COMPARISON OF DIFFERENT ALGORITHMS FOR EXPLOITING THE HIDDEN TRENDS IN DATA SOURCES

Emrah ÖZSEVİM

JUNE, 2003

Comparison of Different Algorithms for Exploting the Hidden Trends in Data Sources

By
Emrah ÖZSEVİM

A Dissertation Submitted to the
Graduate School in Partial Fulfillment of the
Requirements for the Degree of
MASTER OF SCIENCE

Department: Computer Engineering
Major: Computer Software

İzmir Institute of Technology
İzmir, Turkey

May 2003

We approve the thesis of **Emrah ÖZSEVİM**

Date of Signature

10.06.2003

.....
Prof. Dr. Halis PÜSKÜLCÜ
Supervisor
Department of Computer Engineering

10.06.2003

.....
Prof. Dr. Sıtkı AYTAÇ
Department of Computer Engineering

10.06.2003

.....
Prof. Dr. Fikret İKİZ
Ege University
Department of Computer Engineering

10.06.2003

.....
Prof. Dr. Sıtkı AYTAÇ
Head of Department

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude and respects to my advisor Prof. Dr. Halis Püskülcü for his enduring support, advice, suggestions, encouragement and supervision throughout this study.

I do thank to all academic staff of Department of Computer Engineering of İzmir Institute of Technology who have contributed to the accomplishment of this thesis.

Finally, I also owe my special thanks to my family, Mrs. Mürvet ÖZSEVİM and Mr. Bahattin ÖZSEVİM, for their encouragements and patience.

ABSTRACT

The growth of large-scale transactional databases, time-series databases and other kinds of databases has been giving rise to the development of several efficient algorithms that cope with the computationally expensive task of association rule mining.

In this study, different algorithms, Apriori, FP-tree and CHARM, for exploiting the hidden trends such as *frequent itemsets, frequent patterns, closed frequent itemsets* respectively, were discussed and their performances were evaluated. The perfomances of the algorithms were measured at different support levels, and the algorithms were tested on different data sets (on both synthetic and real data sets). The algorihms were compared according to their, *data preparation performances*, *mining performance, run time performances* and *knowledge extraction capabilities*.

The Apriori algorithm is the most prevalent algorithm of association rule mining which makes multiple passes over the database aiming at finding the set of *frequent itemsets* for each level. The FP-Tree algorithm is a scalable algorithm which finds the crucial information as regards the complete set of *prefix paths*, *conditional pattern bases* and *frequent patterns* by using a compact FP-Tree based mining method. The CHARM is a novel algorithm which brings remarkable improvements over existing association rule mining algorithms by proving the fact that mining the set of *closed frequent itemsets* is adequate instead of mining the set of all frequent itemsets.

Related to our experimental results, we conclude that the Apriori algorithm demonstrates a good performance on sparse data sets. The Fp-tree algorithm extracts less association in comparison to Apriori, however it is completelty a feasable solution that facilitates mining dense data sets at low support levels. On the other hand, the CHARM algorithm is an appropriate algorithm for mining closed frequent itemsets (a substantial portion of frequent itemsets) on both sparse and dense data sets even at low levels of support.

“Büyük Veri Gruplarındaki Gizli İlişkilerin Ortaya Çıkarılmasında Kullanılan Algoritmaların Karşılaştırılması”

ÖZ

Geniş ölçekli hareketli (transactional), zaman-serisi ve diğer türdeki veri tabanlarındaki büyüme, ilişki bulma (*association rule*) madenciliği konusunun yoğun işlem sürecinin üstesinden gelebilen etkili birçok algoritmanın geliştirilmesini beraberinde getirmiştir. Bu çalışmada, sırasıyla *frequent itemsets*, *frequent patterns*, *closed frequent itemsets* gibi gizli eğilimleri ortaya çıkarmada kullanılan Apriori, FP-tree ve CHARM gibi çeşitli algoritmalar tartışılmakta ve performansları değerlendirilmektedir. Söz konusu algoritmaların performansları farklı (sentetik ve gerçek) veri grupları üzerinde test edilmiş ve çeşitli eşik (*support*) seviyeleri için ölçülmüştür. Algoritmalar *veri hazırlama*, *madencilik*, *toplam çalışma performansları* ve *bilgi çıkarım yetileri* açısından karşılaştırılmıştır.

İlişki bulma (*association rule*) madenciliğinin en temel algoritması olan Apriori, her seviyedeki *frequent itemset* grubunu bulma amacına yönelik olarak veri tabanı üzerinde çoklu geçişler yapmaktadır. FP-tree algoritması bellekte az yer kaplayan *FP-tree tabanlı* bir madencilik yöntemi kullanarak tüm *prefix paths*, *conditional pattern bases* ve *frequent patterns* gruplarına ilişkin önemli bilgileri bulan ölçeklendirilebilir bir algoritmadır. CHARM, tüm *frequent itemset* grubunu ortaya çıkarmak yerine *closed frequent itemset* grubunu ortaya çıkarmanın yeterli olabileceğini kanıtlayarak mevcut ilişki bulma (*association rule*) madenciliği algoritmaları üzerine kayda değer gelişmeler ekleyen yepyeni bir algoritmadır.

Deneysel sonuçlarımıza dayanarak, Apriori algoritmasının seyrek (*sparse*) veri grupları üzerinde iyi performans gösterdiği sonucuna varmış bulunmaktayız. FP-tree algoritması, Apriori algoritmasına kıyasla daha az ilişki bulmakla beraber, yoğun (*dense*) veri gruplarında düşük eşik (*support*) seviyelerinde de madenciliği mümkün kılan tek algoritmadır. Diğer taraftan, CHARM algoritması hem seyrek (*sparse*) hem de yoğun (*dense*) veri grupları üzerinde düşük eşik (*support*) seviyelerinde *closed frequent itemset* grubu (*frequent itemset* gurubun büyük bir kısmı ya da tamamı) hakkındaki bilgiyi çıkarmak için uygun bulunmuştur.

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1

INTRODUCTION

1.1 Motivation

In the last decade, we have seen an explosive growth in our capabilities to both generate and collect data. Advances in scientific data collection (e.g. from remote sensors or from space satellites), the widespread introduction of bar-codes for almost all commercial products, and the computerization of many business (e.g. credit card purchases) and government transactions (e.g. tax returns) have generated a flood of data. Advances in storage technology, such as faster, higher capacity and cheaper storage devices (e.g. magnetic discs, CD-ROMS), better database management systems, and data warehousing technology, have allowed us to transform this data deluge into "mountains" of stored data.

Representative examples of such huge data, mentioned above, can easily be found in the business world. One of the largest databases in the world was created by Wal-Mart (a U.S. retailer) which handles over 20 million transactions a day [Babcock, 1994]. Most health-care transactions in the U.S. are being stored in computers, yielding multi-gigabyte databases, which many large companies are beginning to analyze in order to control their costs and improve quality. Mobil Oil Corporation, is developing a data warehouse capable of storing over 100 terabytes of data related to oil exploration [Harrison, 1993].

Such volumes of data clearly exceed the performance of traditional manual methods of data analysis such as spreadsheets and ad-hoc queries. Those methods can create informative reports from data, but cannot analyze the contents of those reports in order to focus on important knowledge. For this reason, a significant need exists for a new generation of techniques and tools with the ability to intelligently and automatically assist human in analyzing the mountains of data nuggets of useful knowledge. These techniques are the subject of the emerging field of data mining and knowledge discovery in databases (KDD for short).

Data mining may have simply three major components: Clustering, Classification, Link Analysis (Association Rule Mining or Sequence Analysis).

Mining frequent patterns or itemsets is an essential problem in many data mining applications including association rules, correlations, sequential rules, episodes, multi-dimensional patterns and many other important discovery tasks. Actually, the results of the investigation presented by "www.kdnuggets.com" (given in Figure 1.1 below) shows us that the usage of "*Association Rule Mining Techniques*" possesses the 9% of the overall data mining studies.

Usage Statistics of Data Mining Techniques

Technique	August 2001	October 2002
Clustering	Na	12%
Neural Networks	13%	9%
Decision Trees/Rules	19%	16%
Logistic Regression	14%	9%
Statistics	17%	12%
Bayesian nets	6%	3%
Visualization	8%	6%
Nearest Neighbor	Na	5%
Association Rules	9%	9%
Hybrid methods	4%	3%
Text Mining	2%	4%
Sequence Analysis	Na	3%
Genetic Algorithms	Na	3%
Naive Bayes	Na	2%
Web mining	5%	2%
Agents	1%	Na
Other	2%	2%

Figure 1.1: The percentage of "association rules" among other data mining techniques (http://www.kdnuggets.com).

The general structure of the association rule mining algorithms is that they make multiple passes over the data. Each pass, starts with a seed set for generating new potentially large sets, called *candidate sets*. Then the *support*s of candidate sets during the pass over the data are found (scan step). At the end of the pass, actually large candidate sets are determined (prune step). These candidates become the *seed* for the next pass (for the join step).

1.2 Objective

In this study, we considered the problem of Mining Association Rules between items in large databases of sales transactions. The task of mining association rules over market basket data was first introduced by Rakesh Agrawal,

Tomasz Imielinski and Arun Swami in 1993 [Agrawal, Imielinski, and Swami, 1993(May)], and frequently used since then.

An example of such an association rule might be that 98% of customers that purchase tires and auto accesories also get their automative services done. Other applications include catalog design, add-on sales, store layout, and customer segmentation based on buying patterns. The databases involved in these applications are very large. Therefore, it is imperative to have fast algorithms to handle these computations.

In this study, three algorithms for mining associaiton rules have been discussed (analyzed in detail), implemented and their performances are compared using both different data sets and different treshold (known as *support*) levels. Underlying data sets are "Northwind" database (which is a *sample data set* obtained from Microsoft Access Database), and different kinds of *synthetic data sets.*

1.3 Structure of the Study

As is mentioned in Section 1.1, there are many data mining techniques. These techniques were explained more or less on their own. In Chapter 2, we introduced the scope and the mission of Data Mining and Knowledge Discovery in Databases(KDD) including both the descriptive and the predictive methods.

Chapter 3 discusses the Associaiton Rule Mining Task, introduces Apriori, FP-Tree and CHARM algorithms in detail.

In Chapter 4, features of the sample data set and the synthetic data set are visualized and the generation of a synthetic data sets are described. Afterwards, the proposal with respect to our modification in the CHARM algorithm is given.

Chapter 5 is concerned with the presentation and discussion of all the experimental results. The comparison of the algorithms is also under the scope of this chapter.

Chapter 6 summarizes our main findings and the benefits of the Apriori, FP-tree and CHARM algorithms. It also points out some future research issues.

Chapter 2

DATA MINING AND KNOWLEDGE DISCOVERY

2.1 Overview and Definitions

Databases today can range in size into the terabytes — more than 1,000,000,000,000 bytes of data. Within these masses of data, there may lie hidden information of strategic importance. But when there are so many trees, how meaningful conclusions can be drawn about the forest? The newest answer is data mining, which is being used both to increase revenues and to reduce costs. The potential returns are enormous. Challenging organizations worldwide are already using data mining to locate and appeal to higher-value customers, to reconfigure their product offerings to increase sales, and to minimize their losses [Kennedy, Reed and Roy, 1998] [Fayyad, Piatetsky-Shapiro, Smyth and Uthurusamy, 1996] [Dhar and Stein, 1997].

Data mining is a process that uses a variety of data analysis tools to discover patterns and relationships in data that may be used to make valid predictions. The first and simplest analytical step in data mining is to describe the data — summarize its statistical attributes (such as means and standard deviations), visually review it using charts and graphs, and look for potentially meaningful links among variables (such as values that often occur together).

In data mining process, collecting, exploring and selecting the right data are critically important. But data description alone cannot provide an action plan. A predictive model based on patterns has to be determined from known results, then that model has to be tested on results outside the original sample. A good model should never be confused with reality, but it can be a useful guide for understanding the business.

Historically, the notion of finding useful patterns (or nuggets of knowledge) in raw data has been given various names, including knowledge discovery in databases, data mining, knowledge extraction, information discovery, information harvesting, data archeology, and data pattern processing.

The term *Knowledge Discovery in Databases*, or KDD for short, was first used in 1989 to refer to the broad process of finding knowledge in data, and to emphasize the "high-level" application of particular data mining methods. The term *Data Mining* has been commonly used by statisticians, data analysts and the Management Information Systems (MIS) community, while KDD has been mostly used by artificial intelligence and machine learning researchers.

Some other important terms frequently used throughout this chapter are as follows:

- ***Pattern*:** Pattern is an expression E in a language L describing facts in a subset F_E of F. E is called a pattern if it is simpler than the enumarion of all facts in F_E. For example the expression "If income < $t, then person has defaulted on the loan" would be one such pattern for an appropriate choice of t.
- ***Model*:** Model can be considered as a function prototype in data mining. A model can be descriptive or predictive. A descriptive model helps us to understand processes or behaviors. For example, an association model describes consumer behaviour. A descriptive model is an equation or set of rules that makes it possible to predict an unseen or unmeasured value(the dependent variable or output) from known values(independent variables or input).

  A pattern can be thought of as instantiation of a model, e.g., $f(x) = 3x^2 + x$ is a pattern whereas $f(x) = ax^2 + bx$ is a model.
- ***Induction*:** A technique that infers generalizations from the information in the data.
- ***Deduction*:** Deduction infers information that is a logical consequence of the data.
- ***Training data set*:** A data set used to estimate or train a model.
- ***Test data set*:** A data set independent of the training data set, used to fine-tune the estimates of the model parameters(i.e., weights).

2.2 Goals of Data Mining

Data mining is a tool, not a magic wand. It does not sit in the database watching what happens and send the chief an e-mail when it sees an interesting

pattern. However, it doesn't eliminate the need to understand data, or to understand analytical methods. In this sense, data mining assists business analysts with finding patterns and relationships in the data.

Predictive relationships found via data mining are not necessarily causes of an action or behavior [Westphal and Blaxton, 1998]. For example, data mining might determine that males with incomes between X$ and Y$ who subscribe to certain magazines are likely purchasers of another product. By taking the advantage of this pattern, say by aiming at the sales who fit the pattern, there is no need in terms of market manager to consider any of the other factors cause the customers to buy the product.

To ensure meaningful results, it's vital to understand the data. The quality of the output will often be sensitive to outliers, irrelevant columns or columns that vary together (such as age and date of birth), the way the data encoded, and the data left in and the data excluded. Data mining algorithms vary in their sensitivity to such data issues, but it is still unwise to depend on a data mining product to make all the right decisions on its own.

Data mining will not automatically discover solutions without guidance. Rather than setting the vague goal, data mining might be used to find the characteristics of customers who make large purchase on a specific pattern. Also, the patterns found as a result of the data mining process may be very different. For this reason, it should be better to consider and evaluate all the results.

Although a good data mining tool shelters the analysts from the low level statistical techniques, it requires to understand the workings of the tools and the algorithms on which they are based.

Data mining does not replace skilled satatistical analysts or managers, but rather gives them a powerful new tool to improve the job they are doing. Any company that knows its business and its customers is already aware of many important, high-payoff patterns that its employees have observed over the years. What data mining can do is to confirm such empirical observations and find new, subtle(hard to determine) patterns.

2.3 Data Mining and Data Warehousing

A related area is *data warehousing*, which refers to the recently popular MIS trend for collecting and cleaning transactional data and making them available for online retrieval [Fayyad, Piatetsky-Shapiro, Smyth and Uthurusamy, 1996].

Frequently, the data to be mined is first extracted from an enterprise data warehouse into a data mining database or data mart (Figure 2.1). There is some real benefit if the data is already part of a data warehouse. The problems of cleansing(or cleaning: refers to a step in preparing data for a data mining activity) data for a data warehouse and for data mining are very similar. If the data has already been cleansed for a data warehouse, then it most likely will not need further cleaning in order to be mined [Pyle, 1999].

The data mining database may be a logical rather than a physical subset of the data warehouse, provided that the data warehouse or DataBase Management System (DBMS) can support the additional resource demands of data mining. If it cannot, then a separate data mining database should better be chosen.

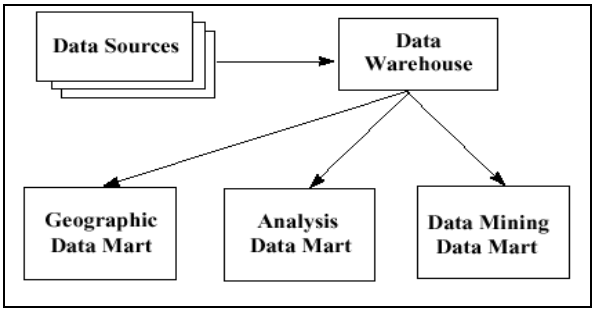


Figure 2.1: Data mining data mart extracted from a data warehouse.

A data warehouse is not a requirement for data mining. Setting up a large data warehouse that consolidates data from multiple sources, resolves data integrity problems, and loads the data into a query database can be an enormous task, sometimes taking years and costing millions of dollars. Data can be mined from one or more operational or transactional databases(Figure 2.2). This new database behaves as a kind of data mart.

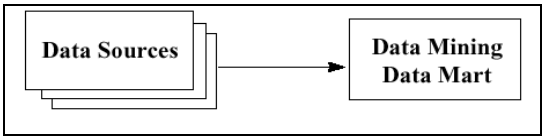


Figure 2.2: Data mining data mart extracted from operational databases.

2.4 Data Mining and OLAP

One of the most common questions is related with the difference between data mining and OLAP (On-Line Analytical Processing). However, they are very different tools that can also complement each other.

After a new set of principles proposed by Codd(1993) [Codd, 1993], OLAP tools focus on providing multi-dimensional data analysis, which is superior to SQL (Standart Query Language) in computing summaries and breakdowns along many dimensions.

OLAP is part of the spectrum of decision support tools. Traditional query and report tools describe what is in a database. OLAP goes further; it is used to answer why certain things are true. The user forms a hypothesis about a relationship and verifies it with a series of queries against the data. For example, an analyst might want to determine the factors that lead to loan defaults. He or she might initially hypothesize that people with low incomes are bad credit risks and analyze the database with OLAP to verify (or disprove) this assumption. If that hypothesis were not borne out by the data, the analyst might then look at high debt as the determinant of risk. If the data did not support this guess either, he or she might then try debt and income together as the best predictor of bad credit risks.

In other words, the OLAP analyst generates a series of hypothetical patterns and relationships and uses queries against the database to verify them or disprove them. OLAP analysis is essentially a deductive process. But what happens when the number of variables being analyzed is in the dozens or even hundreds? It becomes much more difficult and time-consuming to find a good hypothesis (let alone be confident that there is not a better explanation than the one found), and analyze the database with OLAP to verify or disprove it [Berry and Linoff, 1997] [Codd, 1993].

Data mining is different from OLAP because rather than verifying hypothetical patterns, it uses the data itself to uncover such patterns. It is essentially an inductive process. For example, an analyst who wants to identify

the risk factors for loan default uses a data mining tool. The data mining tool might discover that people with high debt and low incomes are at bad credit risks, but it might go further and also discover a pattern the analyst could not think, such as that age is also a determinant of risk. Here is where data mining and OLAP can complement each other.

2.5 Data Mining, Machine Learning and Statistics

Data mining takes advantage of advances in the fields of artificial intelligence (AI) and statistics. Both disciplines have been working on problems of pattern recognition and classification. Both communities have made great contributions to the understanding and application of neural nets and decision trees [Westphal and Blaxton, 1998].

Data mining does not replace traditional statistical techniques. Rather, it is the result of major changes in the statistical investigations. The development of most statistical techniques was, until recently, based on elegant theory and analytical methods that worked quite well on the large amounts of data being analyzed. The increasing power of computers and their lower cost, has been giving rise to the need of analyzing enormous data sets with millions of rows. As a result, new techniques based on brute-force exploration of possible solutions have been developed [Quinlan, 1993].

New techniques include relatively recent algorithms like neural nets and decision trees, and new approaches to older algorithms such as discriminant analysis. With the increasing power of computers on the huge volumes of available data, these techniques can approximate almost any functional form or interaction on their own. Traditional statistical techniques rely on the modeler to specify the functional form and interactions.

The key point is that, data mining is the application of these and other AI and statistical techniques to common business problems in a fashion that makes these techniques available to the skilled knowledge worker as well as the trained statistics professional. Data mining is a tool for increasing the productivity of people trying to build predictive models.

2.6 Data Mining and Hardware/Software Trends

A key enabler of data mining is the major progress in hardware price and performance. The dramatic 99% drop in the price of computer disk storage in just the last few years has radically changed the economics of collecting and storing massive amounts of data. At $10/megabyte, one terabyte of data costs $10,000,000 to store. This doesn't even include the savings in real estate from greater storage capacities [Westphal and Blaxton, 1998].

The drop in the cost of computer processing has been equally dramatic. Generation of chips for each task greatly increases the power of the CPU, while allowing further drops on the cost curve. This is also reflected in the price of RAM (random access memory), where the cost of a megabyte has dropped from hundreds of dollars to around a dollar in just a few years. While the power of the individual CPU has greatly increased, the real advances in scalability stem from parallel computer architectures. Virtually, all servers today support multiple CPUs using symmetric multi-processing, and clusters of these servers can be created that allow hundreds of CPUs to work on finding patterns in the data.

Advances in database management systems to take advantage of this hardware parallelism also benefit data mining. If there is a large or complex data mining problem requiring a great deal of access to an existing database, native DBMS access provides the best possible performance in terms of data mining [Fayyad, Piatetsky-Shapiro, Smyth and Uthurusamy, 1996].

As a result, by the help of these trends many of the performance barriers for finding patterns in large amounts of data have been eliminated.

2.7 Data Mining Applications

Data mining is increasingly popular because of the substantial contribution it can make. It can be used to control costs as well as providing contribution to revenue increases.

Many organizations are using data mining to help manage all phases of the customer life cycle, including acquiring new customers, increasing revenue from existing customers, and retaining good customers. By determining characteristics of good customers (profiling), a company can target prospects with similar characteristics. By profiling customers who have bought a particular product the

company can focus attention on similar customers who have not bought that product (cross-selling). By profiling customers who have left, a company can act to retain customers who are at risk for leaving, because it is usually far less expensive to retain a customer than acquire a new one [Westphal and Blaxton, 1998].

Data mining offers value across a broad spectrum of industries. Telecommunications and credit card companies are two of the leaders in applying data mining to detect the frequent usages in their services. Insurance companies and stock exchanges are also interested in applying this technology to reduce fraud. Medical applications are another fruitful area: data mining can be used to predict the effectiveness of surgical procedures, medical tests or associations between genes. Companies active in the financial markets use data mining to determine market and industry characteristics as well as to predict individual company and stock performance. Retailers are making more use of data mining to decide which products to stock in particular stores (and even how to place them within a store), as well as to assess the effectiveness of promotions and coupons. Pharmaceutical firms are mining large databases of chemical compounds and of genetic material to discover substances that might be candidates for development as agents for the treatments of disease.

2.8 Successful Data Mining

There are two keys to have success in data mining. First is coming up with a precise formulation of the problem. A focused statement usually results in the best payoff. The second key is using the right data. After the data is chosen, it may be needed to be transformed and combined in significant ways. The more the model builder can play with the data, build models, evaluate results, and work with the data some more, the better the resulting model will be [Pyle, 1999].

Consequently, the degree to which a data mining tool supports this interactive data exploration is more important than the algorithm it uses. Ideally, the data exploration tools (graphics/visualization, query/OLAP) are well-integrated with the analytics or algorithms that build the models [Cleveland, 1994].

2.9 Visualization and Summarization of Data

Before building predictive models, the data must be summarized and understood. The summarization studies start by gathering a variety of numerical summaries (including descriptive statistics such as averages, standard deviations and so forth) and looking at the distribution of the data. It may be required to produce cross tabulations (pivot tables) for multi-dimensional data [Wainer, 1997].

Graphing and visualization tools are a vital aid in data preparation. Their importance to effective data analysis cannot be overemphasized. Data visualization most often provides the "Aha!" leading to new insights and success. Some of the common and very useful graphical displays of data are histograms or box plots that display distributions of values. Scatter plots in two or three dimensions of different pairs of variables can be used in order to understand the nature of the data set.

Visualization works because it exploits the broader information bandwidth of graphics as opposed to text or numbers. It allows people to see the forest and zoom in on the trees. Patterns (or itemsets), relationships, exceptional values and missing values are often easier to perceive when shown graphically, rather than as lists of numbers and text.

The problem in using visualization stems from the essential fact that models have many dimensions or variables because it is restricted to show these dimensions on a two-dimensional computer screen or paper. For example, one may wish to view the relationship between credit risk and age, sex, marital status, own-or-rent, years in job, etc. Consequently, visualization tools must use clever representations to collapse n dimensions into two.

Powerful and sophisticated data visualization tools are being developed increasingly, but they often require people to train their eyes through practice in order to understand the information being conveyed.

2.10 The Primary Methods of Data Mining

The two high-level primary goals of data mining in practice tend to be prediction and description [Fayyad, Piatetsky-Shapiro, Smyth and Uthurusamy, 1996]. Prediction involves using some variables or fields in the database to predict

unknown or future values of other variables of interest. Description focuses on finding human-interpretable patterns describing the data. The relative importance of prediction and description for particular data mining applications can vary considerably. However, in the context of data mining, description tends to be more important than prediction. This is in contrast to pattern recognition and machine learning applications (such as speech recognition) where prediction is often the primary goal (see Lehmann 1990, for a discussion from a statistical perspective) [Lehman, 1990].

In predictive models, the values or classes to be predicted are called the *response or dependent variables.* The values used to make the prediction are called the *predictor* or *independent variables.* Predictive models are built, or trained, using data for which the value of the response variable is already known. This kind of training is sometimes referred to as *supervised learning*, because calculated or estimated values are compared with the known results. Some of the essential predictive methods described in this chapter are classification, time-series, neural networks, decision trees, k-nearest neighbour, regression, logistic regression, discriminant analysis and genetic algorithms.

By contrast, descriptive methods are sometimes referred to as *unsupervised learning* because there is no already-known result to guide the algorithms. Some of the important descriptive data mining methods explained in this chapter are clustering and link analysis (association rule discovery and sequence analysis).

2.10.1 Descriptive Methods

2.10.1.1 Clustering

Clustering divides a database into different groups. The goal of clustering is to find groups that are very different from each other, and whose members are very similar to each other. Unlike classification (see Predictive Methods below), in clustering it is unknown what the clusters will be, or by which attributes the data will be clustered. Consequently, someone who is knowledgeable in this technique must interpret the clusters. Often it is necessary to modify the clustering by excluding variables that have been employed to group instances, because upon examination the user identifies them as irrelevant or not meaningful.

After the clusters that reasonably segment the database have been found, these clusters may then be used to classify new data. Some of the common algorithms used to perform clustering include Kohonen Feature Maps and K-means [Berry and Linoff, 1997].

Clustering must not be confused with segmentation. Segmentation refers to the general problem of identifying groups that have common characteristics. Clustering is a way to segment data into groups that are not previously defined, whereas classification is a way to segment data by assigning it to groups that are already defined.

2.10.1.2 Link Analysis

Link analysis is a descriptive approach to exploring data that can help to identify relationships among values in a database. The two most common approaches to link analysis are association discovery (or association rule mining) and sequence discovery (or sequence analysis).

Association discovery finds rules about items that appear together in an event such as a purchase transaction. Market-basket analysis is a well-known example of association discovery.

Sequence analysis is same as association discovery, except that the time sequence of events is also considered. For example, “20% of the people who buy the product X buy the product Y within 4 months”. In sequence analysis the input data is a set of sequences, called data-sequences. Each data sequence is an ordered list of transactions (or itemsets), where each transaction is a set of items (literals). A sequential pattern also consists of a list of sets of items. The problem is to find all sequential patterns with a user-specified minimum support, where the support of a sequential pattern is the percentage of data sequences that contain the pattern. An example of such a pattern is that customers typically rent “Star Wars”, then “Empire Strikes Back” and then “Return of Jedi”.

Associations are written as A => B, where A is called the antecedent or left-hand side (LHS), and B is called the *consequent* or right-hand side (RHS). For example, in the association rule “If people buy a hammer then they buy nails”, the antecedent is “buy a hammer” and the *consequent* is “buy nails.”

Determining the proportion of transactions that contain a particular item or itemset is easy: they are simply counted. The frequency with which a particular association (e.g., the itemset "hammers and nails") appears in the database is called its *support or prevalence*. If 15 transactions out of 1,000 consist of "hammer and nails", the support for this association would be 1.5%. A low level of support (say, one transaction out of a million) that may indicate the particular association isn't very important — or it may indicate the presence of bad data (e.g., "male and pregnant").

To discover meaningful rules, also the relative frequency of occurrence of the items and their combinations must be considered. With occurrence of item A (the antecedent), how often does item B (the consequent) occur? In other words, what is the conditional predictability of B, given A? Using the above example, this would mean asking "When people buy a hammer, how often do they also buy nails?" Another term for this conditional predictability is *confidence*. Confidence is calculated as a ratio: (frequency of A and B)/(frequency of A).

Lift is another measure of the power of an association. The greater the lift, the greater the influence that the occurrence of A has on the likelihood that B will occur. Lift is calculated as the ratio (confidence of A => B) / (frequency of B). Figure 2.3 given below illustrates these concepts in more detail.

It can be seen from the below associations that the likelihood that a hammer buyer will also purchase nails (30%) is greater than the likelihood that someone buying nails will also purchase a hammer (19%). The prevalence of this hammer-and-nails association (i.e., the support is 1.5%) is high enough to suggest a meaningful rule.

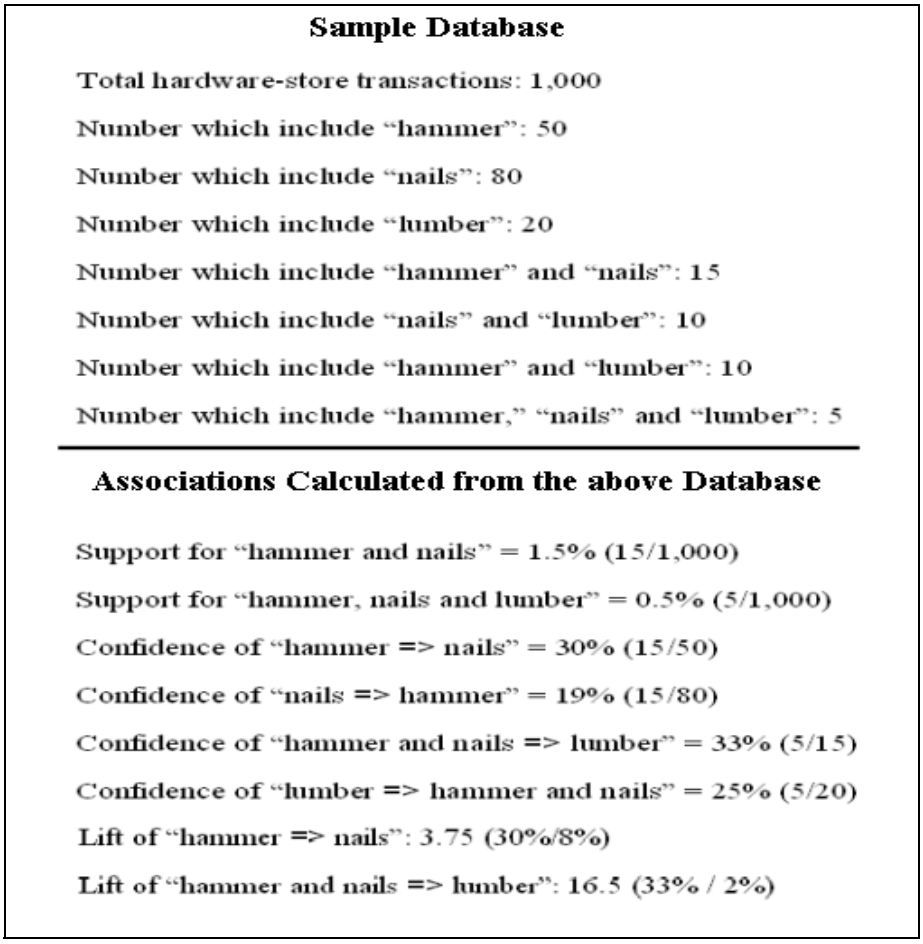


Figure 2.3: Illustration of concepts on a sample database.

Association algorithms find these rules by doing the equivalent of sorting the data while counting occurrences so that they can calculate confidence and support. The efficiency with which they can do this is one of the differentiators among algorithms. This is especially important because of the combinatorial explosion that results in enormous numbers of rules, even for market baskets in the express lane. Some algorithms will create a database of rules, confidence factors, and support that can be queried (for example, "all associations in which ice cream is the consequent, that have a confidence factor of over 80% and a support of 2% or more").

Another common attribute of association rule generators is the ability to specify an item hierarchy. An item hierarchy allows to control the level of aggregation and experiment with different levels (with different support levels).

Association or sequence rules are not really rules, but rather descriptions of relationships in a particular database. There is no formal testing of models on

other data to increase the predictive power of these rules. Rather there is an implicit assumption that the past behavior will continue in the future.

It is often difficult to decide what to do with the discovered association rules. In store planning, for example, putting associated items physically close together may reduce the total value of market baskets — customers may buy less overall because they no longer pick up unplanned items while walking through the store in search of the desired items.

Graphical methods may also be very useful in seeing the structure of links [Cleveland, 1994]. In Figure 2.4 each of the circles represents a value or an event. The lines connecting them show a link. The thicker lines represent stronger or more frequent linkages, thus emphasizing potentially more important relationships such as associations. For instance, looking at an insurance database to detect potential fraud might reveal that a particular doctor and lawyer work together on an unusually large number of cases.

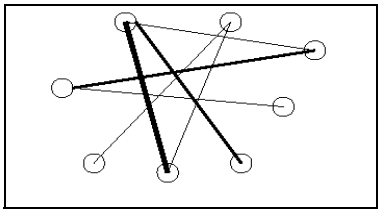


Figure 2.4: Linkage diagram.

2.10.2 Predictive Methods

2.10.2.1 Classification

Classification problems aim to identify the characteristics that indicate the group to which each case belongs. This pattern can be used both to understand the existing data and to predict how new instances will behave [Breiman, Friedman, Olshen and Stone, 1984]. For example, one may want to predict whether individuals can be classified as likely to respond to a direct mail solicitation, vulnerable to switching over to a competing long-distance phone service, or a good candidate for a surgical procedure.

Data mining creates classification models by examining already classified data (cases) and inductively finding a predictive pattern. These existing cases may come from an historical database, such as people who have already undergone a particular medical treatment or moved to a new long-distance service. They may

come from an experiment in which a sample of the entire database is tested in the real world and the results used to create a classifier. For example, a sample of a mailing list would be sent an offer, and the results of the mailing used to develop a classification model to be applied to the entire database. Sometimes an expert classifies a sample of the database, and this classification is then used to create the model which will be applied to the entire database.

2.10.2.2 Regression

Regression uses existing values to forecast what other values will be. In the simplest case, regression uses standard statistical techniques such as linear regression. Unfortunately, many real-world problems are not simply linear projections of previous values [Breiman, Friedman, Olshen and Stone, 1984]. For instance, sales volumes, stock prices, and product failure rates are all very difficult to predict because they may depend on complex interactions of multiple predictor variables. Therefore, more complex techniques (e.g., logistic regression, decision trees, or neural nets) may be necessary to forecast future values.

2.10.2.3 Time Series

Time series forecasting predicts unknown future values based on a time-varying series of predictors. Like regression, it uses known results to guide its predictions. Models must take into account the distinctive properties of time, especially the hierarchy of periods (including such varied definitions as the five- or seven-day work week, the thirteen-"month" year, etc.), seasonality, calendar effects such as holidays, date arithmetic, and special considerations such as how much of the past is relevant [Berry and Linoff, 1997] [Breiman, Friedman, Olshen and Stone, 1984].

2.10.2.4 Neural Networks

Neural networks are of particular interest because they offer a means of efficiently modeling large and complex problems in which there may be hundreds of predictor variables that have many interactions. Neural nets may be used in classification problems (where the output is a categorical variable) or for

regressions (where the output variable is continuous) [Kennedy, Reed and Roy, 1998] [Quinlan, 1993].

A neural network (Figure 2.4) starts with an input layer, where each node corresponds to a predictor variable. These input nodes are connected to a number of nodes in a hidden layer. Each input node is connected to every node in the hidden layer. The nodes in the hidden layer may be connected to nodes in another hidden layer, or to an output layer. The output layer consists of one or more response variables.

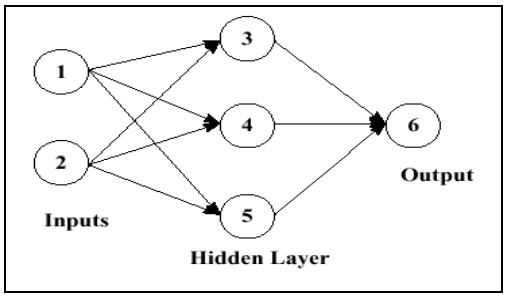


Figure 2.5: A neural network with one hidden layer.

Some of the applications where hundreds of variables may be input into models with thousands of parameters (node weights) include modeling of chemical plants, robots and financial markets, and pattern recognition problems such as speech, vision and handwritten character recognition [Kennedy, Reed and Roy, 1998].

Users must be conscious of several facts about neural networks:

First, neural networks are not easily interpreted. There is no explicit rationale given for the decisions or predictions a neural network makes.

Second, they tend to overfit the training data unless very obligatory measures, such as weight decay and/or cross validation, are used cleverly. This is due to the very large number of parameters of the neural network which will fit any data set arbitrarily well when allowed to train to convergence.

Third, neural networks require an extensive amount of training time unless the problem is very small. Once trained, however, they can provide predictions very quickly.

Fourth, they require a lot of data preparation time than other methods. In this sense, the most successful implementations of neural network involve very careful data cleansing(or cleaning), selection, preparation and pre-processing.

One advantage of neural network models is that they can easily be implemented to run on massively parallel computers with each node simultaneously doing its own calculations [Quinlan, 1993].

2.10.2.5 Decision Trees

Decision tree models are commonly used in data mining to examine the data and its rules that will be used to make predictions. A number of different algorithms may be used for building decision trees including CHAID (Chi-squared Automatic Interaction Detection), CART (Classification And Regression Trees), Quest, and C5.0 [Breiman, Friedman, Olshen and Stone, 1984] [Quinlan, 1993].

Decision trees are a way of representing a series of rules that lead to a class or value. For example, one may wish to classify loan applicants as good or as bad credit risks. Figure 2.6 shows a simple decision tree that solves this problem while illustrating all the basic components of a decision tree: the decision node, branches and leaves.

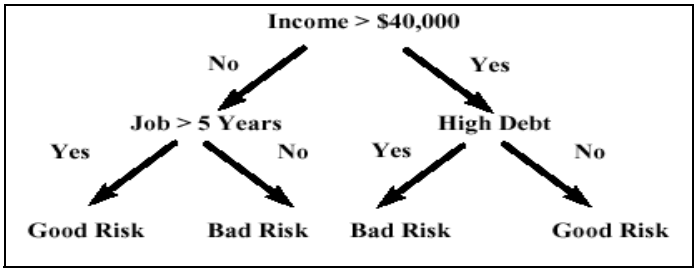


Figure 2.6: A simple decision tree.

Decision trees make few passes through the data (no more than one pass for each level of the tree) and they work well with many predictor variables. As a consequence, models can be built very quickly, making them suitable for large data sets [Breiman, Friedman, Olshen and Stone, 1984].

2.10.2.6 K-Nearest Neighbor and Memory-Based Reasoning (MBR)

When trying to solve new problems, people often look at solutions to similar problems that they have previously solved. K-nearest neighbor (k-NN) is a classification technique that uses a version of this same method. It decides in which class to place a new case by examining some number — the “k” in k-

nearest neighbor — of the most similar cases or neighbors (Figure 2.7). It counts the number of cases for each class, and assigns the new case to the same class to which most of its neighbors belong.

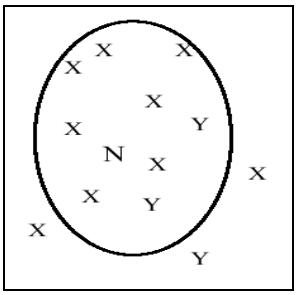


Figure 2.7: K-nearest neighbor (N is a new case. It would be assigned to the class X because the seven X's within the ellipse out number the two Y's).

The main purpose of applying k-NN is to find a measure of the distance between attributes in the data and then calculate it. While this is easy for numeric data, categorical variables need special handling. For example, what is the distance between blue and green? One must then have a way of summing the distance measures for the attributes. Once the distance between cases can be calculated, then, one may select the set of already classified cases to use as the basis for classifying new cases, decide how large a neighborhood in which to do the comparisons, and also decide how to count the neighbors themselves (e.g., more weight might be given to nearer neighbors than farther neighbors).

K-NN models are very easy to understand when there are few predictor variables. They are also useful for building models that involve non-standard data types, such as text. The only requirement for being able to include a data type is the existence of an appropriate metric.

2.10.2.7 Logistic Regression

Logistic regression is a generalization of linear regression. It is used primarily for predicting binary variables (with values such as yes/no or 0/1) and occasionally multi-class variables. Because the response variable is discrete, it cannot be modeled directly by linear regression. Therefore, rather than predicting whether the event itself (the response variable) will occur, the model to predict the logarithm of the odds of its occurrence is built. This logarithm is called the log odds. The odds ratio: *probability of an event occurring / probability of the event*

not occurring, has the same interpretation as in the more casual use of odds in games of chance or sporting events.

When it is said that the odds are 3 to 1 that a particular team will win a soccer game, it means that the probability of their winning is three times as great as the probability of losing. So it is deduced that they have a 75% chance of winning and a 25% chance of losing. Similar terminology can be applied to the chances of a particular type of customer (e.g., a customer with a given gender, income, marital status, etc.) replying to a mailing.

2.10.2.8 Discriminant Analysis

Discriminant analysis is the oldest mathematical classification technique, having been first published by R. A. Fisher in 1936 [Lehman, 1990] to classify the famous Iris botanical data into three species. It finds hyper-planes (e.g., lines in two dimensions, planes in three, etc.) that separate the classes. The resultant model is very easy to interpret because all the user has to do is to determine on which side of the line (or hyper-plane) a point falls. Training is simple and scalable. The technique is very sensitive to patterns in the data. It is used very often in certain disciplines such as medicine, the social sciences, and field biology.

Discriminant analysis is not popular in data mining due to three main reasons. First, it assumes that all of the predictor variables are normally distributed (i.e., their histograms look like bell-shaped curves), which may not be the case. Second, unordered categorical predictor variables (e.g., red /blue/green) cannot be used at all. Third, the boundaries that separate the classes are all linear forms (such as lines or planes), but sometimes the data just can't be separated that way.

2.10.2.9 Genetic Algorithms

Genetic algorithms are not used to find patterns, but rather to guide the learning process of data mining algorithms such as neural nets. Essentially, genetic algorithms act as a method for performing a guided search for good models in the solution space.

They are called genetic algorithms because they follow the pattern of biological evolution in which the members of one generation (of models) compete to pass on their characteristics to the next generation (of models), until the best (model) is found. The information to be passed on is contained in "chromosomes," which contain the parameters for building the right model.

While genetic algorithms are an interesting approach to generating and optimizing models, they add a lot of computational overhead.

Chapter 3

ASSOCIATION RULE MINING AND ALGORITHMS

3.1. Overview

Consider a supermarket with a large collection of items. Typical business decisions that the management bureau of such a supermarket has to make include: what to put on sale, how to design coupons, how to place merchandise on shelves in order to maximize the profit, etc. *Analysis of past transaction data is a commonly used approach in order to improve quality of such decisions*. Until recently, however, only global data about the cumulative sales during some time period (a day, a week, a month, etc.) was available on the computer. However, progress in bar-code technology has made it possible to store the so called basket data that stores items purchased on a per-transaction basis.

In this chapter, we consider the problem of "mining" a large collection of basket data type transactions for finding association rules between sets of items. We present three different association rule mining algorithms: *Apriori algorithm, Frequent Pattern Tree (FP-tree) algorithm and CHARM algorithm.* Although these algorithms serve to the same purpose, they are fundamentally different from each other and each algorithm brings new approaches and solutions to the association rule mining problem.

3.2 Association Rule Mining Problem

3.2.1 Problem Definition and Decomposition

The problem of finding all association rules falls within the purview of database mining [Anwar, Beck and Navathe, 1992] [Agrawal, Imielinski and Swami, 1993(December)] [Holsheimer and Siebes, 1994] [Michalski, Kerschberg, Kaufman and Ribeiro, 1992] [Stonebraker et al, 1993] [Tsur, 1990], also called knowledge discovery in databases [Han, Cai and Cercone, 1992] [Lubinsky, 1989] [Piatestsky-Shapiro, 1991]. Related work includes the induction of classification rules [Chen, Park and Yu, 1998] [Catlett, 1991] [Fayyad, Weir and Djorgovski, 1993] [Han, Cai and Cercone, 1992] [Quinlan, 1990], discovery of

causal structures [Cooper and Herskovits, 1992] [Pearl, 1992], learning of logical definitions [Muggleton and Feng, 1992] [Quinlan, 1990], fitting of functions to data [Langley, Simon, Bradshaw and Zytkow, 1987] [Schaffer, 1990], and clustering [Anwar, Beck and Navathe, 1992] [Cheeseman et al, 1988] [Fisher, 1987].

Association rule mining problem was first introduced in 1993 by R. Agrawal, T. Imielinski and A. Swami [Agrawal, Imielinski, and Swami, 1993(May)]. The following is a formal statement of the problem: Let $I = \{i_1, i_2, \ldots, i_m\}$ be a set of literals, called *item*s. Let D be a set of transactions, where each transaction T is a set of items such that $T \subseteq I$. Associated with, each transaction is a unique identifier, called its transaction identifier (*TID or tid)*. It is said that a transaction T contains X, a set of some items in I, if $X \subseteq T$. An association rule is an implication of the form $X \Rightarrow Y$, where $X \subset I$, $Y \subset I$, and $X \cap Y = \varnothing$. The rule $X \Rightarrow Y$ holds in the transaction set D with confidence c if c% of transactions in D that contain X also contain Y . The rule $X \Rightarrow Y$ has support s in the transaction set D if s% of transactions in D contain $X \cup Y$.

In a more simpler form the problem may be defined as follows: Given a set of transactions D, the problem of mining association rules is to generate all association rules that have support and confidence greater than the user-specified minimum support (called *minsup*) and minimum confidence (called *minconf*) respectively. D could be a data file, a relational table, or the result of a relational expression.

The problem of discovering all association rules can be decomposed into two sub-problems [Agrawal, Imielinski, and Swami, 1993(May)]:

1. Find all sets of items (*itemsets*) that have transaction support above minimum support. The support for an itemset is the number of transactions that contain the itemset. Itemsets with minimum support are called *large itemsets*, and all others *small itemsets*.

2. Use the large itemsets to generate the desired rules. The general idea is that if, say, ABCD and AB are large itemsets, then one can determine if the rule $AB \Rightarrow CD$ holds by computing the ratio conf = support(ABCD) / support(AB). If $conf \geq$ minconf, then the rule holds (the rule will surely have minimum support because ABCD is large).

3.2.2 Discovering Large Itemsets and the Notation Used

Algorithms for discovering large itemsets make multiple passes over the data. In the first pass, we count the support of individual items and determine which of them are large, i.e. have minimum support. In each subsequent pass, we start with a seed set of itemsets found to be large in the previous pass. We use this seed set is used for generating new potentially large itemsets, called *candidate itemsets*, and count the actual support for these candidate itemsets during the pass over the data. At the end of the pass, we determine which of the candidate itemsets are actually large, and they become the seed for the next pass. This process continues until no new large itemsets are found.

The Apriori algorithm and Apriori-like algorithms generate the candidate itemsets to be counted in a pass by using only the itemsets found large in the previous pass without considering the transactions in the database. The basic intuition is that *any subset of a large itemset must be large.* Therefore, the candidate itemsets having k items can be generated by *joining* large itemsets having k-1 items, and *deleting* those that contain any subset that is not large. This procedure results in generation of a much smaller number of candidate itemsets.

It is assumed that items in each transaction are kept sorted in their *lexicographic* order. It is straightforward to adapt these algorithms to the case where the database D is kept normalized and each database record is a <TID, item> pair, where TID is the identifier of the corresponding transaction.

The number of items in an itemset is called its *size*, and can itemset of size k is called a *k-itemset*. Items within an itemset are kept in lexicographic order. The notation c[1] · ... · c[2] · . . . · c[k] is used to represent a k-itemset c consisting of items c[1], c[2], . . .c[k], where c[1]<c[2]< . . . <c[k]. If c = X·Y and Y is an *m-itemset*, Y is also called an *m-extension* of X. Associated with each itemset is a count field to store the "support" for this itemset. The count field is initialized to *zero* when the itemset is first created. In Figure 3.1 the notation used in the Apriori and Apriori-like algorithms (*Apriori-like algorithms are variants of Apriori algorithm, they also depend on the candidate generation philosophy*) is summarized.

k-itemset	An itemset having k items.
L_k	Set of large k-itemsets (those with minimum support). Each member of this set has two fields: i) itemset and ii) support count
C_k	Set of candidate k-itemsets (potentially large itemsets) Each member of this set has two fields: i) itemset and ii) support count
$\overline{C}_k$	Set of candidate k-itemsets when the TIDs of the generating transactions are kept associated with the candidates.

Figure 3.1: Notation used in Apriori and Apriori-like algorithms.

3.3 Apriori Algorithm

An association rule mining algorithm, Apriori, was developed in 1993 by IBM's Quest Project Team [Agrawal, Imielinski and Swami, 1993(December)] for mining large transactional databases. Figure 3.2 gives the Apriori algorithm. The algorithm makes multiple passses over the database, and each pass has three steps: 1) *Scan Step,* 2) *Prune Step,* 3) *Join Step.*

In the first pass the algorithm simply counts item occurences to determine the frequent 1-itemsets (itemsets with 1 item). A subsequent pass, say pass k, consists of two phases. First, the frequent itemsets L_{k-1} (the set of all frequent (k-1)-itemsets) found in the (k-1)th pass are used to generate the candidate itemsets C_k, using the apriori-gen() function. This function first joins L_{k-1} with L_{k-1}, the joining condition being that the lexicographically ordered first k-2 items are the same. Next, it deletes all those itemsets from the results of the join step that have some (k-1) subset that is not in L_{k-1} yielding C_k. The algorithm now scans the database. For each transaction, it determines which of the candidates in C_k are contained in the transaction using the hash-tree data structure and increments the count of those candidates. At the end of each pass, C_k is examined to determine which of the candidates are frequent, yielding L_k. The algorithm terminates when L_k (the set of all frequent itemsets) becomes empty.

```
1) L1 = {large 1-itemsets};
2) for ( k = 2; Lk-1 ≠∅; k++ ) do begin
3)       Ck = apriori-gen(Lk-1); // New candidates
4)       for all transactions t ∈ D do begin
5)             Ct = subset(Ck, t); // Candidates contained in t
6)             for all candidates c ∈ Ct do
7)                   c.count++;
8)       end
9)       Lk = {c ∈ Ck | c:count ≥ minsup}
10) end
11) Answer = ∪k Lk;
```

Figure 3.2: Algorithm Apriori.

3.3.1 Apriori Candidate Generation

The apriori-gen function takes as argument L_{k-1}, the set of all large (k–1)-itemsets. It returns a superset of the set of all large k-itemsets. The function works as follows. First, in the join step, L_{k-1} is joined with L_{k-1}:

insert into C_k
select $p.item_1$, $p.item_2$, ..., $p.item_{k-1}$, $q.item_{k-1}$
from L_{k-1} p, L_{k-1} q
where $p.item_1 = q.item_1$, . . ., $p.item_{k-2} = q.item_{k-2}$, $p.item_{k-1} < q.item_{k-1}$;

Next, in the prune step, all itemsets $c \in C_k$ are deleted such that some (k-1)-subset of c is not in L_{k-1}:

for all itemsets $c \in C_k$ do
    for all (k-1)-subsets s of c do
        if ($s \notin L_{k-1}$) then
            delete c from C_k;

As an example, let L_3 be {{1 2 3}, {1 2 4}, {1 3 4}, {1 3 5}, {2 3 4}}. After the join step, C_4 will be {{1 2 3 4}, {1 3 4 5}}. The prune step will delete the itemset {1 3 4 5} because the itemset {1 4 5} is not in L_3. We will then be left with only {1 2 3 4} in C_4.

3.3.1.1 Correctness

It must be shown that $C_k \supseteq L_k$. Clearly, *any subset of a large itemset must also have minimum support*. Hence, if the algorithm extends each itemset in L_{k-1} with all possible items and then deletes all those whose (k-1)-subsets are not in L_{k-1}, we will be left with a superset of the itemsets in L_k.

The join step is equivalent to extending L_{k-1} with each item in the database and then deleting those itemsets for which the (k-1)-itemset obtained by deleting the (k-1)th item is not in L_{k-1}. The condition $p.item_{k-1} < q.item_{k-1}$ simply ensures that no duplicates are generated. Thus, after the join step, $C_k \supseteq L_k$. By similar reasoning, the prune step, where we delete from C_k all itemsets whose (k-1)-subsets are not in L_{k-1}, also does not delete any itemset that could be in L_k.

3.3.1.2 Counting Candidates of Multiple Sizes in One Pass

Rather than counting only candidates of size k in the kth pass, the algorithm also counts the candidates C'_{k+1}, where C'_{k+1} is generated from C_k, etc. Note that $C'_{k+1} \supseteq C_{k+1}$ since C_{k+1} is generated from L_k. This variation can pay off in the later passes when the cost of counting and keeping in memory additional $C'_{k+1} - C_{k+1}$ candidates becomes less than the cost of scanning the database.

3.3.1.3 Membership Test

The prune step requires testing that all (k−1)-subsets of a newly generated k-candidate-itemset are present in L_{k-1}. To make this membership test fast, large itemsets are stored in a hash table.

3.3.2 Apriori Subset Function

Candidate itemsets, C_k, are stored in a hash-tree. A node of the hash-tree either contains a list of itemsets (a leaf node) or a hash table (an interior node). In an interior node, each bucket of the hash table points to another node. The root of

the hash-tree is defined to be at depth 1. An interior node at depth d points to nodes at depth d+1. Itemsets are stored in the leaves. When an itemset c is added, the algorithm starts from the root and goes down the tree until it reaches a leaf. At an interior node at depth d, the algorithm decides which branch to follow by applying a hash function to the dth item of the itemset. All nodes are initially created as leaf nodes. When the number of itemsets in a leaf node exceeds a specified threshold, the leaf node is converted to an interior node.

Starting from the root node, the subset function finds all the candidates contained in a transaction t as follows. If we are at a leaf, the algorithm finds which of the itemsets in the leaf are contained in t and add references to them to the answer set. If we are at an interior node and we have reached it by hashing the item i, the algorithm hashes on each item that comes after i in t and recursively applies this procedure to the node in the corresponding bucket. For the root node, the algorithm hashes on every item in t.

To see why the subset function returns the desired set of references, consider what happens at the root node. For any itemset c contained in transaction t, the first item of c must be in t. At the root, by hashing on every item in t, the algorithm ensures that it only ignores itemsets that start with an item not in t. Similar arguments apply at lower depths. The only additional factor is that, since the items in any itemset are ordered, if we reach the current node by hashing the item i, the algorithm only needs to consider the items in t that occur after i.

If k is the size of a candidate itemset in the hash-tree, we can find in O(k) (in the order of k) time whether the itemset is contained in a transaction by using a temporary bitmap. Each bit of the bitmap corresponds to an item. The bitmap is created once for the data structure, and reinitialized for each transaction. This initialization takes O(size(transaction)) time for each transaction [Agrawal, Imielinski, and Swami, 1993(May)].

3.3.3 Apriori Buffer Management

In the candidate generation phase of pass k, Apriori needs storage for large itemsets L_{k-1} and the candidate itemsets C_k. In the counting phase, the algorithm needs storage for C_k and at least one page to buffer the database transactions.

First, assume that L_{k-1} fits in memory but that the set of candidates C_k does not. The apriori-gen function is modified to generate as many candidates of C_k as will fit in the buffer and the database is scanned to count the support of these candidates. Large itemsets resulting from these candidates are written to disk, while those candidates without minimum support are deleted. This procedure is repeated until all of C_k has been counted.

3.4 Frequent Pattern Tree Algorithm

The second algorithm evaluated is the Frequent Pattern Tree (FP-tree for short) algorithm which is proposed by J. Han, J. Pei, and Y. Yin in 1999 [Han, Pei and Yin, 1999]. FP-tree algorithm mines the frequent patterns without generating candidate sets.

According to J. Han, J. Pei, and Y. Yin, the algorithm is of a novel FP-tree structure, which is an extended prefix-tree structure for storing compressed, crucial information about frequent patterns. Such a structure facilitates an efficient FP-tree-based mining method, FP-growth, for mining the complete set of frequent patterns by pattern fragment growth. Efficiency of mining is achieved with three techniques: (1) a large database is compressed into a highly condensed, much smaller data structure, which avoids costly, repeated database scans, (2) the FP-tree-based mining adopts a pattern fragment growth method to avoid the costly generation of a large number of candidate sets, and (3) a partitioning-based, divide-and-conquer method is used to decompose the mining task into a set of smaller tasks for mining confined patterns in conditional databases, which dramatically reduces the search space.

Based upon the performance studies managed by J. Han, J. Pei, and Y. Yin [Han, Pei and Yin, 1999], it was claimed that the FP-growth method is efficient and scalable for mining both long and short frequent patterns, and is about an order of magnitude faster than the Apriori algorithm and also faster than some recently proposed frequent pattern mining methods.

The running examples (designed by J. Han, J. Pei, and Y. Yin) presented in this Section are taken from [Han, Pei and Yin, 1999].

3.4.1 Main Idea Behind Frequent Pattern Tree

Most of the studies before the FP-tree algorithm, such as [Agrawal and Srikant, 1994] [Grahne, Lakshmanan and Wang, 2000] [Klemettinen, Mannila, Ronkainen, Toivonen and Verkamo, 1994] [Lent, Swami and Widom, 1997] [Ng, Lakshmanan, Han and Pang, 1998] [Park, Chen and Yu, 1995] [Sarawagi, Thomas and Agrawal, 1998] [Savasere, Omiecinski and Navathe, 1995], adopt an Apriori-like approach, which is based on an anti-monotone Apriori heuristic [Agrawal and Srikant, 1994]: if any length k pattern is not frequent in the database, its length (k +1) super-pattern can never be frequent. The essential idea is to iteratively generate the set of candidate patterns of length (k + 1) from the set of frequent patterns of length k (for $k \geq 1$), and check their corresponding occurrence frequencies in the database.

The Apriori heuristic achieves good performance gain by (possibly significantly) reducing the size of candidate sets. However, in situations with prolific frequent patterns, long patterns, or quite low minimum support thresholds, an Apriori-like algorithm may still suffer from the following two nontrivial costs:

It is costly to handle a huge number of candidate sets. For example, if there are 10^4 frequent 1-itemsets, the Apriori algorithm will need to generate more than 10^7 length-2 candidates and accumulate and test their occurrence frequencies. Moreover, to discover a frequent pattern of size 100, such as $\{a_1,...,a_{100}\}$, it must generate more than $2^{100} \approx 10^{30}$ candidates in total. This is the inherent cost of candidate generation, no matter what implementation technique is applied.

As is seen, it is tedious to repeatedly scan the database and check a large set of candidates by pattern matching, which is especially true for mining long patterns.

After some careful examinations performed by J. Han, J. Pei, and Y. Yin [Han, Pei and Yin, 1999], it is believed that the bottleneck of the Apriori-like method is at the candidate set generation and test. If one can avoid generating a huge set of candidates, the mining performance can be substantially improved. This problem is attacked in the following three aspects:

First, a novel, compact data structure, called frequent pattern tree, or FP-tree for short, is constructed, which is an extended prefix-tree structure storing crucial, quantitative information about frequent patterns. Only frequent length-1

items will have nodes in the tree, and the tree nodes are arranged in such a way that more frequently occurring nodes will have better chances of sharing nodes than less frequently occurring ones.

Second, an FP-tree-based pattern fragment growth mining method, is developed, which starts from a frequent length-1 pattern (as an initial suffix pattern), examines only its conditional pattern base (a "sub-database" which consists of the set of frequent items co-occurring with the suffix pattern), constructs its (conditional) FP-tree, and performs mining recursively with such a tree. The pattern growth is achieved via concatenation of the suffix pattern with the new ones generated from a conditional FP-tree. Since the frequent itemset in any transaction is always encoded in the corresponding path of the frequent pattern trees, pattern growth ensures the completeness of the result. In this context, mining method of the FP-tree algorithm is not Apriori-like restricted generation-and-test but restricted test only. The major operations of mining are count accumulation and prefix path count adjustment, which are usually much less costly than candidate generation and pattern matching operations performed in most Apriori-like algorithms.

Third, the search technique employed in mining is a partitioning-based, divide-and-conquer method rather than Apriori-like bottom-up generation of frequent itemsets combinations. This dramatically reduces the size of conditional pattern base generated at the subsequent level of search as well as the size of its corresponding conditional FP-tree. Moreover, it transforms the problem of finding long frequent patterns to looking for shorter ones and then concatenating the suffix. It employs the least frequent items as suffix, which offers good selectivity. All these techniques contribute to substantial reduction of search costs.

3.4.2 Frequent Pattern Tree Design and Construction

Let $I = \{a_1,a_2,...,a_m\}$ be a set of items, and a transactional database $DB = \{T_1,T_2,...,T_n\}$, where T_i ($i \in [1..n]$) is a transaction which contains a set of items in I. The support (or occurrence frequency) of a pattern A, which is a set of items, is the number of transactions containing A in DB. A, is a frequent pattern if A's support is no less than a predefined minimum support threshold, ξ.

Given a transaction database DB and a minimum support threshold, ξ, the problem of finding the complete set of frequent patterns is called the frequent pattern mining problem.

To design a compact data structure for efficient frequent pattern mining, let's first examine an example.

Example 3.1: Let the transaction database, DB, be the first two columns of Figure 3.3 and support = 3 (ξ = 3).

According to J. Han, J. Pei, and Y. Yin [Han, Pei and Yin, 1999], a compact data structure can be designed based on the following observations:

1. Since only the frequent items will play a role in the frequent pattern mining, it is necessary to perform one scan of DB to identify the set of frequent items (with frequency count obtained as a by-product).

2. If the set of frequent items of each transaction are stored in some compact structure, repeatedly scanning of DB may be avoided.

3. If multiple transactions share an identical frequent item set, they can be merged into one with the number of occurrences registered as count. It is easy to check whether two sets are identical if the frequent items in all of the transactions are sorted according to a fixed order.

4. If two transactions share a common prefix, according to some sorted order of frequent items, the shared parts can be merged using one prefix structure as long as the count is registered properly. If the frequent items are sorted in the *frequency descending order*, there will be better chances that more prefix strings can be shared.

TID	Items Bought	(Ordered) Frequent Items
100	f, a, c, d, g, i, m, p	f, c, a, m, p
200	a, b, c, f, l, m, o	f, c, a, b, m
300	b, f, h, j, o	f, b
400	b, c, k, s, p	c, b, p
500	a, f, c, e, l, p, m, n	f, c, a, m, p

Figure 3.3: A transactional database as running example.

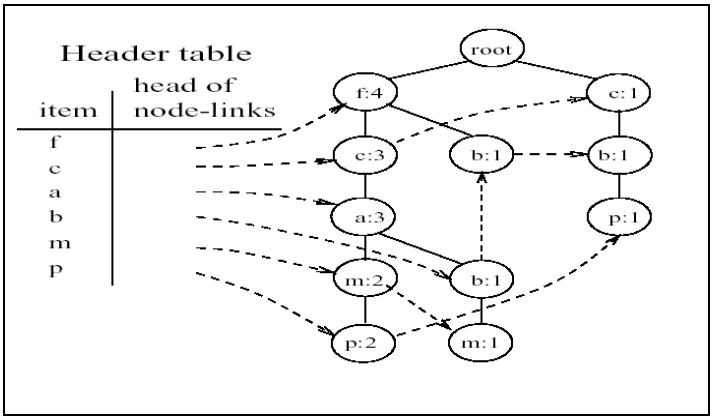


Figure 3.4: The FP-tree in Example 3.1.

With these observations, a frequent pattern tree can be constructed as follows:

First, a scan of DB derives a list of frequent items according to $\xi = 3$, ⟨(f:4); (c:4); (a:3); (b:3); (m:3); (p:3)⟩, (the number after ":" indicates the support), in which items ordered in frequency descending order. This ordering is important since each path of a tree will follow this order. The frequent items in each transaction are listed in this ordering in the rightmost column of Figure 3.3.

Followingly, create the root of a tree and label it "null". Scan the DB for the second time. The scan of the first transaction leads to the construction of the first branch of the tree: ⟨(f:1); (c:1); (a:1); (m:1); (p:1)⟩. Notice that the frequent items in the transaction are ordered according to the order in the list of frequent items. For the second transaction, since its (ordered) frequent item list ⟨f, c, a, b, m⟩ shares a common prefix ⟨f, c, a⟩ with the existing path ⟨f, c, a, m, p⟩, the count of each node along the prefix is incremented by 1, and one new node (b:1) is created and linked as a child of (a:2) and another new node (m:1) is created and linked as the child of (b:1). For the third transaction, since its frequent item list ⟨f, b⟩ shares only the node ⟨f⟩ with the ⟨f⟩ prefix subtree, f's count is incremented by 1, and a new node (b:1) is created and linked as a child of (f:3). The scan of the fourth transaction leads to the construction of the second branch of the tree, ⟨(c:1); (b:1); (p:1)⟩. For the last transaction, since its frequent item list ⟨f, c, a, m, p⟩ is identical to the first one, the path is shared with the count of each node along the path incremented by 1.

To facilitate tree traversal, an item header table is built in which each item points to its occurrence in the tree via a head of node-link. Nodes with the same item-name are linked in sequence via such node-links. After scanning all the transactions, the tree with the associated node-links is shown in Figure 3.4.

3.4.2.1. Formal Definition of a Frequent Pattern Tree

According to the above explanations as regards Example 3.1, a formal definition for a frequent patern tree can be concluded. A frequent pattern tree (or FP-tree in short) is a tree structure defined below:

1. It consists of one root labeled as "null", a set of item prefix subtrees as the children of the root, and a frequent-item header table.

2. Each node in the item prefix subtree consists of three fields: item-name, count, and node-link, where item-name registers which item this node represents, count registers the number of transactions represented by the portion of the path reaching this node, and node-link links to the next node in the FP-tree carrying the same item-name, or null if there is none.

3. Each entry in the frequent-item header table consists of two fields, 1) item-name and 2) head of node-link, which points to the first node in the FP-tree carrying the item-name.

3.4.2.2 Frequent Pattern Tree Construction Algorithm

Based on the above definitions, FP-tree construction algorithm developed by J. Han, J. Pei, and Y. Yin [Han, Pei and Yin, 1999] is as follows:

Input: A transactional database DB and a minimum support threshold ξ.

Output: Its frequent pattern tree, FP-tree

Method: The FP-tree is constructed in the following steps.

1. Scan the transaction database DB once. Collect the set of frequent items F and their supports. Sort F in support descending order as L, the list of frequent items.

2. Create the root of an FP-tree, T , and label it as "null". For each transaction, Trans, in DB do the following: Select and sort the frequent items in Trans according to the order of L. Let the sorted frequent item list in Trans be

([p|P],T), where p is the first element and P is the remaining list. Call insert_tree([p|P], T).

The function insert_tree([p|P], T) is performed as follows: If T has a child N such that N.item-name = p.item-name, then increment N's count by 1; else create a new node N, and let its count be 1, its parent link be linked to T , and its node-link be linked to the nodes with the same item-name via the node-link structure. If P is non-empty, call insert_tree(P,N) recursively.

Analysis: From the FP-tree construction process, it can be seen that one needs exactly two scans of the transaction database, DB: the first collects the set of frequent items, and the second constructs the FP-tree. According to the big-O notation, the cost of inserting a transaction Trans into the FP-tree is O(|Trans|), where |Trans| is the number of frequent items in Trans.

3.4.2.3 Compactness of Frequent Pattern Tree

J. Han, J. Pei, and Y. Yin claims that several important properties of FP-tree facilitates *frequent pattern mining* [Han, Pei and Yin, 1999].

Lemma 3.1: Given a transaction database DB and a support threshold ξ, its corresponding FP-tree contains the complete information of DB in relevance to frequent pattern mining [Han, Pei and Yin, 1999].

Lemma 3.1 states that, each transaction in the DB is mapped to one path in the FP-tree, and the frequent itemset information in each transaction is completely stored in the FP-tree. Moreover, one path in the FP-tree may represent frequent itemsets in multiple transactions without ambiguity since the path representing every transaction must start from the root of each item prefix subtree.

Lemma 3.2: Without considering the (null) root, the size of an FP-tree is bounded by the overall occurrences of the frequent items in the database, and the height of the tree is bounded by the maximal number of frequent items in any transaction in the database [Han, Pei and Yin, 1999].

Lemma 3.2 states that, for any transaction T in DB there exists a path in the FP-tree starting from the corresponding item prefix subtree so that the set of nodes in the path is exactly the same set of frequent items in T. Since no frequent item in any transaction can create more than one node in the tree, the root is the only extra node created not by frequent item insertion, and each node contains one

node-link and one count information, there is a bound on the size of the tree as stated in the Lemma 3.2. The height of any p-prefix subtree is the maximum number of frequent items in any transaction with p appearing at the head of its frequent item list. Therefore, the height of the tree is bounded by the maximal number of frequent items in any transaction in the database, if we do not consider the additional level added by the root.

Lemma 3.2 shows an important benefit of FP-tree: the size of an FP-tree is bounded by the size of its corresponding database because each transaction will contribute at most one path to the FP-tree, with the length equal to the number of frequent items in that transaction. Since there are often a lot of sharing of frequent items among transactions, the size of the tree is usually much smaller than its original database. Unlike the Apriori-like method which may generate an exponential number of candidates in the worst case, under no circumstances, may an FP-tree with an exponential number of nodes be generated.

FP-tree is a highly compact structure which stores the information for frequent pattern mining. Since a single path "$a_1 \rightarrow a_2 \rightarrow ... \rightarrow a_n$" in the a_1-prefix subtree registers all the transactions whose maximal frequent set is in the form of "$a_1 \rightarrow a_2 \rightarrow ... \rightarrow a_k$" for any $1 \leq k \leq n$, the size of the FP-tree is substantially smaller than the size of the database and that of the candidate sets generated in the association rule mining.

The items in the frequent itemset are ordered in the support-descending order: More frequently occurring items are arranged closer to the top of the FP-tree and thus are more likely to be shared. This indicates that FP-tree structure is usually highly compact.

3.4.3 Mining Frequent Patterns using Frequent Pattern Tree

Construction of a compact FP-tree ensures that subsequent mining can be performed with a rather compact data structure. Some interesting properties of FP-tree which facilitate frequent pattern mining are as follows.

3.4.3.1 Node-Link Property

Node-Link property says that: For any frequent item a_i, all the possible frequent patterns that contain a_i can be obtained by following a_i's node-links, starting from a_i's head in the FP-tree header [Han, Pei and Yin, 1999].

This property is based directly on the construction process of FP-tree. It facilitates the access of all the pattern information related to a_i by traversing the FP-tree once following a_i's node-links.

Example 3.2: Let's examine the mining process based on the constructed FP-tree shown in Figure 3.4. Based on node-link property, we collect all the patterns that a node a_i participates by starting from ai's head (in the header table) and following ai's node-links. We examine the mining process by starting from the bottom of the header table.

For node p, it derives a frequent pattern (p:3) and two paths in the FP-tree: ⟨f:4, c:3, a:3, m:2, p:2⟩ and ⟨c:1, b:1, p:1⟩. The first path indicates that string "(f; c; a; m; p)" appears twice in the database. Notice that, although string ⟨f, c, a⟩ appears three times and ⟨f⟩ itself appears even four times, they only appear twice together with p. Thus, to study which string appear together with p, only p's prefix path ⟨f:2, c:2, a:2, m:2⟩ is counted. Similarly, the second path indicates string "(c, b, p)" appears once in the set of transactions in DB, or p's prefix path is ⟨c:1, b:1⟩. These two prefix paths of p, "{(f:2, c:2, a:2, m:2), (c:1; b:1)}", form p's sub-pattern base, which is called p's conditional pattern base (i.e., the sub-pattern base under the condition of p's existence). Construction of an FP-tree on this conditional pattern base (which is called p's conditional FP-tree) leads to only one branch (c:3). Hence, only one frequent pattern (cp:3) is derived. (Notice that a pattern is an itemset and is denoted by a string here.) The search for frequent patterns associated with p terminates.

For node m, it derives a frequent pattern (m:3) and two paths ⟨f:4, c:3, a:3, m:2⟩ and ⟨f:4, c:3, a:3, b:1, m:1⟩. Notice that, p appears together with m as well, however, there is no need to include p here in the analysis since any frequent patterns involving p has been analyzed in the previous examination of p. Similar to the above analysis, m's conditional pattern base is, {(f:2, c:2, a:2), (f:1, c:1, a:1, b:1)}. By constructing an FP-tree on it, m's conditional FP-tree, ⟨f:3, c:3, a:3⟩, a

single frequent pattern path, can be derived. Then, one can call FP-tree-based mining recursively, i.e., call mine(⟨f:3, c:3, a:3⟩|m).

Figure 3.5 shows “mine (⟨f:3, c:3, a:3⟩|m)” function which involves mining three items (a), (c), (f) in sequence. The first derives a frequent pattern (am:3), and a call “mine (⟨f:3, c:3⟩|am)”; the second derives a frequent pattern (cm:3), and a call “mine (⟨f:3⟩|cm)"; and the third derives only a frequent pattern (fm:3). Further recursive call of “mine (⟨f:3, c:3⟩|am)” derives (cam:3), (fam:3), and a call “mine(⟨f:3⟩|cam)”, which derives the longest pattern (fcam:3). Similarly, the call of “mine (⟨f:3⟩|cm)”, derives one pattern (fcm:3). Therefore, the whole set of frequent patterns involving m is {(m:3), (am:3), (cm:3), (fm:3), (cam:3), (fam:3), (fcam:3), (fcm:3)}. This indicates a single path FP-tree can be mined by outputting all the combinations of the items in the path.

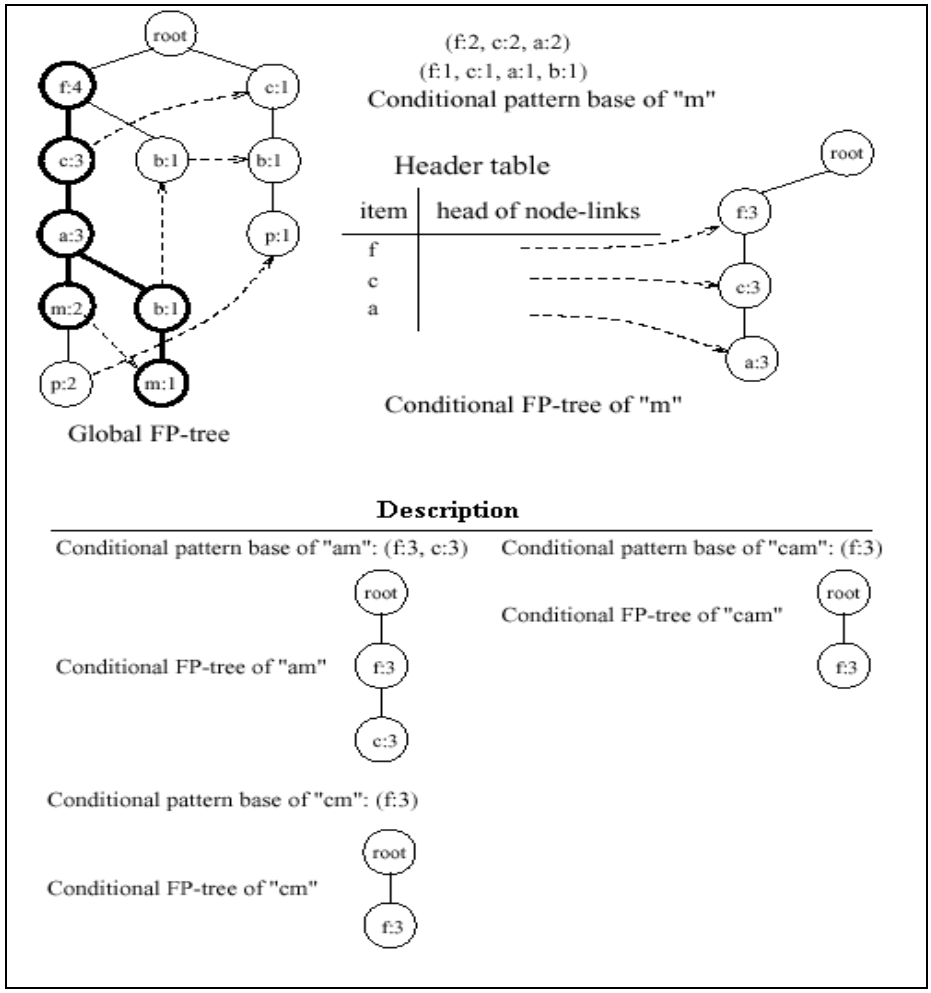


Figure 3.5: A conditional FP-tree built for m, i.e., “FP-tree |m”.

Similarly, node b derives (b:3) and three paths: ⟨f:4, c:3, a:3, b:1⟩, ⟨f:4, b:1⟩, and ⟨c:1, b:1⟩. Since b's conditional pattern base: {(f:1; c:1; a:1), (f:1), (c:1)} generates no frequent item, the mining terminates. Node a derives one frequent pattern {(a:3)}, and one subpattern base, {(f:3; c:3)}, a single path conditional FP-tree. Thus, its set of frequent patterns can be generated by taking their combinations. Concatenating them with (a:3), the set {(fa:3), (ca:3), (fca:3)} is generated. Node c derives (c:4) and one subpattern base, {(f:3)}, and the set of frequent patterns associated with (c:3) is {(fc:3)}. Node f derives only (f:4) but no conditional pattern base. The conditional pattern bases and the conditional FP-trees generated are summarized in Figure 3.6.

Item	Conditional pattern base	Conditional FP-tree
p	{(f:2, c:2, a:2, m:2,), (c:1, a:1)}	{(c:3)}|p
m	{(f:4, c:3, a:3, m:2), (f:4, c:3, a:3, b:1, m:1)}	{(f:3, c:3, a:3}|m
b	{(f:4, c:3, a:3, b:1), (f:4, b:1) (c:1, b:1)}	∅
a	{(f:3, c:3)}	{(f:3), c:2)}|a
c	{(f:3)}	{(f:3)}|c
f	∅	∅

Figure 3.6: Mining of all-patterns by creating conditional (sub)-pattern bases.

3.4.3.2 Prefix Path Property

Prefix Path property says that: In order to calculate the frequent patterns for a node a_i in a path P, only the prefix subpath of node a_i in P need to be accumulated, and the frequency count of every node in the prefix path should carry the same count as node a_i [Han, Pei and Yin, 1999].

Let the nodes along the path P be labeled as $a_1,...,a_n$ in such an order that a_1 is the root of the prefix subtree, a_n is the leaf of the subtree in P, and a_i $(1 \leq i \leq n)$ is the node being referenced. Based on the process of construction of FP-tree, for each prefix node a_k $(1 \leq k < i)$, the prefix subpath of the node a_i in P occurs together with a_k exactly a_i: count times. Thus, every such prefix node should carry the same count as node a_i. Notice that a postfix node a_m (for $i < m \leq n$) along the

same path also co-occurs with node a_i. However, the patterns with a_m will be generated at the examination of the postfix node a_m, enclosing them here will lead to redundant generation of the patterns that would have been generated for a_m. Therefore, it is satisfactory to examine the prefix subpath of a_i in P.

For example, in Example 3.2, node m is involved in a path ⟨f:4, c:3, a:3, m:2, p:2⟩, to calculate the frequent patterns for node m in this path, only the prefix subpath of node m, which is ⟨f:4, c:3; a:3⟩, need to be extracted, and the frequency count of every node in the prefix path should carry the same count as node m. That is, the node counts in the prefix path should be adjusted to ⟨f:2, c:2, a:2⟩.

Based on this property, the prefix subpath of node a_i in a path P can be copied and transformed into a count-adjusted prefix subpath by adjusting the frequency count of every node in the prefix subpath to the same as the count of node a_i. The so transformed prefix path is called the *transformed prefixed path* of a_i for path P.

Notice that the set of transformed prefix paths of a_i form a small database of patterns which co-occur with a_i. Such a database of patterns occurring with a_i is called a_i's *conditional pattern base,* and is denoted as "pattern_base | a_i". Then one can compute all the frequent patterns associated with a_i in this a_i-conditional pattern base by creating a small FP-tree, called a_i's conditional FP-tree and denoted as "FP-tree | a_i". Subsequent mining can be performed on this small, conditional FP-tree. The processes of construction of conditional pattern bases and conditional FP-trees have been demonstrated in Example 3.2.

This process is performed recursively, and the frequent patterns can be obtained by a pattern growth method, based on the following lemmas and corollary [Han, Pei and Yin, 1999].

Lemma 3.3 (Fragment growth): Let α be an item-set in DB, B be α's conditional pattern base, and β be an itemset in B. Then the support of $\alpha \cup \beta$ in DB is equivalent to the support of β in B.

According to the definition of conditional pattern base, each (sub)transaction in B occurs under the condition of the occurrence of α in the original transaction database DB. If an itemset β appears in B times, it appears with α in DB ψ times as well. Moreover, since all such items are collected in the conditional pattern base of α, $\alpha \cup \beta$ occurs exactly ψ times in DB as well.

Corollary (Pattern growth): Let α be a frequent itemset in DB, B be α's conditional pattern base, and β be an itemset in B. Then $\alpha \cup \beta$ is frequent in DB if and only if β is frequent in B.

This corollary is the case when α is a frequent itemset in DB, and when the support of β in α's conditional pattern base B is no less than ξ, the minimum support threshold.

Based on the corollary , mining can be performed by first identifying the frequent 1-itemset, α, in DB, constructing their conditional pattern bases, and then mining the 1-itemset, β, in these conditional pattern bases, and so on. This indicates that the process of mining frequent patterns can be viewed as first mining frequent 1-itemset and then progressively growing each such itemset by mining its conditional pattern base, which can in turn be done similarly. Thus, a frequent k-itemset mining problem can succesfully be transformed into a sequence of k frequent 1-itemset mining problems via a set of conditional pattern bases. The only needed is “pattern growth”. There is no need to generate any combinations of candidate sets in the entire mining process.

Finally, when the FP-tree contains only a single path, the property on mining all the patterns is provided .

Lemma 3.4 (Single FP-tree path pattern generation): Suppose an FP-tree T has a single path P. The complete set of the frequent patterns of T can be generated by the enumeration of all the combinations of the subpaths of P with the support being the minimum support of the items contained in the subpath.

Let the single path P of the FP-tree be $\langle a_1{:}s_1 \rightarrow a_2{:}s_2 \rightarrow \ldots \rightarrow a_k{:}s_k \rangle$. The support frequency s_i of each item a_i (for $1 \leq i \leq k$) is the frequency of a_i co-occurring with its prefix string. Thus any combination of the items in the path, such as $\langle a_i, \ldots, a_j \rangle$ (for $1 \leq i, j \leq k$), is a frequent pattern, with their co-occurrence frequency being the minimum support among those items. Since every item in each path P is unique, there is no redundant pattern to be generated with such a combinational generation. Moreover, no frequent patterns can be generated outside the FP-tree.

3.4.3.3 Pattern Fragment Growth Algorithm

Based on the above lemmas and properties, Pattern Fragment Growth (FP-Growth for short) algorithm is used for mining the frequent patterns on FP-tree [57].

Input: FP-tree constructed based on the algorithm described in Section 3.4.2.2. The parameters in order to construct a FP-tree were a DB (transactional database) and a minimum support threshold ξ.

Output: The complete set of frequent patterns.

Method: Call FP-growth (FP-tree ; null).

```
Procedure FP-growth (Tree; α)
{
(1)     if Tree contains a single path P
(2)     then for each combination (denoted as β) of the nodes in the path P do
(3)     generate pattern β ∪ α with support = minimum support of nodes in β,
(4)     else for each a_i in the header of Tree do {
(5)             generate pattern β = a_i ∪ α with support = a_i.support;
(6)             construct β's conditional pattern base and then β's conditional FP-tree Tree_β;
(7)             if Tree_β ≠ ∅
(8)             then call FP-growth (Tree_β; β)
        }
}
```

Figure 3.7: Pattern Fragment Growth (FP-Growth) Algorithm.

Analysis: With the properties and lemmas in the above sections, it is shown that the FP-growth algorithm, given in Figure 3.7, correctly finds the complete set of frequent itemsets in transaction database DB.

As shown in Lemma 3.3, FP-tree of DB contains the complete information of DB in relevance to frequent pattern mining under the support threshold ξ.

If an FP-tree contains a single path, according to Lemma 3.4, its generated patterns are the combinations of the nodes in the path, with the support being the minimum support of the nodes in the subpath. Thus, we have lines (1)-(3) of the procedure (Figure 3.7). Otherwise, we construct conditional pattern base and mine its conditional FP-tree for each frequent itemset a_i. The correctness and

completeness of prefix path transformation are shown in the Prefix-Path propery section, and thus the conditional pattern bases store the complete information for frequent pattern mining. According to Lemma 3.3 and its corollary, the patterns successively grown from the conditional FP-trees are the set of sound and complete frequent patterns. Especially, according to the fragment growth property, the support of the combined fragments takes the support of the frequent itemsets generated in the conditional pattern base. Therefore, we have lines (4)-(8) of the procedure.

Let's now examine the efficiency of the algorithm. The FP-growth mining process scans the FP-tree of DB once and generates a small pattern-base B_{a_i} for each frequent item a_i, each consisting of the set of transformed prefix paths of a_i. Frequent pattern mining is then recursively performed on the small pattern-base B_{a_i} by constructing a conditional FP-tree for B_{a_i}. As reasoned in the analysis of Fp-tree Construction algorithm, an FP-tree is usually much smaller than the size of DB. Similarly, since the conditional FP-tree, "FP-tree | a_i", is constructed on the pattern-base B_{a_i}, it should be usually much smaller and never bigger than B_{a_i}. Moreover, a pattern-base B_{a_i} is usually much smaller than its original FP-tree, because it consists of the transformed prefix paths related to only one of the frequent items, a_i. Thus, each subsequent mining process works on a set of usually much smaller pattern bases and conditional FP-trees. Moreover, the mining operations consists of mainly prefix count adjustment, counting, and pattern fragment concatenation. This is much less costly than generation and test of a very large number of candidate patterns. Thus the algorithm is efficient.

From the algorithm and its reasoning, one can see that the FP-growth mining process is a divide-and-conquer process, and the scale of shrinking is usually quite dramatic. If the shrinking factor is around 20~100 for constructing an FP-tree from a database, it is expected to be another hundreds of times reduction for constructing each conditional FP-tree from its already quite small conditional frequent pattern base.

Even in the case that a database may generate an exponential number of frequent patterns, the size of the FP-tree is usually quite small and will never grow exponentially. For example, for a frequent pattern of length 100, "$a_1, \ldots, a_{100}$",

the FP-tree construction results in only one path of length 100 for it, such as "$\langle a_1; \rightarrow \ldots \rightarrow a_{100}\rangle$". The FP-growth algorithm will still generate about 10^{30} frequent patterns (if time permits!), such as "a_1, a_2, . . ., a_1 a_2, . . ., $a_1a_2a_3$, . . ., $a_1...a_{100}$". However, the FP-tree contains only one frequent pattern path of 100 nodes, and according to Lemma 3.4, there is even no need to construct any conditional FP-tree in order to find all the patterns.

3.5 Closed Association Rule Mining Algorithm

The task of mining association rules consists of two main steps. The first involves finding the set of all frequent itemsets(frequent subsets of transactions). The second step involves testing and generating all high confidence rules among itemsets. However, it may not be necessary to mine all frequent itemsets in the first step, instead it may be sufficient to mine the set of *closed* frequent itemsets, which is much smaller than the set of all frequent itemsets. Also it may not be necessary to mine the set of all possible rules. Because any rule between frequent itemsets may be equivalent to some rule between closed itemsets. Thus many redundant rules can be eliminated.

In this sense, the CHARM is an efficient algorithm for mining all closed frequent itemsets. It is proposed by M. Zaki and C.-J. Hsiao [Zaki and Hsiao, 1999]. The abbreviation *"CHARM"* stands for *"closed association rule mining"*, the letter 'H' is gratuitous.

The running examples (developed by M. Zaki and C.-J. Hsiao) presented in this Section are taken from [Zaki and Hsiao, 1999].

3.5.1 General Definitions and "Itemset-Tidset" Pairs

The association mining task can be stated as follows: Let I = {1, 2, . . .;m} be a set of items, and let T = {1, 2, . . ., n} be a set of transaction identifiers or *tids*. The input database is a binary relation $\delta \subseteq I \times T$. If an item i occurs in a transaction t, we write it as $(i; t) \in \delta$, or alternately as $i\delta t$. Typically the database is arranged as a set of transactions, where each transaction contains a set of items. For example, consider the database shown in Figure 3.7, used as a running example throughout this section. Here I = {A, C, D, T, W}, and T = {1, 2, 3, 4, 5,

6}. The second transaction can be represented as {Cδ2, Dδ2, Wδ2}; all such pairs from all transactions, taken together form the binary relation δ.

A set $X \subseteq I$ is also called an *itemset*, and a set $Y \subseteq T$ is called a *tidset*. For convenience an itemset {A, C, W} is written as ACW, and a tidset {2, 4, 5} is written as 245. The *support* of an itemset X, denoted σ(X), is the number of transactions in which it occurs as a subset. An itemset is *frequent* if its support is more than or equal to a user-specified *minimum support (minsup)* value, i.e., if $\sigma(X) \geq minsup$.

We know that, an *association rule* is an expression $X_1 \xrightarrow{p} X_2$, where X_1 and X_2 are itemsets, and $X_1 \cup X_2 = \varnothing$. The *support* of the rule is given as $\sigma(X_1 \cup X_2)$ (i.e., the joint probability of a transaction containing both X_1 and X_2), and the *confidence* as $p = \sigma(X_1 \cup X_2) = \sigma(X_1)$ (i.e., the conditional probability that a transaction contains X_2, given that it contains X_1). A rule is frequent if the itemset $X_1 \cup X_2$ is frequent. A rule is *confident* if its confidence is greater than or equal to a user-specified *minimum confidence (minconf)* value, i.e, $p \geq minconf$.

According to these overviews, the association rule mining task consists of two steps [Agrawal, Mannila, Srikant, Toivonen, and Verkamo, 1996]: 1) Find all frequent itemsets, and 2) Generate high confidence rules. Let's now examine these steps in a more realistic way.

3.5.1.1 Finding Frequent Itemsets

This step is computationally and I/O intensive. Consider Figure 3.8, which shows a bookstore database with six customers who buy books by different authors. It shows all the frequent itemsets with minsup = 50% (i.e., 3 transactions). ACTW and CDW are the maximal-by-inclusion frequent itemsets (i.e., they are not a subset of any other frequent itemset).

Let |I| = m be the number of items. The search space for the enumeration of all frequent itemsets is 2^m, which is exponential in m. One can prove that the problem of finding a frequent set of a certain size is NP-Complete, by reducing it to the *balanced bipartite clique problem*, which is known to be NP-Complete [Gunopulos, Mannila, and Saluja, 1997] [Zaki and Ogihara, 1998]. However, if it is assumed that there is a bound on the transaction length, the task of finding all

frequent itemsets is essentially linear in the database size, since the overall complexity in this case is given as $O(r \cdot n \cdot 2^l)$, where $|T| = n$ is the number of transactions, l is the length of the longest frequent itemset, and r is the number of maximal frequent itemsets.

DISTINCT DATABASE ITEMS

Jane Austen	Agatha Christie	Sir Arthur Conan Doyle	Mark Twain	P. G. Wodehouse
A	C	D	T	W

DATABASE

Transcation	Items
1	A C T W
2	C D W
3	A C T W
4	A C D W
5	A C D T W
6	C D T

ALL FREQUENT ITEMSETS

MINIMUM SUPPORT = 50%

Support	Itemsets
100% (6)	C
83% (5)	W, CW
67% (4)	A, D, T, AC, AW CD, CT, ACW
50% (3)	AT, DW, TW, ACT, ATW CDW, CTW, ACTW

Figure 3.8: Generating Frequent Itemsets.

3.5.1.2 Generating Confident Rules

This step is relatively straightforward; rules of the form $X' \xrightarrow{p} X - X'$, are generated for all frequent itemsets X (where $X' \subset X$, and $X' \neq \emptyset$), provided p ≥ *minconf*. For an itemset of size k there are $2^k - 2$ potentially confident rules that can be generated. This follows from the fact that we must consider each subset of the itemset as an antecedent, except for the empty and the full itemset. The complexity of the rule generation step is thus $O(s \cdot 2^l)$, where s is the number of frequent itemsets, and l is the longest frequent itemset (note that s can be $O(r \cdot 2_l)$, where r is the number of maximal frequent itemsets). For example, from the frequent itemset ACW we can generate 6 possible rules (all of them have support of 4):

$$A \xrightarrow{1.0} CW, C \xrightarrow{0.67} AW, W \xrightarrow{0.8} AC, AC \xrightarrow{1.0} W, AW \xrightarrow{1.0} C, \text{and } CW \xrightarrow{0.8} A.$$

3.5.2. Closed Association Rule Mining

The set of closed frequent itemsets (which has smaller cardinality than the set of all frequent itemsets) is necessary and sufficient for the CHARM algorithm to capture all the information about frequent itemsets.

3.5.2.1 Partial Order and Lattices

We first introduce some lattice theory concepts (see [Daveyand and Priestley, 1990] for a good introduction). Let P be a set. A *partial order* on P is a binary relation $\leq$, such that for all x; y; z $\in$ P, the relation is: 1) Reflexive: $x \leq x$. 2) Anti-Symmetric: $x \leq y$ and $y \leq x$, implies $x = y$. 3) Transitive: $x \leq y$ and $y \leq z$, implies $x \leq z$. The set P with the relation $\leq$ is called an *ordered set*, and it is denoted as a pair $(P,\leq)$. We write $x < y$ if $x \leq y$ and $x \neq y$.

Let $(P,\leq)$ be an ordered set, and let S be a subset of P. An element $u \in P$ is an *upper bound* of S if $s \leq u$ for all $s \in S$. An element $l \in P$ is a *lower bound* of S if $s \geq l$ for all $s \in S$. The least upper bound is called the *join* of S, and is denoted as V S, and the greatest lower bound is called the *meet* of S, and is denoted as Λ S. If S = {x, y}, it can also be written x V y for the join, and x Λ y for the meet.

An ordered set $(L, \leq)$ is a *lattice*, if for any two elements x and y in L, the join x V y and meet x Λ y always exist. L is a *complete lattice* if V S and Λ S exist for all $S \subseteq L$. Any finite lattice is complete. L is called a *join semilattice* if only the join exists. L is called a *meet semilattice* if only the meet exists.

Let $\mathcal{P}$ denote the power set of S (i.e., the set of all subsets of S). The ordered set $(\mathcal{P}(S), \subseteq)$ is a complete lattice, where the meet is given by set intersection, and the join is given by set union. For example the partial orders $(\mathcal{P}(I), \subseteq)$, the set of all possible itemsets, and $(\mathcal{P}(T), \subseteq)$, the set of all possible tidsets are both complete lattices.

The set of all frequent itemsets, on the other hand, is only a meet-semilattice. For example, consider Figure 3.9, which shows the semilattice of all frequent itemsets founded in the example database (from Figure 3.8). For any two itemsets, only their meet is guaranteed to be frequent, while their join may or may not be frequent. This follows from the well known principle in association mining that, *if an itemset is frequent, then all its subsets are also frequent*. For example, AC Λ AT = AC $\cap$ AT = A is frequent. For the join, while AC V AT = AC $\cup$ AT = ACT is frequent, AC $\cup$ DW = ACDW is not frequent.

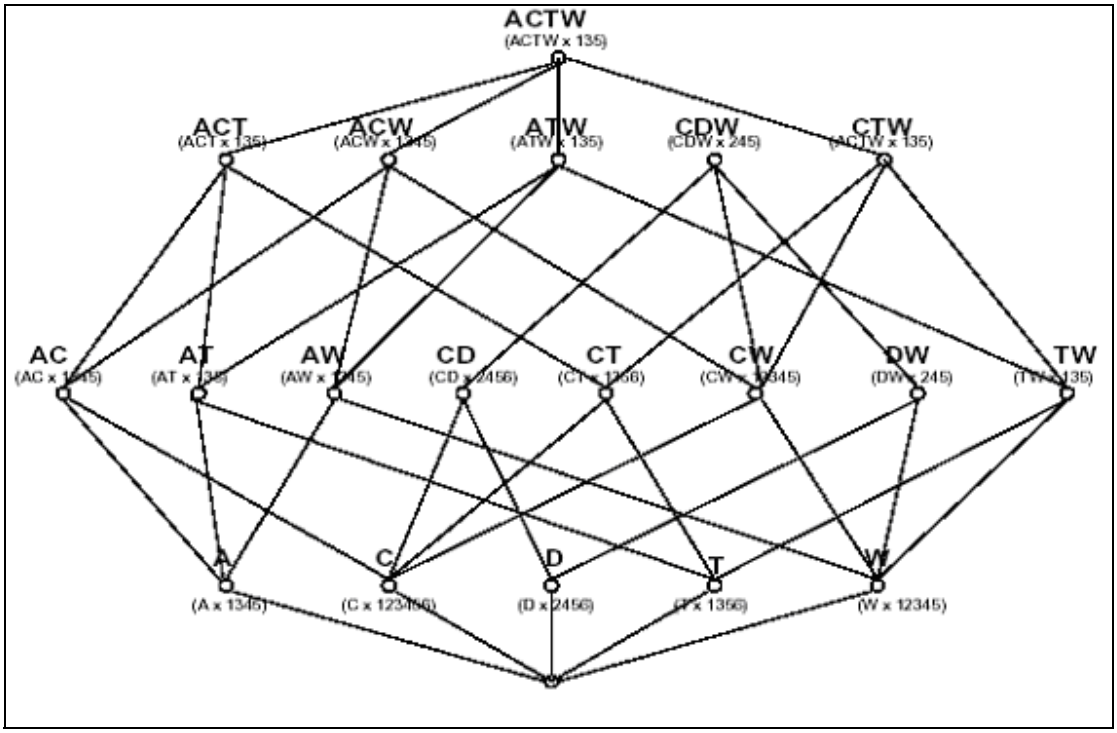


Figure 3.9: Meet Semi–lattice of Frequent Itemsets.

3.5.2.2 Finding Closed Itemsets

Let the binary relation $\delta \subseteq$ I x T be the input database for association mining. Let $X \subseteq I$, and $Y \subseteq T$. Then the mappings

$$t : \mathcal{I} \mapsto \mathcal{T}, \quad t(X) = \{y \in \mathcal{T} \mid \ \forall x \in X, \ x\delta y\}$$
$$i : \mathcal{T} \mapsto \mathcal{I}, \quad i(Y) = \{x \in \mathcal{I} \mid \ \forall y \in Y, \ x\delta y\}$$

define a *Galois connection* between the partial orders ($\mathcal{P}$(I), $\subseteq$) and ($\mathcal{P}$(T), $\subseteq$), the power sets of I and T , respectively. A (X, t(X)) pair is denoted as X x t(X), and a (i(Y), Y) pair is denoted as i(Y) x Y . Figure 3.9 illustrates the two mappings. The mapping t(X) is the set of all transactions (tidset) which contain the itemset X, similarly i(Y) is the itemset that is contained in all the transactions in Y. For example, t(ACW) = 1345, and i(245) = CDW. In terms of individual elements, for example t(ACW) = t(A) $\cap$ t(C) $\cap$ t(W) = 1345 $\cap$ 123456 $\cap$ 12345 = 1345. Also i(245) = i(2) $\cap$ i(4) $\cap$ i(5) = CDW $\cap$ ACDW $\cap$ ACDTW = CDW.

The Galois connection satisfies the following properties (where X, X_1, X_2 $\in \mathcal{P}$(I) and Y, Y_1, $Y_2 \in \mathcal{P}$(T)):

1) $X_1 \subseteq X_2 \Rightarrow t(X_1) \supseteq t(X_2)$.

   For ACW $\subseteq$ ACTW, we have t(ACW) = 1345 $\supseteq$ 135 = t(ACTW).

2) $Y_1 \subseteq Y_2 \Rightarrow i(Y_1) \supseteq i(Y_2)$.

For example, for $245 \subseteq 2456$, we have $i(245) = CDW \supseteq CD = i(2456)$.

3) $X \subseteq i(t(X))$ and $Y \subseteq t(i(Y))$.

For example, $AC \subseteq i(t(AC)) = i(1345) = ACW$.

Let S be a set. A function $c : \mathcal{P}(S) \rightarrow \mathcal{P}(S)$ is a *closure operator* on S if, for all X; $Y \subseteq S$, c satisfies the following properties: 1) *Extension:* $X \subseteq c(X)$. 2) *Monotonicity:* if $X \subseteq Y$, then $c(X) \subseteq c(Y)$. 3) *Idempotency:* $c(c(X)) = c(X)$. A subset X of S is called *closed* if $c(X) = X$.

Lemma 3.5: *Let* $X \subseteq I$ *and* $Y \subseteq T$. *Let* $c_{it}(X)$ *denote the composition of the two mappings* $i \circ t(X) = i(t(X))$. *Dually, let* $c_{ti}(Y) = t \circ i(Y) = t(i(Y))$. *Then* $c_{it}: \mathcal{P}(I) \rightarrow \mathcal{P}(I)$ *and* $c_{ti}: \mathcal{P}(T) \rightarrow \mathcal{P}(T)$ *are both closure operators on itemsets and tidsets respectively.*

A *closed itemset* is defined as an itemset X that is the same as its closure, i.e., $X = c_{it}(X)$. For example the itemset ACW is closed. A *closed tidset* is a tidset $Y = c_{ti}(Y)$. For example, the tidset 1345 is closed.

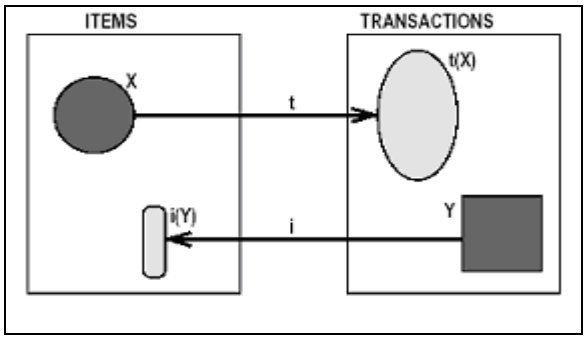


Figure 3.10: Galois Connection.

The mappings c_{it} and c_{ti}, being closure operators, satisfy the three properties of extension, monotonicity, and idempotency. Also the application of $i \circ t$ or $t \circ i$ is called a *round-trip*. Figure 3.11 illustrates this round-trip starting with an itemset X. For example, let X = AC, then the extension property says that X is a subset of its closure, since $c_{it}(AC) = i(t(AC)) = i(1345) = ACW$. Since $AC \neq c_{it}(AC) = ACW$, it is concluded that AC is not closed. On the other hand, the idempotency property says that once we map an itemset to the tidset that contains it, and then map that tidset back to the set of items common to all tids in the tidset,

we obtain a closed itemset. After this, no matter how many such round-trips we make, we cannot extend a closed itemset. For example after one round-trip for AC, the closed itemset ACW is obtained. If another round-trip is performed on ACW, $c_{it}(ACW) = i(t(ACW)) = i(1345) = ACW$ is obtained.

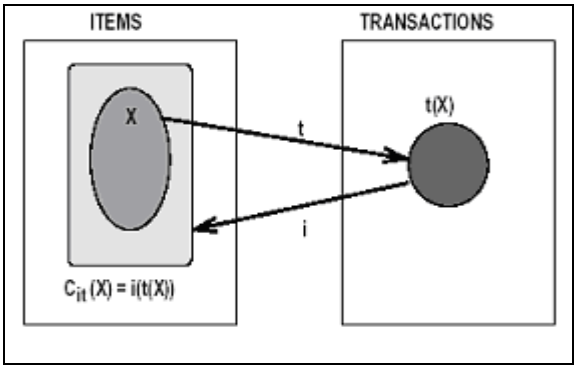


Figure 3.11: Closure Operator: Round-Trip.

For any closed itemset X, there exists a closed tidset given by Y , with the property that Y = t(X) and X = i(Y) (conversely, for any closed tidset there exists a closed itemset). We can see that X is closed by the fact that X = i(Y), then plugging Y = t(X), we get X = i(Y) = i(t(X)) = c_{it}(X), thus X is closed. Dually, Y is closed. For example, we have seen above that for the closed itemset ACW the associated closed tidset is 1345. Such a closed itemset and closed tidset pair X x Y is called a *concept*.

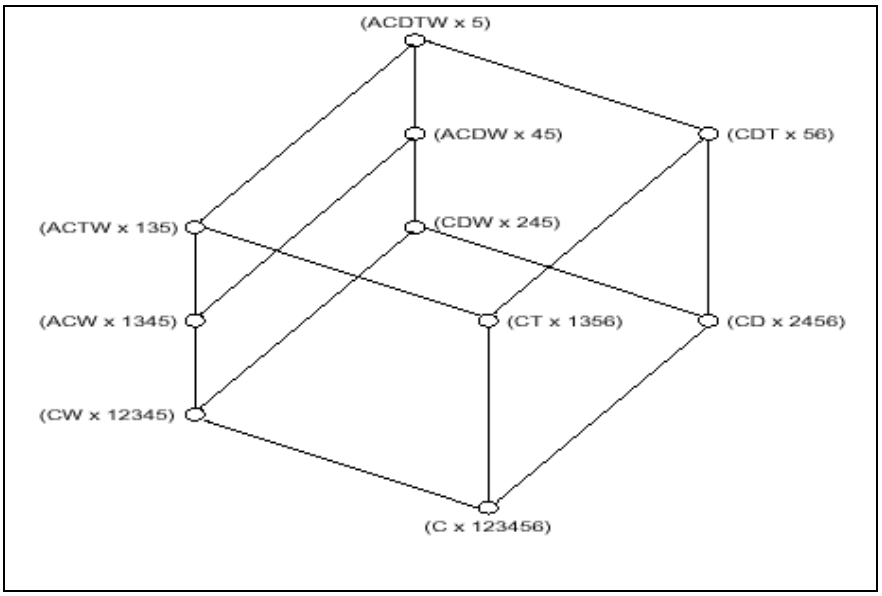


Figure 3.12: Galois Lattice of Concepts.

A concept X_1 x Y_1 is a *subconcept* of X_2 x Y_2, denoted as X_1 x $Y_1 \leq X_2$ x Y_2, iff $X_1 \subseteq X_2$ (iff $Y_2 \subseteq Y_1$). Let B(δ) denote the set of all possible concepts in the database, then the ordered set (B(δ), ≤) is a complete lattice, called the *Galois* lattice. For example, Figure 3.12 shows the Galois lattice for our example database, which has a total of 10 concepts. The least element is the concept C x 123456 and the greatest element is the concept ACDTW x 5. Notice that the mappings between the closed pairs of itemsets and tidsets are anti-isomorphic, i.e., concepts with large cardinality itemsets have small tidsets, and vice versa.

3.5.3 Closed Frequent Itemsets Versus All Frequent Itemsets

We may now define the join and meet operations on the concept lattice (see [52] for the formal proof): The set of all concepts in the database relation δ, given by (B(δ),≤) is a (complete) lattice with join and meet given by

join: $(X_1 \text{ x } Y_1) \text{ V } (X_2 \text{ x } Y_2) = c_{it}(X_1 \cup X_2) \text{ x } (Y_1 \cap Y_2)$

meet: $(X_1 \text{ x } Y_1) \Lambda (X_2 \text{ x } Y_2) = (X_1 \cap X_2) \text{ x } c_{ti}(Y_1 \cup Y_2)$

For the join and meet of multiple concepts, we simply take the unions and joins over all of them. For example, consider the join of two concepts, (ACDW x 45) V (CDT x 56) = c_{it}(ACDW ∪ CDT) x (45 ∩ 56) = ACDTW x 5. On the other hand their meet is given as, (ACDW x 45) Λ (CDT x 56) = (ACDW ∩ CDT) x c_{ti}(45 ∪ 56) = CD x c_{ti}(456) = CD x 2456. Similarly, we can perform multiple concept joins or meets; for example, (CT x 1356) V (CD x 2456) V (CDW x 245) = c_{it}(CT ∪ CD ∪ CDW) x (1356 ∩ 2456 ∩ 245) = c_{it}(CDTW) x 5 = ACDTW x 5.

The support of a closed itemset X or a concept X x Y is defined as the cardinality of the closed tidset Y = t(X), i.e, σ(X) = |Y| = |t(X)|. A closed itemset or a concept is *frequent* if its support is at least *minsup*. Figure 3.13 shows all the frequent concepts with *minsup* = 50 % (i.e., with tidset cardinality at least 3). The frequent concepts, like the frequent itemsets, form a meet-semilattice, where the meet is guaranteed to exist, while the join may not.

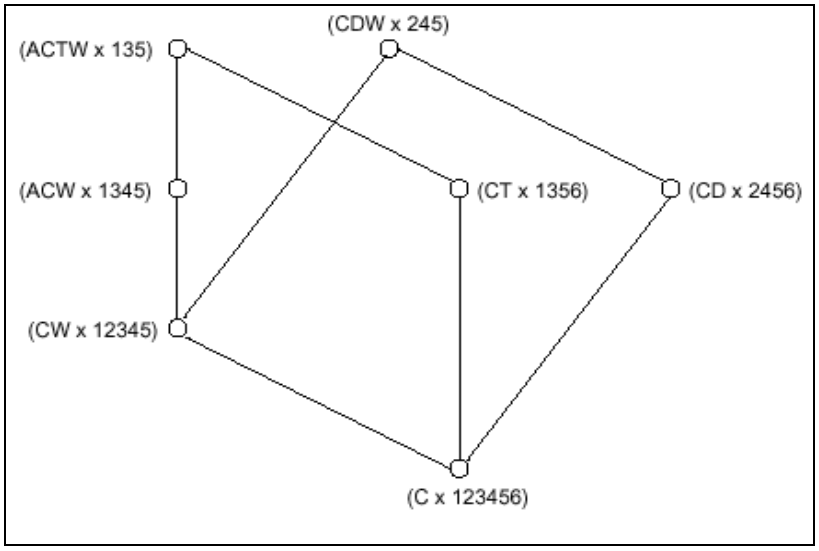


Figure 3.13: Frequent Concepts.

Theorem 3.1: For any itemset X, its support is equal to the support of its closure, i.e., $\sigma(X) = \sigma(c_{it}(X))$.

Proof: The support of an itemset X is the number of transactions where it appears, which is exactly the cardinality of the tidset t(X), i.e., $\sigma(X) = |t(X)|$. Since $\sigma(cit(X)) = |t(cit(X))|$, to prove the Theorem 3.1, $t(X) = t(c_{it}(X))$ have to be shown.

Since c_{ti} is closure operator, it satisfies the extension property, i.e., $t(X) \subseteq c_{ti}(t(X)) = t(i(t(X))) = t(c_{it}(X))$. Thus $t(X) \subseteq t(c_{it}(X))$. On the other hand since c_{it} is also a closure operator, $X \subseteq c_{it}(X)$, which in turn implies that $t(X) \supseteq t(c_{it}(X))$, due to property 1) of Galois connections. Thus $t(X) = t(c_{it}(X))$.

According to M. Zaki and C.-J. Hsiao [Zaki and Hsiao, 1999], this theorem states that all frequent itemsets are uniquely determined by the frequent closed itemsets (or frequent concepts). Furthermore, the set of frequent closed itemsets is bounded above by the set of frequent itemsets, and is typically much smaller, especially for dense datasets (where there can be orders of magnitude differences). To illustrate the benefits of closed itemset mining, contrast Figure 3.9, showing the set of all frequent itemsets, with Figure 3.13, showing the set of all closed frequent itemsets (or concepts). It is seen that while there are only 7 closed frequent itemsets, there are 19 frequent itemsets. This example clearly illustrates the benefits of mining the closed frequent itemsets.

3.5.4 Closed Association Rule Mining Algorithm Design

CHARM is unique in that it simultaneously explores both the itemset space and tidset space, unlike all previous association mining methods which only exploit the itemset space [Zaki and Hsiao, 1999]. Furthermore, CHARM avoids enumerating all possible subsets of a closed itemset when enumerating the closed frequent sets, which rules out a pure bottom-up search. This property is important in mining dense domains with long frequent itemsets, where bottom-up approaches are not practical (for example if the longest frequent itemset is l, then bottom-up search enumerates all 2^l frequent subsets).

The exploration of both the itemset and tidset space allows CHARM to use a novel search method that skips many levels to quickly identify the closed frequent itemsets, instead of having to enumerate many non-closed subsets. Further, CHARM uses a two-pronged pruning strategy. It prunes candidates based not only on subset infrequency (i.e., no extensions of an infrequent itemset are tested) as do all association mining methods, but it also prunes branches based on non-closure property, i.e., any non-closed itemset is pruned. Finally, CHARM uses no internal data structures like Hash-trees [Agrawal, Mannila, Srikant, Toivonen, and Verkamo, 1996] or Tries [Brin, Motwani, Ullman, and Tsur, 1997]. The fundamental operation used is an union of two itemsets and an intersection of their tidsets.

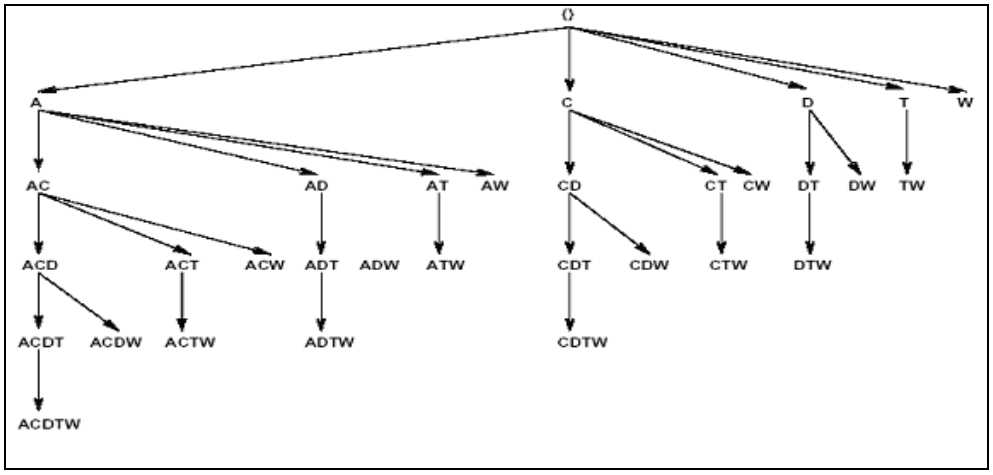


Figure 3.14: Complete Subset Lattice.

Consider Figure 3.14 which shows the complete subset lattice over the five items in the example database given in Figure 3.9. The idea in CHARM is to process each lattice node to test if its children are frequent. All infrequent, as well

as non-closed branches are pruned. Notice that, the children of each node are formed by combining the node by each of its siblings that come after it in the branch ordering. For example, A has to be combined with its siblings C; D; T and W to produce the children AC; AD; AT and AW.

A sibling need not be considered if it has already been pruned because of infrequency or non-closure. While many search schemes are possible (e.g., breadth-first, depth-first, best-first, or other hybrid search), CHARM performs a depth-first search of the subset lattice.

It is assumed that for any itemset X, there is access to its tidset t(X), and for any tidset Y there is access to its itemset i(Y). CHARM actually enumerates all the frequent concepts in the input database. Recall that a concept is given as a pair X x Y , where X = i(Y) is a closed itemset, and Y = t(X) is a closed tidset. The algorithm can start the search for concepts over the tidset space or the itemset space. However, typically the number of items is a lot smaller than the number of transactions, and since the CHARM ultimately interested in the closed itemsets, the algorithm starts the search with the single items, and their associated tidsets.

3.5.4.1 Basic Properties of Itemset-Tidset Pairs

Let $f : \mathcal{P}(I) \rightarrow N$ be a one-to-one mapping from itemsets to integers. For any two itemsets X_1 and X_2, it is said that $X_1 \leq X_2$ iff $f(X_1) \leq f(X_2)$. f defines a total order over the set of all itemsets. For example, if f denotes the lexicographic ordering, then itemset AC < AD. As another example, if f sorts itemsets in increasing order of their support, then AD < AC if support of AD is less than the support of AC.

Let's assume that the algorithm is processing the branch X_1 x $t(X_1)$, and it is needed to combine it with its sibling X_2 x $t(X_2)$. That is $X_1 \leq X_2$ (under a suitable total order f). According to the foundations of M. Zaki and C.-J. Hsiao [Zaki and Hsiao, 1999], the main computation in CHARM relies on the following *properties*:

1. If $t(X_1) = t(X_2)$, then $t(X_1 \cup X_2) = t(X_1) \cap t(X_2) = t(X_1) = t(X_2)$. Thus we can simply replace every occurrence of X_1 with $X_1 \cup X_2$, and remove X_2 from further consideration, since its closure is identical to the closure of $X_1 \cup X_2$. In other words, we treat $X_1 \cup X_2$ as a composite itemset.

2. If $t(X_1) \subset t(X_2)$, then $t(X_1 \cup X_2) = t(X_1) \cap t(X_2) = t(X_1) \neq t(X_2)$. Here we can replace every occurrence of X_1 with $X_1 \cup X_2$, since if X_1 occurs in any transaction, then X_2 always occurs there too. But since $t(X_1) \neq t(X_2)$ we cannot remove X_2 from consideration; it generates a different closure.

3. If $t(X_1) \supset t(X_2)$, then $t(X_1 \cup X_2) = t(X_1) \cap t(X_2) = t(X_2) \neq t(X_1)$. In this we replace every occurrence of X_2 with $X_1 \cup X_2$, since wherever X_2 occurs X_1 always occurs. X_1 however, produces a different closure, and it must be retained.

4. If $t(X_1) \neq t(X_2)$, then $t(X_1 \cup X_2) = t(X_1) \cap t(X_2) \neq t(X_2) \neq t(X_1)$. In this case, nothing can be eliminated; both X_1 and X_2 lead to different closures.

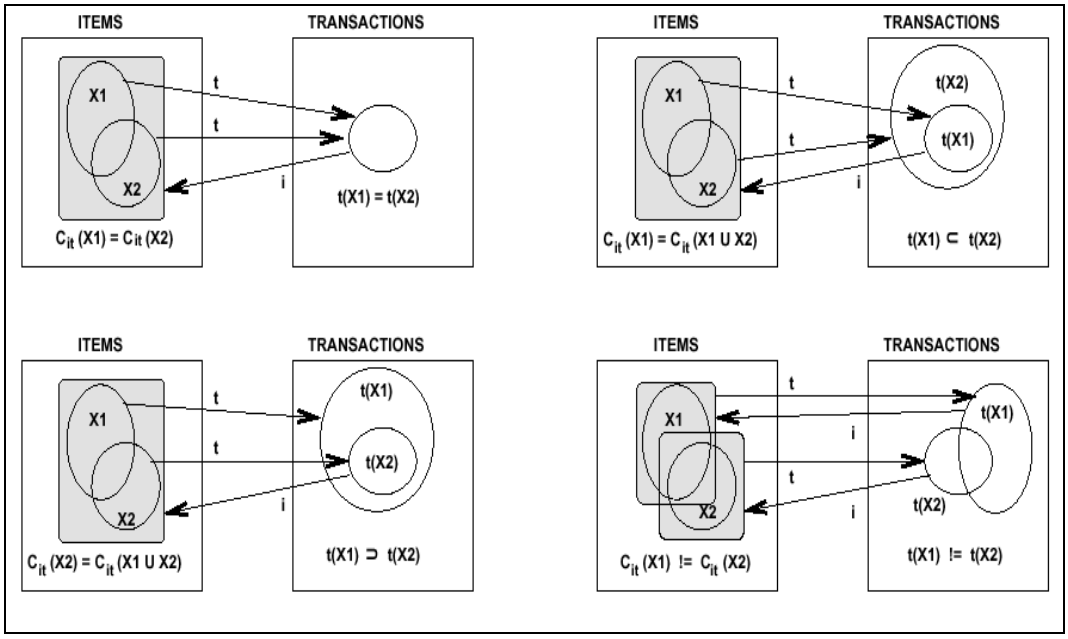


Figure 3.15: Basic Properties of Itemsets and Tidsets.

Figure 3.15 pictorially depicts the four cases. It is seen that only closed tidsets are retained after two itemset-tidset pairs are combined. For example, if the two tidsets are equal, one of them is pruned (property 1). If one tidset is a subset of another, then the resulting tidset is equal to the smaller tidset from the parent, and that parent is eliminated (properties 2 and 3). Finally if the tidsets are unequal, then those two and their intersection are all closed.

Example 3.3: Before formally presenting the CHARM algorithm, we want to show how the four basic properties of itemset-tidset pairs are exploited in CHARM to mine the closed frequent itemsets.

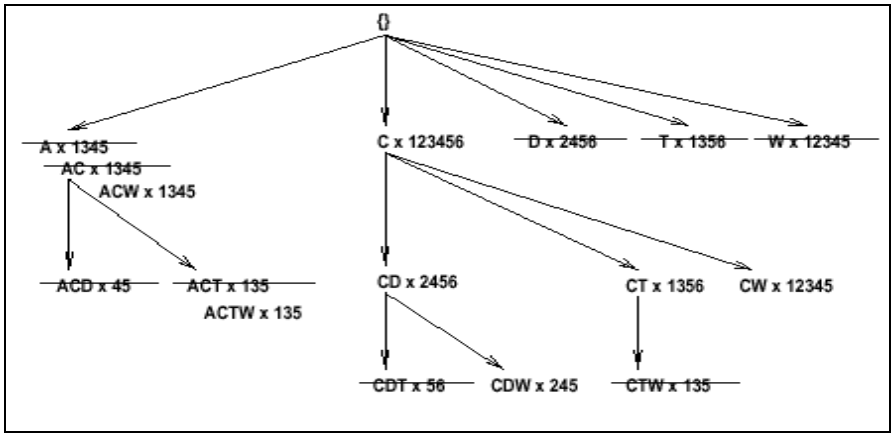


Figure 3.16: CHARM: Lexicographic Order.

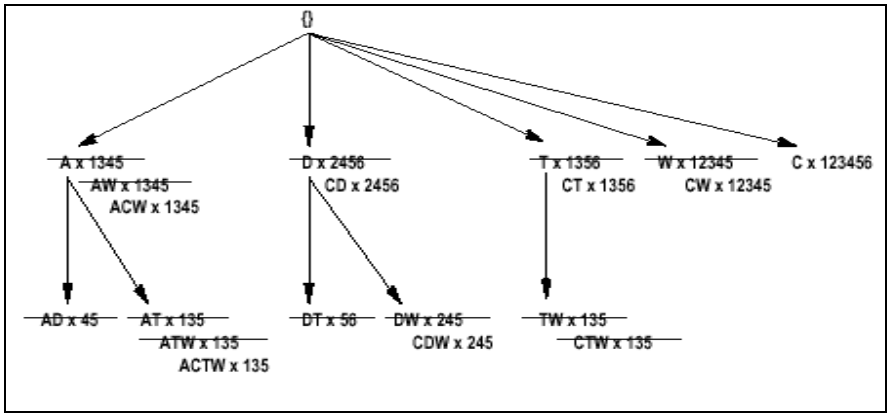


Figure 3.17: CHARM: Sorted by Increasing Support.

Consider Figure 3.16, initially there are five branches, corresponding to the five items and their tidsets from the example database (recall that the algorithm uses *minsup* = 3 as an input parameter). To generate the children of item A (or the pair A x 1345) the CHARM algorithm combines it with all siblings that come after it. When two pairs X_1 x $t(X_1)$ and X_2 x $t(X_2)$ are needed to be combined, the resulting pair can be calculated as $(X_1 \cup X_2)$ x $(t(X_1) \cap t(X_2))$. In other words, CHARM is supposed to perform the intersection of corresponding tidsets whenever it needs to combine two or more itemsets.

When A is tried to be extended with C, the property 2 will be processed, i.e., $t(A) = 1345 \subseteq 123456 = t(C)$. Thus, A will be removed and replaced with AC. Combining A with D produces an infrequent set ACD, which is pruned. Combination of AC (A) with T produces the pair ACT x 135; property 4 holds here, so nothing can be pruned. When A is tried to be combined with W it will be seen that $t(A) \subseteq t(W)$. According to property 2, the algorithm replaces all

unpruned occurrences of A with AW. Thus, AC becomes ACW and ACT becomes ACTW. At this point there is nothing further to be processed from the A branch of the root.

The algorithm now starts to process the C branch. When C is needed to be combined with D, it will be observed that property 3 holds, i.e., t(C) ⊃ t(D). This means that wherever D occurs C always occurs. Thus D can be removed from further consideration, and the entire D branch is pruned; the child CD is replaced with D. Exactly the same scenario occurs with T and W. Both the branches are pruned and are replaced by CT and CW as children of C. Continuing in a depth-first manner, the algorithm next processes the node CD. Combining it with CT produces an infrequent itemset CDT, which is pruned. Combination with CW produces CDW and since property 4 holds, nothing can be removed. Similarly the combination of CT and CW produces CTW. At this point all branches have been processed.

Finally, CTW x 135 is removed since it is contained in ACTW x 135. As it is seen, in just 10 steps the CHARM algorithm is able to identify the all 7 closed frequent itemsets.

3.5.4.2 Pseudo-Code Description

The algorithm starts by initializing the set of nodes to be examined to the frequent single items and their tidsets in Line 1. The main computation is performed in CHARM-EXTEND which returns the set of closed frequent itemsets C. The pseudo-code of the algorithm, proposed by M. Zaki and C.-J. Hsiao [Zaki and Hsiao, 1999], is given in Figure 3.18 below.

```
CHARM (δ ⊆ I × T, minsup):
1.  Nodes = {I_j × t(I_j) : I_j ∈ I ∧ |t(I_j)| ≥ minsup}
2.  CHARM-EXTEND (Nodes, C)

CHARM-EXTEND (Nodes, C):
3.  for each X_i × t(X_i) in Nodes
4.      NewN = ∅ and X = X_i
5.      for each X_j × t(X_j) in Nodes, with f(j) > f(i)
6.          X = X ∪ X_j and Y = t(X_i) ∩ t(X_j)
7.          CHARM-PROPERTY(Nodes, NewN)
8.      if NewN ≠ ∅ then CHARM-EXTEND (NewN)
9.      C = C ∪ X //if X is not subsumed

CHARM-PROPERTY (Nodes, NewN):
10. if (|Y| ≥ minsup) then
11.     if t(X_i) = t(X_j) then //Property 1
12.         Remove X_j from Nodes
13.         Replace all X_i with X
14.     else if t(X_i) ⊂ t(X_j) then //Property 2
15.         Replace all X_i with X
16.     else if t(X_i) ⊃ t(X_j) then //Property 3
17.         Remove X_j from Nodes
18.         Add X × Y to NewN
19.     else if t(X_i) ≠ t(X_j) then //Property 4
20.         Add X × Y to NewN
```

Figure 3.18: The CHARM Algorithm.

CHARM-EXTEND is responsible for testing each branch for viability. It extracts each itemset-tidset pair in the current node set, Nodes (X_i x t(X), Line 3), and combines it with the other pairs that come after it, (X_j x t(X_j), Line 5), according to *support-based ordering* shown in Figure 3.17 above. The combination of the two itemset-tidset pairs is computed in Line 6. The routine CHARM-PROPERTY tests the resulting set for required support and also applies the four properties discussed above. Note that this routine may modify the current node set by deleting itemset-tidset pairs that are already contained in other pairs. It also inserts the newly generated children frequent pairs in the set of new nodes NewN. If this set is non-empty, the CHARM-EXTEND is processed recursively in depth-first manner (Line 8). Then the possibly extended itemset X, of X_i, is inserted in the set of closed itemsets, since it cannot be processed further; at this stage any closed itemset containing X_i has already been generated. Then the algorithm returns to Line 3 to process the next (unpruned) branch.

The routine CHARM-PROPERTY simply tests if a new pair is frequent, discarding it if it is not. It then tests each of the four basic properties of itemset-tidset pairs, extending existing itemsets, removing some subsumed branches from the current set of nodes, or inserting new pairs in the node set for the next (depth-first) step.

3.5.4.3 Branch Reordering

The CHARM algorithm purposely lets the itemset-tidset pair ordering function in Line 5 remain unspecified. The usual manner of processing is in lexicographic order shown in Figure 3.16, but one may specify any other total order. The most promising approach is to sort the itemsets based on their support. The motivation is to increase opportunity for non-closure based pruning of itemsets. A quick look at properties 1 and 2 tells us that these two situations are preferred over the other two cases. For property 1, the closure of the two itemsets is equal, and thus X_j can be discarded and X_i can be replaced with $X_i \cup X_j$. For property 2, X_i can still be replaced with $X_i \cup X_j$. Note that in both these cases the algorithm does not insert anything in the set of new nodes. Thus the more the occurrence of case 1 and 2, the fewer levels of search the CHARM performs. In contrast, the occurrence of cases 3 and 4 results in additions to the set of new

nodes, requiring additional levels of processing. Note that the reordering is applied for each new node set, starting with the initial branches.

Since $t(X_i) = t(X_j)$ or $t(X_i) \subset t(X_j)$ is a desired situation, it follows that the itemsets should be sorted in increasing order of their support. Thus, larger tidsets occur later in the ordering and the occurrence of properties 1 and 2 is maximized. By similar reasoning, sorting by decreasing order of support doesn't work very well, since it maximizes the occurrence of properties 3 and 4, increasing the number of levels of processing.

Example 3.4: Figure 3.14 shows how CHARM works on the example database if we sort itemsets in increasing order of support. We will use the pseudo-code to illustrate the computation.

The algorithm initializes Nodes = {A x 1345, D x 2456, T x 1356, W x 12345, C x 123456} in Line 1. At Line 3 the algorithm first processes the branch A x 1345 (X = A is set in Line 4); it will be combined with the remaining siblings in Line 5. AD is not frequent and is pruned. The algorithm next looks at A and T; since $t(A) \neq t(T)$, AT is simply inserted in NewN. Next, it is found that $t(A) \subset t(W)$. Thus, all occurrences of A is replaced with AW (that is X = AW), which means that also AT in NewN is changed to ATW. Looking at A and C, the algorithm finds that $t(A) \subset t(C)$. Thus, AW becomes ACW (X = ACW), and ATW in NewN becomes ACTW. At this point CHARM-EXTEND is invoked with the non-empty NewN (Line 8). But since there is only one element, the CHARM-EXTEND immediately returns after adding ACTW x 135 to the set of closed frequent itemsets C (Line 9).

When the CHARM-EXTEND returns, the A branch has been completely processed, and X = ACW is added to C. The other branches are examined in turn, and the final C is produced as shown in Figure 3.17. One final note; the pair CTW x 135 produced from the T branch is not closed, since it is subsumed by ACTW x 135, and it is eliminated in Line 9.

3.5.5 Correctness and Efficiency

Theorem 3.2 (Correctness): The CHARM algorithm enumerates all closed frequent itemsets.

Proof: CHARM correctly identifies all and only the closed frequent itemsets, since its search is based on a complete subset lattice search. The only branches that are pruned as those that either do not have sufficient support, or those that result in non-closure based on the properties of itemset-tidset pairs. Finally, CHARM eliminates the few cases of non-closed itemsets that might be generated by performing subsumption checking before inserting anything in the set of all closed frequent itemsets C.

Theorem 3.3 (Computational Cost): The running time of CHARM is O(l – |C|), where l is the average tidset length, and C is the set of all closed frequent itemsets.

Proof: Note that starting with the single items and their associated tidsets, while a branch is being processed the following cases might occur: Let X_c denote the current branch and X_s the sibling it is tried to be combined with it. The X_s branch is pruned if $t(X_c) = t(X_s)$ (property 1). X_c is extended to become $X_c \cup X_s$ if $t(X_c) \subset t(X_s)$ (property 2). X_s is pruned if $t(X_c) \supset t(X_s)$, and X_s is extended to become $t(X_c) \subset t(X_s)$ (property 3). Finally a new node is only generated if there remained a new possibly closed set due to properties 3 and 4. Also note that each new node in fact represents a closed tidset, and thus indirectly represents a closed itemset, since there exists a unique closed itemset for each closed tidset. Thus CHARM performs on the order of O(|C|) intersections. If each tidset is on average of length l, an intersection costs at most $2 \cdot l$. The total running time of CHARM is thus $2 \cdot l \cdot |C|$ or $O(l \cdot |C|)$.

Theorem 3.4 (I/O cost): The number of database scans made by CHARM is given as $O\left(\frac{|C|}{\alpha \cdot |I|}\right)$, where C is the set of all closed frequent itemsets, I is the set of items, and α is the fraction of database that fits in memory.

Proof: The number of database scans required is given as the total memory consumption of the algorithm divided by the fraction of database that will fit in memory. Since CHARM computes on the order of O(|C|) intersections, the total memory requirement of CHARM is $O(l \cdot |C|)$, where l is the average length of a tidset. The total database size is $l \cdot |I|$, and the fraction that fits in memory is given as $\alpha \cdot l \cdot |I|$. The number of data scans is then given as $(l \cdot |C|)=(\alpha \cdot l \cdot |I|) = |C|=(\alpha \cdot |I|)$.

Note that in the worst case |C| can be exponential in |I|, but this is rarely the case in practice.

Chapter 4

IMPLEMENTATION OF ALGORITHMS AND SAMPLE DATA SETS

In order to compare the algorithms, experimental studies using different data sets (two synthetic data sets and a real data set) and different support levels have been conducted. In this chapter we present our approaches with respect to the implementation stages of the Apriori, the FP-tree and the CHARM algorithms.

All the experiments were performed on a 600 MHz Intel Pentium PC machine with 128 megabytes main memory, running on Microsoft Windows/NT. All the programs were source coded in JAVA due to its Collections Framework (java.util package)'s advantages which provides magical facilities for implementing the data structures. As a Java Virtual Machine we utilized the one running on Java Standart Development Kit - Version 1.4.1.

4.1 Implementation of Algorithms and Comparison Criteria

The Apriori, the FP-tree and the CHARM algorithms are explained in detail in Chapter 3. In this section, we present our implementation techniques and programming approaches for the algorithms.

4.1.1 Input Parameters of the Algorithms

All algorithms take the following input parameters:

- Support (user specified treshold) percentage.
- Input file name (an input data set).
- Output file name (the file that contain the output of the algorithm).

4.1.2 Outputs Generated by the Algorithms

4.1.2.1 Data Features

- input file name
- number of elements
- number of fields(or items)
- number of transactions
- longest transaction size
- average transaction size

4.1.2.2 Performance Statistics

- data preparation time
- mining time
- run time

4.1.2.3 Number of Associations

The associations extracted by the algorithms are not identical. In other words, they depend on the nature of the algorithm. Apriori algorithm finds all "*frequent itemsets*" which satisfy the treshold. Since FP-tree algorithm designed for pattern the mining problem, it extracts all "*prefix path*", "*conditional pattern base*" and "*frequent pattern*" information. CHARM algorithm eliminates non-closed itemsets while determining frequent itemsets, and finally it presents us all "*closed frequent itemsets*".

4.1.3 Implementation of Apriori Algorithm

Most of the algorithms which have candidate generation and testing procedures(like Apriori algorithm) use special data structures or techniques in order to manage the "search" task. These sophisticated techniques play remarkable roles on the performances of the algorithms.

When we take a look at IBM's Quest team project described in Section 3.3, we see that the Apriori algorithm has apriori-gen() and subset() sub-functions performing candidate generation and membership (subset) testing respectively. To reduce the time needed for the membership testing, the Apriori algorithm uses a hash-tree for storing large itemsets.

Since the FP-tree and the CHARM algorithms weren't need any special search techniques, like-wise, we ordinarily prefered to use the sequential search technique in the implementation of the Apriori algorithm. Considering this, the pseudo-code of the Apriori algorithm which we used in in our source code implementation has been given in Figure 4.1.

```
C_k: candidate item set of size k
L_k: frequent item set of size k

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

Figure 4.1: Pseudo-code of Apriori algorithm used in our study.

4.1.4 Implementation of FP-Tree Algorithm

As we described in Chapter 3, the FP-tree algorithm includes the following two steps:

1) Composing the frequent pattern tree by "FP-tree Construction algorithm".
2) Mining frequent patterns with "Pattern Fragment Growth algorithm".

These steps are described in Section 3.4.2.2 and in Section 3.4.3.3 respectively. In our source code implementation we obeyed the same routines.

4.1.5 Implementation of CHARM Algorithm

In the course of both the investigation step and the source code implementation step, we realized that the pseudo-code description of the CHARM algorithm, proposed by M.J. Zaki and C.-J. Hsiao [Zaki and Hsiao, 1999] had some notation deficiencies due to which we could not manage to find all closed frequent itemsets.

4.1.5.1 Deficiency Observations in the Pseudo-Code Description

The main idea behind the CHARM algorithm is described in Chapter 3. Example 3.3 describes how the algorithm works in a bird's eye view. However,

when we take a close look at the pseudo-code we can easily see the following fundamental deficiency in CHARM-EXTEND (in Figure 3.11 - line 6):

For all X_i, since the variable X itself is combined with each node according to the condition f(j) > f(i), it will also contain the non-frequent items. This observation caused us to make some modifications in the pseudo-code of the algorithm.

4.1.5.2 Modifications in the Pseudo-Code Description

We first substituted the assignment $X = X \cup X_j$ (in CHARM-EXTEND-line 6) with $X = X_i \cup X_j$. This substitution yielded adding two more variables, $\overline{X}$ and $\overline{Y}$, to the pseudo-code. Succedingly, the variable X in line 4 substituted with the variable $\overline{X}$ and moreover the assignment $\overline{Y} = t(X_i)$ added to line 4. In this sense, variable X in line 9 substituted with $\overline{X}$ and the assignment $C = C \cup \overline{Y}$ added to line 9. As shown in Figure 4.2 (b), also the sub-routine CHARM-PROPERTY was re-arranged by modifying the lines 4 and 6.

Through the help of these modifications, we can obtain the set of closed frequent itemsets, C, for each item in Nodes. However, when the set of all closed frequent itemsets for the transactional database is considered, it goes without saying that we need one more sub-routine to find all closed frequent itemsets for the whole transactional database.

The GET-CLOSED function has been designed to determine and prune the non-closed itemsets in C. Consider C = { ACTW x 135, ACW x 1345, CDW x 245, CD x 2456, CTW x 135, CT x 1356, CW x 12345, C x 123456 } be a set of closed itemset-tid pairs processed by CHARM-EXTEND. Since the itemsets ACTW and CTW have the same tidset, GET-CLOSED simply will the prune the pair CTW x 135 as it is also a subset of ACTW. Note that the main idea behind the GET-CLOSED function is based on the basic properties of itemset-tidset pairs.

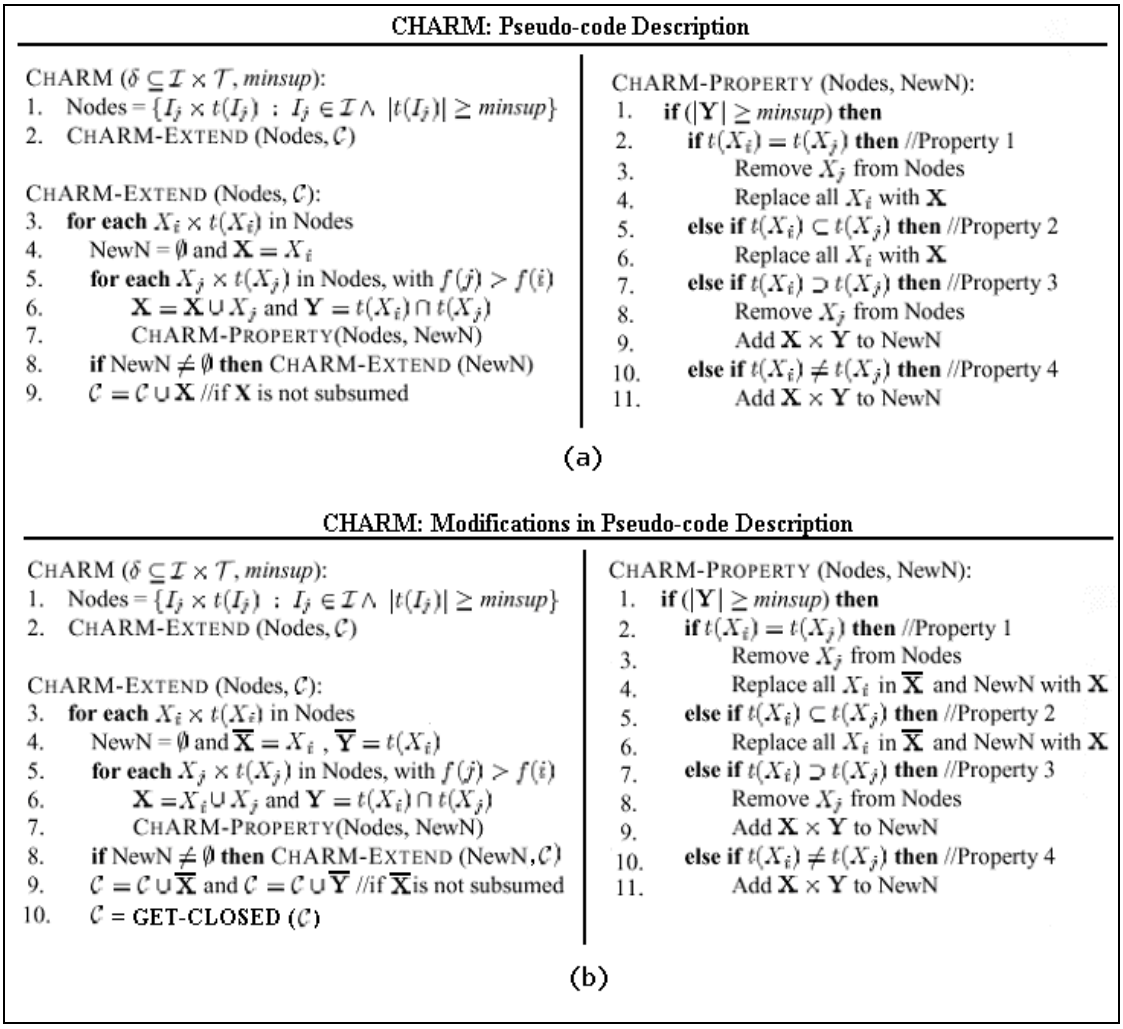


Figure 4.2: The main pseudo-code of the CHARM algorithm and Modifications in the pseudo-code are shown in (a) and (b) respectively.

4.1.5.3 Impacts of Modifications on CHARM Algorithm

After the proposed modifications, the CHARM algorithm is as follows:

Let C be a set of closed frequent itemsets for each node. The main computation for the composition of C is performed by the CHARM-EXTEND function. It extracts each itemset-tidset pair, (X_i x $t(X_i)$), in the current node set, Nodes, and combines it with the other pairs that come after it, (X_j x $t(X_j)$). The CHARM-PROPERTY function tests the resulting set for the required support and also applies the itemset-tidset properties. Note that, this routine modifies the current node set by deleting itemset-tidset pairs that are already contained in other pairs. It also inserts the newly generated children frequent pairs in the set of new nodes, newN. If this set is non-empty, the next(unpruned) branch is processed recursively in the dept-first manner. Then, the possibly extended itemset $\overline{\mathbf{X}}$ with

the current tidset $\overline{Y}$ can be inserted to the set of closed itemsets, since it cannot be processed furher. At this stage, any closed itemset containing $\overline{X}x\overline{Y}$ pair has already been granted. After finding all the closed frequent itemsets for each node, the GET-CLOSED function is called. This function simply prunes the smaller-length itemsets in C that have the same tidsets. As a result, the remaining set refers to the set of of all closed frequent itemsets.

4.1.6 Comparison Criteria

Algorithms were compared according to their data preparation times, mining times, run times and the number of associations they extracted.

4.1.6.1 Data Preparation Time

Before mining the data set the algorithm has to prepare the data for its requirements (for the mining process). Thus, data preparation step can vary from algorithm to algorithm.

Since Apriori already makes multiple passes over the data during the mining process, just reading the data set is adequate for its data preparation. The CHARM algorithm is supposed to compose the “Nodes” in order to start the mining process. So that, the data preparation for the CHARM algorithm includes first reading the data set and then composing the “Nodes” in an ascending order. Data preparation for the FP-tree algorithm includes scanning the transactional database once, collecting the set of frequent items and their supports and sorting the frequent items in a support descending order. We did not consider the FP-tree construction process as a data preparation step, because it is a special kind of process on its own.

4.1.6.2 Mining Time

Each algorithm evaluated in this study uses different strategies to mine data sets. Apriori algorithm makes multiple passes over the database. Each pass consists of scanning, pruning and joining steps. At the end of each joining step, the algorithm generates new candidates for the next pass. The mining process continues until no more candidades can be generated. The FP-tree algorithm first constructs a FP-tree and then starts to mine the frequent patterns according to

pattern fragment growth(FP-growth) algorithm. The CHARM uses itemset-tidset properties in order to perform the mining process. It searches the subset lattice in a dept-first manner. The routine CHARM-PROPERTY tests the resulting set for required support and also applies the four itemset-tidset properties. This routine also inserts the newly generated children frequent pairs in the set of new nodes, NewN. If this set is non-empty we recursively call the CHARM-EXTEND function and process new nodes in a depth-first manner (Figure 4.2 (b) CHARM-EXTEND – line 8). This mining process continues until no more branches left.

4.1.6.3 Run Time

The run time of the algorithms has been calculated by simply adding the mining time to the data preparation time: *run_time = data_preparation_time + mining_time.* As we were involved in the performance measurement we did not take into account of the time needed for IO (Input / Output) routines in the calculation of the run time.

4.1.6.4 Number of Associations

The word "association" includes frequent patterns, frequent itemsets, sequential patterns, etc. The Apriori Algorithm finds all frequent itemsets, the FP-tree algorihm first determines the prefix paths, then develops the conditional pattern bases and finally finds all the frequent patterns. The CHARM algorithm finds the closed frequent itemsets which may have smaller cardinalities than the set of all frequent itemsets.

4.2 Synthetic Data Generation and Statistical Anaylsis of the Data Sets

The algorithms were tested on three different data sets: A real data set and two synthetic data sets.

The real data set, "*Northwind",* is a sample database taken from *Microsoft Access Database*. Only the *"order_details"* table of this database is used in this study. The statistical features of Northwind (only the *order_details* table) table are decribed in Section 4.2.2.

Having tested the algorithms on the Northwind database, we had been involved in the observation of behaviours of the algorithms on synthetic data sets

having less and more elements than Northwind. In this sense, we produced two different data sets which have 1000 and 5000 transactions. These synthetic data sets were simulated using a pre-defined probability mass tables, that is, the probability of occurences of the items and the number of transactions that the data set contain were pre-designated. The synthetic data generation routines that we used in our study are described in Section 4.2.1 and the statistical features of the synthetic data sets with 1000 and 5000 transactions are described in Section 4.2.3 and Section 4.2.4 respectively.

4.2.1 Synthetic Data Set Generation

We managed 5 steps for generating a synthetic data sets. Our data generation procedure as follows:

1) Determine the borders of the data sets. Such as;

- maximum transaction size: 20
- number of items (or fields): 50
- number of transactions: 1000 and 5000

2) To hold the "Transaction Size Distributions" and the "Item Distributions" define two matrices, Transaction_Size_Matrix (1 x 20) and Item_Matrix (1 x 50). The Transaction_Size_Matrix and the Item_Matrix are shown in Figure 4.3.

Transaction Size Distribution

Transaction Size	Probability of Occurence
1	0.03
2	0.04
3	0.05
4	0.07
5	0.08
6	0.08
7	0.09
8	0.10
9	0.08
10	0.07
11	0.07
12	0.05
13	0.04
14	0.03
15	0.04
16	0.03
17	0.02
18	0.01
19	0.01
20	0.01

Item Distribution

Item ID	Probability of Occurence	Item ID	Probability of Occurence
1	0.077	26	0.013
2	0.069	27	0.011
3	0.056	28	0.011
4	0.055	29	0.011
5	0.049	30	0.010
6	0.048	31	0.010
7	0.046	32	0.009
8	0.044	33	0.008
9	0.036	34	0.007
10	0.035	35	0.007
11	0.034	36	0.006
12	0.032	37	0.006
13	0.030	38	0.006
14	0.030	39	0.005
15	0.028	40	0.004
16	0.025	41	0.004
17	0.023	42	0.004
18	0.022	43	0.003
19	0.020	44	0.002
20	0.018	45	0.002
21	0.017	46	0.002
22	0.016	47	0.002
23	0.015	48	0.001
24	0.015	49	0.001
25	0.014	50	0.001

Figure 4.3: Pre-designated probability distributions for synthetic data generation: Transaction Size Distribution and Item Distribution.

3) Sort the probabilitiy of occurences that Transaction_Size_Matrix contain (in a descending order).

4) Extend the probability of occurences of items that Item_Matrix hold. In other words, create a 1 x 1000 matrix (let it be A_Matrix) that will contain the items according to their probability of occurences. For example, in A_Matrix, the item "1" will occupy the first 1000*0.077 place and item 2 will occupy the following 1000*0.069 place,...,etc. As a result, the A_Matrix will contain the items from 1 to 50 in a consecutive order: <1...1 , 2...2 , 3...3 , . . . , 47 47 , 48, 49 , 50>. In this sense, you can see that the size of the A_Matrix will be 1000.

5) Having determined the data features, designated the probability of item occurences and the probability of transaction size occurences in 4 steps, the data generation algorithm can be proceeded as it is shown in Figure 4.4.

```
1.  dataGeneration()
2.      for(counter1=0; counter1<numberOfTransactions; counter1++)
3.          transactionSize = getTransactionSize();
4.          for(counter2=0; counter2<transactionSize; counter2++)
5.              itemBought = getItem();
6.              addToTransaction(itemBought); //IO routine
```

Figure 4.4: Synthetic data generation algorithm.

The sub-routine getTransactionSize() in line 3 generates a random number (according to uniform distribution) between 0 and 1 and returns the index which corresponds to the random number in Transaction_Size_Matrix.

The sub-routine getItem() in line 5, generates a random number (according to uniform distribution) between 1 and 1000 (the size of the A_Matrix) and returns the value(item id) that corresponds to the random number in A_Matrix.

4.2.2 Statistical Analysis of "Northwind" Database

The features and the statistical analysis of the order_details table of the Northwind database are given in Figure 4.5 and Figure 4.6 respectively.

Name	Size
Number of elements	2155
Number of fields	77
Number of transactions	830
Longest transaction size	25
Average transaction size	2.596

Figure 4.5: Features of "order-details" table of "Northwind" database.

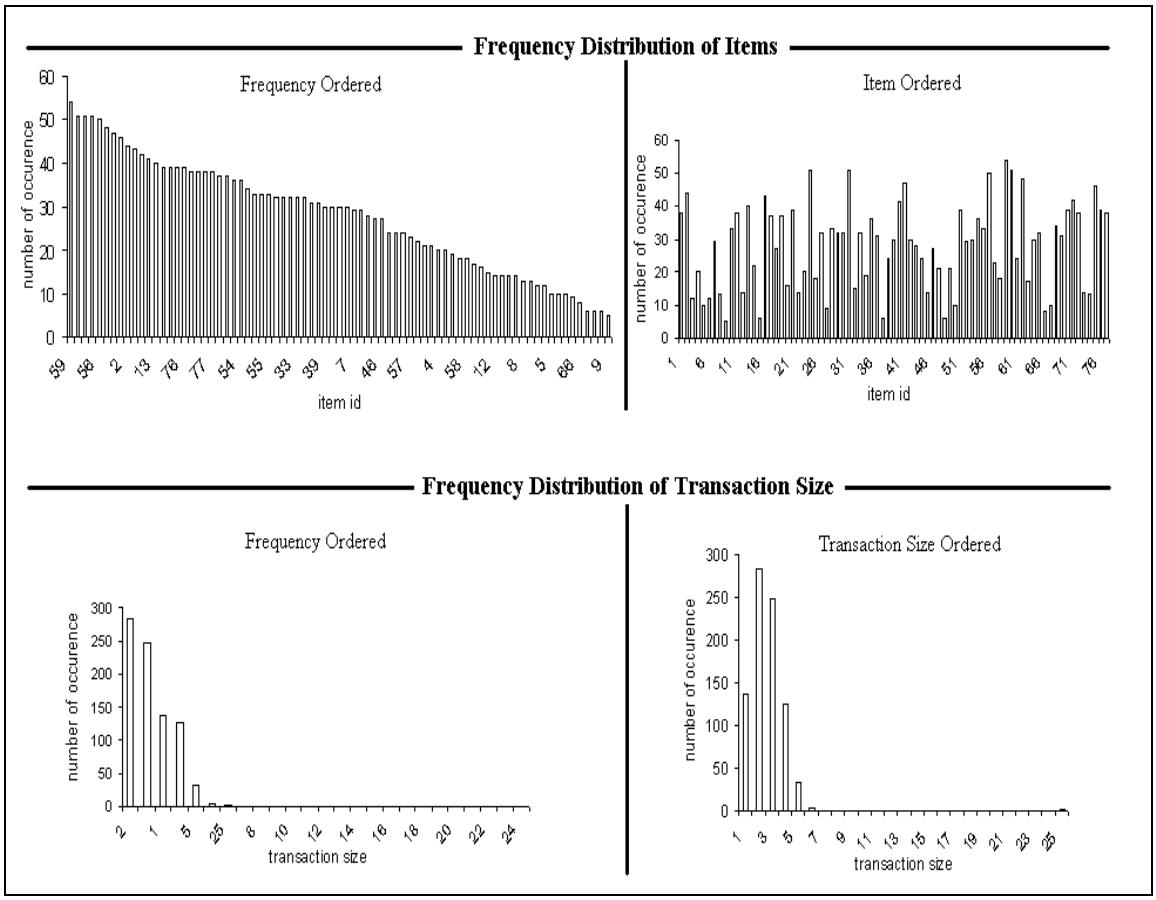


Figure 4.6: Statistical anaysis of "order-details" table of "Northwind" database.

4.2.3 Statistical Analysis of "Synthetic Data Set with 1000 Transactions"

The features and the statistical analysis of the "Synthetic Data Set with 1000 Transactions" are given in Figure 4.7 and Figure 4.8 respectively.

Name	Size
Number of elements	7134
Number of fields	50
Number of transactions	1000
Longest transaction size	20
Average transaction size	7.134

Figure 4.7: Features of "Synthetic Data Set with 1000 Transactions".

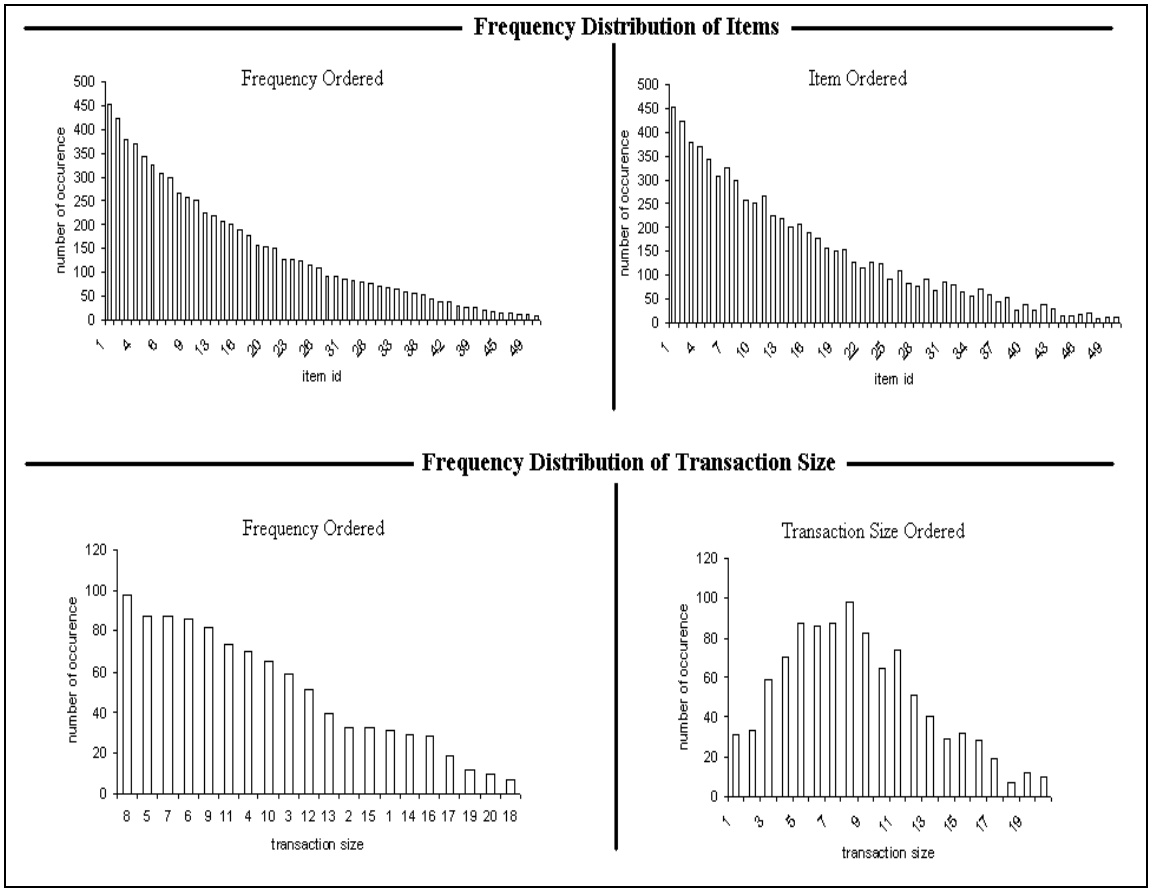


Figure 4.8: Statistical anaysis of "Synthetic Data Set with 1000 Transactions".

4.2.4 Statistical Analysis of "Synthetic Data Set with 5000 Transactions"

The features and the statistical analysis of the "Synthetic Data Set with 1000 Transactions" are given in Figure 4.9 and Figure 4.10 respectively.

Name	Size
Number of elements	35861
Number of fields	50
Number of transactions	5000
Longest transaction size	20
Average transaction size	7.132

Figure 4.9: Features of "Synthetic Data Set with 5000 Transactions".

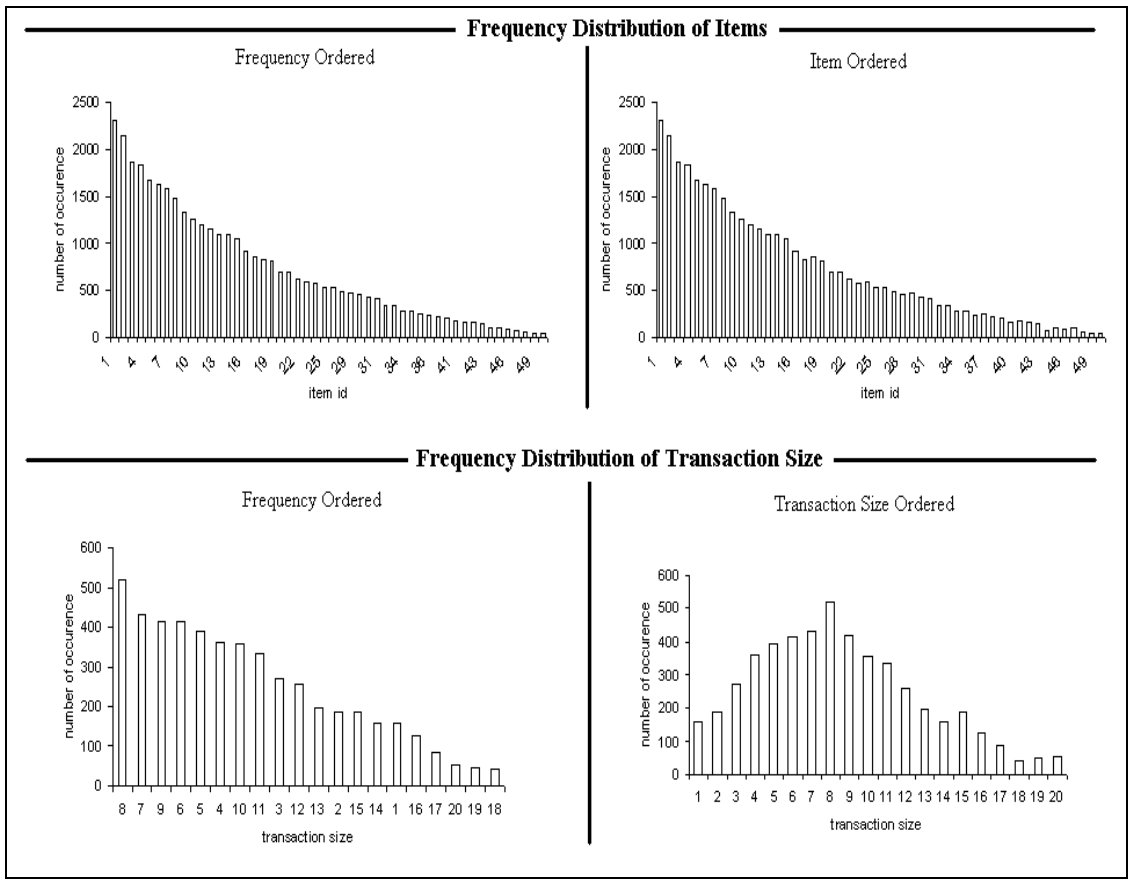


Figure 4.10: Statistical anaysis of "Synthetic Data Set with 5000 Transactions".

Chapter 5

RESULTS AND DISCUSSION

As the result of the experimental study, we revealed the performance statistics and the knowledge extraction capabilities of the algorithms. In this chapter, we visualize, evaluate and compare the results obtained from our experimental study as regards the "*performance statistics"* and *knowledge extraction capabilities"* of the algorithms.

5.1 Comparison of the Data Preparation Times

As is mentioned in Section 4.1.6.1, all algorithms have to make preparations over the data set before they start mining. Since each algorithm utilize a different strategy for its mining process, it is expected that the data preparation times of the algorithms differ from each other.

We know that, since the Apriori algorithm already makes multiple passes over the data during the mining process, it is sufficient for the algorithm just reading the data set once for its data preparation step. However, as described in Section 4.1.6.1, the FP-tree the CHARM algorithms have additional steps for their data preparations. When these extra steps of FP-tree and CHARM algorithms are taken into consideration, it is expected that the Apriori algorithm should be more efficient in terms of data preparation performance than the FP-tree and the CHARM algorithms.

Accordingly, when we take a glance at the experimental results in Figure 5.1 we see that the CHARM algorithm spends much more time than the FP-tree algorithm. This inferiority of CHARM stems from the time spent for generating the set of *Nodes*. Actually, while the FP-tree algorithm composes the *header table* and determines the frequent items of each transaction with reference to the order of the items in the header table, the CHARM algorithm first finds the frequent items and then determines the *tidset*s for each frequent item to be inserted to the set of *Nodes*.

The experimental results shown in Figure 5.1(a), Figure 5.1(b) and Figure 5.1(c) reveal that the features of the data set which the algorithm is executed for influence the data preparation performance of the algorithm to a large extend. When the size (the number of elements) of the data set grows, the gap between the data prepration time of the algorithms exaggerates.

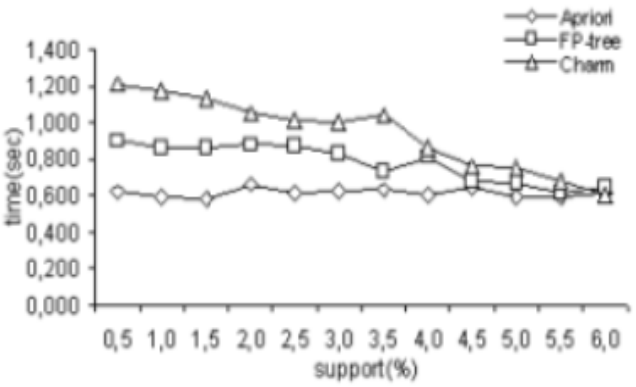
The support levels may also effect the data preparation performances of some algorithms. As shown in Figure 5.1(a), Figure 5.1(b) and Figure 5.1(c) respectively, at high support levels (where many items are able to be pruned at the outset) the FP-tree and the CHARM algorithms spend less time for the data preparation (i.e., for Northwind data as we increase the support from 0,5% to 6,0% support, the data preparation time for the CHARM algorithm decreases from 1,212 secs to 0,601 secs). Since the Apriori algorithm independently preparares the data set for its mining process from the support level, the data preparation performance of the algorithm goes linear (i.e., for Northwind data it is approximately 0,6 secs) through all supports levels. Additionally, at high support levels (such as 6,0%) due to the elimination of infrequent items at the initial steps, the data preparation performances of the algorithms are almost similar [Figure 5.1(a)].

Consequently, due to the fact that the algorithms initially should read the data set at least once, the FP-tree and the CHARM algorithms can never exceed the data preparation performance of the Apriori algorithm even sparse data sets or high support levels concerned.

Data Preparation Times of the Algorithms

For Northwind Data Set

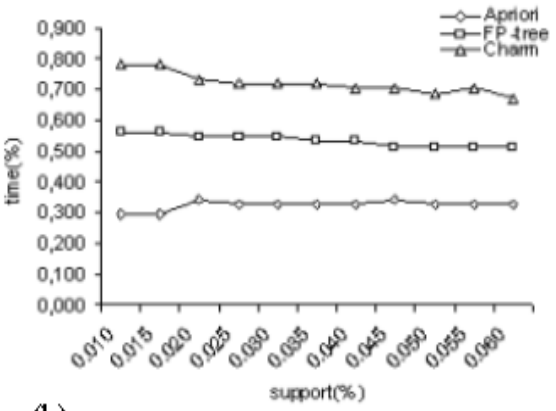
	Data Preparation Time(sec)		
Support(%)	Apriori	FP-tree	Charm
0,5	0,620	0,902	1,212
1,0	0,591	0,861	1,172
1,5	0,581	0,862	1,131
2,0	0,661	0,881	1,052
2,5	0,611	0,871	1,011
3,0	0,621	0,831	1,002
3,5	0,631	0,731	1,042
4,0	0,601	0,811	0,861
4,5	0,641	0,681	0,761
5,0	0,591	0,671	0,751
5,5	0,591	0,611	0,681
6,0	0,611	0,650	0,601



(a)

For Synthetic Data Set with 1000 Transactions

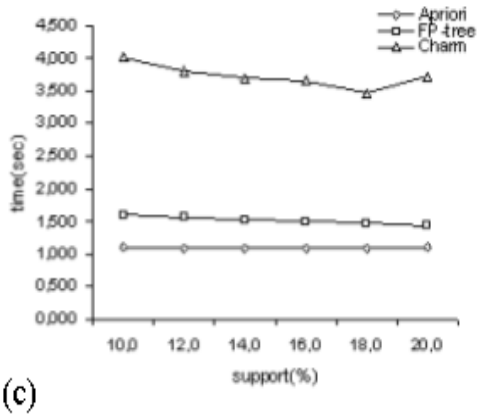
	Data Preparation Time(sec)		
Support(%)	Apriori	FP-tree	Charm
0.010	0,297	0,562	0,781
0.015	0,297	0,563	0,781
0.020	0,344	0,547	0,734
0.025	0,328	0,547	0,718
0.030	0,329	0,547	0,719
0.035	0,328	0,531	0,719
0.040	0,328	0,531	0,704
0.045	0,344	0,516	0,704
0.050	0,328	0,516	0,688
0.055	0,328	0,516	0,704
0.060	0,328	0,516	0,672



(b)

For Synthetic Data Set with 5000 Transactions

	Data Preparation Time(sec)		
Support(%)	Apriori	FP-tree	Charm
10,0	1,109	1,594	4,031
12,0	1,078	1,547	3,813
14,0	1,078	1,515	3,703
16,0	1,078	1,500	3,656
18,0	1,079	1,469	3,453
20,0	1,093	1,438	3,719



(c)

Figure 5.1: The "Data Preparation Times" of the algorithms for different data sets and at different support levels are given in (a), (b) and (c).

5.2 Comparison of the Mining Times of the Algorithms

As is described in Chapter 3, each algorithm uses a different kind of mining technique developed in the wake of its herediterial philosophy.

The Apriori algorithm makes as many passes in order to find the most frequent itemsets as the user specified support permits. Each pass includes the *scanning*, the *pruning* and the *joining* (*candidate generation*) steps. At the end of each pass the algorithm prepares a new data set (called the *candidate set*) for the next pass. For this reason, it is expected that Apriori may not show a good performance on dense data sets (where many itemsets cannot be pruned and form the candidate sets for the next pass).

The FP-tree algorithm first constructs the FP-tree and then mines the frequent patterns. The mining process is performed by the FP-growth algorithm. As is described in Section 3.4.3.3, the FP-growth algorithm uses a partitioning-based divide-and-conquer process over the compact FP-tree structure. Apart from the Apriori algorithm, the FP-tree algorithm never deals with the candidate set generation and testing. Additionally, the compactness of the FP-tree provides many advantages in terms of mining performance when compared with Apriori.

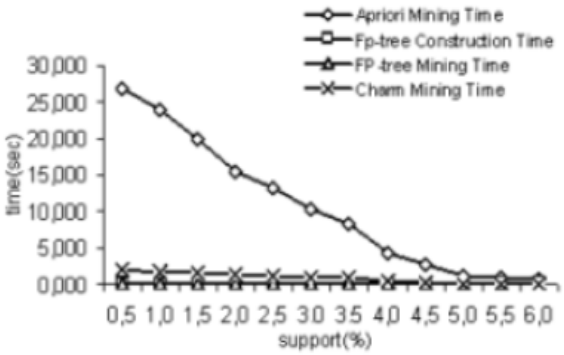
The CHARM algorithm implements a depth-first mining strategy for mining the closed frequent itemsets instead of mining the set of all frequent itemsets. The *itemset-tidset properties*, described in Section 3.5.4.1, make significant contributions to the performance of the aforesaid mining process. However, in circumstances including the frequent occurences of the 3rd and the 4th itemset-tidset properties for the pairs in *newN* (Section 3.5.4.2), the CHARM algorithm is expected to show an inferior mining performance in comparison to Apriori.

Mining Times of the Algorithms

For Northwind Data Set

Mining Time(sec)

Support(%)	Apriori	FP-tree construction	FP-tree	Charm
0,5	26,859	0,040	0,060	1,963
1,0	23,864	0,040	0,050	1,702
1,5	19,908	0,060	0,040	1,542
2,0	15,423	0,041	0,060	1,332
2,5	13,169	0,040	0,050	1,172
3,0	10,395	0,030	0,050	0,941
3,5	8,202	0,040	0,020	0,971
4,0	4,246	0,030	0,030	0,421
4,5	2,754	0,020	0,020	0,270
5,0	1,212	0,000	0,010	0,130
5,5	0,881	0,000	0,010	0,070
6,0	0,701	0,010	0,000	0,030

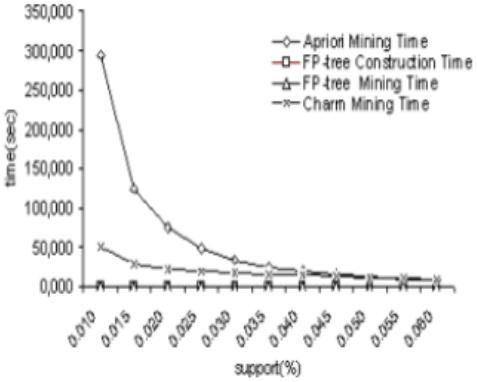


(a)

For Synthetic Data Set with 1000 Transactions

Mining Time(sec)

Support(%)	Apriori	FP-tree construction	FP-tree	Charm
0.010	294,610	0,063	0,000	51,078
0.015	124,423	0,062	0,188	28,406
0.020	75,639	0,062	0,188	22,453
0.025	49,297	0,063	0,172	18,953
0.030	34,343	0,047	0,188	17,657
0.035	25,188	0,063	0,203	15,735
0.040	20,125	0,063	0,187	13,922
0.045	15,626	0,062	0,188	12,515
0.050	12,672	0,063	0,187	11,516
0.055	9,813	0,063	0,172	11,218
0.060	8,329	0,047	0,172	9,547

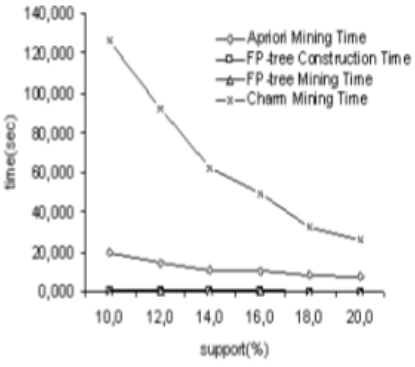


(b)

For Synthetic Data Set with 5000 Transactions

Mining Time(sec)

Support(%)	Apriori	FP-tree construction	FP-tree	Charm
10,0	19,625	0,656	0,000	126,422
12,0	14,171	0,500	0,360	92,453
14,0	10,625	0,360	0,328	62,375
16,0	10,000	0,344	0,359	49,500
18,0	8,423	0,203	0,219	32,813
20,0	7,626	0,156	0,187	26,235



(c)

Figure 5.2: The "Mining Times" of the algorithms for different data sets and at different support levels are given in (a), (b) and (c).

Obviously, we see from the experimental results [Figure 5.2(a), Figure 5.2(b), Figure 5.2(c)] that although the mining performance of the FP-tree algorithm remains almost the same, the mining performance of the Apriori and the

CHARM algorithms are effected from the changes in support levels and from the change in the size of the data sets, to a great extend. In other words, when compared with Apriori and CHARM, the FP-tree algorithm shows a robust mininig performance [i.e., the mining time is almost the same at the supports 0,015% and 0,060% as shown in Figure 5.2(b)].

The experimental results show that, on small data sets [Figure 5.2(b) and Figure 5.2(c)] of few transactions the CHARM algorithm shows a better mining performance than Apriori. However, for mining large data sets, like 5000 transactions [Figure 5.2(c)], the Apriori algorithm is considerably superior to CHARM.

Another obvious point of the experimental results is that, when compared with the Apriori and the CHARM, even in unfavourable conditions (such as at low support levels or on large transactional data sets) the FP-tree algorithm shows a spectacularly rapid mining performance. This result can be explained by the compactness of the FP-tree which is described in Section 3.4.2.3. In other words, the whole size of an FP-tree is bounded by the overall occurrences of the frequent items in the database. For this reason, the FP-tree algorithm shows almost the same mining performance for all conditions (robustness).

The experiments [Figure 5.2(a), Figure 5.2(b), Figure 5.2(c)] show that, the time needed for the FP-tree construction decreases by the increase in the support level. This situation stems from the following reason: As the size of the *header table* shrinks at higher support levels, the FP-tree construction algorithm uses less insert-tree() calls and this leads to spend less time for the FP-tree construction process.

The Apriori is the fundamental algorithm of association rule mining, thus many algorithms like CHARM have been developed over the Apriori algorithm. Nonetheless, they still cannot compete with the Apriori algorithm in some cases: Apriori demonstrates a better mining performance on *relatively sparse* data sets in comparison to CHARM. By "*relatively sparse*" we mean the data sets or databases containing more enough zero entries(unmarked fields or items), in other words, the ratio *number of fields / number of elements* for the data database is smaller. Since the Apriori algorithm stores and processes only the non-zero entries, it takes the advantage of pruning most of the infrequent items during the

first few passes. This advantage of the Apriori algorithm can be seen from the experiments shown in Figure 5.2(a), Figure 5.2(b), Figure 5.2(c) obviously. While the CHARM algorithm shows a better mining performance on small and dense data sets [Figure 5.2(a)], Apriori's mining performance is superior to CHARM on large and sparse data sets [Figure 5.2(b), Figure 5.2(c)].

However, the increase in the ratio of *number of fields / number of elements* provides considerably faster mining performance to the CHARM algorithm. On such cases [Figure 5.2(a)], the inrease in the probability of enumarating the unique tidsets for itemsets will decrease the probability of the occurence of the 3rd and the 4th itemset-tidset properties for the pairs in *newN*; thus, CHARM will consume less time for mining the complete set of closed frequent itemsets.

5.3 Comparison of the Run Times of the Algorithms

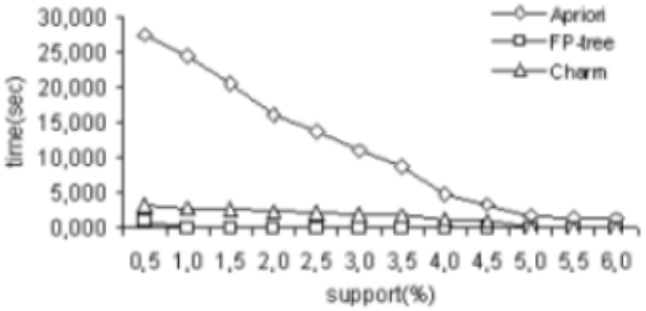
In our study, as we described in Section 4.1.6.3, the run time was calculated simply by adding the *mining time* to the *data preparation time*, and the output routines were discarded during the calculation of run time.

Since the most time consuming computations of the Apriori and the CHARM algorithms are carried out during the mining process, the run time performance graphics of these two algorithms resemble their mining time graphics. However, even the ratio of *(average mininig time+ FP-tree construction time) / average data preparation time* obtained from the experiments regarding the FP-tree algorithm never exceeds 1.

Run Times of the Algorithms

For Northwind Data Set

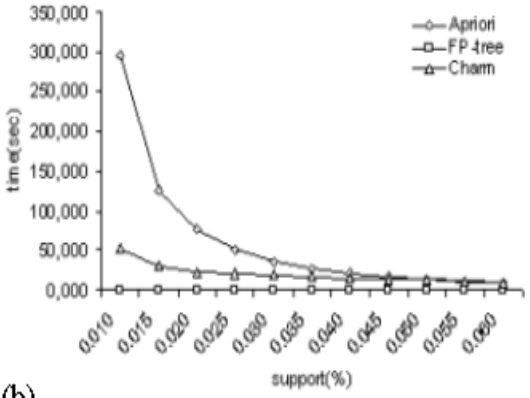
Support(%)	Run Time(sec) Apriori	FP-tree	Charm
0,5	27,479	1,002	3,175
1,0	24,455	0,951	2,873
1,5	20,489	0,962	2,673
2,0	16,084	0,982	2,384
2,5	13,780	0,961	2,183
3,0	11,016	0,911	1,943
3,5	8,833	0,791	1,833
4,0	4,847	0,871	1,282
4,5	3,395	0,721	1,031
5,0	1,803	0,681	0,881
5,5	1,472	0,621	0,751
6,0	1,312	0,660	0,631



(a)

For Synthetic Data Set with 1000 Transactions

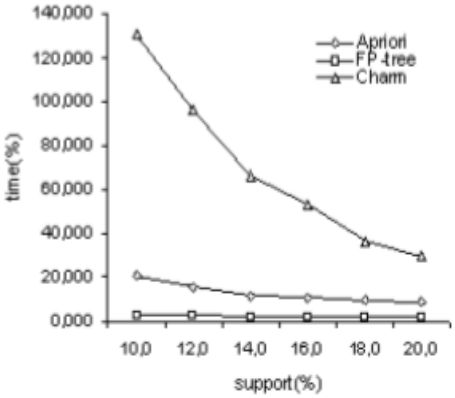
Support(%)	Run Time(sec) Apriori	FP-tree	Charm
0.010	294,907	0,812	51,859
0.015	124,720	0,813	29,187
0.020	75,983	0,797	23,187
0.025	49,625	0,782	19,671
0.030	34,672	0,782	18,376
0.035	25,516	0,797	16,454
0.040	20,453	0,781	14,626
0.045	15,970	0,766	13,219
0.050	13,000	0,766	12,204
0.055	10,141	0,750	11,922
0.060	8,657	0,735	10,219



(b)

For Synthetic Data Set with 5000 Transactions

Support(%)	Run Time(sec) Apriori	FP-tree	Charm
10,0	20,734	2,688	130,453
12,0	15,250	2,407	96,266
14,0	11,703	2,203	66,078
16,0	11,078	2,203	53,156
18,0	9,502	1,891	36,266
20,0	8,719	1,781	29,954



(c)

Figure 5.3: The "Run Times" of the algorithms for different data sets and at different supports are given in (a), (b) and (c).

According to the experimental results [Figure 5.3(a), Figure 5.3(b), Figure 5.3(c)], the Apriori (which can be referred as an obselete algorithm) still shows a better performance on sparse data sets and at high support levels than the CHARM algorithm. That is, when the measurement for the data set having 5000 transactions given in Figure 5.3(c) is observed, it is seen that while the run time of CHARM responds with an exponential increase to the support level decreases, the run time of the Apriori algorithm almost goes linear. However, the idea behind enumarating the only the closed frequent itemsets instead of coping with all frequent itemsets and the depth-first mining strategy of the CHARM algorithm presents us many agvanteges for mining databases of small transactions [Figure 5.3(a), Figure 5.3(b)] in comparison to Apriori.

When the results of the overall processing performances, “run times”, of the algorithms taken in all round, the Apriori and the CHARM algorithms cannot compete with the FP-tree algorithm which uses a divide-and-conquer search tecnique for mining the frequent patterns on compact FP-trees.

5.4 Comparison of the Knowledge Extraction Capabilities of the Algorithms

To compare the knowledge extraction capabilities of the algorithms, we measured the number of associations extracted by the algorithms at different support levels on the 3 three data sets cited.

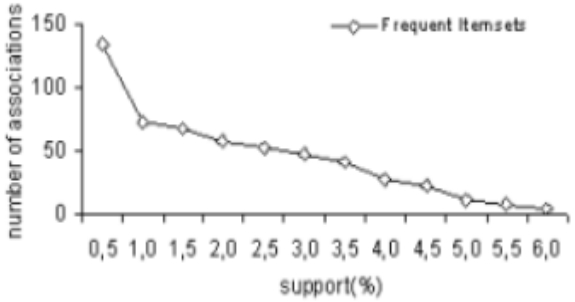
As it is described in Section 4.1.6.4, patterns, itemsets or sequences are all considered as a part of associations. In our study, since we were simply involved in the investigation of the algorithm which was capable of finding much more associations than the other algorithms, we never evaluated what the information, a pattern or an itemset, tells.

Experimental results given in Figure 5.4, Figure 5.5 and Figure 5.6, shows us that the number associations are quite sensible to the support levels. As expected, algorithms extract more information at low support levels. The number of associations as to Apriori and CHARM, presented in Figure 5.5(a) and Figure 5.5 (c) respectively, increase exponentially as the support decreases. But in small data sets this increase can almost be linear [Figure 5.4(a) and Figure 5.4(c)].

Associations for Northwind Data Set

Associations Extracted by the Apriori Algorithm

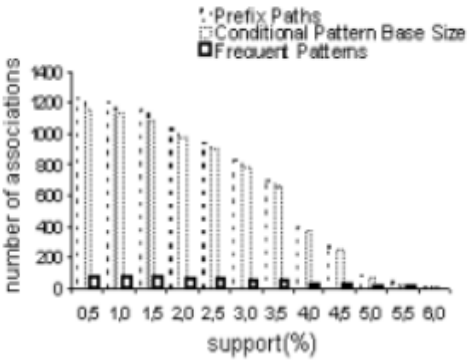
Support(%)	Number of Associations Frequent itemset
0,5	135
1,0	74
1,5	68
2,0	59
2,5	54
3,0	48
3,5	42
4,0	29
4,5	23
5,0	12
5,5	8
6,0	5



(a)

Associations Extracted by the FP-tree Algorithm

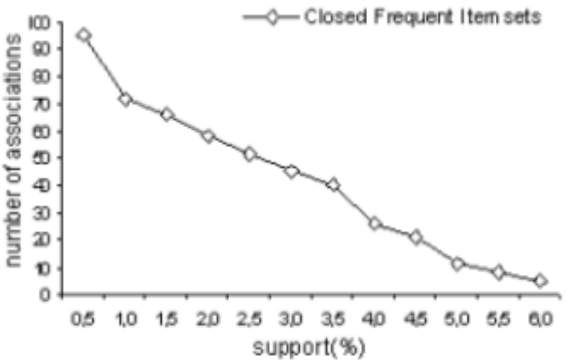
Support(%)	Number of Associations Prefix path	Conditional pattern base	Frequent pattern
0,5	1222	1155	77
1,0	1199	1132	73
1,5	1152	1088	68
2,0	1031	975	59
2,5	950	897	54
3,0	835	787	48
3,5	699	657	42
4,0	400	371	29
4,5	277	254	23
5,0	86	74	12
5,5	36	28	8
6,0	15	10	5



(b)

Associations Extracted by the CHARM Algorithm

Support(%)	Number of Associatons Closed frequent itemset
0,5	95
1,0	72
1,5	66
2,0	58
2,5	52
3,0	45
3,5	40
4,0	26
4,5	21
5,0	11
5,5	8
6,0	5



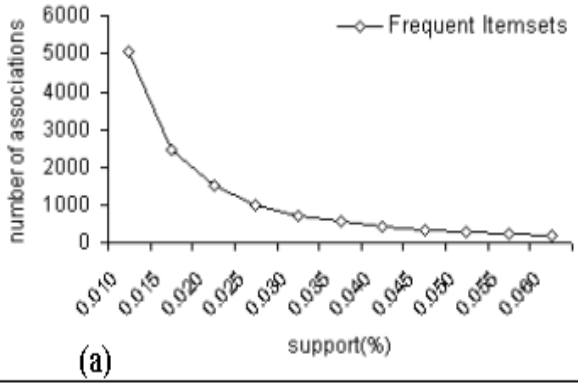
(c)

Figure 5.4: The number of associations for Northwind Data Set generated by the algorithms at different support levels are given in (a), (b) and (c).

Associations for Synthetic Data Set with 1000 Transactions

Associations Extracted by the Apriori Algorithm

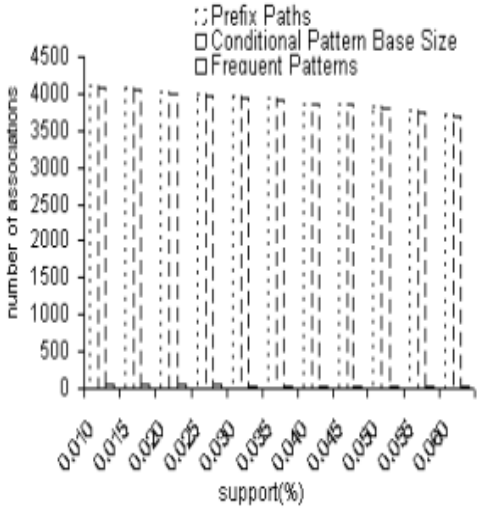
	Number of Associations
Support(%)	Frequent itemset
0.010	5079
0.015	2489
0.020	1484
0.025	1000
0.030	724
0.035	550
0.040	440
0.045	349
0.050	283
0.055	224
0.060	192



(a)

Associations Extracted by the FP-tree Algorithm

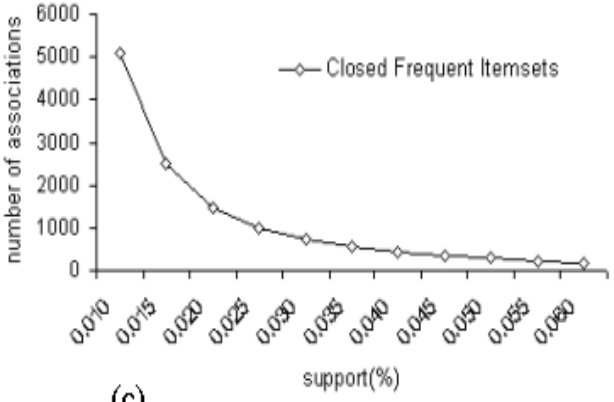
	Number of Associations		
Support(%)	Prefix path	Conditional pattern base	Frequent pattern
0.010	4111	4086	49
0.015	4090	4065	47
0.020	4022	3997	43
0.025	3998	3974	42
0.030	3973	3949	41
0.035	3943	3919	40
0.040	3868	3845	38
0.045	3868	3845	38
0.050	3822	3800	37
0.055	3770	3748	36
0.060	3714	3692	35



(b)

Associations Extracted by the CHARM Algorithm

	Number of Associations
Support(%)	Closed frequent itemset
0.010	5073
0.015	2489
0.020	1484
0.025	1000
0.030	724
0.035	550
0.040	440
0.045	349
0.050	283
0.055	224
0.060	192



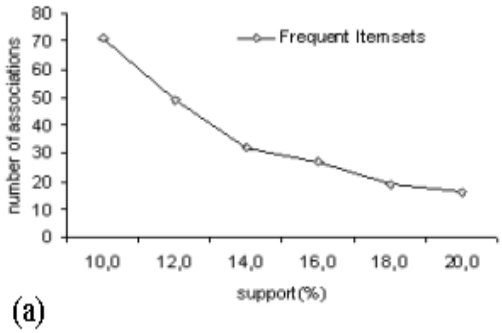
(c)

Figure 5.5: The number of associations for Synthetic Data Set with 1000 Transactions generated by the algorithms at different support levels are given in (a), (b) and (c).

Associations for Synthetic Data Set with 5000 Transactions

Associations Extracted by the Apriori Algorithm

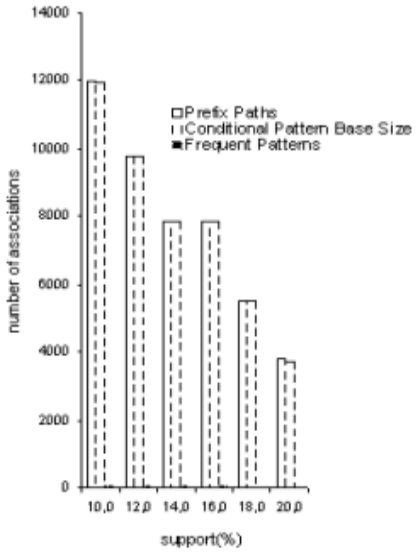
Support(%)	Number of Associations Frequent itemset
10,0	71
12,0	49
14,0	32
16,0	27
18,0	19
20,0	16



(a)

Associations Extracted by the FP-tree Algorithm

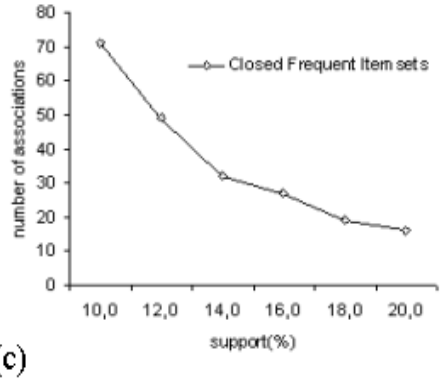
Support(%)	Number of Associations Prefix path	Conditional pattern base	Frequent pattern
10,0	11984	11958	26
12,0	9784	9762	22
14,0	7858	7839	19
16,0	7858	7839	19
18,0	5516	5500	16
20,0	3800	3720	15



(b)

Associations Extracted by the CHARM Algorithm

Support(%)	Number of Associations Closed frequent itemset
10,0	71
12,0	49
14,0	32
16,0	27
18,0	19
20,0	16



(c)

Figure 5.6: The number of associations for Synthetic Data Set with 5000 Transactions generated by the algorithms at different support levels are given in (a), (b) and (c).

According to the results shown in Figure 5.5(a) and Figure 5.5(c), at low support levels (like 0,010%), the number of associations (closed itemsets) extracted by CHARM is less than the number of associations (itemsets) found by

the Apriori algorithm. This situation can be seen also from the results of the experiments for the Northwind data. As is seen from the Figure 5.4(a) and Figure 5.4(c), although the Apriori and the CHARM algorithms extract the same number of associatons at high support levels (such as 6,0%), when we decrease the support from 6,0% to 0.5% Apriori becomes more consistent succeedingly. Since we managed the tests at very high support levels on the Synthetic data set with 5000 transactions [Figure 5.6(a), Figure 5.6(b) and Figure 5.6(c)], the number of associations is considerably low when compared with the test results as to Synthetic data set with 1000 transactions [Figure 5.5(a), Figure 5.5(b) and Figure 5.5(c)].

As is described in Section 3.4, since the term *pattern* refers to both an itemset and its occurrence(such as <{a,b,d,e,f}, 3>), patterns include crucial information in terms of association rule mining. But unfortunately, when we observe the results of the experiments shown in Figure 5.4(b), Figure 5.5(b) and Figure 5.6(b), we see that the number of patterns generated by the FP-tree algorithm are remarkably at low levels when compared with the itemsets extracted by the Apriori and the CHARM algorithms. The experimental results show us that even at low support levels the number of patterns are still in unconsiderable amounts in comparison to Apriori and CHARM. However, since the frequent patterns are generated by composing the set of prefix paths and conditional pattern bases respectively, one may claim that also these two sets serve us information to some extend. But as we emphasized, such kind of semantical claims are not under the scope of this study.

Chapter 6

CONCLUSION

6.1 Introduction

The principal purpose of this study is the comparison of association rule algorithms. The typical application of association rules in practice is the market basket analysis, where the items represent products and the records represent the point of sales data at large departmental stores. These kinds of databases are generally sparse, i.e. the longest frequent itemsets are relatively short. However, there are many real-life data sets that are very dense, i.e they contain very long frequent itemsets of 30-40 items. Over these data sets it is almost impossible to overcome the exponential (NP-Complete) computation task through the use of unsophisticated algorithms. Although sophisticated algorithms developed in order to break the so-called bottle-neck of association rule mining task, it is not easy for the administrative stuff to choose the appropriate algorithm that is able to meet his needs. For this purpose, many algorithms (such as Apriori, Apriori-some, Apriori-all, AClose, CMaxMiner, MAFIA, MaxMiner, CHARM, Pincer-Search, FP-tree, All-MFS, etc.) have been proposed, evaluated and compared in the literature.

Although there have been several studies conducted to compare different algorithms for association rules [Agrawal, Imielinski and Swami, 1993(December)] [Agrawal, Imielinski, and Swami, 1993(May)] [Sarawagi, Thomas and Agrawal, 1998] [Savasere, Omiecinski and Navathe, 1995] [Agrawal, Mannila, Srikant, Toivonen, and Verkamo, 1996] [Brin, Motwani, Ullman, and Tsur, 1997] [Gunopulos, Mannila, and Saluja, 1997] [Zaki and Hsiao, 1999] [Han, Pei and Yin, 1999], we have not found any study for comparison of the three most commonly used algorithms using different support levels, different database sizes and structures. The algorithms evaluated in this study are:

- Apriori Algorithm
- Frequent Pattern Tree(FP-tree) Algorithm
- CHARM Algorithm

The above algorithms were chosen for the implementation and comparison due to the fact that although they serve to the same purpose they use different techniques for mining. The source codes were written in JAVA. JAVA was choosen to be environment since it provides many facilities both to implementation of the data structures by its Collections Framework package (java.util) and access the databases by Java Database Connectivity package (java.sql). The source codes of the algorithms were processed on an Intel Pentium machine with 600 MHz processor and 128 megabytes main memory on three different data sets containing 2155, 7134, 35,861 elements and 77, 50, 50 fields (items) respectively.

6.2 Evaluation of the Algorithms

According to our observations, the performances of the algorithms are strongly depend on the support levels and the features of the data sets (the nature and the size of the data sets).

Apriori uses the *downward closure property* to prune the search space. In this sense, a pass over the database is made at each level to find the frequent itemsets among the candidates. In high levels of support or in sparse data sets such as market basket data, Apriori is an efficient algorithm which is able to give all associations for each length of itemset.

In comparison to the Apriori and the CHARM algorithms, the FP-tree algorithm finds less associations. However, it is a fast, compact and a scalable (robust) algorithm that solves the candidate generation problem. Therefore, it is feasible to use the FP-tree algorithm for mining dense data sets on low levels of support(when fewer associations satisfy us).

The CHARM algorithm provides orders of magnitude improvement over existing methods of association rules when mining times and run times are considered. It is based on the *closure properties* that help us for mining the closed frequent itemsets. Closed frequent itemsets are the unique subsets of frequent itemsets that contains the substantial portion of all frequent itemsets. Therefore, the CHARM algorithm is completely an appropriate solution for mining both the dense data sets on low levels of support(when some of the frequent itemsets may be ignored).

6.3 Future Work

Mining Larger Data Sets: The results of the experiments were adequate for us to arrive at a conclusion with respect to the relative performances of the algorithms. However, investigating the behaviours of the algorithms on larger data sets of 1,000,000 elements can be interesting.

Storage of Larger Data Sets: As is mentioned above, the task of storing such huge data sets on the memory is a serious problem. Accordingly, even if the hardware permits to store the whole data set in the memory, storing the length-2 candidates of complexity $2^{\text{number of frequent items}}$ for the Apriori algorithm will be impossible, no matter what the capacity of the hardware will be. To overcome the overflow problem, the elements of the database can be considered as bits including 0 and 1, so that the memory allocation for the data set can be reduced from *average transaction length * number of transactions* (bytes) to *number of transactions * number of fields* (bits).

Correlations among the Elements of Synthetic Data Set: The two synthetic data sets, used in this study, only differ from each other by the number of transactions (also by the number of elements) they contain. The comparison of the algorithms can be performed on the data sets that have correlations among the items and among the size of transactions. Through the help of such an investigation, the tendency of the knowledge extraction capabilities and the sensitivity of the performances of algorithms to data sets of various distributions may be concluded.

REFERENCES

[Agrawal and Srikant, 1994] R. Agrawal and R. Srikant. Fast algorithms for Mining Association Rules. In VLDB'94, pp. 487-499, 1994.

[Agrawal and Srikant, 1995] R. Agrawal and R. Srikant. Mining Sequential Patterns. In ICDE'95, 3-14, 1995.

[Agrawal, Imielinski and Swami, 1993(December)] R. Agrawal, T. Imielinski and A. Swami. Database mining: A Performance Perspective. IEEE Transactions on Knowledge and Data Engineering, 5(6), 914-925, Special Issue on Learning and Discovery in Knowledge-Based Databases, December 1993.

[Agrawal, Imielinski, and Swami, 1993(May)] R. Agrawal, T. Imielinski and A. Swami. Mining Association Rules between Sets of Items in Large Databases. In Proc. of the ACM SIGMOD Conference on Management of Data, 207-216, Washington, D.C., May 1993.

[Agrawal, Mannila, Srikant, Toivonen, and Verkamo, 1996] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen and A. Inkeri Verkamo. Fast Discovery of Association Rules. In U. Fayyad and et al, editors, Advances in Knowledge Discovery and Data Mining, 307?328. AAAI Press, Menlo Park, CA, 1996.

[Anwar, Beck and Navathe, 1992] T. M. Anwar, H. W. Beck and S. B. Navathe. Knowledge Mining by Imprecise Querying: A Classification-Based Approach. In IEEE 8th Int. Conf. on Data Engineering, Phoenix, Arizona, February 1992.

[Anwar, Navathe and Beck, 1992] T. M. Anwar, S. B. Navathe and H. W. Beck. Knowledge Mining in Databases: A United Approach through Conceptual Clustering. Technical report, Georgia Institute of Technology, May 1992.

[Babcock, 1994] C. Babcock, Parallel Processing Mines Retal Data. Computer World, 6, September 26, 1994.

[Bayardo, 1998] R. J. Bayardo. Efficiently Mining Long Patterns from Databases. In SIGMOD'98, 85-93, 1998.

[Berry and Linoff, 1997] M. Berry and G. Linoff, Data Mining Techniques, John Wiley, 1997.

[Breiman, Friedman, Olshen and Stone, 1984] L. Breiman, J. H. Friedman, R. A. Olshen and C. J. Stone. Classification and Regression Trees, Wadsworth, 1984.

[Brin, Motwani, Ullman, and Tsur, 1997] S. Brin, R. Motwani, J. Ullman and S. Tsur. Dynamic Itemset Counting and Implication Rules for Market Basket Data. In ACM SIGMOD Conf. Management of Data, May 1997.

[Catlett, 1991] J. Catlett. Mega-Induction: A Test Sight. In 8th Int. Conf. on Machine Learning, June 1991.

[Cheeseman et al, 1988] P. Cheeseman et al. Auto Class: A Bayesian Classification System. In 5th Int. Conf. on Machine Learning. Morgan Kaufman, June 1988.

[Chen, Park and Yu, 1998] M. -S. Chen, J. S. Park and P.S. Yu. Efficient Data Mining for Path Traversal Patterns. Knowledge and Data Engineering, 10(2):209,221, 1998. [Cleveland, 1994] W. S. Cleveland, The Elements of Graphing Data, revised, Hobart Press, 1994.

[Codd, 1993] E. F. Codd, Providing OLAP (On-Line Analytical Processing) to User Analysts: An IT Mandate. E.F. Code and Associates, 1993.

[Cooper and Herskovits, 1992] G. Cooper and E. Herskovits. A Bayesian Method for the Induction of Probabilistic Networks from Data. Machine Learning, 1992.

[Daveyand and Priestley, 1990] B.A.Daveyand and H.A.Priestley. Introduction to Lattices and Order. Cambridge University Press, 1990.

[David Shepard Associates, 1990] David Shepard Associates. The New Direct Marketing. Business One Irwin, Illinois, 1990.

[Dhar and Stein, 1997] V. Dhar and R. Stein, Seven Methods for Transforming Corporate Data into Business Intelligence, Prentice Hall 1997.

[Fayyad, Piatetsky-Shapiro, Smyth and Uthurusamy, 1996] U. Fayyad, G. Piatetsky-Shapiro, Smyth and Uthurusamy, Advances in Knowledge Discovery and Data Mining, MIT Press, 1996.

[Fayyad, Weir and Djorgovski, 1993] U. Fayyad, N. Weir and S.G. Djorgovski. Skicat: A Machine Learning System for Automated Cataloging of Large Scale Sky Surveys. In 10th Int. Conf. On Machine Learning, June 1993.

[Fisher, 1987] D. H. Fisher. Knowledge Acquisition via Incremental Conceptual Clustering. Machine Learning, 2(2), 1987.

[Ganter and Wille, 1999] B. Ganter and R. Wille. Formal Concept Analysis: Mathematical Foundations. Springer Verlag, 1999.

[Grahne, Lakshmanan and Wang, 2000] G. Grahne, L. Lakshmanan and X. Wang. Efficient mining of Constrained Correlated Sets. In ICDE'00, 2000.

[Gunopulos, Mannila, and Saluja, 1997] D. Gunopulos, H. Mannila and S. Saluja. Discovering All the Most Specific Sentences by Randomized Algorithms. In Int. Conf. on Database Theory, January 1997.

[Han, Cai and Cercone, 1992] J. Han, Y. Cai and N. Cercone. Knowledge Discovery in Databases: An Attribute Oriented Approach. In Proc. of the VLDB Conference, 547-559, Vancouver, British Columbia, Canada, 1992.

[Han, Pei and Yin, 1999] J. Han, J. Pei and Y. Yin. Mining Frequent Patterns without Candidate Generation. In CS Tech. Rep. 99-10, Simon Fraser University, July 1999.

[Harrison, 1993] D. Harrison, Backing Up. Network Computing, October 15, 98-104, 1993.

[Holsheimer and Siebes, 1994] M. Holsheimer and A. Siebes. Data Mining: The Search for Knowledge in Databases. Technical Report CS-R9406, CWI, Netherlands, 1994.

[Houtsma and Swami, 1993] M. Houtsma and A. Swami. Set Oriented Mining of Association Rules. Research Report RJ 9567, IBM Almaden Research Center, San Jose, California, October 1993.

[Kennedy, Reed and Roy, 1998] R. Kennedy, L. Reed and Van Roy, Solving Pattern Recognition Problems, Prentice-Hall, 1998.

[Klemettinen, Mannila, Ronkainen, Toivonen and Verkamo, 1994] M. Klemettinen, H. Mannila, P. Ronkainen, H. Toivonen, and A.I. Verkamo. Finding Interesting Rules from Large Sets of Discovered Association Rules. In CIKM'94, 401-408, 1994.

[Langley, Simon, Bradshaw and Zytkow, 1987] P. Langley, H. Simon, G. Bradshaw and J. Zytkow. Scientific Discovery: Computational Explorations of the Creative Process. MIT Press, 1987.

[Lehman, 1990] E. L. Lehman, Model Specification: The Views Fisher and Neyman and Later Developments. Statistical Science, 5(2), 160-168, 1990.

[Lent, Swami and Widom, 1997] B. Lent, A. Swami and J. Widom. Clustering Association Rules. In ICDE'97, 220-231, 1997.

[Lubinsky, 1989] D. J. Lubinsky. Discovery from Databases: A Review of AI and Statistical Techniques. In IJCAI-89 Workshop on Knowledge Discovery in Databases, 204-218, Detroit, August 1989.

[Michalski, Kerschberg, Kaufman and Ribeiro, 1992] R.S. Michalski, L. Kerschberg, K.A. Kaufman and J.S. Ribeiro. Mining for Knowledge in Databases: The INLEN Architecture, Initial implementation, and First Results. Journal of Intelligent Information Systems, 85-113, 1992.

[Muggleton and Feng, 1992] S. Muggleton and C. Feng. Efficient Induction of Logic Programs. In Steve Muggleton, Editor, Inductive Logic Programming. Academic Press, 1992.

[Ng, Lakshmanan, Han and Pang, 1998] R. Ng, L. V. S. Lakshmanan, J. Han and A. Pang. Exploratory Mining and Pruning Optimizations of Constrained Associations Rules. In SIGMOD'98, 13-24, 1998.

[Park, Chen and Yu, 1995] J.S. Park, M.S. Chen, and P.S. Yu. An effective Hash-Based Algorithm for Mining Association Rules. In SIGMOD'95, 175-186, 1995.

[Pearl, 1992] J. Pearl. Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference, 1992.

[Piatestsky-Shapiro, 1991] G. Piatestsky-Shapiro, Editor. Knowledge Discovery in Databases. AAAI/MIT Press, 1991.

[Pyle, 1999] D. Pyle, Data Preparation for Data Mining, Morgan Kaufmann, 1999.

[Quinlan, 1990] J. R. Quinlan. Learning Logical Defnitions from Examples. Machine Learning, 5(3), 1990.

[Quinlan, 1993] J. R. Quinlan. C4.5: Programs for Machine Learning. Morgan Kaufman, 1993.

[Sarawagi, Thomas and Agrawal, 1998] S. Sarawagi, S. Thomas, and R. Agrawal. Integrating Association Rule Mining with Relational Database Systems: Alternatives and Implications. In SIGMOD'98, 343-354, 1998.

[Savasere, Omiecinski and Navathe, 1995] A. Savasere, E. Omiecinski, and S. Navathe. An Efficient Algorithm for Mining Association Rules in Large Databases. In VLDB'95, 432-443, 1995.

[Schaffer, 1990] C. Schaffer. Domain-Independent Function Finding. PhD thesis, Rutgers University, 1990.

[Srikant, Vu and Agrawal, 1997] R. Srikant, Q. Vu and R. Agrawal. Mining Association Rules with Item Constraints. In KDD'97, 67-73, 1997.

[Stonebraker et al, 1993] M. Stonebraker et al. The DBMS Research at Crossroads. In Proc. of the VLDB Conference, Dublin, August 1993.

[Tsur, 1990] S. Tsur. Data Dredging. IEEE Data Engineering Bulletin, 13(4):58, December 1990.

[Wainer, 1997] H. Wainer, Visual Revelations, Copernicus, 1997.

[Westphal and Blaxton, 1998] C. Westphal and T. Blaxton, Data Mining Solutions, John Wiley, 1998.

[Zaki and Hsiao, 1999] M. J. Zaki and C.-J. Hsiao. CHARM: An Efficient Algorithm for Closed Association Rule Mining. Technical Report 99-10, Computer Science Dept., Rensselaer Polytechnic Institute, October 1999.

[Zaki and Ogihara, 1998] M. J. Zaki and M. Ogihara. Theoretical Foundations of Association Rules. In 3rd ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery, June 1998.