1	1	1

Table 5.10: Comparison on Whole Database Scan for Database *Connect4*

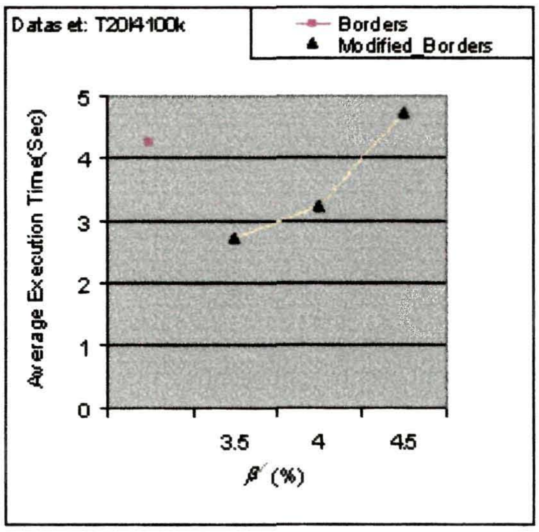
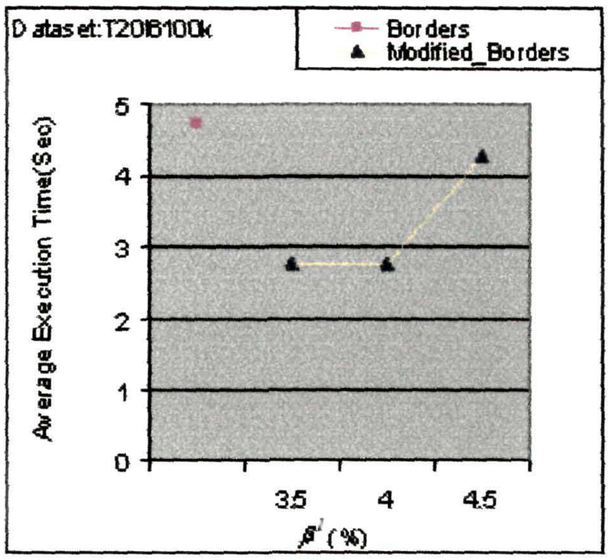


Figure 5.2: Average Execution Time for *Borders* & *Modified_Borders* (For Various Values of β') - I



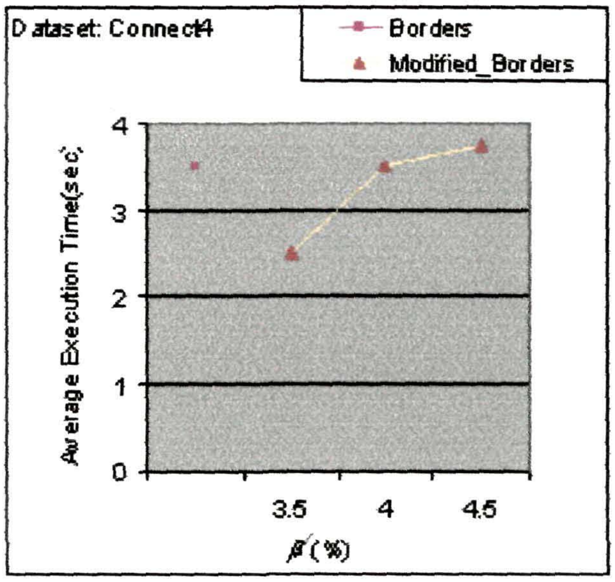


Figure 5.3: Average Execution Time for *Borders* & *Modified_Borders* (For Various Values of β') - II

Observations

Followings are some observations made from the experimental results.

- *Tables* 5.8 on page 112, 5.9 on page 113 & 5.10 on page 113 clearly show that *Borders* requires whole scan of the database several number of times, whereas *Modified_Borders* requires whole scan of database a few number of times. Thus, *Modified_Borders* saves the execution time.
- The value of β' has a great effect on the performance of *Modified_Borders* algorithm. As the value of β' increases, number of whole scan also increases. In the experiments, it was found that when β'=4.5%, number of full scan of the database is almost same in both the algorithms.
- As far as execution time is concerned, *Modified_Borders* takes much less time than that of *Borders* when β' is small (*Figures* 5.2 on page 113 & 5.3 on the preceding page). As the value of β' increases, *Modified_Borders* tend to take little more time. This is because number of full scan tends to increase with the increase in value of β'.

Selection of β'

Value of β' plays an important role in the algorithm. When β' tends to *minsup*, the algorithm tends to become *Borders* algorithm. With the decrease in value of β', performance of *Modified_Borders* becomes better than *Borders* algorithm in terms of execution time. However, with the decrease in β' value the memory requirement increases due to the additional candidate sets. So, value of β' cannot be decreased too much. This cost of additional memory requirement is quite negligible in comparison to the requirement of full database scanning, particularly when database is very large. So, there should be some trade off in choosing the value of β'. If there is not enough memory and the database is dense, the value of β' can be set to a higher value. For sparse databases, β' can be set to a lower value.

Here is a very simple technique to choose value of β'. The algorithm is run with small random sample of the database, incremental database, and an initial lower value of β'. If the result is not satisfactory, β' is increased by a little amount and the algorithm is rerun . The process is continued several times until number of candidates is manageable and gives desired results. When optimum results are obtained, the corresponding value of β' can be used for the entire database. It has been found that the value of β' in range of 80% - 90% of *minsup* gives satisfactory results. As for example, if *minsup* is 5% then β' will be in the range of 4% - 4.5%.

5.3 *Distributed_Borders*

Basically, there are three approaches for distributed algorithms - fully distributed, central machine based and self-organized. In case of fully distributed algorithm. processing takes place in every node and every node can act as a merger machine. Nodes also can exchange data among themselves. In case of central machine based algorithm, most of the processing takes place in a central machine (server). The disadvantage is that data replication is required in this environment. Self-organized distributed algorithms are more intelligent. The algorithm assigns tasks to different nodes according to requirement. These systems are more robust and fault-tolerant.

This section presents a fully distributed version of the *Borders* algorithm. The *Borders* algorithm discussed above is sequential in nature and is meant for centralized database. However, nowadays. most of the databases are distributed in nature. So, a distributed version of *Borders* algorithm called *Distributed_Borders* has been proposed. It can also be used in a centralized database by partitioning the database and placing the partitions in different nodes of a distributed systems. This process reduces the number of candidate sets to a great extent resulting in high flexibility, scalability and low cost performance ratio [CHN+96a]. However, distributed algorithms posses some problems such as locally large or border sets may not be globally large or border sets. Again, message passing being a costly affair, processing should be confined in the local sites as much as possible.

Let us consider a transactional database, where each record is a transaction in a supermarket made by the customers. Each transaction is of the form

$< TID$,1.1,0..,0,1>. Here, TID is the transaction id, which is unique for each transaction. 1 and 0 represents the corresponding item has been bought and not bought respectively in the transaction. It is also assumed that the database is horizontally partitioned and allocated in n_s sites S_i(i=1,2,3...n_s) in a fully distributed system; incremental database also is partitioned and added to database of each site. Now. the task is to maintain the global frequent itemsets and global border sets in this distributed environment when the database is updated.

5.3.1 Distributed Algorithm For Maintaining Frequent Itemsets in Dynamic Database

Here, *Borders* algorithm has been examined in a distributed environment. Let T_{old} be the old transaction database distributed in n_s sites. T^i_{old} is the old transaction database at the site i(i=1,2,3...n_s). T_{new} and T^i_{new} are the new transactions to be added to the whole database and to the transactions at the site i respectively. For a given minimum support threshold *minsup*, an itemset X is globally large in the old database (updated database) if $Sup(X)_{T_{old}} \geq minsup$ ($Sup(X)_{T_{whole}} \geq minsup$). Similarly, an itemset X is locally large in the old database (updated database) at some site i, if $Sup(X)_{T^i_{old}} \geq minsup$ ($Sup(X)_{T^i_{whole}} \geq minsup$). Like the *Borders* algorithm, this algorithm also uses the concept of *border* set and *promoted* border set. The only difference is that all the concepts have been used in the context of the distributed environment. An itemset X is a *global border*, if X is not globally large, but all its subsets are globally large. An itemset X becomes *globally promoted border* on adding the new transactions, if X is a globally border in the old database and globally large in the updated database. L_{old} is the global large itemsets in the old database and B_{old} is the global border sets in the old database. Given L_{old} and B_{old}, the problem is to find the updated large itemsets L_{whole} and border sets B_{whole} for the updated database T_{whole}. The main purpose of the *Distributed_Borders* algorithm is to reduce the number of candidate sets and in turn reduce the number of messages to be passed across the network and execution time. To reduce the number of messages, polling technique as discussed in *DMA* [CNF+96] was used.

Some interesting observations, which are listed below, can be made relating to large, border and promoted border sets in distributed environment. Some of

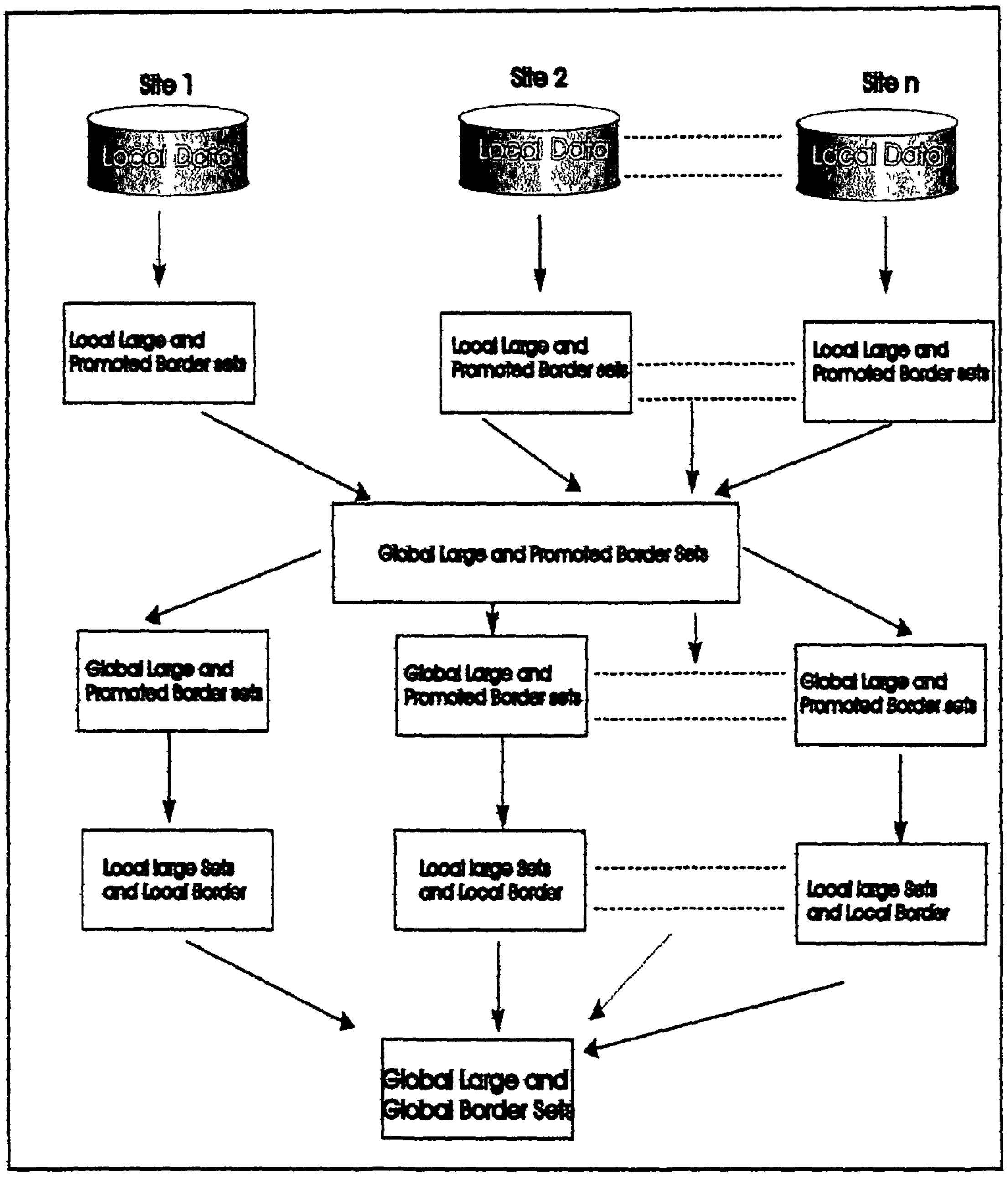

Site 1
Site 2
Site n
Local Data
Local Data
Local Data
Local Large and Promoted Border sets
Local Large and Promoted Border sets
Local Large and Promoted Border sets
Global Large and Promoted Border Sets
Global Large and Promoted Border sets
Global Large and Promoted Border sets
Global Large and Promoted Border sets
Local large Sets and Local Border
Local large Sets and Local Border
Local large Sets and Local Border
Global Large and Global Border Sets

Figure 5.4: *Distributed_Borders* Architecture

these observations were discussed in [CNF+96].

Observation 5.1 *Every global large itemset X must be large in at least one site S_i.*

Observation 5.2 *If an itemset X is locally large at some site S_i, than all its subsets are also locally large in the site S_i.*

Observation 5.3 *If an itemset X is globally large at some site S_i, then all its subsets are also globally large at the site S_i.*

Observation 5.4 *If an itemset X is globally large or promoted border, then X must be large in at least one site i.*

Proof. This is obvious, because if an itemset X is small in all the sites, it can not be large in the whole database. □

Observation 5.5 *If an itemset X is a global promoted border set, then X must be large in T^i_{new} for some site S_i.*

Proof. Let an itemset X is a promoted border set in the updated database T_{whole}. Then $Sup(X)_{T_{old}} < minsup$ and $Sup(X)_{T_{whole}} \geq minsup$. Since T_{whole}= $T_{old} \cup T_{new}$. $Sup(X)_{T_{new}} \geq minsup$. So, $Sup(X)_{T^i_{new}} \geq minsup$ for at least one site S_i. □

Observation 5.6 *If X is global border set, then there exist $Y \subset X$ so that Y is local border in some site S_i.*

Proof. If X is a global border set, then X must be small/infrequent in at least one site S_i . Therefore there exist at least one subset of X, which is a local border set in the site. □

Observation 5.7 *If a new candidate set c in T_{whole} has to be large or border in T_{whole}, then either c or one of its immediate subsets must be locally large in one T^i_{new}.*

Proof: If c is a new candidate set there can be two possible cases:

1. *c may be large in the updated database T_{whole}*: In this case c must be large in T_{new} i.e. c must be large in T^i_{new} for some i.
2. *c may be a border set in the updated database T_{whole}*: In this case c will be small and all of its subsets will be large in the updated database T_{whole}. So, there exists at least one $c' \subset c$, which was small in the old database T_{old}. Otherwise. c would have been generated in the old database T_{old}. This c' will be large in the updated database T_{whole}. So, c' must be large in the T_{new} i.e. c' must be large in T^i_{new} for some i. □

Observation 5.8 *If a candidate set X in the updated database is either large or border set, all of its immediate subsets must be either $\in F \cup PB$ or large in at least one site i.*

Proof: There can be two possible cases:

1. *X is large in the updated database*: If X is large then all the subsets of X must also be large in the updated database. Let $Y \subset X$. Then Y is either a candidate set or $Y \in F \cup PB$. If Y is a candidate and Y is large, then Y must be large in at least one site because if Y cannot be large if it is small in all the sites.
2. *X is border in the updated database*: In this case, all the subsets of X must be large in the updated database. So, by first option, all the subsets are either $\in F \cup PB$ or large in at least one site. □

5.3.2 Local Pruning

Using the above observations many unnecessary candidates can be pruned locally. If an itemset X is locally small in all the sites, then X can not be large globally. That is why, itemsets are first checked if they are locally large or not. Their global supports are found only when they are locally large in at least one site S_i. Similarly some promoted border sets also can be pruned away locally using

```
Input: $L_{old}$, $B_{old}$, $T^i_{new}$, $T^i_{old}$ and $minsup$.
Output: Updated $L_{whole}$ and $B_{whole}$

Repeat the following steps at each site $i$ distributively
 1. Scan $T^i_{new}$ and count the support of all the itemsets $X \in \{L_{old} \cup B_{old}\}$ and find
      (a) $PB^i = \{X | X \in B_{old}$ and $Sup(X)_{T^i_{new}} \geq minsup\}$ ;
      (b) $L^i_{whole} = PB^i \cup \{X | X \in L_{old}$ and $Sup(X)_{T^i_{whole}} \geq minsup\}$; (by observation 5.4 on page 119)
 2. Broadcast $X \in L^i_{whole}$ to other sites along with their supports;
 3. Prune $L^i_{whole}$ and $PB^i$:
    $L^i_{whole} = \{X | X \in L^i_{whole}$ and $Sup(X)_{T_{whole}} \geq minsup\}$;
    $PB^i = \{X | X \in PB^i$ and $Sup(X)_{T_{whole}} \geq minsup\}$;
 4. Compute $PB = \bigcup PB^i$ and $L_{whole} = \bigcup L^i_{whole}$;
 5. $B_{whole} = \{X | \forall x \in X, X - \{x\} \in L_{whole}\}$;
    Generate candidates :
 6. $m = \max\{i | PB(i) \neq \phi\}$;
 7. i=1;
 8. While ($L_i \neq \phi$ or $i \leq m$) do
 9.      $C_{i+1} = \{X = S_1 \cup S_2 |$ (i) $|X| = i + 1$,
10.                  (ii) $\exists x \in X, X - \{x\} \in PB(i) \cup L_i$,
11.                  (iii) $\forall x \in X, X - \{x\} \in L_{whole}(i) \cup L_i\}$;
12.      Scan $T^i_{new}$ and compute $Sup(X)_{T^i_{new}}$ for all $X \in C_{i+1}$;
13.      Remove any candidate set $X \in C_{i+1}$, which is or at least one of its immediate subsets is not large in $T^i_{new}$; (by observation 5.7 on page 119)
14.      Scan $T^i_{old}$ and find the support $Sup(X)_{T^i_{whole}}$ for all $X \in C_{i+1}$; ( $T^i_{new}$ has already been scanned).
15.      $C_{i+1} = \{X \in C_{i+1} | \forall x \in X, Y = X - \{x\}, Y \in L_{whole}$ or $Sup(Y)_{T^i_{whole}} \geq minsup\}$(by observation 5.8 on the previous page)
16.      Collect all $Sup(X)_{T^i_{whole}}$ and find $Sup(X)_{T_{whole}}$ for all $X \in C_{i+1}$;
17.      $L_{i+1} = \{X | X \in C_{i+1}$ and $Sup(X)_{T_{whole}} \geq minsup\}$;
18.      $L_{whole} = L_{whole} \cup L_{i+1}$;
19.      $B_{whole} = B_{whole} \cup (C_{i+1} - L_{i+1})$;
20.      i=i+1;
21. Enddo
22. Return $L_{whole}$ and $B_{whole}$;
```

Algorithm 5.4: *Distributed_Borders*

Observation 5.4 on page 119. So, if a border set X is not large in any site, then it is not tested for global promoted border. *Observation* 5.7 on page 119 is very significant in the pruning away of the candidate sets locally. After the candidate sets are generated , support of the candidate sets are counted in the incremental part T^i_{new}. If any candidate set or at least one of its immediate subsets is not large in at least one T^i_{new}, it can be pruned away because it can be neither large set nor border set by *Observation* 5.7 on page 119 . *Observation* 5.8 on page 120 is also helpful in pruning away unnecessary candidate sets.

5.3.3 Explanation of the Algorithm

It is assumed that L_{old} and B_{old} are available with the local support to all the sites. The algorithm starts with scanning the incremental portion T^i_{new} and finds local support for all $X \in L_{old} \cup B_{old}$. This is because frequent and border sets. which are locally large in at least one site, can only be large globally. Then comes to the second step, which finds the global support for L^i_{whole}. This can be done by simply broadcasting the local support of $X \in L^i_{whole}$. If all the items are broadcast to all the sites, then for each item X, $O(n_s^2)$ messages will be required, where n_s is the number of sites. Here, polling techniques as described in [CHN+96a] can be used to reduce the number of messages to $O(n_s)$ for each itemset. Third step prunes away the $X \in L^i_{whole}$, which are not globally large. The fourth step just broadcasts the L^i_{whole} to other sites and receives the same from other sites to compute L_{whole} and PB. It can be noted that, all the sites will be having the same set of L_{whole} and PB. Steps 6-11 are responsible for generating the candidate sets. The candidate sets are generated using methods like *Apriori*. Some kind of pruning techniques are required to prune away some unnecessary candidate sets. *Observation* 5.4 on page 119 helps prune away some candidate sets. Steps 13-15 are basically pruning steps. It scans the T^i_{new} and finds the $Sup(X)_{T^i_{new}}$ for all the candidate sets. According to *Observation* 5.4 on page 119, a candidate X can be neither large nor border if neither X is large nor at least one of its immediate subsets is large in any T^i_{new}. So, the candidates, which do not conform to *Observation* 5.4 on page 119, can easily be removed from the candidate sets. At last, step 16 finds the global support for all the candidate sets. Polling technique as given in [CHN+96a] can be used here also.

5.3.4 Experimental Results

The algorithm was simulated on a share-nothing environment. A 10/100 Mb LAN was used to connect six PIV machines running Windows NT. Each machine had 20GB disk space and 256MB memory. The datasets used in the experiments were T20I4200K and T20I6200K, which were generated using the technique given in [AMS+94]. Each dataset contained 200K tuples(transactions). Each dataset was partitioned and corresponding partitions were loaded in the machines before the experiments started.

Data Set	$\|T\|$	$\|ML\|$	$\|D\|$
T20I42000K	20	4	200K
T20I6200K	20	6	200K

Table 5.11: Parameters for Synthetic Databases - VII

Three experiments were carried out. In the first experiment, three machines(sites) were used. The purpose of the experiment was to find the execution time for different minimum supports. Each machine initially contained 63K transactions and 3K transactions were added to each machine as incremental database. The results are given *Figure* 5.5 on the next page.

The second experiment was the scale up experiment. The testbed of the second experiment was same as that of first experiment. Here also, three machines(sites) were used. The purpose of the second experiment was to evaluate the scalability performance of the algorithm. Three machines initially contained 30%. 30% and 25% transactions respectively. Size of incremental database was 5% for each of the machines and minimum support was 1%. The results are given in *Figure* 5.6 on page 125.

The third experiment was the speedup experiment. For n sites, *speedup factor* is defined as $S(n) = T(1)/T(n)$ and *efficiency* is defined as $S(n)/n$, where $T(n)$ is the execution time with n sites. Here, number of machines(sites) were increased from 1 to 6. Sizes of initial database and incremental database were taken as 80% and 20% respectively. Initial and incremental database were divided equally among the machines and minimum support was taken as 1%. When 1 machine(site) was used, it was the sequential run time of the *Borders* algorithm. The results are given in *Figure* 5.7 on the next page.

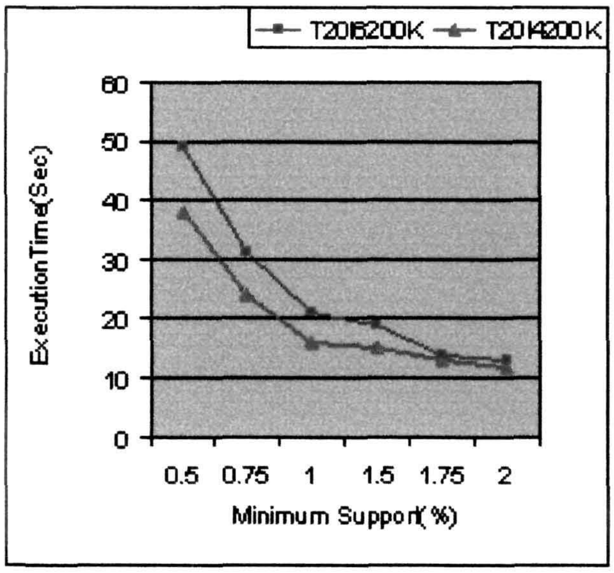


Figure 5.5: Execution Time of *Distributed_Borders* for different Minimum Supports (*minsup*)

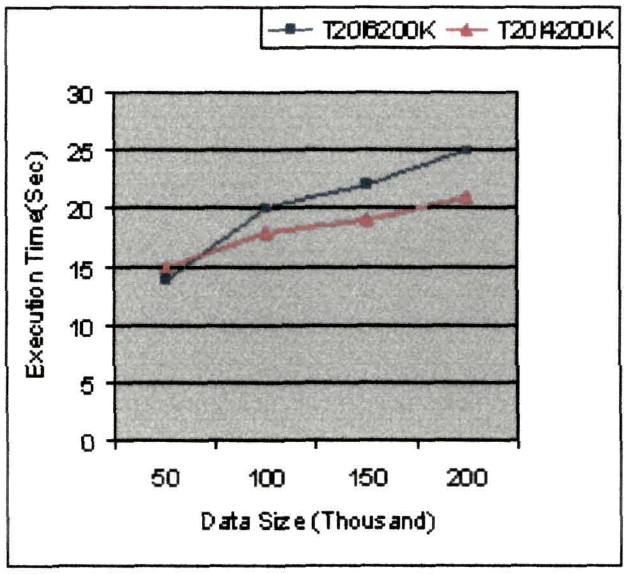


Figure 5.6: Execution Time of *Distributed_Borders* for different Database Sizes

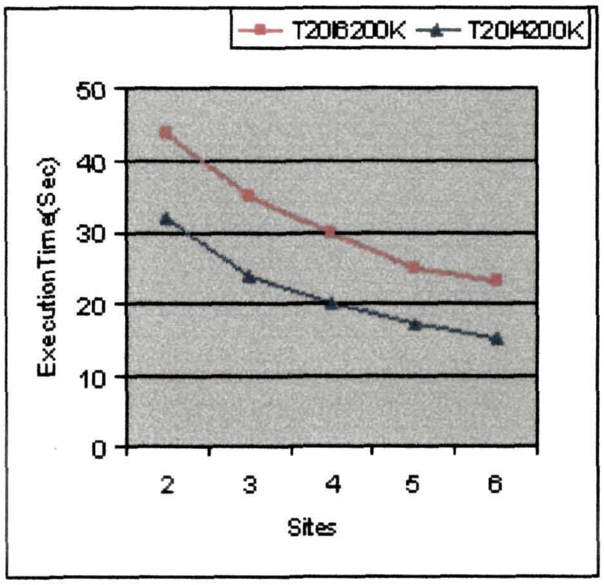


Figure 5.7: Execution Time of *Distributed_Borders* for different Number of Sites

Observations : The results of the first experiment was obvious and straightforward. In some cases, execution time performance did not improve with the increase of minimum support. This was because whole scan of the database might be required in some sites. It was evident from the second experiment that execution time increased with the increase of the size of initial database and incremental database. However, it increased linearly. Third experiment measured *speedup* and *efficiency* of the algorithm. Average efficiency of 63% and 67% for T20I6200K and T20I4200K respectively were found, which showed that the algorithm achieved *sublinear speedup*. This speedup is acceptable for any distributed algorithm. However, like other distributed algorithms, performance of this algorithm also depends on the factors such as database types, distribution of data, skewness of data, network speed and other network related problems.

5.4 Discussion

The chapter has presented two enhanced versions of the *Borders* algorithm: *Modified_Borders* and *Distributed_Borders*. *Modified_Borders* has tried to reduce the execution time by avoiding full scan of the database in most of the cases. On the other hand, *Distributed_Borders* is the modification of the *Borders* algorithm to make it suitable for distributed dynamic databases.

Based on the algorithms reported so far. it is quite evident that finding frequent itemsets is a crucial phase in association rule mining. With the increase in dimensionality of databases, the cost of frequent itemset finding also increases. Therefore, the frequent itemset finding task should be limited to those features appropriate or relevant to the task, which increases the efficiency of the algorithms to a great extent. Next chapter attempts to highlight some of the popular feature selection methods and presents a novel method for relevant feature selection.

Chapter 6

Feature Selection

Almost, all the databases contain some irrelevant/redundant features. That is why, removing the redundant features from the databases has been an interesting research problem for some decades. In the real-world situations, relevant features are unknown apriori. Therefore, many candidate features are introduced to better represent the domain. It has been found from the experiments that many of the features are either irrelevant or redundant to the target concept. An irrelevant feature does not affect the target concept in any way, and a redundant feature does not add anything new to the target concept [JKP94]. In many applications, the size of the dataset is so large that learning algorithms might not work as well before removing the unwanted features. Reducing the number of irrelevant or redundant features drastically reduces the running time of a learning algorithm and yields more general concept of a real-world classification problem [KS95], [KS96].

There are basically two categories of feature selection methods - *supervised*, where each instance is associated with a class label and *unsupervised*, where instances are not related with any class label. In case of supervised feature selection. the relevant features are selected to increase the accuracy of prediction of the class label for a given instance. Unsupervised feature selection is used as a preprocessing of other machine learning techniques to reduce the dimensionality of the domain space without much loss of information content.

Feature selection is defined by many authors in different ways. Some of them are as follows.

Idealized: It finds the minimally sized feature subset, necessary and sufficient for a target concept [KR92].

Classical: It selects a subset M features from a set of N features, $M < N$, such that value of a criterion function is optimized over all subsets of size M.

Improving Prediction Accuracy: Here, aim of feature selection is to choose a subset of features for improving prediction accuracy or decreasing the size of the structure without significantly decreasing prediction accuracy of the classifier built using only selected features [KS96].

Approximating original class distribution: Here, goal of feature selection is to select a small subset such that the resulting class distribution given only the values for the selected features is as close as possible to the original class distribution given all feature values.

Generally, feature selection attempts to select the minimally sized subset of features according to two basic criteria: (i) the classification accuracy does not significantly decrease and (ii) the resulting class distribution, given only the values for the selected features, is as close as possible to the original class distribution. given all the features. Some feature selection methods find the best feature subset in terms of some evaluating function among the possible 2^N subsets. Basically. there are four steps in a typical feature selection algorithm.

1. A *generation procedure* to generate the next candidate subset.
2. An *evaluation function* to evaluate the subset under examination.
3. A *stopping criterion* to decide when to stop.
4. A *validation procedure* to check if the selected subset is valid or not.

A good number of algorithms have been developed for feature selection over the years [Doa92]. Some of the prominent feature selection algorithms are *Branch & Bound* [NF77], *Focus* [AD91], *Relief* [KR92], *LVF* [LS96], etc. Next section presents a brief discussion on some of these algorithms.

This chapter also introduces a new supervised feature selection algorithm for binary classification based on frequent (or large) features of association rule mining technique [AMS+94, AMS+96].

6.1 Some Existing Feature Selection Algorithms

In this section, some of the popular and prominent feature selection algorithms are reproduced.

6.1.1 *Branch and Bound*

This is an exponential search algorithm and was proposed by Narendra and Fukonaga in 1977. It follows top-down approach with backtracking and based on the assumption that the criterion function is monotonous. Suppose, it is required to select 2 features out of four features (f^1, f^2, f^3, f^4) by using *Branch and Bound*. *Branch and Bound* first constructs the search tree as given in *Figure* 6.1 on the following page, where root denotes the set of all features and leaves denotes the set of two features. Nodes of level k is constructed by removing k features from the root. Nodes in the kth level represents the subset of N -k features. where N is the total number of features. The algorithm starts with searching from the root and every time it reaches a leaf, it updates the *bound* (current maximum) with the corresponding criterion value of the leaf. The advantage of the algorithm over exhaustive search is that it is not required to construct a branch of a node if the criterion value of the node is less than the current *bound* because of monotonous property of the criterion function.

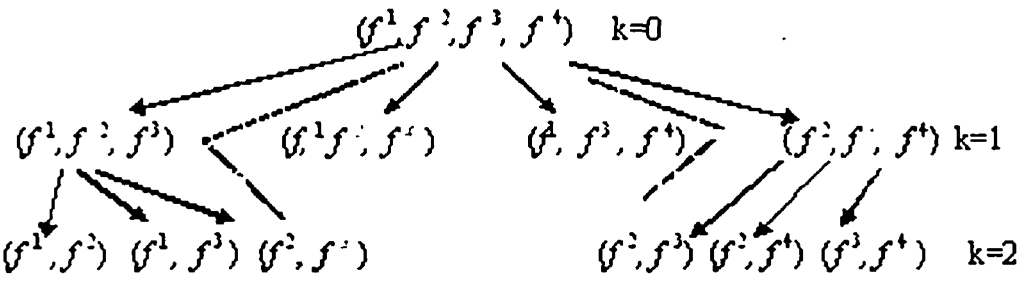


Figure 6.1: *Branch and Bound* Search Tree

The algorithm is presented in *Algorithm* 6.1 on the next page. The algorithm needs inputs of required number of features (M) and it attempts to find out the best subset. The algorithm uses two functions. The function $isbetter(X, Y)$ checks if the set X is better than the set Y and the function $Card(X)$ finds cardinality of the set X . However, the algorithm suffers from the following drawbacks.

1. It does not perform well. if the criterion function is of high computational complexity.
2. It does not guarantee to remove enough sub-trees.
3. Criterion value computation is slower, nearer to the root.
4. Removal of sub-trees is less, nearer to the root.

6.1.2 *Relief*

Relief uses heuristic technique to generate candidate feature subset and distance to evaluate a candidate subset. It is a feature weight-based algorithm and uses statistical method to choose the relevant features. It also uses the concept of *NearHit* and *NearMiss*. *NearHit* of an instance is defined as the instance having minimum Euclidean distance among all instances of the same class as that of the instance. *NearMiss* of an instance is defined as the instance having minimum

```
Input: $D$, $F$, $M$.
Output: $S_f$.

B&B($D, F, M$)

 1. If $Card(F) \neq M$ then /*subset generation*/
 2.     $j$=0;
 3.     $S_f$= $F$;
 4.     For all features $f \in F$ begin
 5.         $S_j = F - f$; /*remove one feature at a time */
 6.         If ($S_j$ is legitimate) then
 7.             If $isbetter(S_j, S_f)$ then
 8.                 $S_f = S_j$;
    /*recursion*/
 9.         B&B($S_j, M$);
10.     Endfor
11.     $j++$;
12. Endif
13. Return $S_f$;
```

Algorithm 6.1: *Branch & Bound*

Euclidean distance among all instances of different class. The algorithm finds the weights of the features from a sample of instances and chooses the features with weight greater than a threshold. The algorithm uses one function *diff*() to find difference of same features in two different records . The algorithm is given in *Algorithm* 6.2 on the following page. The advantage of the algorithm is that it can work for noisy and correlated features. However, it suffers from following drawbacks.

1. It cannot work with redundant features and hence generates non-optimal features, if the database contains redundant features.

2. It works only with binary classes.

3. Another problem is the selection of optimum values of *NoSample* and *Threshold* is not clear.

6.1.3 *Focus*

This is an inductive learning algorithm and it is based on the concept of *Min-feature* bias. According to *Min-feature* bias, if two functions are consistent with the training examples, the functions with minimum features will be preferred. The algorithm first identifies p features that are required to define a binary function over n boolean input features. Then, it applies some learning procedures that focus on those p features. In other words, the algorithm generates all possible feature subsets and uses consistency measure to evaluate the subsets. The algorithm has been found to work well with noise-free data. However, the main disadvantage is how to select correct inconsistency measure. The algorithm is given in *Algorithm* 6.3 on the next page.

```
Input : $D$, $F$, $NoSample$, $Threshold$.
Output : $S_f$;

1.    $S_f = \phi$;
2.    Initialize all weights, $W_j$ to zero;
3.    For $i = 1$ to $NoSample$
4.        Randomly choose an instance $t$ in $D$;
5.        Find its $NearHit$ and $NearMiss$;
6.        For $j = 1$ to $N$
7.            $W_j = W_j - diff(f^j, NearHit(j))^2 + diff(f^j, NearMiss(j))^2$;
8.        For $j = 1$ to $N$
9.            If $W_j \geq Threshold$
10.               Append feature $f^j$ to $S_f$;
11.   Return $S_f$;
```

Algorithm 6.2: *Relief*

```
Input: $D$, $F$.
Output: $S_f$.

1. $S_f = F$;
2. For $i$=0 to $N$ /*$N$ is number of features */
3.     For each subset $X$ of size $i$
4.         If no inconsistency in the training set $D$ then
5.             $S_f = X$;
6.             Return $S_f$;
```

Algorithm 6.3: *Focus*

6.1.4 *LVF*

LVF generates the candidate subsets randomly and uses consistency measure to evaluate a subset. It randomly searches the subset space and calculates an inconsistency count for the subset. To search the optimal subset, the algorithm uses *Las Vegas* algorithm [BB96]. The algorithm calculates the inconsistency count based on the intuition that most frequent class label among those instances matching this subset of features is the most probable class label. An inconsistency threshold is assumed and any subset with inconsistency measure greater than that value is rejected. The algorithm can find optimal subset even for datasets with noise and user does not have to wait too long because it outputs any subsets that is better than the previous best. The algorithm is given in *Algorithm* 6.4 on the following page. The algorithm has used two functions : $Card(X)$ and $InConCal(X, Y)$. $Card(X)$ finds cardinality of the set X and $inConCal(X, Y)$ finds inconsistency between sets X and Y. This algorithm is efficient, as only the subset having the number of features smaller than that of the current best subset are checked for inconsistency. In addition to that, the algorithm is easy to implement and is guaranteed to find the optimal subset. However, the algorithm suffers from two major drawbacks.

1. Selection of optimum inconsistency threshold (*ucon*) is difficult.
2. Selection of number of samples ($Maxtries$) is also a difficult decision.

6.1.5 Discussion

In this section, some of the popular feature selection algorithms have been discussed. It has been observed that, different algorithms have used different concepts of relevant features to select the features. In addition to that, different algorithms used different assumptions. As for example, *Branch and Bound* has assumed that criterion function is monotonous; *Relief* is based on the concept of *NearHit* and *NearMiss* and so on. So, different algorithms give optimum results in different environments and with different data sets. In other words, no algorithm is suitable for all environments or can select relevant features in all types of data sets. As a matter of fact, it will be virtually impossible to design

```
Input : $D$,$F$,$Maxtries$,$ucon$.
Output : $S_f$, Set of relevant features.

 1. $S_f = F$;
 2. For $i$=1 to $Maxtries$;
 3.    Randomly choose a subset of features, $X$;
 4.     If $Card(X) \leq Card(S_f)$
 5.       If $InConCal(X, D) \leq ucon$
 6.          $S_f = X$;
 7.           Output $X$;
 8.       Else
 9.           Append $X$ to $S_f$;
10.          Output $X$ as 'another solution';
11. Endfor
12. Return $S_f$;
```

Algorithm 6.4: *LVF*

an algorithm, which will find most relevant features in all environments and for all types of data sets.

Researchers are trying to use different concepts to design algorithms to select relevant features. One such useful concept is the frequency count(support count), as can be found in association rule mining technique. To the best of our knowledge, there has not been much attempt to find relevant features based on frequent features/items of association rule mining technique [AMS$^+$94, AMS$^+$96]. J Moore used association rules to select features in a web page for web page clustering. V Jovanoski and N Lavrac used association rules in inductive concept learning i.e. to device a classifier. They also used association rules to measure the accuracy of other feature selection algorithms.

This chapter (next section) presents a supervised feature selection algorithm for binary classification based on frequent items/features of association rule mining technique [AMS+94, AMS+96]. In binary classification, instances of a database are associated with only one class label. The class label is either 1, which represents the instance belongs to the class, or 0, which represents the instance does not belong to the class. So, the instances in the database may belong to the class or may not belong to the class. The chapter also presents comparative results of the proposed algorithm and some existing algorithms.

6.2 The *FFC* Algorithm

The proposed algorithm, *FFC* (Feature selection using Frequency/support Count), is meant for relevant feature selection from binary classified data and is based on frequent items/features of association rule mining technique [AMS+94, AMS+96]. Here, frequency count refers to support count of association rule mining. Let us consider a binary classified database of instances, where each instance either belongs to a class or does not belong to the class. It is also assumed that each instance is of the form $< TID, f^1, f^2, ...f^n, C_l >$, where TID is the unique identification number of the instance , f^i is the i^{th} feature and C_l is the class of the instance. f^i can be either 1, if the feature has occurred in the instance, or 0 , if the feature has not occurred in the instance. Similarly, C_l can be either 1, if the instance belong to the class, or 0 , if the instance does not belong to the class. . As for example, one instance may be (11, 1, 0, 1,... ,1). The first number 11 is the instance number. Among the rest, 1 represents that corresponding feature has occurred and 0 represents that corresponding feature has not occurred. The last value represents the occurrence of the class, where 1 represents that the instance belongs to the class and 0 represents that the instance does not belong to the class.

It has been observed that in binary classified data, where instances are associated with only one class label, if a feature f is relevant or has some influence on the occurrence of the class C_l then there may be two possible cases: One is that the class C_l occurs when f occurs and the other is that class C_l occurs when f does not occur in most of the instances. In association rule mining terminology, a feature f will be relevant with respect to the class C_l, if either fC_l or $f'C_l$ is frequent, where f' represents the non-occurrence of f. The algorithm explores these observations to find the relevant features.

The algorithm works as follows. The inputs to the algorithm are D, *minsup* , *incr* and *minf*. The output is S_f. Initially, the algorithm assumes that all the features are relevant. So, S_f contains all the features. Then it generates L_1 and L'_1 followed by generation of C. Each element of C consists of two elements and is of the form fC_l, where f is a feature and C_l is the class such that either f or f' is frequent. Then the algorithm finds the support count for all the elements of C. The algorithm uses bitmaps [HLL03] of the features to find the support count because it reduces the execution time to a great extent. At the end, the relevant features are extracted. The relevant features are those which are included in at least one frequent element of C. Then *minsup* is incremented by *incr*. The purpose of increasing the minimum support is to find the features with maximum possible support. This process is repeated as long as number of features in S_f is greater than the minimum number of required features. If the number of selected features become less than the minimum number of required features, the immediate previous set of selected features is returned. Otherwise, S_f is returned. The algorithm is presented in *Algorithm* 6.5 on the next page.

The algorithm uses the concept of frequent itemset. As a result, it may not be able to remove the redundant/correlated features in small database. The value of *minsup* , *incr* and *minf* plays a major role to determine the execution time and how quickly the algorithm will converge. The larger value of *minsup* and *incr*, the more quickly the algorithm will converge. However, larger value of *incr* may miss some relevant features. As far as *minf* is concerned, the execution time decreases as the value of *minf* increases for less number of iterations over the database. In the experiments, it was found that the value of *minsup* between 0.01 and 0.05, and the value of *incr* between .005 and 0.01 give good results. For dense database, *minsup* can be set to a higher value. So, there should be some trade-off in choosing the values of the above mentioned parameters.

```
Input : $D$, $minsup$, $incr$, $minf$.
Output : $S_f$ (set of relevant features).

1. $S_f$= All features;
2. Scan the database and find the bitmaps of all the features and the class
   label.
3. Do while($|S_f| > minf$)
4.     $C=\phi$;
5.     $L_1 = \{f|Sup(f) \geq minsup\}$; /*Features whose occurrence is large*/
6.     $L'_1=\{f|Sup(f') \geq minsup\}$; /*Features whose non-occurrence is
   large*/
7.     $C=C \cup \{xC_l|x \in L_1 \cup L'_1\}$; // support count of $C$
8.     For all $c \in C$
9.         Find support count ($c.count$) of $c$ using bitmaps;
10.    $F1 = S_f$;
11.    $S_f = \{f|c \in C, c = fC_l/f'C_l, c.count \geq minsup\}$;
12.    $minsup$=$minsup$+$incr$;
13. Enddo
14. If $|S_f| < minf$ then
15.         $S_f= F1$;
16. Endif
17. Return $S_f$;
```

Algorithm 6.5: *FFC*

6.3 Experimental Results

Feature selection methods can be validated either by using artificial datasets or by real-world datasets. Artificial datasets are constructed with some known relevant features and some noise features. Feature selection methods are run over these datasets to check if they can find the known relevant features or not. In case of real-world datasets, relevant features are unknown. Accuracy of a feature selection method is determined with the help of a suitable classifier. However. selecting suitable classifier is difficult because different classifiers support different datasets. So, four artificial datasets were chosen to evaluate the performance of the algorithms.

6.3.1 Datasets Used

The datasets, which are described below, have combinations of relevant, correlated, irrelevant and redundant features. These datasets are available in *UCI Machine Learning Repository* (http://www.ics.uci.edu). Here, the attempt is to evaluate the strengths and weaknesses of the proposed method along with the other methods.

CorrAL Dataset [KAH96]

The dataset consists of 32 instances, binary classes and six boolean features(A_0, A_1, B_0, B_1, I, C), where I is irrelevant and C is the class level. Here, relevant features are A_0. A_1, B_0 and B_1.

Modified Par3+3 Dataset

The dataset contains 64 instances. It consists of binary classes and twelve boolean features. Among them, A_1, A_2, A_3 are relevant and A_7, A_8, A_9 are redundant.

Monk1 and Monk3 Dataset [TBB+91]

Monk1 consists of five discrete features(A_1, A_2, A_3, A_4 and A_5) and the binary class, out of which A_1, A_2 and A_5 are relevant to the target concept.

Monk3 consists of six discrete features(A_1, A_2, A_3, A_4, A_5 and A_6) and the binary class, out of which A_2, A_4 and A_5 are relevant to the target concept.

6.3.2 Experimental Setup

The proposed algorithm was implemented using a Intel PIV machine. The value of *minsup* and *incr* were taken as 0.05 and 0.005 respectively for all the datasets. The value of *minf* was taken as 4 for *CorrAL* and 3 for other datasets.

6.3.3 Results

Average experimental results are presented in *Table* 6.1. All the algorithms took very less amount of time and there was a little variation in the execution times.

Method (RA)	*CorrAL* (A0,A1,B0,B1)	*Monk3* (A2,A4,A5)	*Monk1* (A1,A2,A5)	*Modified Par 3+3* ({A1,A7}, {A2.A8} ,{A3,A9})
Relief	A0,B0,B1,C	A2,A5 always & one or both of A3,A4	A1,A2,A5	A1,A2,A3,A7,A8.A9
B&B	A0,A1,B0,I	A1,A3,A4	A1,A2,A4	A1,A2,A3
Focus	A0,A1,B0,B1	A1,A2,A5	A3,A4,A5	A1.A2,A3
LVF	A0,A1,B0,B1	A2,A4,A5	A1,A2,A5	A2,A3,A7
FFC	A0,A1,B0,B1,I,C	A1,A2,A4,A5	A1,A2,A5	A1,A2,A3

Table 6.1: Experimental Results of *Relief, B &B, Focus, LVF and FFC*

6.3.4 Observations

Followings are some observations from the experimental results.

1. The proposed algorithm could not remove the Correlated/Redundant features from *CorrAL* data which is evident in the result. However, it found all the relevant features in *Monk1* and *Monk3* datasets.

2. In case of *Monk1*, it could select all relevant features. The discrepancy in the results could be attributed to the inherent nature of frequent itemset, type and size of the database.

6.4 Discussion

This chapter has presented an algorithm for relevant feature selection in binary classified data using the concept of frequent itemset. Based on experimentation, it has been found that the proposed algorithm is equally good when compared with the other counterparts. One disadvantage of the algorithm is that it could not find all the relevant features in all the datasets. However, this cannot be considered as a major disadvantage, because no algorithm can find all the relevant features in all kinds of data. The main advantage of the algorithm is the simplicity and easy implementation compared to its counterparts. So, the algorithm can be very useful to find relevant features in binary classified data.

Chapter 7

Data Cube Materialization

OLAP(On-line Analytical Processing) operations deal with aggregate data. Hence, materialization or pre-computation of summarized data are often required to accelerate the DSS(Decision Support System) query processing and data warehouse design. Otherwise, DSS queries may take long time due to huge size of data warehouse and complexity of the query , which is not acceptable in DSS environment. Different techniques like query optimizers and query evaluation techniques [CS94, GHQ95, YL95] are being used to reduce query execution time. View materialization is also one very important technique, which is used in DSS to reduce query response time. Therefore, researchers are always in search of better algorithms, which can select best views to be materialized. In this chapter also, there has been attempt to develop a better view materialization technique by exploitation of density concept and association rule mining technique. Different indexing techniques like bit-map index, join index, etc. are also used to reduce the query response time to a great extent.

Query response time largely depends on the data structure used to represent the aggregates. One of the most efficient data structures is data cube [GCB+97] , which is used widely to represent multidimensional aggregates in data warehouse systems. A data cube allows data to be modeled and viewed in multiple dimensions. In SQL terminology, data cube is nothing, but collection of group-bys. Let us take an example. Suppose, an organization keeps sales data of a particular product with respect to time(t), location(l) and branch(b) without any hierarchy as has been shown in *Figure* 7.1 on the next page. Here, the data cube consists of eight possible group-bys: ***tlb***, ***tl***, ***tb***, ***bl***, ***t,l,b*** and ***none***. Each individual group-by is called ***sub-cube*** or ***cuboid*** or ***view***. In data warehouse systems, query response time largely depends on the efficient computation of data cube. However, creating data cube on the fly is very much time and space consuming.

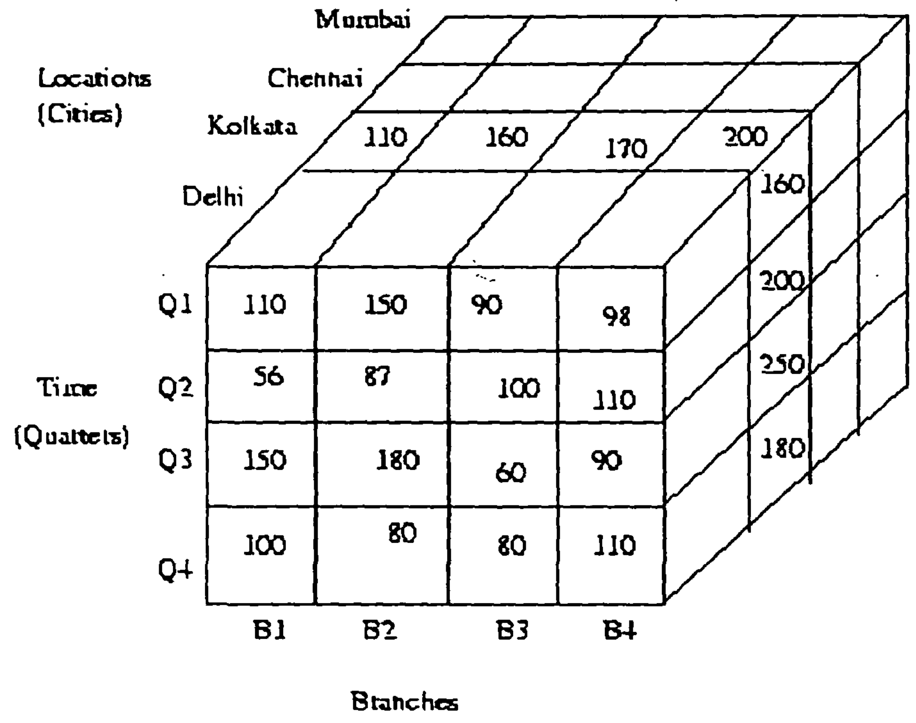


Figure 7.1: A 3-D Data Cube Representation

So, one common technique used in data warehouse is to materialize(i.e. pre-compute) cuboids of a data cube. To do this, there are three possibilities:

1. Materialize the whole data cube: This is the best solution in terms of query response time. However, computing every cuboid and storing them will take maximum space if data cube is very large, which will affect indexing and the query response time.

2. No Materialization: Here, cuboids are computed as and when required. So, the query response time fully depends on database which stores the raw data.

3. Partial materialization: This is the most feasible solution. In this approach, some cuboids or cells of a cuboid are pre-computed. However, the problem

is how to select these cuboids and cells to be pre-computed. Generally. cuboids and cells which can help in computing other cells or cuboids, are pre-computed.

There exists some view materialization algorithms. However, most of them work on some constraints such as space to store the views, time to update the views, etc. Some well known algorithms are *BPUS* [HRU96], *PBS* [SDN98]. *PVMA* [URT99], A^* [GYC+03]. etc. *BPUS* is a greedy algorithm, which selects the views with the highest benefit per unit space. The complexity of the algorithm is $O(k.n^2)$, where k is the number of views to be selected and n is the total number of views. The main disadvantage of the algorithm is that its execution time increases exponentially with the increase of number of views. Otherwise, the algorithm selects better views in terms of benefits. *PBS*(Pick By Size) algorithm selects the views on the basis of view size. However, *PBS* is meant only for SR(Size Restricted)-Hypercube lattice. A^* algorithm is one of the recent algorithms. The algorithm is interactive, flexible and robust enough to find the optimal solution under disk space constraint and the algorithm has been found to be useful when disk-space constraint is small. The algorithm has used two powerful pruning techniques (H-pruning and F-pruning) and two sliding techniques(sliding-left and sliding-right) to further improve the running efficiency of the search. Above all. there is one algorithm called *PVMA*(Progressive View Materialization Algorithm) [URT99]. The algorithm is based on the concept of Nearest Materialized Parent Views (NMPV). To the best of our knowledge. this is the first algorithm to have used access frequency of queries to select the views. It also considers updates on views and view size to calculate benefits of the views. So, this algorithm can be found to select better views than other algorithms [URT99].

This chapter has discussed performance analysis of *PVMA* algorithm in detail for the reasons given above and attempted to present a faster view materialization algorithm(*DVMAFC*) based on the notion of density and frequency count (support count) of the views. The algorithm basically forms clusters of views and selects the core views for materialization. The concept of density has been taken from the algorithm *DBSCAN* [EKS+96], which is a well-known clustering algorithm. The algorithm *DVMAFC* also has applied the concept of cost/benefit of *PVMA* to form the clusters of views. In addition to that, the algorithm has used the supports of the frequent (or large) sub-views to calculate the benefits. because it has been observed that supports of the frequent (or large) views plays an important role to select better views to be materialized. At the end, the chapter has compared the performance of *DVMAFC* with *PVMA*. It has been observed that in most of the cases *DVMAFC* selects better views and works much faster than *PVMA*.

7.1 Data Cube Lattice

All the view materialization algorithms are required to use some data structures to represent the data cube. One useful data structure is data cube lattice [HRU96], which has been used by many algorithms to represent a data cube. Let us consider the above example of sales data. The group-bys(views) can be organized in the form of a lattice as shown in *Figure* 7.2, which is a directed acyclic graph. The top view *tlb* is known as *fact table*. An edge from a view *u* to view *v* in the graph means that *v* can be calculated from *u*. [HRU96] also has shown that this relationship is in partial order. *DVMAFC* has also used data cube lattice to represent the views.

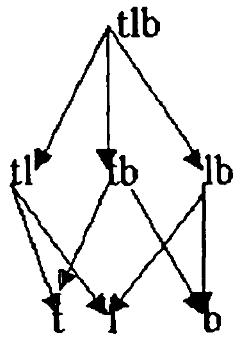


Figure 7.2: A Lattice

7.2 Progressive View Materialization Algorithm (*PVMA*)

PVMA [URT99] assumes that data cube is represented in the form of lattice as discussed in [HRU96] and selects the appropriate views to be materialized. which minimizes the query response time and maintenance cost. The feature, which distinguishes the algorithm from other algorithms is the use of size of the views. access frequency of queries (views). updates(insert, edit and delete) on each view to select the views for materialization. The algorithm also uses number of rows affected by each of the update operations. These parameters information are usually available and can be kept track easily in a data warehouse system by the warehouse administrator, considering the fact that data warehouse is updated in off peak period. Followings are some of the concepts used in the algorithm:

- *Nearest Materialized Parent Views* (*NMPV*): A view u is a parent view of v, if v can be computed from u. *NMPV* of of v at iteration k, denoted by $NMPV_k(v)$ is a materialized view u such that the difference between size of view v and size of view u is minimum among all materialized views in the iteration k of the algorithm. So, $NMPV_k(v) = \min(R(u), R(v)) \forall u \in S$ and $u \rightarrow v$.

- *Benefit*: If a view v is materialized then view v and its children receive the benefits because children can be computed from v whose size is smaller than the fact table. The benefit of v in the iteration k is calculated as

$$benefit_k(v) = \left(\frac{R(NMPV(v)) - R(v)}{bf} \sum_{u \in child(v) \cup v} f_u \right) T_{rba} \qquad (7.1)$$

- *Cost*: Each change(insert, delete and update) in the fact table results in update to each corresponding view. So, cost calculation includes the number of operations(insert, delete and update), their frequencies and time for random block access. The cost is calculated as:

$$cost(v) = \left(\sum_{i \in SI} 4N_i f_i + \sum_{d \in SD} 4N_d f_d + \sum_{u \in SU} 3N_u f_u \right) T_{rba} \qquad (7.2)$$

It is to be noted that the cost is same for all the views because the formula does not contain any information of the view.

- *Profit*: Profit of a view v at iteration k, denoted by $profit_k(v)$, is calculated as $benefit_k(v) - cost(v)$.

7.2.1 The Algorithm

The algorithm is very simple and works as follows. Base cuboid(fact table) is always to be materialized because any cuboid can be calculated from the base cuboid. The algorithm calculates cost, benefit and profit of all the views, which are not included in NR, where NR is a set of views with negative profit. The views with negative profit are discarded. Then, the algorithm selects the view with maximum positive profit. The process continues until all the views are either discarded or selected for materialization. The algorithm is given in *Algorithm* 7.1 on the next page.

Example 7.1

Let us consider the lattice given in *Figure* 7.2 on page 145. Let size(number of rows) of the cuboids *tlb, tl, tb, lb, t, l, b* be 100, 70, 60, 50, 40, 30, 20 respectively. Let access frequency of the cuboids be 10. 5. 5. 6, 5, 3, 1 respectively. Let us also take bf. T_{rba} and *cost* as 100, 10msec and 5msec respectively. Based on the above assumptions, three iterations of the algorithm are shown in *Table* 7.1 on page 149, where *s* and *nr* represents the cuboid is selected and included in NR respectively. In the first step *tlb* is selected because it is the fact table; in the second step *lb* is selected; in the third step *tb* is selected and *b* is included in NR because the profit is negative.

```
Input : Lattice of the views. $V$, Access frequency of the views.
Output : $S$.

1. $S = v_1$; ($v_1$ is the base cuboid)
2. $NR = \phi$;
3. $cost = \left( \sum_{i \in SI} 4N_i f_i + \sum_{d \in SD} 4N_d f_d + \sum_{u \in SU} 3N_u f_u \right) T_{rba}$
   ;
4. For $k$=1 to $|V|$
5. Begin
6.         For all views $v$
7.         Begin
8.           If ($v \in S$ & $v \notin NR$) then
9.           $benefit_k(v) = \left( \frac{R(NMPV(v)) - R(v)}{bf} \sum_{u \in child(v) \cup v} f_u \right) T_{rba}$
   ;
10.            $profit_k(v) = benefit_k(v) - cost$;
11.            If $profit_k(v) \leq 0$ then add $v$ into $NR$;
12.        End
13. End
14. Find $P_{view}$ from all the views $v \in S$;
15. Add $P_{view}$ to $S$;
```

Algorithm 7.1: *PVMA*

Cuboid	First step		Second step		Third step	
•	Benefit	Profit	Benefit	Profit	Benefit	Profit
tlb	-	*s*	-	-	-	-
tl	-	-	39	34	39	34
tb	-	-	44	39	44	39(*s*)
lb	-	-	50	45(*s*)	-	-
t	-	-	30	25	30	25
l	-	-	21	16	6	1
b	-	-	8	3	3	-2(*nr*)

Table 7.1: *PVMA* Example

7.2.2 Analysis

The complexity of the algorithm is $O(V^2 + SI + SD + SU)$. So, it is clear that the complexity heavily depends on V. Complexity increases exponentially with the increase of V. However, the algorithm has been found to be superior to other algorithms and considers access frequency of the views, size of the views and maintenance cost of the views to select the views [URT99]. The algorithm performs better in situations which involve databases with more dimensions and different access frequencies of views.

7.3 Density-based View Materialization Algorithm using Frequency Count (*DVMAFC*)

DVMAFC also assumes that data cube is represented in the form of lattice as discussed in [HRU96] and selects the appropriate views to be materialized. which minimizes the query response time and maintenance cost. Like *PVMA*, it also uses size of the views, access frequency of queries (views), frequency of updates (insert, edit and delete) on each view and number of rows affected by each of the update operations to select the views for materialization.

The important concept used in *DVMAFC* is the use of concept of density [EKS+96] to form clusters of views and then select the views to be materialized in a data warehouse system. A cluster in *DVMAFC* consists of views. The main characteristic of the clusters is that the *benefit* of the neighborhood of any view in a cluster must be at least some pre-defined value. Another new concept is the use of frequency/supports of the frequent sub-views to select the views because it has been observed that supports of the sub-views help select better views for materialization.

7.3.1 Definitions

Followings are some definitions [EKS+96], which are of importance in the context of the algorithm. For all these definitions, it is assumed that views are arranged in the form of a lattice as explained in previous sections.

Definition 7.1

Neighborhood : Neighborhood of a view v with respect to $MaxD$, denoted by $N(v)$, is defined by $N(v) = v \cup \{w | w \in child(v) \text{ and } R(v) - R(w) \leq MaxD\}$.

Definition 7.2

Core View : A view v is said to be core view if $benefit(N(v)) \geq MinBen$, where $MinBen$ is the minimum benefit.

Definition 7.3

Directly-Density-Reachable: A view v is directly-density-reachable from a view w, if w is a core view and v is in the neighborhood of w.

Definition 7.4

Density-Reachable: A view v_i is density reachable from another view v_j with respect to $MinBen$. if there exist a chain of views $v_1, v_2, ...v_k$ such that $v_1 = v_j$ and $v_k = v_i$ and v_e is directly-density-reachable from v_{e+1}.

Definition 7.5

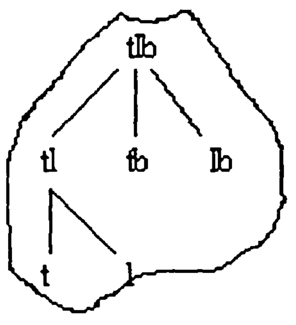
Density-Connected : Two views v_1, v_2 are density-connected if there exists another view v_3 such that v_1 and v_2 are density reachable from v_3.

Definition 7.6

Cluster : A cluster Cl of views with respect to *MinBen* and $MaxD$ is a non-empty set of views with the following conditions

1. For two views $v_1, v_2 \in V$, $v_2 \in Cl$ if $v_1 \in Cl$ and v_2 is density-reachable from v_1.

2. Two views $v_1, v_2 \in Cl$ are density connected.



t, l, fb, lb : Border points.
tlb, tl: Core points.
tl is directly density-reachable from tlb.
t is density- reachable from tlb.
t, lb are density connected to tlb.

Figure 7.3: Neighborhood, Core Points, Density-Reachable and Density-Connected

There are three categories of views - *classified, unclassified and noise. Classified* views are already associated with a cluster; *unclassified* views are not yet associated with any cluster; *noise* views do not belong to any cluster . So, it is understood that neighborhoods of classified and noise views are already calculated. Another category of views, called *leader* view, has been introduced. A leader view is an unclassified view, of which all the parents are either classified(not materialized) or declared noise.

7.3.2 Frequent Sub-views

Sub-views of a view(group-bys) are basically views consisting of the subsets of the view. As for example, sub-views of a view (u, v, w) are (u, v), (v, w), etc. In other words, sub-views of a view are the descendants of the view in the data cube lattice(*Figure* 7.2 on page 145). It has been observed that frequent sub-views play an important role in predicting future views. As an example, let us consider five views: (v_1, v_2, v_3), (v_1, v_3), (v_1, v_2, v_3), (v_1, v_2) and (v_1, v_2, v_5). Here, the sub-view (v_1, v_2) is frequent and present in 60% views. So, it can be predicted that future queries may be based on views which are superset of the sub-view (v_1, v_2). In other words, views which are superset of the frequent sub-views should be materialized so that any query on those views can be answered instantly. So. supports of the sub-views should also be considered to calculate benefits of the views.

Finding frequent sub-views may be challenging task, particularly when the view (query) database is very large. For this purpose, frequent itemset finding algorithms. as discussed in *chapter* 3, can be of great help. To calculate the frequencies of the sub-views, view database can be represented easily in the form of market-basket database. Let us consider the above example again. The equivalent market-basket database of the five views is given in the *Table* 7.2. Each

v_1	v_2	v_3	v_4	v_5
1	1	1	0	0
1	0	1	0	0
1	1	0	1	0
0	1	0	1	0
1	1	0	0	1

Table 7.2: Representation of Views

transaction represents one view, where 1 represents that the corresponding attribute has occurred in the view and 0 represent that corresponding attribute has not occurred in the view. Now. frequent itemset finding algorithms, as discussed in *Chapter* 3, can be used to find frequent sub-views with the corresponding supports and these supports will be used to calculate the benefits of the views.

7.3.3 Benefit of a Neighborhood

Benefit is an important concept and the views are selected for materialization on the basis of benefits of the views. The more the benefit of a view, the more likely the view will be selected for materialization. However, benefits of the neighborhoods of the views will be used, instead of the views themselves. Benefit of $N(v)$, denoted by $benefit(N(v))$ (*Formula* 7.3), is calculated in the same way as benefit of a view v is calculated in $PVMA$ [URT99], which is based on size of the view and access frequencies of the children views. In addition to that, supports of the frequent sub-views have been used, as discussed above, to calculate the benefits of the views. However, only the sub-views of the neighborhoods have been used, because it has been observed that lower level sub-views do not have much effect on the view.

$$benefit(N(v)) = \left((R(NMPV(v)) - R(v)) \sum_{u \in N(v) \cup v} f_u \right) + \sum_{u \in N(v) \cap Fv} Sup(u) \tag{7.3}$$

7.3.4 The Algorithm($DVMAFC$)

The algorithm centers around forming the clusters of views. While creating clusters, the algorithm has to calculate benefits of neighborhoods. The benefit is based on the view size(number of rows), access frequency of views and frequency count of the sub-views. Frequencies of view access are easily available in any data warehouse system and frequency of the sub-views can be easily calculated using the algorithms discussed in *Chapter* 3 . View sizes can also be calculated easily using the methods given in [SDN98, LS96].

The algorithm assumes that views are selected independently, there is no space constraint and OLAP uses relational database system. The algorithm also assumes that views are organized in the form of lattice as explained the previous sections. The working principle of the algorithm is very simple. It first finds all the clusters of views and then selects the core views of the clusters for materialization. The algorithm always selects the fact table for materialization. So, top view(fact table) is not included in the creation of the clusters; clusters are created from the rest of the views.

The algorithm works as follows. The algorithm starts with finding the smallest leader view v among the leaders with highest dimensions because clusters are created from the top of the lattice. Then it calculates the benefit of $N(v)$. If the benefit is less than the minimum benefit($MinBen$), it is marked as a noise. Otherwise, a cluster starts at v, and all the unclassified child views are put into a list of candidate views. Then, one view from the candidate views is picked up and benefit is calculated. If it is a core view, all the unclassified child views are included in the list of candidate views. Otherwise, it is marked as classified. The process continues until the list becomes empty. This way one cluster is formed. Similarly, other clusters are formed. At the end, core views of the clusters are selected for materialization. Here, each view will require to compute the neighborhood only once. So, average run time complexity of the algorithm is $O(VlogV)$.

The algorithm needs two important parameters - $MinBen$ and $MaxD$. $MinBen$ can be set to any arbitrary positive value according to requirement. However. optimum value can be calculated in the same way as cost of a view is calculated in $PVMA$. Similarly, optimum value for $MaxD$ can be determined in the same way as Eps has been determined in [EKS$^+$96].

7.4 Experimental Results

Setup : Performance of *DVMAFC* and *PVMA* was compared with two synthetic datasets (TD1 and TD2) and a PIV machine with 256 MB RAM.

Test data :Two synthetic data sets(TD1 and TD2) were used for the experiments. Each of them contained 8 dimensions without any hierarchy, one measure attribute and 2 lacs tuples. Each of 255 possible views was indexed from 1 to 255 . Values of each dimension and measure attribute were chosen randomly. It was assumed that queries on any view were equally likely. The analytical formula presented in [SDN98, LS96] was used to estimate the size of views. One view(query)

```
Input: Lattice of the views, $V$. Access frequency of the views, $Fv$. $MaxD$ and
$MinBen$. Output: $S$.

Set all views of $V$ as leader;
$S = \{$ fact table $\}$; $Temp$= "True";
$clid$ = Get a new cluster id;
Do while there is a leader view
        Find leader view $v$ with smallest in size (R(v)) among leader views;
           with maximum dimensions;
        $Temp$= CreateCluster($V$, $v$,$clid$, $MaxD$, $MinBen$);
        If $Temp$ = "True" then
           clid = Get a new cluster id;
        Endif
End DO

CreateCluster($V$, $v$.$clid$. $MaxD$, $MinBen$ )
(Form the cluster with cluster id as $clid$)
If benefit($N(v)$) < $MinBen$ Then
        $v$.noise= "True";
        Return "False";
Else
        $v$.classified= "True"; $S$=$S \cup v$;
        $seeds$= $\{w | w \in N(v)$ and w.classified= "False" $\}$;
        For all $s \in seeds$ set s.classified= True;
        While Empty($seeds$)="False" Do
            For each $s \in seeds$
               If $benefit(N(s)) \geq MinBen$ then
                  $S = S \cup s$; $Results = \{w | w \in N(s)\}$;
                  For each $r \in Results$
                     If $r$.classified = "False" then
                        $seeds = seeds \cup r$; $r$.classified= "True";
                     Endif
                  EndFor
               Endif
               $seeds = seeds$-s;
            Endfor
        EndWhile
        Return "True";
Endif
```

Algorithm 7.2: *DVMAFC*

database was created with about 1000 views to calculate access frequencies of the views and frequent sub-views. Frequent sub-views and their frequency were calculated using *Modified_Bit_Assoc* algorithm with minimum support as 5%. A constant value for $MinBen$ was taken, because this parameter also does not change even if some views have been materialized. bf and T_{rba} were also not considered because these values are constant for all the views and do not affect the selection of views.

Experimental results are shown in the figures 7.4 on the next page, 7.5 on page 158, 7.6 on page 158 & 7.7 on page 159. *Figures* 7.4 on the next page & 7.5 on page 158 gives the average query cost(in '000 tuples). *Figures* 7:6 on page 158 & 7.7 on page 159 reports the execution time.

Observations : Experimental results showed that both the algorithms selected almost same views and average query costs were also almost same for both the algorithms. In case of TD1(*Figure* 7.4 on the next page), *PVMA* selected slightly better views than that of *DVMFC*, resulting in slightly better performance in terms of average query cost. In case of TD2(*Figure* 7.5 on page 158), *PVMA* outperformed *DVMAFC* marginally in the beginning, when number of materialized views was small. However, as the number of materialized views increased. *DVMAFC* outperformed *PVMA* in terms of average query cost. This could be attributed to the selection of better views by *DVMAFC*. Another point to be noted is that average query cost becomes almost constant with the increase of number of materialized views. This shows that materialization of too many views does not reduce the query cost. As far as execution time (*Figures* 7.6 on page 158 & 7.7 on page 159) is concerned, *DVMAFC* takes much less time than that of *PVMA*. This is the main advantage of the *DVMAFC* over *PVMA*. The gain in execution time could be attributed to the difference in the time complexities of the algorithms.

7.5 Discussion

This chapter has presented a view materialization algorithm called *DVMAFC*. which has used density concept to select better views. The most important feature of the algorithm is the use of frequency count of the views to select better views. To find frequency count of the views, the frequent itemsets finding algorithms reported in the previous chapters(*Chapter* 3) may be of great help. Followings are the other important features of the algorithm.

- Complexity of the algorithm is only O($n\log n$), where n is the number of views.
- As far as view selection is concerned, it selects almost same views as that of *PVMA*.
- The algorithm is scalable due to its low complexity.

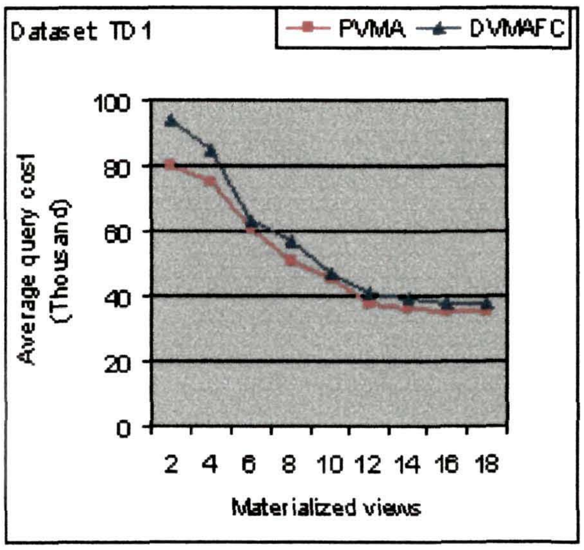


Figure 7.4: Average Query Cost ('000 tuples) of *DVMAFC* & *PVMA* - I

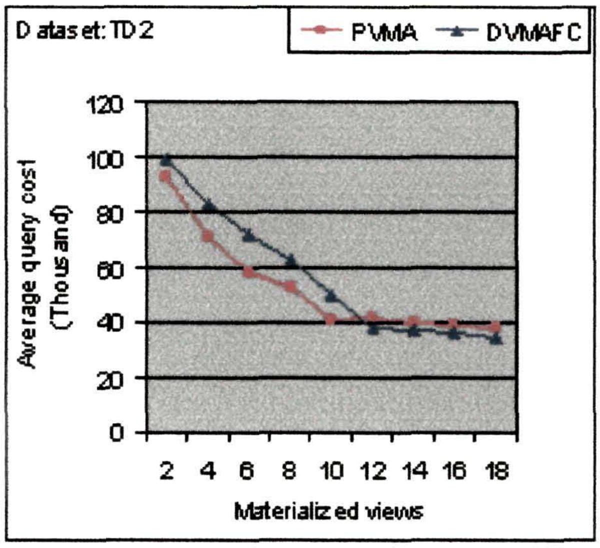


Figure 7.5: Average Query Cost ('000 tuples) of *DVMAFC* & *PVMA* - II

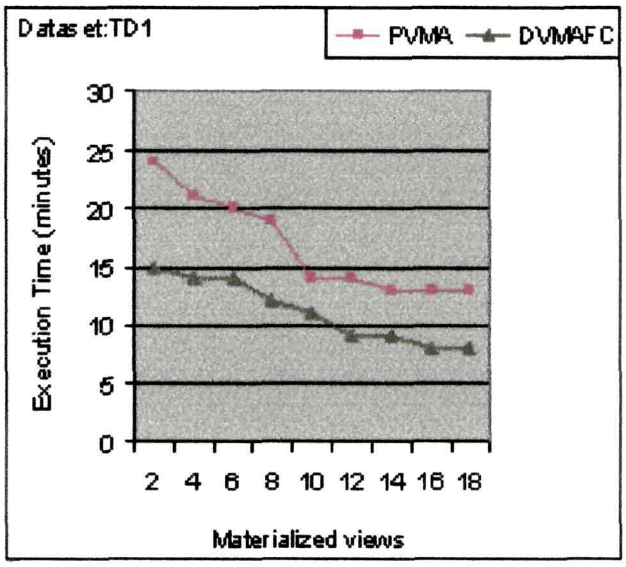


Figure 7.6: Execution Times of *DVMAFC* & *PVMA* - I

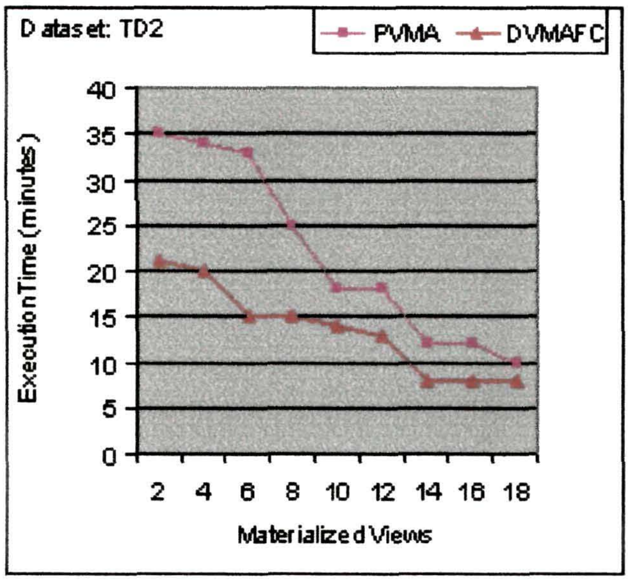


Figure 7.7: Execution Time of *DVMAFC* & *PVMA* - II

Chapter 8

Conclusion and Future Works

Association rule mining in large databases is an important data mining technique and it is used extensively in the field of knowledge discovery. Some of the uses of association rules are understanding customers' buying patterns, detecting crime patterns in a particular city, detecting frauds in credit card systems, etc.

8.1 Finding Frequent Itemsets Plays an Important Role in Association Rule Mining

Generally, association rule mining is a two step process - finding frequent itemsets and finding association rules among the frequent itemsets. Out of these two steps. finding frequent itemsets is more important, difficult and challenging because if the frequent itemsets are available, finding association rules is a trivial task. There exist algorithms to find frequent itemsets from large databases. Among them, *Apriori* is one of the earliest and important algorithms. The algorithm is robust enough to find all frequent itemsets. The algorithm first finds candidate itemsets and then finds frequent itemsets from those candidate sets. However. it has been observed that the ratio of number of candidate itemsets to that of frequent itemsets is very high. In other words, very few candidate itemsets become actually frequent itemsets. There exist algorithms such as *Bit_AssocRule*. which use bitmap technique to find frequent itemsets. There also exist algorithms which find frequent itemsets without candidate generation. *FP-growth* is one such algorithm. However, it has been observed that performance of the algorithm degrades with the increase of minimum support.

8.1.1 Solutions Provided

Following solutions are provided to address the above problems.

- A faster version of *Apriori* algorithm, which generates less number of candidate sets has been proposed.
- A modified and faster version of *Bit-AssocRule* algorithm has also been proposed.
- To improve the performance of *FP-growth*, a vertical partition based *FP-growth* has been proposed.

8.1.2 Partitioning is Another Good Approach

Horizontal partitioning of databases is a good approach to find frequent itemsets from large databases. However, it does not perform well for high dimensional databases. This problem has been addressed by vertical partitioning of the databases. Vertical partitioning gives better results than horizontal partitioning in case of high dimensional databases.

8.2 Finding Frequent Itemsets for Dynamic Databases

One important feature of most of the databases is that they are dynamic in nature because records are added, updated, deleted very frequently. During the study of the algorithms, it has been observed that existing algorithms are not efficient to find frequent itemsets in dynamic databases. Moreover, to the best of our knowledge, no algorithm exists for distributed dynamic databases. Among the existing algorithms for finding frequent itemsets in dynamic databases, *Borders* is the most important one. However, the algorithm scans the old database very frequently. To address this problem, a modified version of the algorithm has been proposed. The modified version does not require to scan the database frequently. Another problem is that *Borders* cannot be used directly for distributed dynamic databases. To solve this problem, a fully distributed version of *Borders* has been proposed. This distributed version can also be used for centralized database by partitioning the database and placing the partitions in different sites.

8.3 Feature Selection

Selecting relevant features is an important task in Decision Support Systems. There exist algorithms to select relevant features. These algorithms work for different types of databases and use different criteria to select relevant features. Moreover, these algorithms are very complex to implement. The thesis has reported a very simple algorithm to select relevant features using support count of the features. The algorithm is very simple and comparable to its counterparts in terms of relevant feature selections.

8.4 View Materialization

View materialization is an important technique used in data warehouse systems to reduce query response time. There exist algorithms for this purpose. All these algorithms have to work under some constraints such as disk-space constraint. Moreover, it has been observed that no algorithm has used support of the views for view selection. So, an algorithm has been proposed to select views for materialization. The important feature of the algorithm is that it has used density concept and support count of the views.

8.5 Future Works

Association rule mining is a vast area of research. So, it is not possible to cover every aspect of it in a stipulated period of time. Although it has been tried to cover as many aspects as possible, yet there are ample scopes for future works. Followings are some of the future works.

1. To develop a frequent itemset finding algorithm, which will generate the frequent itemsets without candidate generations for any high dimensional large market-basket databases and will also be capable of handling the minimum support condition i.e. the performance of the algorithm will not degrade even if value of minimum support count varies.

2. To extend the existing developments reported so far in the preceding chapters, to enable to work over spatial data, temporal data and any high dimensional categorical data.

3. To introduce soft-computing approach in the frequent itemset generation as well as in rule generation to discover more comprehensive and interesting patterns.

4. To develop better and robust dynamic rule mining algorithm over market-basket data as well as other huge data sources.

5. To incorporate data clustering technique or functional dependency approach to enable association mining over categorical and mixed types of data.

6. To analyze and develop quantitative association rule mining algorithms.

7. To develop a better feature selection technique using linear and non-linear manifolding techniques.

8. To explore the possibility of developing deviation analyzer (association rule mining based) for intrusion detection system.

List of Publications

1. A Das and D K Bhattacharyya. Efficient rule mining with modified *Apriori*. In *Proceedings of 1st National Workshop on Soft Data Mining and Intelligent Systems (SDIS'01)*, pages 117-122, Tezpur. India, September 2001.

2. A Das and D K Bhattacharyya. An efficient algorithm for rule mining. In *Proceedings of 5th International Conference on Information Technology (CIT 2002)*, pages 319-320, Bhubneswar, India, December 2002.

3. A Das and D K Bhattacharyya. Efficient rule mining for dynamic databases. In *Proceedings of International Conference on Information Technology (ITPC 2003)*, pages 25-30, Kathmandu, Nepal. May 2003.

4. A Das. D K Bhattacharyya and J Kalita. Horizontal Vs vertical partitioning in association rule mining: a comparison. In *Proceedings of CINC2003*. pages 30-36, USA, August 2003.

5. A Das and D K Bhattacharyya. Rule mining for dynamic databases. In *Proceedings of International Workshop on Distributed Computing (IWDC2004)*, LNCS 3326, Springer, pages 46-51, Kolkata, India, December 2004.

6. A Das and D K Bhattacharyya. Feature selection using frequency count. In *Proceedings of 12th International Conference on Advanced Computing and Communications(ADCOM2004)*. pages 620-624, Ahmedabad, India, December 2004.

7. A Das and D K Bhattacharyya. Faster algorithms for association rule mining. In *Proceedings of 12th International Conference on Advanced Computing and Communications (ADCOM2004)*, pages 629-635, Ahmedabad, India, December 2004.

8. A Das and D K Bhattacharyya. Rule mining for dynamic databases. *Australian Journal for Information Systems*, 13(1):19-39, September, 2005.

9. A Das and D K Bhattacharyya. Density-based view materialization. In *Proceedings of 1st International Conference on Pattern Recognition and*

Machine Intelligence (PReMI2005), LNCS 3776, Springer, pages 589-594, Kolkata, India, December 2005.

10. A Das and D K Bhattacharyya. Finding frequent itemsets over high dimensional data using partitioning *FP-growth*. In *Proceedings of National Workshop on Trends in Advanced Computing (NWTAC 2006)*, pages 130-138, Narosa, Tezpur, India, January 2006.

11. A Das and D K Bhattacharyya. View materialization using density concept. *Information Visualization* (Communicated).

12. A Das and D K Bhattacharyya. Using frequency count in feature selection. *Australian Journal for Information Systems* (Communicated).

Bibliography

[AD91] H Almuallim and T G Dietterich. Learning with many irrelevant features. In *Proceedings of 9th National Conference On Artificial Intelligence*, pages 547-552, Anahein, California, 1991.

[AIS93] R Agarwal, T Imielinski, and A Swami. Mining association rules between sets of items in large databases. In *ACM SIGMOD Conference on Management of Data*, pages 207–216, Washington, May 1993.

[AK93] T Anand and G Kahn. Opportunity explorer: Navigating large databases using knowledge discovery templates. In *Proceedings of AAAI-93 Workshop in Knowledge Discovery in Databases*. pages 45–51, Washington, DC, July 1993.

[AMS+94] R Agarwal, H Mannila, R Shrikant, et al. Fast algorithms for mining association rules in large databases. In *20th International Conference on Very Large Databases*, pages 487–499. Chile, September 1994.

[AMS+96] R Agarwal, H Mannila, R Shrikant, et al. Fast discovery of association rules. In *Advances in Knowledge Discovery and Data Mining*. pages 307–328. AAAI Press, 1996.

[AS95] R Agarwal and R Shrikant. Mining sequential patterns. In *Proceedings 11th International Conference Data Engineering*, pages 3–14. Taipei, Taiwan, March 1995.

[AYJ00] S Amer-Yahia and T Johnson. Optimizing queries on compressed bitmaps. In *Proceedings of 20th VLDB Conference*. pages 329–338. Cairo, Egypt, September 2000.

[BA99] R J Bayardo Jr. and R Agarwal. Mining the most interesting rules. In *Proceedings of The 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 145–154. San Diego United States, 1999.

[BAG99] R J Bayardo, R Agarwal, and D Gunopulos. Constraint-based rule mining in large, dense database. In *Proceedings of 15th International Conference on Data Engineering*, pages 188–197, Washington, DC. 1999.

[BB96] G Brassard and P Bratley. *Fundamentals of Algorithms*. Prentice Hall, New Jersy, 1996.

[BFO+83] L Breiman, J H Freidman, R A Olshen, et al. *Classification and Regression Trees*. Wordsworth. Belmont, 1983.

[Bit92] D Bitton. *Bridging the Gap between Database Theory and Practice*. MIT Press, 1992.

[BMS97] S Brin, R Motowani, and C Silverstein. Beyond market basket: Generalizing association rules to correlation. In *Proceeding of ACM-SIGMOD International Conference on Management of Data*. pages 265–276, Tucson, AZ, May 1997.

[BMU+97] S Brin, R Motwani, J D Ullman, et al. Dynamic itemset counting and implication rules for market basket data. In *Proceedings of 1997 International Conference on Management of Data (ACM SIGMOD)*, pages 255–264, Arizona, USA, May 1997.

[Bor] C Borgelt. An implementation of FP-growth algorithm. http://fuzzy.cs.uni-magdeburg.de/ borgelt/papers/fpgrowth.pdf.

[CB91] P Clerk and R Boswell. Rule induction with CN2: Some recent improvements. In *Proceedings of 5th European Conference*, pages 151–163, Porto, Portugal, March 1991.

[CHN+96a] D W Cheung, J Han, V T Ng, et al. A fast distributed algorithm for mining association rules. In *Proceedings of 4th International Conference on Parallel and Distributed Information Systems*, pages 31–42, Miami Beach, USA, 1996.

[CHN+96b] D W Cheung, J Han, V T Ng, et al. Maintenance of discovered association rules in large databases: An incremental updating technique. In *12th International Conference on Data Engineering*, pages 106–114, New Orleans, Louisiana, 1996.

[CLK97] D W Cheung. S D Lee, and B Kao. A general incremental technique for maintaining discovered association rules. In *Proceedings of the 5th International Conference on Database System for Advanced Applications*, pages 185–194, Melbourn, Australia, 1997.

[CNF+96] D W Cheung, V T Ng. A W Fu, et al. Efficient mining of association rules in distributed databases. *IEEE Transactions on Knowledge and Data Engineering*, 8(6):911–921. December 1996.

[CS94] S Choudhury and K Shim. Including group-by in query optimization. In *Proceedings of the 20th International Conferences on Very Large Databases (VLDB)*. pages 354-366, Santiago, Chile, 1994.

[DB04] A Das and D K Bhattacharyya. Faster algorithms for association rule mining. In *Proceedings of 12th International Conference on Advanced Computing and Communications(ADCOM2004)*, pages 629–635, Ahmedabad. India, December 2004.

[Doa92] J Doak. An evaluation of feature selection methods and their application to computer security. Technical Report CSE-92-18, CA: University of California, Department of Computer Science, 1992.

[DT99] L Dehaspe and H Toivonen. Discovery of frequent datalog patterns. *Data Mining and Knowledge Discovery*, 3(1):7–36, 1999.

[Ege91] M J Egenhofer. Reasoning about binary topological relations. In *Proceedings of 2nd International Symposium on Advances in Spatial Databases, SSD'91*, pages 143–160, Zurich, Switzerland, August 1991.

[EH94] M J Egenhofer and J R Herring. Categorizing binary topological relations between regions, lines and points in geographic databases. Technical Report 94-1, U S National Center For Geographic Information Analysis, 1994.

[EKS+96] M Ester, H P Krigel, J Sander, et al. A density- based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of 2nd International Conference on Knowledge Discovery and Data Mining*, pages 226–231, Portland, 1996.

[ES02] C I Ezeife and Y Su. Mining incremental association rules with generalized FP-tree. In *Proceedings of 15th Canadian Conference on Artificial Intelligence, AI2002*, pages 147–160, Calgary, Canada, May 2002.

[FAA+97] R Feldman, Y Aumann, A Amir, et al. Efficient algorithms for discovering frequent sets in incremental databases. In *Proceedings of ACM SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery (DMKD)*, pages 59–66, 1997.

[FAL+99] R Feldman, Y Aumann, O Lipshtat, et al. Borders: An efficient algorithm for association generation in dynamic databases. *Journal of Intelligent Information System*, 12(1):61–73, 1999.

[Fla98] P Flach. From extensional to intensional knowledge: Inductive logic programming techniques and their approaches to deductive databases. In *Transaction and Change in Logic Databases*, volume LNCS 1472. Springer-Verlag. Berlin, 1998.

[FMM+96] T Fukuda, Y Morimoto, S Morishita, et al. Data mining using two-dimensional optimized association rules :schemes algorithms and visualization. In *Proceedings 1996, ACM-SIGMOD International Conference Management of data (SIGMOD'96)*, pages 13–23, Montreal. Canada, June 1996.

[FR94] A S Fotheringham and P A Rogerson. *Spatial Analysis and GIS*. Taylor and Prancis. 1994.

[FWD93] U M Fayyad, N Weir, and G Djorgovski. SKICAT: A machine learning system for automated cataloging for large scale sky serveys. In *Proceedings of 10th International Conference On Machine Learning (ICML 1993)*, pages 112–119, June 1993.

[GCB+97] J Gray, S Chaudhury, A Bosworth, et al. Data cube: A relational aggrègation operator generalizing group-by, cross-tab and sub-totals. *Data Mining and Knowledge Discovery*, 1(1):29–54, 1997.

[GHQ95] A Gupta, V Harinarayan, and D Quass. Aggregate-query processing in data warehousing environments. In *Proceedings of 21st International VLDB Conference*, pages 358–369, California, USA, 1995.

[GK84] S C Gupta and V K Kapoor. *Fundamentals of Mathematical Statistics*. Sultan Chand and Sons, New Delhi, 1984.

[GN04] A Ghosh and B Nath. Multi-objective rule mining using genetic algorithm. *Information Sciences: An International Journal*, 163:123–133, June 2004.

[Gra94] G Graefe. Volcano, an extensible and parallel query evaluation system. *IEEE Transaction on Knowledge and Data Engineering*. 6(1):120–135, 1994.

[GYC+03] G Gau, J Xu Yu, C H Choi, et al. An efficient and interactive A* algorithm with pruning power: Materialized view selection revisited. In *Proceedings of 8th International Conference On Database Systems for Advanced Application (DASFAA)*, page 231. 2003.

[HCC92] J Han, Y Cai, and N Cercone. Knowledge discovery in databases: An attribute oriented approach. In *Proceedings of 18th VLDB Conference*, pages 547–559, Vancouver, Canada. 1992.

[HF95] J Han and Y Fu. Discovery of multiple level of association rule from large databases. In *Proceedings of International Conference on VLDB (VLDB'95)*, pages 420–431, Zurich, September 1995.

[HK01] J Han and M Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, San Fransisco, Usa, 2001.

[HKK00] E H Han, G Karypis, and V Kumar. Scalable parallel data mining for association rules. *IEEE Transactions on Knowledge and Data Engineering*, 12(3):337–352, 2000.

[HKS97] J Han, K Koperski, and N Stephanovic. GEOMINER: A system prototype for spatial data mining. In *Proceeding of The ACM-SIGMOD International Conference on Management of Data.* pages 553–556, 1997.

[HLL03] X Hu, T Y Lin, and E Louie. Bitmap techniques for optimizing decision support queries and association rule algorithm. In *Proceedings of International Databases Engineering and Application Symposium.* pages 17–23. Hongkong, July 2003.

[HPY00] J Han. J Pei, and Y Yin. Mining frequent patterns without candidate generation. In *Proceedings of The 2000 ACM-SGMOD International Conference on Management of Data*, pages 1–12, Texas, USA. May 2000.

[HRU96] V Harinarayan. A Rajaraman, and J D Ullman. Implementing data cubes efficiently. In *Proceedings 1996 ACM-SIGMOD International Management of Data*, pages 205–216, Montreal, Canada, 1996.

[HS93] M Houtsma and A Swami. Set oriented mining of association rules. Technical Report RJ9567, IBM Almaden Research Center. California, October 1993.

[Inm96] W H Inmon. *Building The Data Warehouse.* John Wiley & Sons, New York, 1996.

[JDO99] C Jermaine, A Data, and E Omiecinski. A novel index supporting high volume data warehouse insertion. In *Proceedings of 25 VLDB Confernece*, pages 235–246, Sanfrancisco, USA, 1999.

[JKP94] G H John, R Kohavi, and K Pfleger. Irrelevant features and subset selection problem. In *Proceedings of The 11th International Conference On Machine Learning*, pages 121–129. 1994.

[Joh98] T Johnson. Performance measurement of compressed bitmap indices. In *Proceedings of 25th VLDB Conference*, pages 278–289. Edingburg, September 1998.

[KAH96] K Koperski, J Adhikary, and J Han. Spatial data mining: Progress and challenges. In *Proceedings of Workshop on Research Issues on Data Mining and Knowledge Discovery*, pages 1–10, Montreal. Canada, 1996.

[KBJ$^+$99] A J Knobbe, H Blockeel, A P J M Siebes, et al. Multi-relational data mining. Technical Report INS-R9908, CWI, 1999.

[KH95] K Koperski and J Han. Discovery of spatial association rule in geographic information databases. In *Advances in Spatial Databases*. LNCS 951, pages 47–66. Springer-Verlag, 1995.

[KHC97] M Kamber, J Han, and J Y Chiang. Metarule-guided mining of multidimensional association rules using data cubes. In *Proceedings of 1997 International Conference Knowledge Discovery and Data Mining (KDD'97)*, pages 207–210, Newport, CA. August 1997.

[KR92] K Kira and L A Rendell. The feature selection problem: Traditional methods and a new algorithm. In *Proceedings of 10th National Conference on Artificial Intelligence*, pages 129–134, Menlo Park, 1992.

[KS95] R Kohavi and D Sommerfield. Feature subset selection using the wrapper method: Over fitting and dynamic search space topology. In *Proceedings. of 1st International Conference On Knowledge Discovery and Data Mining*, pages 192–197, 1995.

[KS96] D Koller and M Sahani. Towards optimal feature selection. In *Proceedings of International Conference on Machine Learning*, pages 284–292, Bari, Italy, July 1996.

[LCK98] S D Lee, D W Cheung, and B Kao. Is sampling useful in data mining? a case in the maintenance of discovered association rules. *Data Mining and Knowledge Discovery*, 2(3):233–262, 1998.

[LHC97] B Liu, W Hsu, and S Chen. Using general impression to analyze discovered classification rules. In *Proceedings of International Conference On Knowledge Discovery and Data Mining*, pages 31–36, Newport Beach, CA, August 1997.

[LK98] D Lin and Z M Kedem. Pincer Search: A new algorithm for discovering the maximum frequent set. *Lecture Notes in Computer Science*. 1377:105–119, 1998.

[LL00] E Louie and T Y Lin. Finding association rules using fast bit computation: Machine-oriented modeling. In *LNCS*, volume 1932, pages 486–494. Springler-Verlag, 2000.

[LS96] H Liu and R Setiono. A probabilistic approach to feature selection - a filter solution. In *Proceedings of International Conference on Machine Learning*, pages 319–327, Bari, Italy, 1996.

[Man97] H Mannila. Inductive databases and condensed representation for data mining. In *Proceedings of International Symposium on Logic Programming*, pages 21–30, Port Washington, USA, 1997.

[MEL$^+$00] D Malerba, F Esposito, A Lanza, et al. Discovering geographic knowledge: The INGENS system. In *LNCS*, volume 1932/2000, page 40. Springer-Verlag, Berlin, 2000.

[MFM$^+$98] Y Morimoto, T Fukuda, H Matsuzawa, et al. Algorithms for mining association rules for binary segmentation of huge categorical databases. In *Proceedings of 24th International Conference on Very Large Databases*, pages 380–391, Francisco, 1998.

[ML01] D Malerba and F A Lisi. An ILP method for spatial association rule mining, 2001.

[Mor98] S Morishita. On classification and regression. In *Lecture Notes in Artificial Intelligence*, LNAI 1532, pages 40–57. Springer-Verlag, Berlin, 1998.

[MR87] H Manila and K J Raiha. Dependency inference. In *Proceedings of 13th International Conference on VLDB*, pages 155–158, Brigton, England, 1987.

[MY97] R J Miller and Y Yang. Association rules over interval data. In *Proceedings of SIGMOD International Conference on Management of Data(SIGMOD'97)*, pages 452–461, Tucson, Az, USA, 1997.

[MZ98] T Morzy and M Zakrzewicz. Group bitmap index: A structure for association rule retrieval. In *Proceedings of 4th ACMSIGMOD International Conference on Knowledge Discovery and Data Mining*, pages 284–288, Newyork. USA, 1998.

[NF77] P M Narendra and K Fukunaga. A branch and bound algorithm for feature selection. *IEEE Transaction on Computers*, C-26(9):917–922, September 1977.

[NG95] P O Neil and G Graefe. Multi-table joins through bitmapped join indices. *ACM SIGMOD*, 24(3):8–11, September 1995.

[NH94] R T Ng and J Han. Efficient and effective clustering method for spatial data mining. In *International Conference on VLDB*, pages 144–155, Santiago, Chile, September 1994.

[NLH+98] R Ng, L V Lakshmanan, J Han, et al. Exploratory mining and pruning optimization of constrained association rules. In *Proceedings of 1998 ACM SIGMOD International Conference on Management of Data*, pages 13–24, Seattle, Washington, June 1998.

[NM] A Nakaya and S Morishita. Fast parallel search for correlated association rules. Unpublished Manuscript.

[PCY95a] J S Park, M S Chen, and P S Yu. An effective hash-based algorithm for mining association rules. In *Proceedings of 1995 ACM-SIGMOD International Conference Management of Data*, pages 175–186, San Jose, CA, May 1995.

[PCY95b] J S Park, M S Chen, and P S Yu. Efficient parallel data mining for association rules. In *Proceedings of 1995 International Conference on Information and Knowledge Management*, pages 31–36, Maltimore, November 1995.

[Pia91] G Piatsky-Shapiro. Discovery, analysis and presentation of strong ruels. In G Piatsky-Shapiro, editor, *Knowledge Discovery in Databases*. AAAI/MIT Press, 1991.

[PM94] G Piatetsky-Shapiro and C J Matheus. The interestingness of deviations. In *AAAI Workshop on Knowledge Discovery in Database*. pages 25–36. Seattle,WA, 1994.

[Pop98] L Popelimsky. Knowledge discovery in spatial data by means of ILP. In *Principles of Data and Mining Knowledge Discovery*, LNCS 1510. pages 271–279. Springer-Verlag, Berlin, 1998.

[Puj01] A K Pujari. *Data Mining Techniques*. University Press, Hyderabad. India, 2001.

[RS02] R Rastogi and K Shim. Mining optimized association rules with categorical and numeric attributes. *Knowledge and Data Engineering*. 14(1):29–50, 2002.

[SA95] R Shrikant and R Agarwal. Mining generalized association rules. In *Proceedings of 1995 International. Conference on Very Large Databases(VLDB'95)*, pages 407–419, Zurich, September 1995.

[SA96] R Shrikant and R Agrawal. Mining quantitative association rules in large relational tables. In *Proceedings of 1996 ACM SIGMOD International Conference on Management of Data(SIGMOD'96)*, pages 1–12. Montreal, Canada, June 1996.

[SDN90] J Shieinvald, B Dom, and W Niblack. A modeling approach to feature selection. In *Proceedings of 10th International Conference on Pattern Recognition*, pages 535–539, June 1990.

[SDN98] A Shukla, P M Despande, and J F Naughton. Materialized view selection for multidimensional dataset. In *Proceedings of 24th VLDB Conference*, pages 488–499, Sanfrancisco, USA, 1998.

[SMO$^+$94] W M Shen, B Mitbandar, K Ong, et al. Using metaqueries to integrate inductive learning and deductive database technology. In *Proceedings of AAAI'94 Workshop on Knowledge Discovery in Databases*, pages 335–346, Seattle, Washington, July 1994.

[SON95] A Savasere, E Omiecinski, and S Navathe. An efficient algorithm for mining association rules in large databases. In *Proceedings of*

21st Conference on Very Large Databases, pages 432–444, Zurich. September 1995.

[SVA97] R Shrikant, Q Vu, and R Agarwal. Mining association rules with item constraints. In *Proceedings of International Conference Knowledge Discovery and Data Mining(KDD'97)*, pages 67-73, Newport Beach.CA, August 1997.

[TBA$^+$97] S Thomas, S Bodagala, K Alsabti, et al. An efficient algorithm for the incremental updation of association rules in large databases. In *Proceedings of The 3rd International Conference On KDD And Data Mining (KDD'97)*, pages 263-266, New port, 1997.

[TBB$^+$91] S B Thrun, J Bala, E Bloedorn, et al. The Monk's problem : A performance comparison of different algorithms. Technical Report CS-91-197, CMU-CS, Carnegie Mellon University, December 1991.

[URT99] H Uchiyama, K Runapongsa, and T J Teorey. A progressive view materialization algorithm. In *Proceedings of 2nd International Data Warehousing and OLAP Workshop*, pages 36–41, Kansas City. November 1999.

[Web95] G I Webb. OPUS: An efficient admissible algorithm for unordered search. *Journal For Artificial Intelligence Research*, 3:431–465, 1995.

[YFM$^+$97] K Yoda. T Fukuda. Y Morimoto, et al. Computing optimized rectilinear regions for association rules. In *Proceedings of 3rd International Conference on Knowledge Discovering and Data Mining (KDD'97)*, pages 96–103, Newport, CA, August 1997.

[YL95] W Yan and P A Larson. Eager aggregation and lazy aggregation. In *Proceedings of 21st International Conference on VLDB*, pages 345–357, Zurich, September 1995.

[Zak00] M J Zaki. Generating non-redundant association rules. In *Proceedings of 6th ACM-SIGMOD International Conference on KDD*, pages 34–43, Boston, USA, 2000.

[ZE01] Z Zhou and C I Ezeife. A low-scan incremental association rule maintenance method based on apriori property. In *Proceedings of 14th Canadian Conference on Artificial Intelligence, AI2001*, pages 26–35, Ottawa, Canada, June 2001.

[ZH99] M J Zaki and C J Hsiao. CHARM: An efficient algorithm for closed association rule mining. Technical Report 99-10, Rensselaer Polytechnic Institute, New York, October 1999.

[ZPO+97] M J Zaki, S Parthasarathy, M Ogihara, et al. New algorithms for fast discovery of association rules. In *Proceedings of 3rd International Conference on KDD*, pages 283–296, 1997.