MINING XML DOCUMENTS WITH ASSOCIATION RULE ALGORITHMS

A Thesis Submitted to

the Graduate School of Engineering and Sciences of

Izmir Institute of Technology

in Partial Fulfillment of the Requirements for the Degree of

MASTER OF SCIENCE

in Computer Software

By
Görkem GÜREL

July, 2008

We approve the thesis of **Görkem GÜREL**

Prof. Dr. Halis PÜSKÜLCÜ
Supervisor

Dr. Belgin ERGENÇ
Committee Member

Prof. Dr. Halil ŞENGONCA
Committee Member

17 June 2008

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

Prof. Dr. Hasan BÖKE
Dean of the Graduate School of Engineering and Sciences

ACKNOWLEDGMENTS

First and foremost I would like to thank my advisor, Prof. Dr. Halis Püskülcü, for his guidance, patience, and support. I would also like to thank my parents and my sister Gizem Gürel for their support throughout my education.

ABSTRACT

MINING XML DOCUMENTS WITH ASSOCIATION RULE ALGORITHMS

Following the increasing use of XML technology for data storage and data exchange between applications, the subject of mining XML documents has become more researchable and important topic. In this study, we considered the problem of Mining Association Rules between items in XML document. The principal purpose of this study is applying association rule algorithms directly to the XML documents with using XQuery which is a functional expression language that can be used to query or process XML data. We used three different algorithms; Apriori, AprioriTid and High Efficient AprioriTid. We give comparisons of mining times of these three apriori-like algorithms on XML documents using different support levels, different datasets and different dataset sizes.

ÖZET

EŞLEŞTİRME KURALI ALGORİTMALARI İLE XML DOKÜMANLARI MADENCİLİĞİ

XML teknolojisinin veri saklama ve uygulamalar arası veri aktarımında kullanımının artmasının ardından XML dokümanları madenciliği alanı çok daha önemli ve araştırılabilir bir konu olmuştur. Bu çalışmada XML dokümanındaki nesneler arasındaki eşleştirme kurallarının madenciliği problemini göz önünde bulundurduk. Bu çalışmanın ana amacı; XML dokümanları üzerine eşleştirme kuralı algoritmalarını, XML verilerini sorgulayıp işleyebilen, işlevsel bir tanımlama dili olan XQuery kullanarak doğrudan uygulamaktır. Bu çalışmada üç farklı algoritma kullanılmıştır, bunlar Apriori, AprioriTid ve HEA algoritmalarıdır. Çalışmanın sonunda farklı veri kümeleri, veri küme boyutları ve destek düzeyleri kullanarak bu üç apriori benzeri algoritmanın XML dokümanları üzerindeki madencilik sürelerinin karşılaştırmaları sunulmuştur.

TABLE OF CONTENTS

LIST OF FIGURES

Figure **Page**

LIST OF TABLES

Table **Page**

CHAPTER 1

INTRODUCTION

1.1. Motivation

EXtensible Markup Language (XML) has emerged as the dominant standard for describing data and exchanging data on the web. It's nested; self-describing structure provides a simple yet flexible means for applications to exchange data (Hassan, et al. 2003). Most recently, XML is passing into virtually all areas of internet application programming, producing huge amount of data encoded in XML. The ability to extract knowledge from XML data sources turned into a very important and necessary characteristic with the continuous growth in XML data.

Data mining, appearing during the late 1980s, has improved during the 1990s especially in transforming vast amounts of data into useful knowledge, and is expected to continue to grow rapidly in the future. (Han and Kamber 2001). Nevertheless, compared to the successful performances in mining well-structured data such as relational databases and object-oriented databases, mining in the semi-structured XML world still remains at a preliminary stage and is confronted with more challenges due to the intrinsic characteristics of XML in both structure and semantics (Feng, et al. 2005). XML data have a more complex hierarchical structure than a database record. Different from a database record, elements in XML data have contextual positions, which thus carry the order notion XML data appear to be much bigger than traditional data. According to these needs, the traditional data mining technology have to be regenerated and reformed for extracting knowledge from XML structure. The aim of XML mining is to integrate the emerging XML technology into data mining technology.

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 (Han, et al. 2001). Actually, the results of the investigation presented by "www.kdnuggets.com" shows us that the usage

of “Association Rule Mining Techniques” possesses the 21% of the data mining studies in 2007.

The association rule mining algorithms’ general structure is making multiple passes over the data. Each pass, starts with a seed set for generating new potentially large sets, called candidate sets. Then the supports 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).

The aim of this thesis is mining XML documents with using association rule algorthims such as Apriori, AprioriTid and HEA (High Efficient AprioriTid). Algorithms implemented in XQuery which is a query language for XML documents defined by W3C. Without preprocessing, documents directly mined by algorithms and results turns as an XML file.

1.2. Objective

In this study, we considered the problem of Mining Association Rules between items in XML document. (Agrawal, et al. 1993) had first introduced the problem of mining association rules. Since then, numerous works have been done in various directions. The task of mining association rules over XML data was first introduced by (Braga, et al. 2001). The idea of mining XML documents with XQuery was first published by (Wan, et al. 2003). They used apriori algorithm with XQuery but they didn’t published the test results or XQuery expressions of the functions and noticed that they haven’t applied the algorithm to large item sets. Our objectives for this topic was applying this algorithm for large item sets and apply other association rule algorithms with XQuery and compare the results of these algorithms in XML documents.

In this study, three algorithms for mining associaiton rules have been discussed, implemented in XQuery and their performances are compared using both different data sets and different treshold (known as support) levels with different size synthetic datasets.

1.3. Structure of the Study

As is mentioned in Section 1.1, there are many data mining techniques. In Chapter 2, we introduced the scope and the mission of Data Mining and Knowledge Discovery in Databases (KDD) including the descriptive and the predictive methods.Chapter 3 discusses the XML document structure, points out advantages of XML, introduces XML query languages in detail. In Chapter 4, scope of xml mining , xml mining taxonomy, techniques and related work about xml mining are explained. Chapter 5 expresses association rule mining and algorithms we used (Apriori, Apriori-TID, HEA) briefly and XQuery representations of the algorithm functions with examples. In Chapter 6, the synthetic data sets are visualized, the generation of synthetic data sets and their statistical analysis are described.Chapter 7 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 8 summarizes our main findings and the benefits of the Apriori, AprioriTID and HEA algorithms with implementing XQuery. It also points out some future research issues on XML mining.

CHAPTER 2

DATA MINING AND KNOWLEDGE DISCOVERY

2.1 Overview

In the last decade we have witnessed a dramatic increase in the amount of data that is publicly available. With the stupendous amount of data stored in files, databases, and other repositories, it is getting important to develop powerful means for analysis and interpretation of such data and for the extraction of interesting knowledge that could help in decision making.

Data Mining is the discovery of hidden information found in large quantities of data and can be viewed as a step in the knowledge discovery process (Fayyad 1996). Data mining defined as a set of computer-assisted techniques designed to automatically mine large volumes of integrated data for new, hidden or unexpected information, or interesting patterns. With small set of data, traditional statistical analysis can be efficiently used. 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 (Edelstein 1998).

Data mining involves the use of sophisticated data analysis tools to discover previously unknown, valid patterns and relationships in large data sets. These tools can include statistical models, mathematical algorithms, and machine learning methods (algorithms that improve their performance automatically through experience, such as neural networks or decision trees). (Cooney 2006)

Data mining processes have required an integration of techniques from multiple disciplines such as, statistics, machine learning, database technology, pattern recognition, neural networks, information retrieval and spatial data analysis. Consequently, data mining consists of more than collecting and managing data, it also includes analysis and prediction.

Data mining differs from other research method in that it is intended to work on data without starting from a particular hypothesis, assumption or even a particular

question. Essentially, it reverses the scientific method, starting from data and moving towards hypotheses instead of following the traditional order (Berry, et al. 2000).

Data mining can be used in different kinds of databases (e.g. relational database, transactional database, object-oriented database and data warehouse) or other kinds of information repositories (e.g. spatial database, time-series database, text or multimedia database, legacy database and the World Wide Web) (Han 2001) Therefore, data to be mined can be numerical data, textual data or even graphics and audio.

While data mining and knowledge discovery in databases (KDD) are frequently treated as synonyms, data mining is actually part of the knowledge discovery process (Zaiane 1999). The following figure (Figure 2.1) shows data mining as a step in knowledge discovery process.

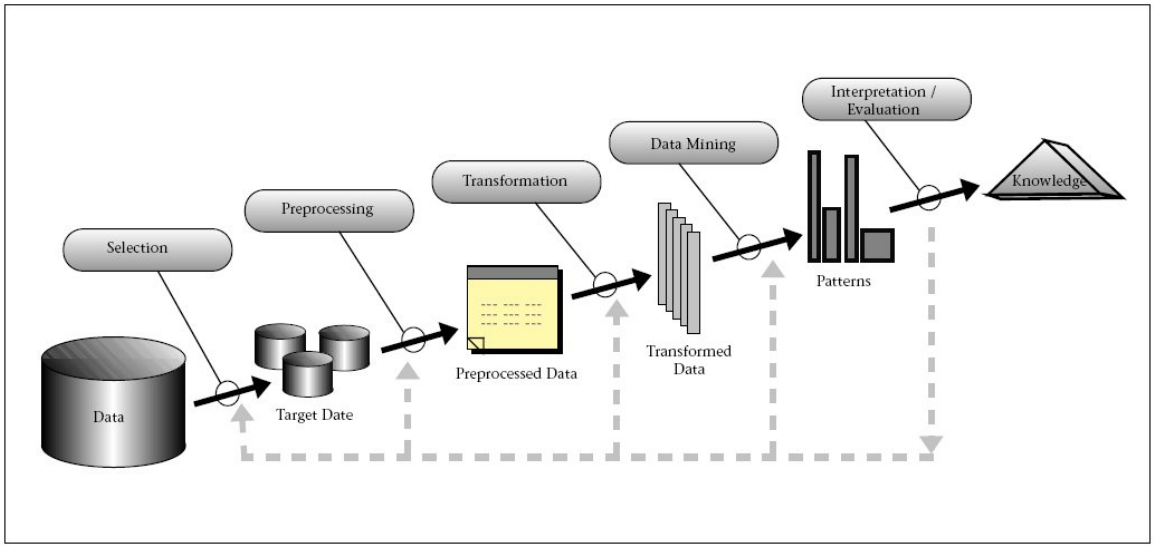

Figure 2.1. An overview of steps that compose KDD process
(Source: Fayyad, et al. 1996)

Data extraction process extracts useful subsets of data for mining. Goals of the extraction process are, identifying concerned information in the database and processing the database into some suitable form to analysis by the data mining algorithms.

Data preparation is one of the most important steps in the data mining process. Large database systems generally contain errors in the stored data. Examine the data for errors, outlines and missing values to the quality of the data. This is the most time consuming and most important step in the data preparation process. Robustness is an important property for the data mining systems. So, some techniques are used to

managing to data in this process. Data cleaning can be applied to remove noise and inconsistency in the data. There are many reasons for noisy and incomplete data. Some of methods are used to filling in the missing values. The most famous one is the regression methods for data cleaning. Data integration merges data from multiple sources. These sources may include multiple databases, data cubes, or flat files. The main problem of the integration is data confliction. Data transformation operations used for normalizations and aggregation. Data are transformed or integrated into form suitable for mining. Data transformation process includes smoothing, generalization, normalization and aggregation techniques. Data reduction operation used for reduce the data size by using one of the data aggregation, dimension reduction or data comparison methods. Data reduction methods can be used to minimizing representation of the data, while reducing the loss of information content.

All of the raw data are created and cleaned in the previous steps. Thus, data are prepared for the data mining stage. Prepared data might contain many attributes and we have to select a subset of the attributes for using in data mining process.

A Data mining algorithm takes data as input and produces output in the form of models or patterns. In this step an intelligent methods are applied in order to extract data patterns. Visualization, classification, clustering, regression or association algorithms are used for different problem. There are many algorithmic approaches to extracting useful information from data.

Data mining system generate lots of patterns, or rules. Only small fractions of the patterns which are generated from the data mining system are interested to any given problem. Patterns have to be easily understood, useful, and interesting. An interesting pattern represents knowledge. After determination of the knowledge function, some measured functions which are used to separate uninteresting patterns from knowledge, are used for the data mining process. For example, association rules use support and confidence measure functions to determine the rules.

An important goal of data mining is to apply what has been learned. In the knowledge presentation step some possible actions formed from the successful application. Creation of a report or technical article, relocation of retail items for purchase or placement of selected items on sale together, funding of a new scientific study motivated by what has been learned from a knowledge discovery process are some possible example.

2.2 Aims of Data Mining

The data mining goals are defined by the intended use of the system. We can distinguish two types of goals:

- Verification and
- Discovery.

With verification, the system is limited to verifying the user's hypothesis. With discovery, the system autonomously finds new patterns. We further subdivide the discovery goal into prediction, where the system finds patterns for predicting the future behavior of some entities, and description, where the system finds patterns for presentation to a user in a human-understandable form (Fayyad 1999).

2.3 Methods of Data Mining

The two high-level primary goals of data mining in practice tend to be prediction and description. 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 (Shapiro, et al. 1996). Clustering and association rules are descriptive methods of data mining. Some of the predictive methods described in this chapter are classification, neural networks, and decision trees, nearest neighbor, regression and genetic algorithms.

2.3.1 Classification

Classification is learning a function that maps (classifies) a data item into one of several predefined classes (Weiss and Kulikowski 1991). Examples of classification methods used as part of knowledge discovery applications include the classifying of trends in financial markets (Apte and Hong 1996) and the automated identification of objects of interest in large image databases (Djorgovski and Weir 1996).

Classification is an operation that enables to discover patterns in large or complex data sets in order to solve specific problems. Classification is classifies a data item into one or several predetermined categories. For example, banks want to evaluate the credit risks of card applicants in order to decide whom to issue their credit card. You first examine the existing customer data for which the group each customer belongs to is already known. Then variables or attributes of each group are identified. You can

classify new customers by examining their attributes. Classification models are widely used to other marketing tasks such as target marketing and direct mailing decision. The popular mining tools to conduct classification tasks include logistic regressions, neural networks, and tree-based models.

A raw method can classify customers by whether their income is above or below a certain amount. A little more subtle approach tries to find a linear relationship between two different factors like income and age to divide a data set into two groups. Real world classification problems usually involve many more dimensions and therefore require a much more complex delimitation between different classes.

2.3.2 Clustering

Another kind of data mining technique is a clustering model, also called a segmentation model. Clustering is the process of grouping the data into class. The goal of the clustering is to take a set of objects which are records in a database or data warehouses and to partition them into number of groups or clusters. (Berkhin 2002).

Similar to classification, clustering segments customers into several groups. However, clustering differs from classification in that there are no predefined groups. That is, you do not know how many groups you have or which groups each customer belongs to. The algorithm will find some number of clusters such that customers within the same cluster are very similar while customers from different clusters are very different.

Clustering techniques have been studied overall in statistics, machine learning and data mining. Clustering can be used on the principle of maximizing the intra-class similarity and minimizing the interclass similarity (Dunham 2003). Cluster analyzes is used in different area. In marketing, characterize customer groups based on purchasing pattern. In Earthquake studies, observed earthquake epicenters should be clustered.

2.3.3 Association

Associations are the tasks to determine which items or events occur together. An interesting example of mining algorithm to perform association task is market basket analysis. It tells us which products tend to be purchased together by customers and/or in particular sequence. For example, it analyzes customer's transaction data in a supermarket and provides some "interesting" patterns in the form of association rule

like: "*If* a customer purchases a coffee, *then* she tends to buy sugar." These kinds of information are useful for retailers to make decisions on shelf-space allocation, bundling products, store layout, a promotional design, and so on (Berry and Linoff 1997). Market basket analysis is especially useful for exploratory data analysis when managers do not know what the "interesting" patterns are. Association rule technique has mainly been applied to the point-of-purchase data so far, but has lots of potentially interesting applications to any industries with large number of customers and products.

Association is a rule, which implies certain association relationships among set of objects such as occur together or one implies the other. (Kumar, et al. 2000) Goal of association rule is finding associations among items from a set of transactions which contain a set of items. A formal statement of the association rule problem is (Agrawal, et al. 1993) (Cheung, et al.1996): Let $I = \{I_1, I_2, \ldots, I_m\}$ be a set of m distinct attributes, also called *literals*. Let D be a database, where each record (tuple) T has a unique identifier, and contains a set of items such that $T \subseteq I$ An association rule is an implication of the form $X \Rightarrow Y$, where $X, Y \subset I$, are sets of items called itemsets, and $X \cap Y = \phi$. Here, X is called antecedent, and Y consequent.

The *support (s*) of an association rule is the ratio (in percent) of the records that contain $X \cup Y$ to the total number of records in the database. Therefore, if we say that the support of a rule is 5% then it means that 5% of the total records contain $X \cup Y$. Support is the statistical significance of an association rule. For a given number of records, *confidence (*α*)* is the ratio (in percent) of the number of records that contain $X \cup Y$ to the number of records that contain X. Thus, if we say that a rule has a confidence of 75%, it means that 75% of the records containing X also contain Y. The aim of the algorithm is to discover all association rules with

Support ≥ minimum_Sup and
Confidence ≥ minimum_Conf.

2.3.4 Neural Networks

Based on the model similar to human brain, artificial neural networks learn and generalize from external inputs. When exposed to training examples, neural networks discover patterns and relationships. Their approach is fundamentally different from the way that traditional digital computer solves problems.

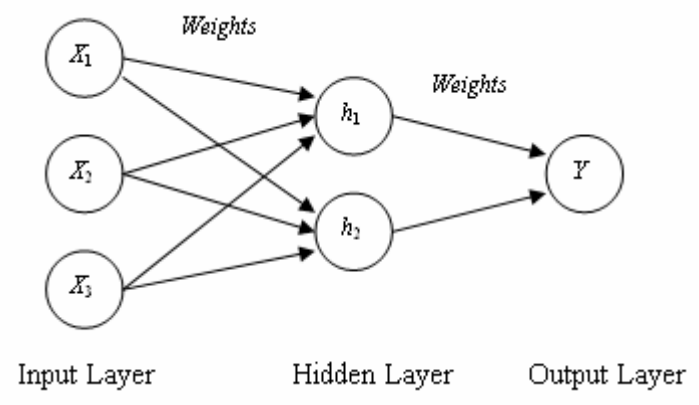

Figure 2.2. Neural Network

A neural network is an information processing unit that transforms a set of inputs into a set of output values. Inputs are interconnected to produce outputs by several weights or parameters. The layer located between the output and the input layer is the hidden layer. The units in hidden layer produce their output values which are functions of the weighted sum of the input values feeding into them. A neural network can theoretically have any number of hidden layers. The wider the hidden layer, the network can capture more sophisticated patterns or relationships.

2.3.5 Decision trees

Decision trees are mainly used for classification task. The tree-based models produce a series of rules as outputs.

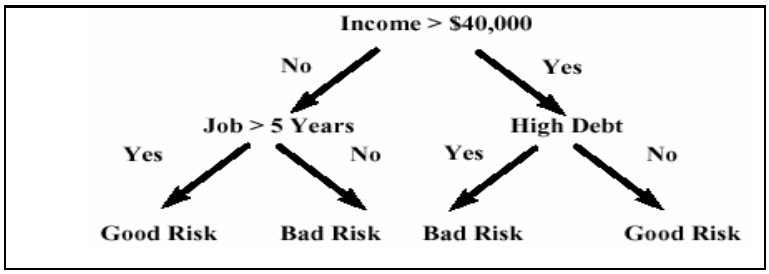

Figure 2.3. A Decision tree

In order to generate a decision tree from the training set of data we need to split the data into progressively smaller subsets. Each iteration considers the data in only one node. The first iteration considers the root node that contains all the data. Subsequent

iterations work on derivative nodes that will contain subsets of the data. 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, et al. 1984).

2.3.6 Nearest Neighbor (NN)

In this technique unlike other predictive algorithms, the training data is not scanned or processed to create the model. Instead, the training data is the model. When a new case or instance is presented to the model, the algorithm looks at all the data to find a subset of cases that are most similar to it and uses them to predict the outcome. There are two principal drivers in the k-NN algorithm: the number of nearest cases to be used (k) and a metric to measure what is meant by nearest. To classify a new case, the algorithm computes the distance from the new case to each case (row) in the training data. The new case is predicted to have the same outcome as the predominant outcome in the k closest cases in the training data (Rinsa-Runtala 2001).

K-NN requires large computational efforts because the calculation time increases as the factorial of the total number of points. It's a rapid process to apply a decision tree or neural network but this technique requires making a new calculation on each new case (Jovanovic, et al. 2002).

2.3.7 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 (Werbos 1976).

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 (Jovanovic, et al. 2002).

2.3.8 Regression

Regression is the oldest statistical data mining technique which used to fit an equation to a dataset. 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, et al. 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. Regression works well with continuous quantitative data such as age, speed etc but not significant categorical data like name, gender or color. Therefore, more complex techniques like decision trees or neural nets may be necessary to forecast future values.

2.4 Data Mining Applications

Currently, data mining is being used in wide variety of business areas for many purposes. Most organizations use data mining in order to manage their customer life cycle such as acquiring new customers, increasing revenue of existing ones and retaining good customers (Edelstein 2001). When a company defines the characteristics of its customers by using historical data, it can predict the future behaviors of existing and candidate ones, so that it can develop required strategies. And also, a lot of organizations, such as telecommunication, credit card and insurance companies, use data mining in order to detect and reduce fraudulent use of their services.

Some organizations use data mining techniques to marketing. For example for this purpose is to analyze customer buying habits. To obtain of such information can help retailers to improve marketing strategies. Banks use data mining to predict credit risks. Insurance companies use data mining to analyze claims patterns for fraud and to predict high risk situations. Other applications include stock market analysis, predicting foreign exchange rates based on current financial indicators, determining commonalities or anomalies among classes of medical patients, modeling proteins, and finding genes in DNA sequences. Another data mining application area is within the operations and communications of the organization itself. Security offices and law enforcement agencies have been applying data mining technologies to their data sets. They have analyzed all sorts of data sets including telephone tool calls, narcotics operations,

financial crime enterprises, criminal organizations, border crossing, terrorist activities and a wide range of other activities. (Westphal, et al. 1998).

CHAPTER 3

XML DOCUMENTS

3.1 Introduction

With the advent of the World Wide Web an abundance of information is stored and disseminated from different application domains. The Hyper Text Markup Language HTML currently used to describe information for web publishing allows the specification of the documents display properties e.g. font, size, color. However, it did not take long for the World Wide Web consortium to realize that a more expressive language is needed to support computation on web documents.

3.2 Emergence of XML

In response to the need for a more powerful language for modeling web information the eXtensible Markup Language XML was proposed by the World Wide Web Consortium in 1997. XML is a subset of Standard Generation Markup Language SGML which is the meta-language of HTML simplified upon the requirements of web applications. XML, Extensible Markup Language, is a widely accepted standard for exchanging information between systems (W3C). While other markup languages, such as HTML, uses only predefined elements, XML is a meta-markup language because it allows defining other markup languages. Converting data to XML document can reduce the complexity of creating data that can be read by different types of applications (Besprozvannykh 2003). Most recently, XML is passing into virtually all areas of internet application programming, producing huge amount of data encoded in XML.

3.3 XML Document Structure

```
<? xml version=”1.0” ?>
<DEPARTMENT>
<PhDCourses>
<Course teacher="fp1" title="Advanced Data Mining">
        <TimeTable>...</TimeTable>
    <Student ref="ps1" />
        <Student ref="ps2" />
</Course>
<Course teacher="fp3" title="Intricacies of XML parsers">
        <TimeTable>...</TimeTable>
<Student ref="ps2" />
<Student ref="ps3" />
</Course>
</PhDCourses>

<People>

<PhDStudent id="ps2" advisor="fp3">
<PersonalInfo email="fp3@cs.atlantis.edu">
<Name>...</Name>
</PersonalInfo>
<Subscription year="2001" />
<Publications> ... </Publications>
</PhDStudent>

<FullProfessor id="fp3">
<PersonalInfo email="fp3@cs.atlantis.edu">
<Name> ... </Name>
</PersonalInfo>
<Publications>
<Article title="Golden Data Mines in Atlantis">
<Author>Wilson</Author>
<Author>Holmes</Author>
<Conference name="VLDB" year="2001" />
</Article>
<Article title="P is just like NP - The Final Proof">
<Author>...</Author>
<Journal year="2000" month="4" volume="4" name="DMKD" publisher=
"Kluwer" />
</Article>
<Book year="2001" title="XML Query Languages">
<Author>...</Author>
<Publisher>...</Publisher>
<Keyword>XML</Keyword>...<Keyword>Xquery</Keyword>
</Book>
</Publications>
<Award year="2001" society="IEEE">This award..</Award>
</FullProfessor>
</People>
</DEPARTMENT>
```

Figure 3.1. Example of an XML Document
(Source: Braga, et al. 2003)

The XML document consists of items, which are either nodes or atomic values. The nodes are of six kinds: elements, attributes, text, comment, processing instruction, and namespace. The XML document is composed of:

- *Elements*: An XML element contains a name, a list of attributes, a list of in-scope namespaces, and a list of children. The children of an element are serialized between the open and close tags of their parent. When an element has no children, the element is said to be empty. Every XML document has exactly one top-element known as the document element. (Skonnard and Gudgin 2001)
  E.g.: <Article title="Golden Data Mines in Atlantis">
- *Attributes* gives the information about the elements. Attributes appear as name/value pairs separated by an equal sign.
  E.g.: title="Golden Data Mines in Atlantis"
- *Comments* are enclosed in <!—comment -- > and are used to provide information to humans about the XML content.
- *Processing:* Instruction allows documents to contain instructions for applications. These instructions contains information about how to process and how to display the XML document.
- *Namespaces* provide a way to distinguish deterministically between XML elements that have same local name. Namespace declarations appear in elements start tag.

3.3.1 Document Type Definition (DTD)

W3C introduced a model to describe the structure of an XML document: the Document Type Definition (DTD).A document Type Declaration defines the constraints on the sequence and nesting of element tag and attributes. Figure 3.2 shows an example of DTD.

```
<! ELEMENT DEPARTMENT (PhDCourses, People)>
<! ELEMENT PhDCourses (Course+)>
<! ELEMENT Course (TimeTable?, Student*)>
<! ATTLIST Course teacher IDREF #REQUIRED title CDATA #REQUIRED>
<! ELEMENT TimeTable (#PCDATA)>
<! ELEMENT Student EMPTY>
<! ATTLIST Student ref IDREF #REQUIRED>
<! ELEMENT People (PhDStudent*, FullProfessor+)>
<! ELEMENT PhDStudent (PersonalInfo, Subscription, Publications, Award*)>
<! ATTLIST PhDStudent id ID #REQUIRED advisor IDREF #REQUIRED>
<! ELEMENT PersonalInfo (Name)>
<! ATTLIST PersonalInfo email CDATA #REQUIRED>
<! ELEMENT Name (#PCDATA)>
<! ELEMENT Subscription EMPTY>
<! ATTLIST Subscription year CDATA #REQUIRED>
<! ELEMENT FullProfessor (PersonalInfo, Publications, Award*)>
<! ATTLIST FullProfessor id ID #REQUIRED>
<! ELEMENT Publications (Article*, Book*)>
<! ELEMENT Article (Author+, (Conference|Journal), Keyword*)>
<! ATTLIST Article title CDATA #REQUIRED>
<! ELEMENT Conference EMPTY>
<! ATTLIST Conference name CDATA #REQUIRED year CDATA #REQUIRED>
<! ELEMENT Journal EMPTY>
<! ATTLIST Journal year CDATA #REQUIRED month CDATA #IMPLIED volume
CDATA #IMPLIED name CDATA #REQUIRED publisher CDATA #REQUIRED>
<! ELEMENT Book (Author+, Publisher, Keyword*)>
<! ATTLIST Book title CDATA #REQUIRED year CDATA #REQUIRED>
<! ELEMENT Author (#PCDATA)>
<! ELEMENT Publisher (#PCDATA)>
<! ELEMENT Keyword (#PCDATA)>
<! ELEMENT Award (#PCDATA)>
<! ATTLIST Award year CDATA #REQUIRED society CDATA #REQUIRED>
```

Figure 3.2. The DTD of the document showed in Figure 3.1.
(Source: Braga, et al. 2003)

There are four kinds of declarations in DTD; elements, attribute lists, entities and notations.

3.3.1.1 Element Declarations

To define an element the definition should begin with "<! ELEMENT" followed by the name of the element, and then the content of element. The definition ends with ">".

3.3.1.2 Attribute Declarations

In the DTD, XML element attributes are declared with an ATTLIST declaration. There are ten types that can be assigned to attributes:

- CDATA - text
- enumerated - an exact list of options
- ID - a unique name for the element
- IDREF - the value of an ID type attribute
- IDREFS - multiple IDs, separated by whitespace
- ENTITY - the name of an entity declared in the DTD
- ENTITIES - multiple entities, separated by whitespace
- NMTOKEN - an XML name
- NMTOKENS - multiple XML names, separated by whitespace
- NOTATION - the name of a notation declared in the DTD

3.3.1.3 Entity Declarations

Entity Declarations can be either internal, external or parameter entities. An internal entity contains a block of text. External entities are references to files that may contain textual or binary data. Parameter entities allow reusable text throughout the DTD code.

3.3.1.4 Notation Declarations

Notation Declarations can be used to identify specific types of external binary data. Notations allow including that data in the documents by describing the format it and allowing applications to recognize and handle it. The format for a notation is: <! NOTATION name system "external_ID">

3.3.2 XML Schema

The purpose of the XML schema language is to provide an inventory of XML markup constructs with which to write schemas. The purpose of a schema is to define and describe a class of XML documents by constraining and documenting the meaning usage and relationships of their document parts the schema specifies data types,

elements and their content attributes and their values entities and their contents and notations. Schema constructs may also provide for the specification of implicit information such as default values Schemas document their own meaning usage and function. The XML schema language can be used to define, describe and catalogue XML vocabularies for classes of XML documents (World Wide Web Consortium 1999).

XML Schema used to define the allowable structure of elements for a given application or application domain in a valid XML document uses XML document syntax. Declarations in XML Schema can have richer and more complex internal structures than declarations in DTDs. Schema designers can take advantage of XML's containment hierarchies to add extra information where appropriate (Laurent 1999).

```
<product effDate="2001-04-02"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="chapter01.xsd">
 <number>557</number>
 <size>10</size>
</product>
```

Figure 3.3. A part of an XML Document

Figure 3.3 shows an example XML document. This documents XML Schema shown in Figure 3.4.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="product" type="ProductType"/>
 <xs:complexType name="ProductType">
  <xs:sequence>
   <xs:element name="number" type="xs:integer"/>
   <xs:element name="size" type="SizeType"/>
  </xs:sequence>
  <xs:attribute name="effDate" type="xs:date"/>
 </xs:complexType>
 <xs:simpleType name="SizeType">
  <xs:restriction base="xs:integer">
   <xs:minInclusive value="2"/>
   <xs:maxInclusive value="18"/>
  </xs:restriction>
 </xs:simpleType>
</xs:schema>
```

Figure 3.4. An XML Schema

XML Schemas provide an improved data typing system which has data-oriented data types in addition to the more document-oriented data types compared to DTDs, making XML more suitable for data interchange applications. Built-in data types include strings, booleans, and time values. The XML Schema provides a mechanism for generating additional data types. Besides, XML Schema supports namespaces and the notion of keys to uniquely identify elements in an XML document (Mulchandani 2003).

3.4 Advantages of XML

XML is a simple and flexible text-based language that markup tags like <name> and <author> can be specified what kind of information is contained in a tag. Tags and contents are concatenated together; information can be understood without the application that created them. XML documents are not only easy to read for humans but also can be easily processed by machines.

XML is very different from HTML. HTML contains both data and presentation information in a single document, but XML separates presentation from data. Extensible Stylesheet Language provides a mechanism to transform one XML document structure into another new XML document, or format XML document into desired HTML display or devices.

The structure of XML represents the semantics of the data, which provides a great opportunity for information retrieval. XML documents can be categorized by their schemas; results can be more exactly by limiting the searching to specific schemas (Egnor and Lord 2000). If we want to search a keyword “Aydın”, without any information, the search engines cannot identify if it is a search for a person's name, or a state name. If “Aydın" is specified like \<city>Aydın</city>", then search results will be much better. With the structure information, a specific relevant part of an XML document can also be retrieved.

XML is an extensible meta-language. With XML, we can define whatever information representation we need. One of the goals of the XML standard is its extensibility. XML is preferred as a standardization mechanism to define platform-independent grammar for specific purposes and schemas for all kinds of data models. DTD can be used to define the syntax and structure of XML documents. XML Schema is a much more comprehensive schema language for XML, expressed with the syntax of XML. XML documents can be composed from relational tables (Wang, 2004).

XML can represent most languages in the world because it is based on Unicode. XML tools can be written once and used for most languages. This makes the data and document exchange easy in a multilingual world.

XML standard is supported by most major software vendors. They heavily invest on XML, and actively contribute to the standardization of XML (Wang, 2004). With XML as the base standard, there are many XML related application standards, such as GML and ebXML. Already, an impressive number of commercial document tool vendors have announced planned product versions supporting XML-based documents (Swen 1997). This support coincides with the growing number of available commercial XML parsers that provide a consistent interface for manipulating XML documents (Goldfarb 1998).

3.5 XML Query Languages

Ease of use and performance are the advantages of the XML query languages to programming languages. Most general-purpose programming languages treat XML as any other API, instead of as a first-class part of the language. Instead of providing operators for constructing and navigating XML directly, you have to access it through an API layer. Just as text manipulation is easier in Perl than in, say, Fortran, so a single line of an XML query language like XSLT or XQuery can accomplish the equivalent of hundreds of lines of C, C#, Java, or some other general-purpose language (Brundage 2004).

3.5.1 XSLT

The Extensible Stylesheet Language for transformation is an official recommendation of the World Wide Web Consortium which published in 1999. It provides a flexible, powerful language for transforming XML files and uses XML syntax to define transformation rules that are applied to an input XML document to result in a text document that has not to be an XML document. This result can be an HTML document, another XML file, PDF, SVG, java code or a text file.

An XSLT transformation is provided as an XML document called a stylesheet, or XSLT stylesheet. A stylesheet is applied to an input XML document, which means that the input XML document is transformed according to the stylesheet into an output

document. A stylesheet can be looked at as a set of rewriting rules. These rules are called as a template rule. Each rewriting rule is equipped with a pattern and a body expression. When a stylesheet is applied to an input XML document, a rewriting rule is found, whose pattern matches the root document node of the input XML document. This rule's body expression defines a transformation for the root node, so the rule's body expression is evaluated to compute the output.

Subset of XPath can be used to define patterns in template rules that are matched against nodes in an input XML document and full XPath 2.0 expressions can be used inside template's bodies (Hlousek 2005).

3.5.2 XPath

XPath is a language for addressing parts of an XML document which is a standard recommended by W3C. XPath defines a library of standard functions but is not itself written in XML because it defines how to locate parts of an XML document, forms the basis for a query language on XML, such as XSLT or Xquery. XPath models an XML document as a tree of nodes of which there are different types, including element nodes, attribute nodes and text nodes (World Wide Web Consortium 1999).

XPath 1.0 has been designed to easily identify or match nodes in an XML document with the intention to be used either standalone or in XSLT 1.0 and XPointer. The result of evaluating an XPath 1.0 expression is either an atomic value or a set of nodes from the source XML document usually referred to as nodeset. As the most complicated structure of a result is a set, there is no ordering information about the items in the result set. The absence of order information in a result has been understood as a particular disadvantage. Specifically, information about document order of nodes is lost in a result.

The main purpose of the XPath 2.0 is to address nodes in an XML document, which is the same as for the 1.0 version. The main difference between XPath 1.0 and XPath 2.0 is that an XPath 2.0 expression returns a sequence of items instead of a nodeset. Thus, the items in the returned sequence have now their order defined.

3.5.3 XQuery

XQuery is currently still under development by the W3C (XQuery 1.0), and is also known as W3C XML Query. The purpose of XQuery is extracting data from entire XML documents, collections of XML documents, or only document fragments.

XQuery is derived from an XML query language called Quilt, which in turn borrowed features from several other languages, including XPath 1.0, XQL, XML-QL, SQL, and OQL. XQuery 1.0 is the superset of XPath 2.0 both in syntax and semantics.

XQuery is a functional expression language that can be used to query or process XML data or any data that can be represented within the same model as XML. Being purely an expression language, XQuery programs are easier to understand and maintain than XSLT, because they do not include the complexities or management of templates (rule-based system) (Funderburg, et al. 2002). This is especially true for highly structured data, and for longer programs. XQuery will still be able to effectively process semi-structured data. The query language is small and powerful. Moreover, XQuery is a full-fledged programming language. It provides if/then statements, loops, variables, quantified expressions and a set with the most important functions (Hollenstein 2005). Applications are made simpler by performing a single XQuery request over these views and receiving satisfactory results in one step also it has both an easy, human readable form and an XML representation (IBM Journal 2002).

There are several operators provided to filter the documents for conditions which match either content or structure (Hollenstein 2005). The core of queries is a FLWOR expression. FLWOR is coming from ’for, let, where, order by, return’. With for and let XML fragments can be bound to a variable which then can be further processed in the where-clause and finally outputted in the return-clause. An example query can be seen in Figure 3.5.

```
for $book in fn:docs("Books.xml")//Book
for $review in fn:docs("Reviews.xml")//Review
where $book/ISBN = $review/ISBN
order by $book/Title
return
<Desc>{$book/Title, $review/Description}</Desc>
```

Figure 3.5. An example to XQuery
(Source: Hollenstein 2005)

- **For clauses**: Associate one or more variables to expressions, creating a tuple stream in which each tuple binds a given variable to one of the items to which its associated expression evaluates.
- **Let clause**s: Bind variables to the entire result of an expression, adding these bindings to the tuples generated by a “for clause” or creating a single tuple to contain these bindings if there is no for clause.
- **Where clauses**: Filter tuples, retaining only those tuples that satisfy a condition.
- **Order by clauses**: Sort the tuples in a tuple stream.
- **Return clauses**: Build the result of the FLWOR expression for a given tuple.

3.5.4 Advantages of XQuery to other XML query languages

XQuery can be used to query XML data which has no schema at all or XML Schema or by a Document Type Definition (DTD). The data model used by XQuery has some differences from the classical relational model. Unlikely to relational model, it has no hierarchy, treats order as insignificant, and does not support identity. XQuery is a functional language, instead of executing commands as procedural languages do, every query is an expression to be evaluated, and expressions can be combined quite flexibly with other expressions to create new expressions (Robie 2004).

When we compare XPath and XQuery, we see that XPath only designed for select a node out of an existing XML document or database. XPath can't create new XML, it can't select only part of an XML node, and it can be hard to read and understand. XPath also can't define variables or namespace bindings and it has a very simple type system, essentially just string, boolean, double, and nodeset. If we need to work with date values, calculate the maximum of a set of numbers, or sort a list of strings, then XPath is not suitable for our application.

XQuery takes a different approach from XSLT 1.0. They both produce same results but XQuery is more functional. XQuery is very good at expressing joins and sorts and can operate sequences of values and nodes in arbitrary order and document order. XQuery takes a procedural approach to query processing, making it easy to write user defined functions, including recursive ones, but more difficult to perform pattern

matching. Support for XML Schema 1.0 is built into XQuery, and XQuery was designed with optimization in mind (Brundage 2004).

Funderburg (Funderburg, et al. 2002) points that number of special features can be added to the database XQuery processor which makes it different than other query languages. A default view can be generated across the entire database. If this is supported, seamless queries against meta-data and data will be possible. For example, one can ask for all the tables that have a column named salary and have a value larger than 10000. XML query languages naturally query across meta-data (tags) and data (node values). Exposing any XML view affords this ability. As more data are placed in the view, the queries can become more powerful and abstract. For example, a view could also expose type, ownership, and data cataloging information as well as data values (Reinwald 2002).

CHAPTER 4

XML MINING

4.1 Introduction

Following the increasing use of XML technology for data storage and data exchange between applications, the subject of mining XML documents has become more researchable topic. Understandably, the users wanted to be able to store, query and mine the growing amount of data represented in XML format (Rusu, et.al 2006).

XML mining, first named in IEEE International Conference on Data Mining in 2001 (J. Lee, et al. 2001), is a unique application of data mining to XML contents. Since its introduction, XML has reached more attention such in business applications collaborations such as ebXML and Web Services and recently web personalization such as Web 2.0. In spite of its frequent use, we have accomplished very little to reason about XML (Jeong, et al. 2006).

(Jeong, et al. 2006) defines XML mining as a special type of web content mining, but also unique in that contents in an XML document are modular and well-structured, while a web content is more likely a plain text document. That is, XML mining must be capable of manipulating the structure of contents as well as the contents themselves.

Jeong points that XML mining is certainly distinguished from text and web mining, in that it deals with modularly structured contents while text and web mining handle un-semistructured ones, such as HTML documents.

Recently, there has growing research interest to mine XML data. (Zhao, et al. 2005) defines existing work on mining XML data as frequent substructure mining, classification and association rule mining.

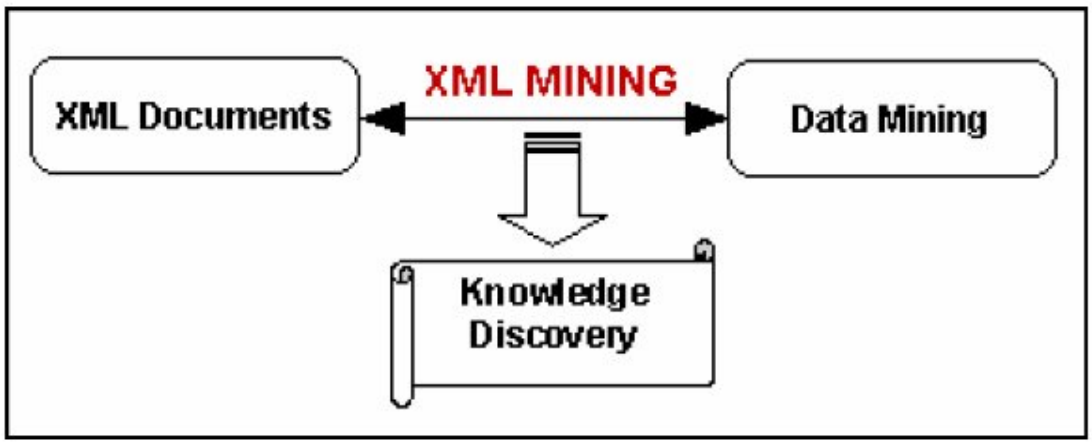

Figure 4.1. Role of XML Mining
(Source: Nayak 2003)

Mining of XML documents significantly differs from structured data mining and text mining. XML allows the representation of semi-structured and hierarchal data containing the values of individual items and the relationships between data items. Element tags and their nesting there in dictate the structure of an XML document. Due to the inherent flexibility of XML, in both structure and semantics, discovering knowledge from XML data is faced with new challenges as well as benefits. Mining of structure along with content provides new insights and means into the process of knowledge discovery (Nayak, et al. 2006).

4.2 Taxonomy of XML Mining

Since XML provides a mechanism for tagging names with data, knowledge discovery on the semantics of the documents becomes easier for improving document retrieval on the Web.

Mining of XML structure is essentially mining of schema including intra-structure mining, and inter-structure mining. (Nayak, et al. 2002)

(Nayak, et al. 2002) defines the Intra-structure mining is concerned with the structure within an XML document. Knowledge is discovered about the internal structure of XML documents in this type of mining.

Inter-structure mining is concerned with the structure between XML documents. Knowledge is discovered about the relationship between subjects, organizations, and nodes on the Web in this type of mining.

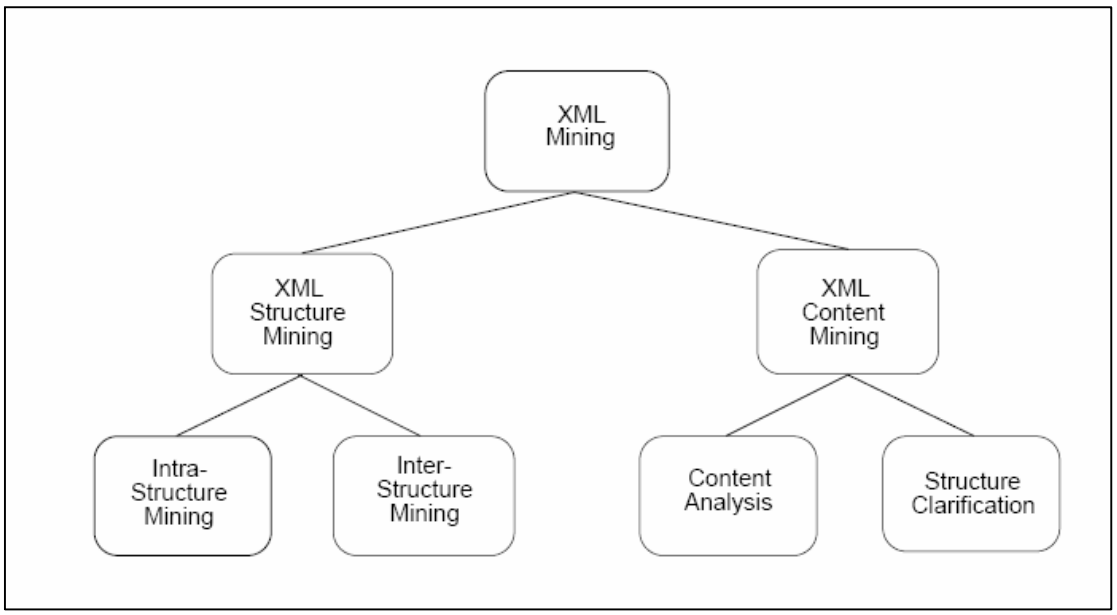

Figure 4.2. A Taxonomy of XML Mining
(Source: Nayak 2002)

Content is the text between each start and end tag in XML documents. Mining for XML content is essentially mining for values (an instance of a relation) including content analysis and structural clarification. Content analysis is concerned with analyzing texts within XML documents.

4.3 XML Mining Techniques

One of the simple methods to mine XML documents is probably to transform the data from XML to relations. However, the drawbacks of this method are: (Zhang, Yao 1999)

1) The transformation itself is usually complex and time-consuming;

2) The transformation may lose some important information for generating rules of interests, for example, some explicit hierarchical relationship between XML elements may become inexplicit when transformed into relations.

Before knowledge discovery in XML documents occurs, it is necessary to querying XML tags and content to prepare the XML material for mining. A SQL based query can extract data from XML documents. As mentioned before in chapter 2 XQuery and XPath could be used for querying xml documents.

4.3.1 Tree Mining over XML

The method proposed in (Zaki, et.al 2003) finds frequent structures within XML documents in order to classify them, i.e., a set of preclassified XML documents (training dataset) is used to develop a model to classify XML documents (test dataset) that still do not belong to a class. The model is created using the underlying structure of the preclassified documents.

Zaki models an XML document as an ordered, labeled, rooted tree. There is no distinction between attributes and elements of an XML document; both are mapped to the label set. More precisely an XML document is denoted as

T = (V, B), where V is the set of labeled nodes, and B the set of branches. The label of each node is taken from a set of items

L = {1, 2, 3, . . . ,m}; different nodes can have the same label.

Each branch, b = (x, y), is an ordered pair of nodes, where x is the parent of y. Consider a node x in a tree T with root r, then any node y on the unique path from r to x is called an ancestor of x, and is denoted as y ·l x, where l is the length of the path from y to x. If y ·1 x (i.e., y is an immediate ancestor of x), then y is called the parent of x, and x the child of y. A tree S = (Vs,Bs) is an embedded subtree of T = (V,B), denoted as S ¹ T, if and only if Vs µ V , b = (x, y) 2 Bs, and x is an ancestor of y in T. Note that in the traditional definition of an induced subtree, for each branch b = (x, y) 2 Bs, x must be a parent of y in T.

4.3.2 Mining association rules using XQuery

Wan and Dobbie presented a new native XML data mining approach in (Wan and Dobbie 2003).The authors show that extracting association rules from XML documents without any preprocessing or post-processing using XQuery is possible. They propose the XQuery implementation of the well-known Apriori algorithm.

The XML document structure proposed by the authors to mine association rules over XML reflects simply the relational model of the Association Rule Mining problem. The set of transactions is identified by the tag <transactions> and each transaction in the transactions set is identified by the tag <transaction>. The set of items in each transaction is identified by the tag <items> and an item is identified by the tag <item>. The Apriori algorithm is implemented in a classical way: first an XQuery expression is

used to create the set of frequent items then another XQuery expression is used to obtain the association rules from the frequent item sets.

Scientio XML Miner

XML Miner classifies XML fragments within one or more XML documents. The training set and test set are selected by using XPath expressions. The classifier model is created using Fuzzy Logic, i.e., an extension to conventional logic that allows representing the truth of an assertion by any real number between 1 and 0. Conventional logic allows only the value 1 that states the truthful of an assertion, and 0 that states is falsehood. Using Fuzzy Rules, XML Miner generates a rule set that explains and predicts selected values in a test dataset. The resulting rule set is expressed in Metarule, a dialect of XML. As an example the Fuzzy Logic uses the numeric values of the characteristic of the Iris species to define Fuzzy concepts of small, medium and large for each characteristic of the flower. Then the model is applied to the rest of the data to predict the species.

4.4 Related work

The topic of mining XML data has received little attention, as the data mining community has focused on the development of techniques for extracting common structure from heterogeneous XML data. For instance, (Termier, et al. 2002) has proposed an algorithm to construct a frequent tree by finding common subtrees embedded in the heterogeneous XML data. On the other hand, some researchers focus on developing a standard model to represent the knowledge extracted from the data using XML.

First studies in the area used techniques derived from Text Mining. Text Mining is an area of data mining that focused on finding repeating patterns inside text databases, i.e. Text Mining find frequent pattern of words inside a collection of phrases. In this framework an XML document is considered as a bag of words, and patterns are extracted from such bag. A second approach, native XML data mining is quite new to the area of data mining. Some studies in this area focus on mining frequent trees inside XML files. Tree mining over XML was first proposed in (Facca, et al. 2004); a similar approach is also proposed in (Fraternali, et al. 2004). This task is exploited to find

similarities between XML documents. In (Wan and Dobbie 2003) an approach to mine association rules from XML documents using an XQuery implementation of the Apriori algorithm is proposed. Besides these research efforts, XML Miner, a commercial product, has been developed by Scientio.

Some of the initial work for XML data mining is based on the use of the XPath language as the main component to query XML documents (Braga, et al 2002; Braga, et al 2003). Braga presented the XMINE operator, which is a tool developed to extract XML association rules for XML documents. The operator is based on XPath and inspired by the syntax of XQuery. It allows us to express complex mining tasks, compactly and intuitively. XMINE can be used to specify indifferently and simultaneously mining tasks both on the content and on the structure of the data, since the distinction in XML is slight. (Braga et al, 2003)

Other works for XML data mining focus on extracting the frequent tree patterns from the structure of XML data such as TreeFinder (Termier, et al. 2002) and TreeMiner (Zaki 2002). TreeFinder uses an Inductive Logic Programming approach. Notice that TreeFinder cannot produce complete results. It may miss many frequent subtrees, especially when the support threshold is small or trees in the database have common node labels. TreeMiner can produce the complete results by using a novel vertical representation for fast subtree support counting.

CHAPTER 5

ASSOCIATION RULE MINING AND ALGORITHMS

5.1. Overview

Association rule mining, one of the most important and well researched techniques of data mining, was first introduced in (Agrawal, et al. 1993). It aims to extract interesting correlations, frequent patterns, associations or casual structures among sets of items in the transaction databases or other data repositories. (Zhao, et al. 2005)

Association is a rule, which implies certain association relationships among set of objects such as occur together or one implies the other (Kumar 2001). Goal of association rule is finding associations among items from a set of transactions which contain a set of items.

Association rule mining is to find out association rules that satisfy the predefined minimum support and confidence from a given database. The problem is usually decomposed into two sub problems. One is to find those itemsets whose occurrences exceed a predefined threshold in the database; those itemsets are called frequent or large itemsets. The second problem is to generate association rules from those large itemsets with the constraints of minimal confidence (Kotsiantis, et al. 2006).

In this chapter, we consider the problem of "mining" a large collection of basket data type transactions on XML documents for finding association rules between sets of items. We present three different association rule mining algorithms: Apriori, AprioriTid and HEA (High Efficient AprioriTid) and their XQuery implementations.

5.2 Association Rule Mining Problem

5.2.1 Problem Definition and Decomposition

Association rule mining problem was first introduced in 1993 by R. Agrawal, T. Imielinski and A. Swami (Agrawal, et al. 1993). The following is a formal statement of the problem: (Agrawal, et al. 1993)

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, et al. 1993):

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 ≥ minconf, then the rule holds (the rule will surely have minimum support because ABCD is large).

5.2.2 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 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.(Grabusts 2004)

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.(Agrawal, et al. 1994)

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 $c[1] \cdot ... \cdot c[2] \cdot . . . \cdot c[k]$ 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 $c = X \cdot Y$ 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 5.1 the notation used in the Apriori and Apriori-like algorithms 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 5.1. Notation used in Apriori and Apriori-like algorithms.

5.3 Apriori Algorithm

An association rule mining algorithm, Apriori, was developed in 1993 by IBM's Quest Project Team (Agrawal, et al. 1993) for mining large transactional databases.

Figure 5.2 gives the Apriori algorithm. The algorithm makes multiple passes 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 5.2. The Apriori Algorithm.

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. First, in the join step, L_{k-1} is joined with L_{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}:

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.

5.4 Implementing Apriori Algorithm with XQuery

The query language XQuery was proposed by the W3C. The purpose of XQuery is to provide a flexible way to extract XML data and provide the necessary interaction between the web world and database world. More information about XQuery explained in Chapter 3.

```
<transactions>
        <transaction id="1">
                <items>
                        <item> a </item>
                        <item> d </item>
                        <item> e </item>
                </items>
        </transaction>
        <transaction id="2">
                <items>
                        <item> b </item>
                        <item> c </item>
                        <item> d </item>
                </items>
        </transaction>
        <transaction id="3">
                <items>
                        <item> a </item>
                        <item> c </item>
                </items>
        </transaction>
        <transaction id="4">
                <items>
                        <item> b </item>
                        <item> c </item>
                        <item> d </item>
                </items>
        </transaction>
        <transaction id="5">
                <items>
                        <item> a </item>
                        <item> b </item>
                </items>
        </transaction>
</transactions>
```

Figure 5.3. Transaction document (transactions.xml)
(Source: Wan and Dobbie 2003)

The XML document on Figure 5.3 contains set of transactions which are identified by the tag <transactions> and each transaction in the transactions set is identified by the tag <transaction>. The set of items in each transaction are identified by the tag <items> and an item is identified by the tag <item>.

```
let $src := doc("dataset.xml")//items
let $minsupp := 0.4
let $total := count($src) * 1.00
let $C := distinct-values($src/*)
let $L := (for $itemset in $C let $items :=(for $item in $src/*
where $itemset = $item  return $item)
let $sup := (count($items) * 1.00) div $total where $sup >= $minsupp
return <largeitemset>
<item> {$itemset} </item>
<support> {$sup} </support>
</largeitemset>)
 return
 <largeItemsets>{ local:apriori($l, $L,$minsup, $total, $src,$i) }</largeItemsets>
```

Figure 5.4. An XQuery expression which computes the association rules

The computation of the above expressions begins with several let clauses where we specify the data source $src, the support threshold $minsup and total number of transaction $total in the data source. After that, it starts to generate the candidate 1-itemsets $C. Once the candidate 1-itemsets are available, the algorithm is ready to generate the large 1-itemset $l by scanning through the transactions document to obtain the support value of each candidate 1-itemset and remove the one which is less than minsup. The variable $L is used to collect all the large itemsets in the transaction document. Finally, it will pass the information (e.g. $l, $L, $minsup, $total) to the recursive function Apriori in the return clause to generate the other large itemsets.

The function Apriori is called once in every level. Function generates the candidate set C in the current level by combining the large itemsets in the previous level. It then removes the unnecessary itemsets from C and obtains the large itemsets by reading the database to calculate the support. Figure 5.5 shows the Apriori function.

```
declare function local:apriori($l, $L, $minsup, $total ,$src)
 {
 let $C := local:removeDuplicate(local:candidateGen($l, ,$i))
 let $l := local:getLargeItemsets($C, $minsup, $total, $src)
 let $L := $l union $L
 return
 if (empty($l)) then $L
 else
 local:apriori($l, $L, $minsup, $total, $src)
}
```

Figure 5.5. Apriori function

Apriori function calls several necessary functions. These are

- CandidateGen( ),
- getLargeItemsets( ) ,
- removeDuplicate( ).

CandidateGen function generates candidate sets using previous levels' large itemset ($l). CandidateGen function returns current level candidate set. Figure 5.6 shows XQuery expression of AprioriGen function. Function uses join( ) and prune( ) functions to generate candidate itemsets.

```
declare function local:candidateGen($l,$j)
 {
 for $freqSet1 in $l
 let $items1 := $freqSet1//items/*
 for $freqSet2 in $l
 let $items2 := $freqSet2//items/*
 where $freqSet2 >> $freqSet1 and count($items1)+$j=count($items1 union $items2) and
local:prune(local:join($items1,$items2), $l)
 return
 <items>
 {local:join($items1,$items2)}
 </items>
 };
```

Figure 5.6. CandidateGen Function

. The function showed in Figure 5.7 joins two sequences and eliminates duplicates. It works like set join operation.

```
 declare function local: join($X, $Y)
 {
 let $items := ( for $item in $Y where every $i in $X satisfies $i != $item return $item )
 return $X union $items
 };
```

Figure 5.7. Join function

GetLargeItemsets function takes $C, $minsup, $total, $src as parameters and returns large itemset and its support value. $C represents the candidate itemset of the current level, $minsup represents the minimum support, $total symbolizes count of candidate itemset members and finally $src is the dataset for scanning. Function selects the itemsets which are supported above $minsup.

```
declare function local:getLargeItemsets($C,$minsup, $total, $src)
 {
 for $items in $C
 let $trans := (for $tran in $src where every $item1 in $items/* satisfies
 (some $item2 in $tran/* satisfies $item1 = $item2)
 return $tran )
 let $sup := (count($trans) * 1.00) div $total where $sup >= $minsup
 return
 <largeItemset>
 {$items} <support> {$sup} </support>
 </largeItemset>
 };
```

Figure 5.8. getLargeItemsets Function

RemoveDuplicate function called in apriori function because the candidates generated by candidateGen function may contain duplicate itemset. This function removes duplicates by comparing equality of the itemsets. The helper function named commonIts( ) works like a set intersect operation.

```
declare function local:removeDuplicate( $C )
 {
 for $itemset1 in $C
 let $items1 := $itemset1/*
 let $items :=(for $itemset2 in $C
 let $items2 := $itemset2/* where $itemset2>>$itemset1 and
count($items1)=count(local:commonIts($items1, $items2))
 return $items2 )
 where count($items) = 0
 return $itemset1
 };
```

Figure 5.9. Function removeDuplicate

Figure 5.10 illustrates the process of generating large itemsets. Minimum support value is defined as 0.4. Figure 5.10 shows all the large itemsets generated by our XQuery expression with the input file in Figure5.3.

```
<largeItemsets>
        <largeItemset>
                <items>
                        <item> a </item>
                </items>
                <support> 0.6 </support>
        </largeItemset>
        <largeItemset>
                <items>
                        <item> d </item>
                </items>
                <support> 0.6 </support>
        </largeItemset>
        <largeItemset>
                <items>
                        <item> b </item>
                </items>
                <support> 0.6 </support>
        </largeItemset>
                <items>
                        <item> c </item>
                </items>
                <support> 0.6 </support>
        </largeItemset>
        <largeItemset>
                <items>
                        <item> d </item>
                        <item> b </item>
                </items>
                <support> 0.4 </support>
        </largeItemset>
        <largeItemset>
                <items>
                        <item> d </item>
                        <item> c </item>
                </items>
                <support> 0.4 </support>
        </largeItemset>
        <largeItemset>
                <items>
                        <item> b </item>
                        <item> c </item>
                </items>
                <support> 0.4 </support>
        </largeItemset>
        <largeItemset>
                <items>
                        <item> d </item>
                        <item> c </item>
                        <item> b </item>
                </items>
                <support> 0.4 </support>
        </largeItemset>
</largeItemsets>
```

Figure 5.10. Output of the Apriori Algorithm

5.5 AprioriTid Algorithm

An association rule mining algorithm, AprioriTid, also was developed in 1993 by IBM's Quest Project Team (Agrawal, et al. 1993) for mining large transactional databases. Figure 5.11 shows the AprioriTid algorithm.

Contrasted to Apriori algorithm, the AprioriTid algorithm has the additional property that the database is not used at all for counting the support of candidate itemsets after the first pass. Rather, an encoding of the candidate itemsets used in the

previous pass is employed for this purpose. In later passes, the size of this encoding can become much smaller than the database, thus saving much reading effort. The AprioriTid algorithm also uses the Apriori-gen function to determine the candidate itemsets before the pass begins. The interesting feature of this algorithm is that the database D is not used for counting support after the first pass.

```
1)   L_1 := {frequent 1-itemsets};
2)   C̄_1 := database D;
2a)  k := 2;  // k represents the pass number
3)   while ( L_{k-1} ≠ ∅ ) do begin
4)       C_k := New candidates of size k generated from L_{k-1};
5)       C̄_k := ∅;
6)       forall entries T ∈ C̄_{k-1} do begin
7)           // determine candidate itemsets in C_k contained in
             // the transaction with identifier T.TID
             C_t := {c ∈ C_k | (c - c[k]) ∈ t.set-of-itemsets ∧ (c - c[k-1]) ∈ t.set-of-itemsets};
8)           Increment the count of all candidates in C_t.
10)          if (C_t ≠ ∅) then C̄_k += ⟨t.TID, C_t⟩;
11)      end
12)      L_k := All candidates in C_k with minimum support.
12a)     k := k + 1;
13)  end
14)  Answer = ⋃_k L_k;
```

Figure 5.11. AprioriTid Algorithm
(Source: Agrawal, et al. 1993)

The AprioriTid algorithm also uses the Apriori candidate generation function to determine the candidate itemsets before the pass begins. The interesting feature of this algorithm is that the database D is not used for counting support after the first pass rather the set Ck is used for this purpose. Each member of the set $\overline{C}_k$ is of the form <TID, $\{X_k\}$> where each X_k is a potentially frequent k-itemset present in the transaction with identifier TID. For k=1, $\overline{C}_k$ corresponds to the database D, although conceptually each item i is replaced by the itemset {i}. For k > 1 $\overline{C}_k$ is generated by the algorithm. The member of $\overline{C}_k$ corresponding to transaction T is

< t.TID, {c ∈ Ck | c contained in t}>.

If a transaction does not contain any candidate k-itemset, then $\overline{C}_k$ will not have an entry for this transaction. Thus, the number of entries in $\overline{C}_k$ may be smaller than the number of transactions in the database, especially for large values of k. In addition, for large values of k, each entry may be smaller than the corresponding transaction because

very few candidates may be contained in the transaction. However, for small values for k, each entry may be larger than the corresponding transaction because an entry in $\overline{C}_k$ includes all candidate k-itemsets contained in the transaction.

The XQuery expression of AprioriTid algorithm differs from Apriori in large itemset generation function and a new function needed to compute $\overline{C}_k$. This new function is named as generatenewCk. The function that finds the large itemset has some changes in the parameter values. $\overline{C}_k$ is used at all for counting the support of candidate itemsets in the large itemset generator function.

```
declare function local:generatenewCk($L,$src)
                                                {
let $cx:=local:removeDuplicate(local:candidateGen($L))
let $ck:= (for $k in 1 to count($L//items)
let $L:=(for $i in 1 to count($src[$k]//item)
return <largeitemset>
<items><items> {distinct-values($src[$k]//item[$i])} </items></items>
<support> {$sup} </support>
</largeitemset>)
return local:candidateGen($L) )
let $item1:=count($ck)
let $item2:=$cx
for $items in $ck
 let $trans := ( for $tran in $cx  where every $item1 in $items/* satisfies
 (some $item2 in $tran/* satisfies $item1 = $item2)
 return $tran)
return $trans}:
```

Figure 5.12. generatenewCk function

5.6 High Efficient AprioriTid Algorithm (HEA)

HEA algorithm is improved AprioriTid algorithm (Chao Li et al. 2005), whose idea: on the basis of theorem, only use itemsets of C_{k-1}, which supports are equal to or greater than minsup, to build $\overline{C}_{k-1}$; so we renew the relationship of transaction t with entry, defined as

$\overline{C}_k = < t.TID, \{c \in Ck \mid c \subseteq t$ and c.sup port $\geq$ minsup$\} >$;

This will excessively decrease size of stored data in $\overline{C}_k$; moreover, because searching data scale is reduced when we compute support of itemsets in C_K. At the same time, it will reduce much time of I/O and running.

1) L_1={largel-itemsets};
2) $\overline{C_1} = database\ D$;
3) For (k=2; $L_{k-1} \neq \varnothing$; K++) do begin
4) $C_k = appriori - gen(L_{k-1})$;
// New candidates
5) $C_k = \varnothing$;
6) For all $c \in C_k$ do begin
7) $T_c = \{t.TID \mid t \in \overline{C_{k-1}}, (c - c[k]) \in t.set - of - itemsets \wedge (c - c[k-1] \in t.set - of - itemsets)\}$
8) If $|T_c| \geq \min sup$ then begin
9) $L_k = L_k \cup \{c\}$
10) For all $p \in T_c$ do
11) $\overline{C_k} = \overline{C_k} \cup < p, c >$;
12) end
13) end
14) end
15) Answer= $\cup L_k$

Figure 5.13. HEA Algorithm
(Source: Chao Li, et al. 2005)

In AprioriTid algorithm, from step 6 to step 11, it is to compute new $\overline{C_k}$ and support of itemset in C_k. So it can be added to the sentence, which will delete itemset that support is smaller than minsup in $\overline{C_k}$, to optimize the algorithm after step 11. However, in application of data mining, frequent itemset also can be build from SQL sentence on the basis of structure of table. So through analysis, we also use circle of in $\overline{C_k}$ HEA to instead of circle of $\overline{C_k}$ in AprioriTid step 6

Firstly, Confirming itemset c in C_k, then transaction set T_c, presented with TID, including the item of c, is computed; Secondly, the number of entry is computed in T_c, defined as $|Tc|$, which is support of itemset c; Thirdly, if $|Tc| \geq \min sup$, c is included into L_k and $\overline{C_k}$ otherwise deleting c. Through above process, with the computed support in C_k, void itemset can be directly deleted from $\overline{C_k}$ or added to $\overline{C_k}$ and L_k.

The XQuery expression of this algorithm uses Apriori helper functions like AprioriTid algorithm. The difference of this expression is in function which generates the $\overline{C_k}$.

```
declare function local:HEAnewCk($L,$src,$total)
{
let $cx:=local:removeDuplicate(local:candidateGen($L))
let $ck:= (for $k in 1 to count($L//items)
let $L:=(for $i in 1 to count($src[$k]//item)
return <largeitemset>
<items><items> {distinct-values($src[$k]//item[$i])} </items></items>
<support> {$sup} </support>
</largeitemset>)
return local:candidateGen($L) )
let $item1:=count($ck)
let $item2:=$cx
for $items in $ck
 let $trans := ( for $tran in $cx  where every $item1 in $items/* satisfies
 (some $item2 in $tran/* satisfies $item1 = $item2 and )
 return $tran)
 let $sup := (count($trans) * 1.00) div $total
 where $sup >= $minsup
return $trans
};
```

Figure 5.14. HEAnewCk function

CHAPTER 6

SAMPLE DATA SETS

In order to compare the algorithms, experimental studies using different data sets (three synthetic data sets) and different support levels have been conducted. In this chapter we present our creating process of the datasets and statistical analysis.

All the data sets were performed on a 1.86 GHz Intel Pentium PC machine with 1 gigabyte main memory, running on Microsoft Windows/XP. All the dataset creation programs were source coded in JAVA due to its Collections Framework (java.util package). As a Java Virtual Machine we utilized the one running on Java Standart Development Kit - Version 1.5.

6.1 Synthetic Data Generation and Analysis of the Data Sets

The algorithms were tested on different data sets: three synthetic data sets. We produced three different data sets which have 100,1000 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 and the statistical features of the synthetic data sets with 100,1000 and 5000 transactions are described .

For generating synthetic data sets firstly we determine the borders of the data sets. Such as;

- maximum transaction size: 20
- number of items: 50
- number of transactions: 100 ,1000 and 5000

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). These matrices pre-desined before the synthetic dataset generation.Then we sort the probabilitiy of occurences that Transaction_Size_Matrix contain in a descending order. We extend the probability of occurences of items that Item_Matrix hold. Briefly we

create a 1 x 1000 matrix that will contain the items according to their probability of occurences. For example, in this new matrix, the item "1" will occupy the first 1000*0.077 place (because in Item_Matrix "1" refers 0,077) and item 2 will occupy the following 1000*0.069 place (because in Item_Matrix "2" refers 0,069),...,etc. As a result, the new matrix (new_matrix) will contain 1000 items from 1 to 50 in a sequential order.

```
dataGeneration()
for (i=0;i<transnumber;i++)//trnasnumber could be 100, 1000 or 5000
    transsize=gettranssize();//define the transaction size using
transaction_size_matrix
    for(j=0;j<transize;j++)
        takenitem=getItem();//define the item using new_matrix
        addtotrans(takenitem);// add item to the dataset.
```

Figure 6.1. Synthetic data generation algorithm.

After generating this new matrix data generation starts. The data generation algorithm structure can be seen in Figure 6.1. The function getTransSize() generates a random number using uniform distribution between 0 and 1 and returns the index which corresponds to the random number in Transaction_Size_Matrix. The function getItem() generates a random number using uniform distribution between 1 and 1000 (the size of the new_matrix) and returns the value that corresponds to the random number in new_matrix .

6.2.1 Analysis of "Synthetic Data Set with 100 Transactions"

The features and the analysis of the "Synthetic Data Set with 100 Transactions" are given in Figure 6.2 and Figure 6.3 respectively. This dataset contains 1048 items. Maximum transaction size is 20. Average transaction size is 10.57.

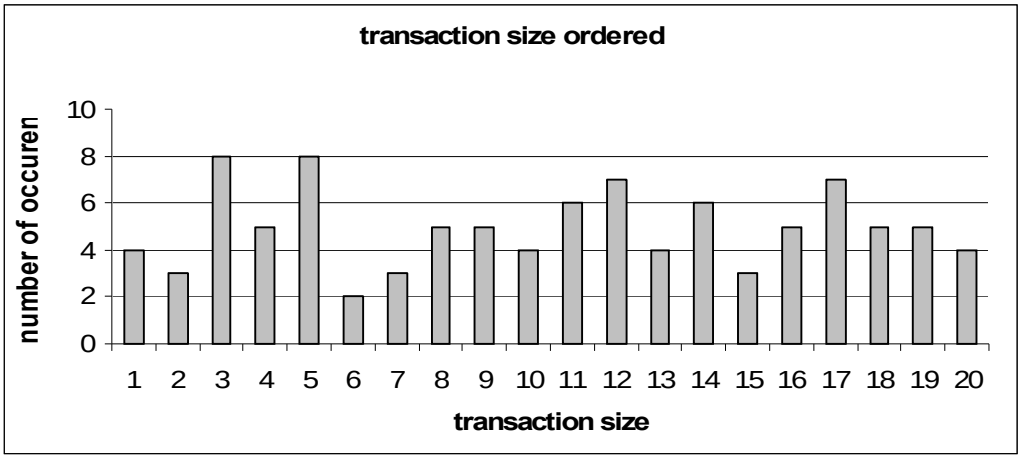

Figure 6.2. Frequency of transaction size on dataset with 100 transactions

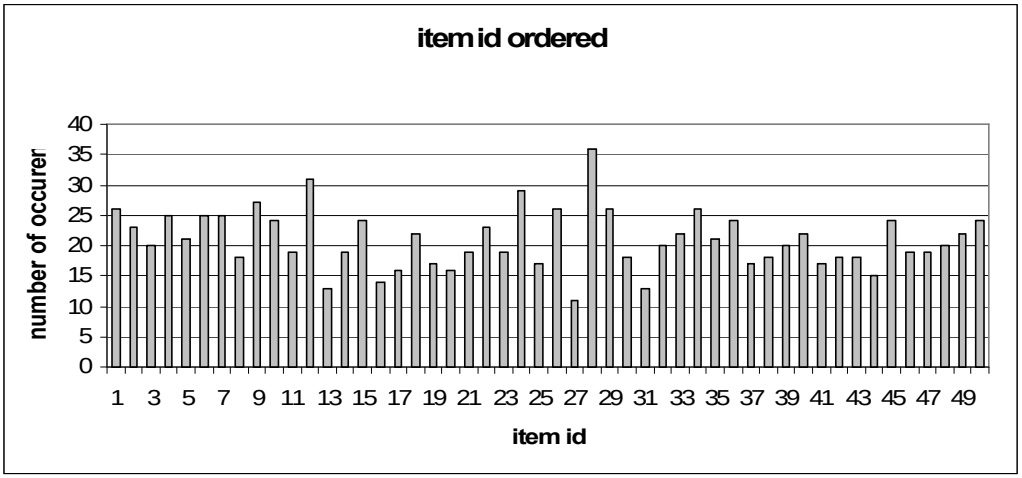

Figure 6.3. Frequency of items on dataset with 100 transactions

6.2.2 Analysis of "Synthetic Data Set with 1000 Transactions"

The features and the analysis of the "Synthetic Data Set with 1000 Transactions" are given in Figure 6.4. and Figure 6.5. respectively. This dataset contains 11530 items. Maximum transaction size is 20. Average transaction size is 9.54.

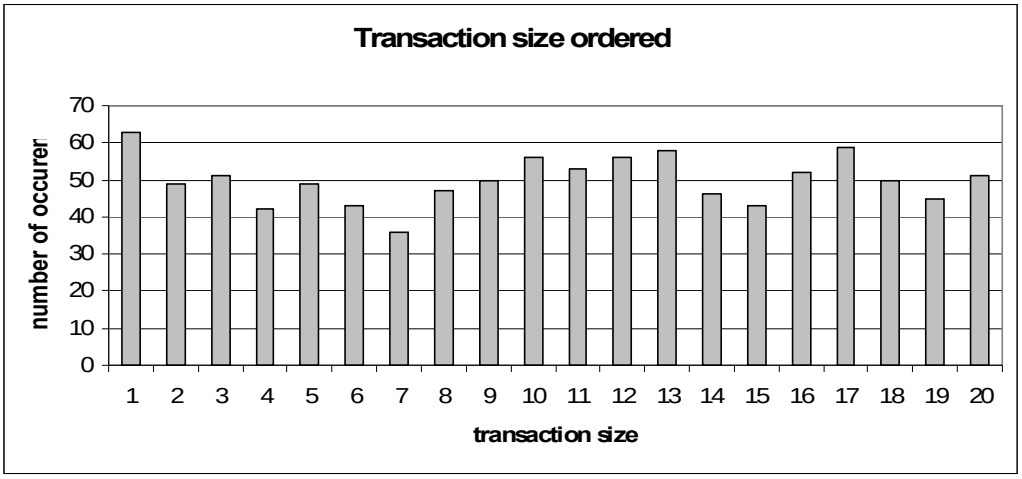

Figure 6.4. Frequency of transaction size on dataset with 1000 transactions

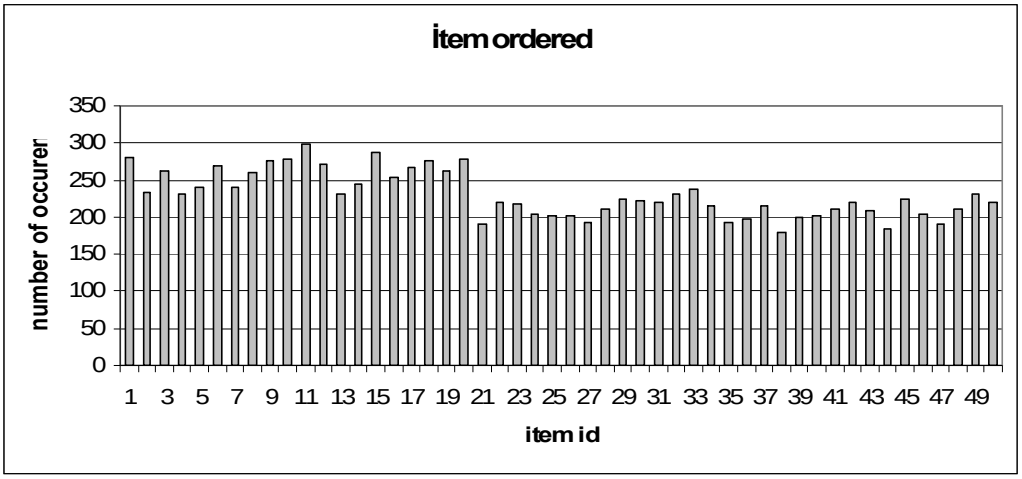

Figure 6.5. Frequency of items on dataset with 1000 transactions

6.2.3 Analysis of "Synthetic Data Set with 5000 Transactions"

The features and the analysis of the "Synthetic Data Set with 5000 Transactions" are given in Figure 6.6 and Figure 6.7 respectively. This dataset contains 35716 items. Maximum transaction size is 20. Average transaction size is 9.37

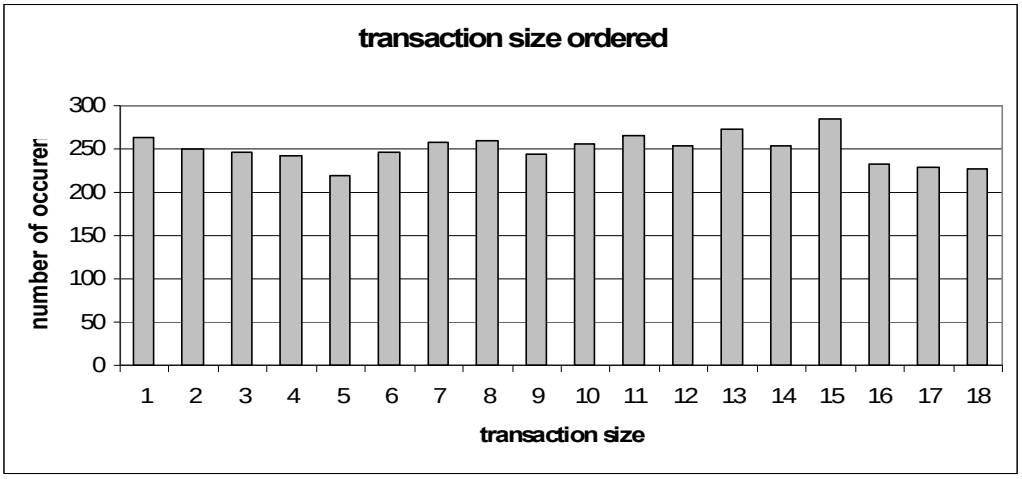

Figure 6.6. Frequency of transaction size on dataset with 5000 transactions

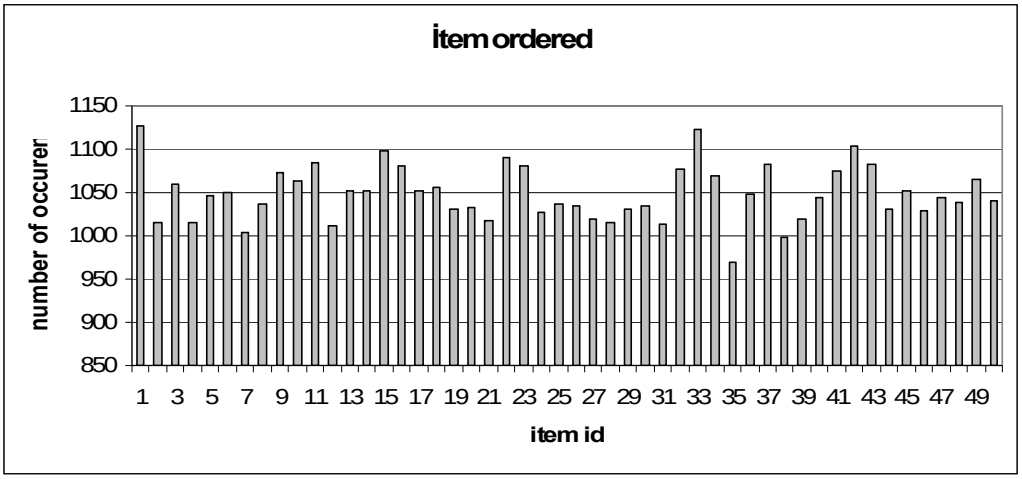

Figure 6.7. Frequency of items on dataset with 5000 transactions

CHAPTER 7

RESULTS AND DISCUSSION

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

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.

Contrasted to Apriori algorithm, the AprioriTid algorithm has the additional property that the database is not used at all in defining large itemsets for counting the support of candidate itemsets after the first pass. Instead of this an encoding of the candidate itemsets used in the previous pass is employed for this purpose called as $\overline{C}_k$. In later passes, the size of this encoding can become much smaller than the database, thus saving much reading effort. The AprioriTid algorithm also uses the AprioriGen function to determine the candidate itemsets before the pass begins.

The High Efficient AprioriTid algorithm is improvement of AprioriTid algorithm. This algorithm only uses encoding of candidate itemsets, which supports are equal to or greater than minsup. This approach will decrease size of stored data in encoding of candidate itemsets.

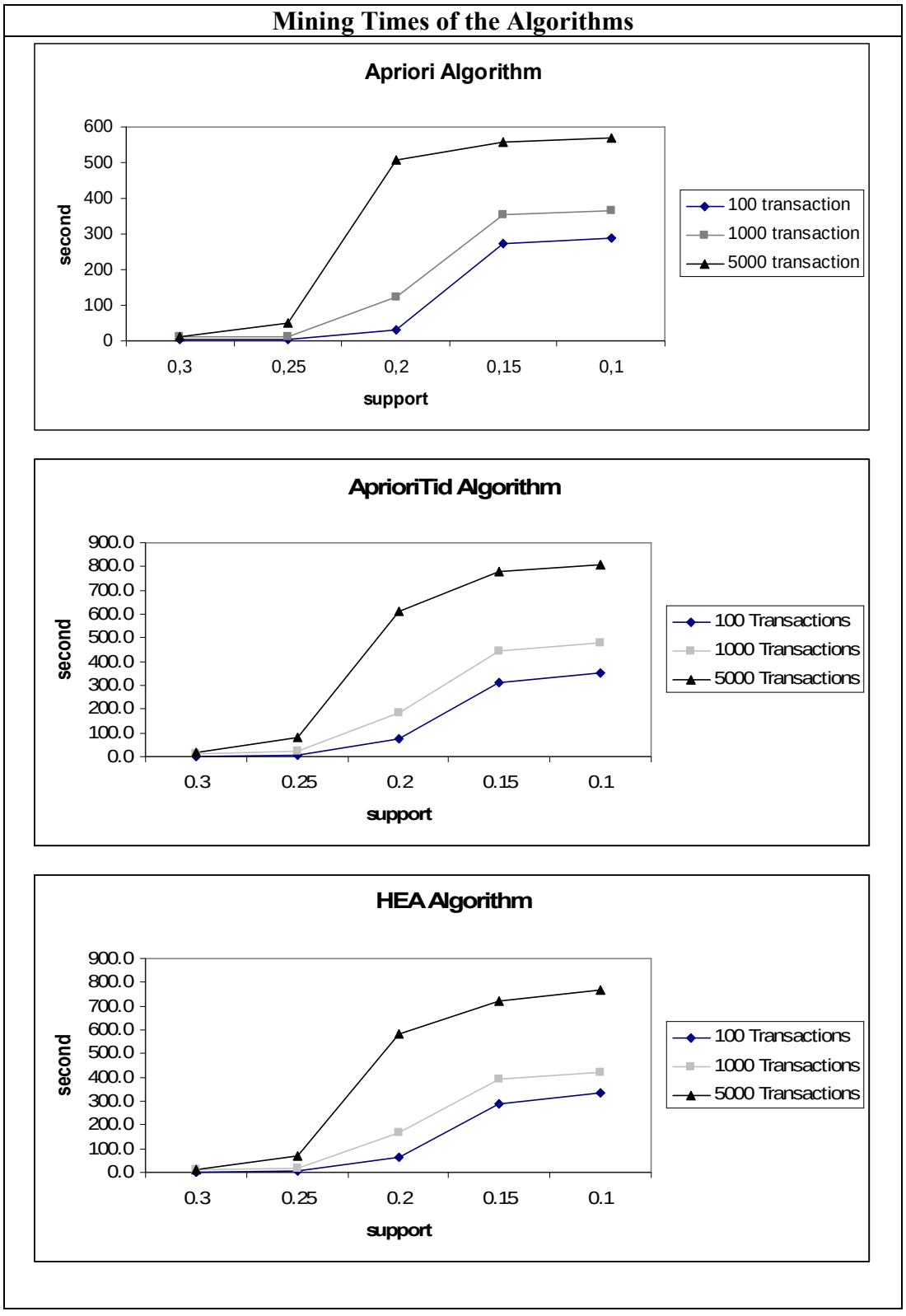

Figure 7.1. The "Mining Times" of the algorithms for different data sets and at different support levels are given in (a), (b) and (c)

4 different datasets have been used and their average mining times are taken in order to compare their performances. These results can be seen in Figure 7.2, 7.3 and 7.4.

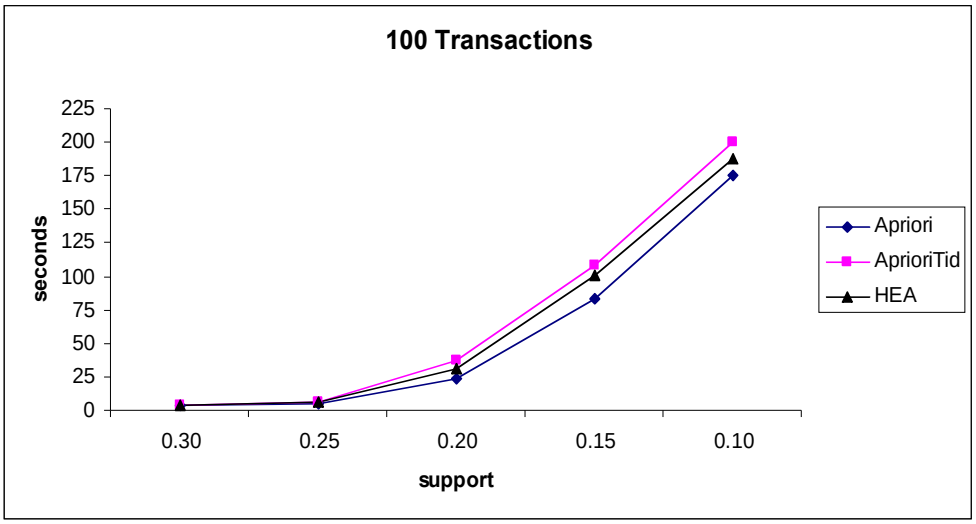

Figure 7.2. Performance of the algorithms in dataset with 100 transactions

Figure 7.2 shows the performance of algorithms in dataset with 100 transactions. AprioriTid has the longest time in mining process with 100 transactions.

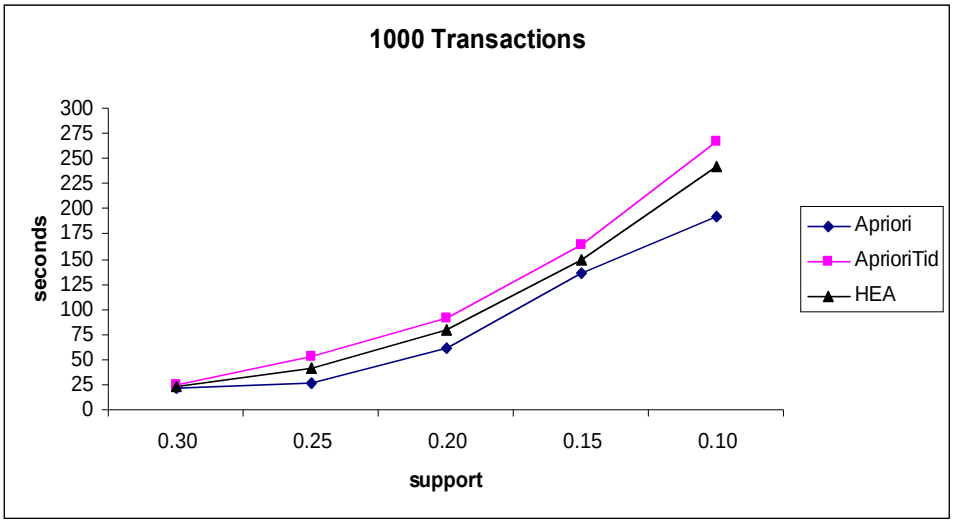

Figure 7.3. Performance of the algorithms in dataset with 1000 transactions

When algorithms compared in dataset which has 1000 transactions , similar results can be seen in Figure 7.3 as in Figure 7.2. However, the perfromances are slightly different then Figure 7.2 for support values 0.25 and 0.20.

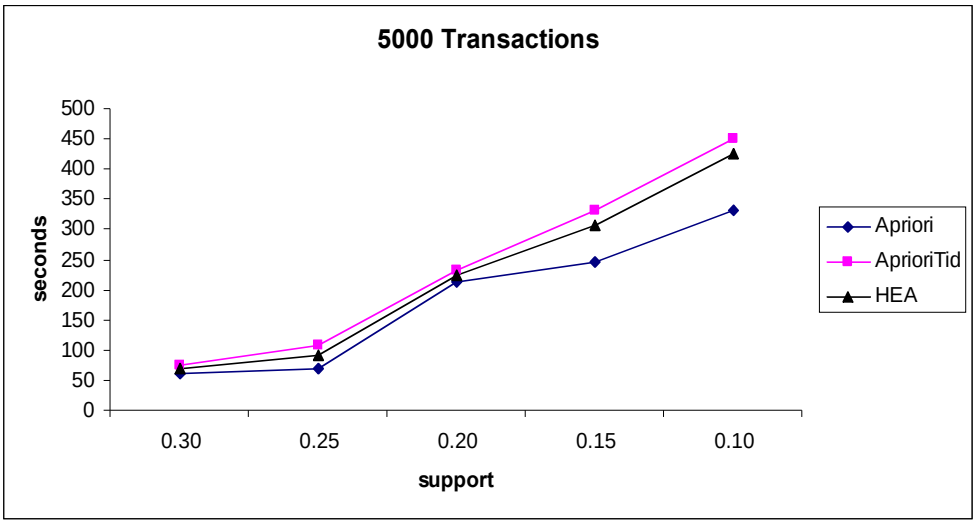

Figure 7.4. Performance of the algorithms in dataset with 5000 transactions

For all algorithms as the minimum support decreases, the execution times of all the algorithms increase because of increases in the total number of candidate itemsets and large itemsets.

In order to compare the effects of factors of mining times a univariate analysis of variance was conducted. For this purpose SPSS tool was used. We examined four parameters to see how they affect on mining times. “Datasetno” represents for 4 different datasets. “Support” is representing for support values like 0.3, 0.25, 0.2, 0.15 and 0.1. “Dsize” symbolizes the size of the databases which are 100, 1000 and 5000. “Algorithm” represents different types of algorithms like Apriori, AprioriTid and HEA. Tests of Between-Subjects Effects can be seen in Table 7.1.

Table 7.1. Tests of Between-Subjects Effects

Tests of Between-Subjects Effects Dependent Variable: time					
Source	Type III Sum of Squares	df	Mean Square	F	Sig.
Datasetno	1519494,124	3	506498,041	34,250	,000
Support	1479691,485	4	369922,871	25,015	,000
Dsize	729660,106	2	364830,053	24,670	,000
Algorithm	34794,915	2	17397,458	1,176	,312
dsize * algorithm	9702,168	4	2425,542	,164	,956
support * algorithm	18173,853	8	2271,732	,154	,996
support * dsize	153210,837	8	19151,355	1,295	,251
support * dsize * algorithm	7160,782	16	447,549	,030	1,000
Error	1952040,953	132	14788,189		
Corrected Total	5903929,223	179			

Univariate analysis of variance is significant when α is smaller than 0.05 for 95% confidence interval. Results of univariate analysis of variance shows that significance values for "Datasetno", "Support" and "Dsize" are smaller than 0.05. Therefore it's concluded that these factors affect the mining time significantly. These differences can be seen in Table 7.3, 7.4 and 7.5 However as indicated in the Table 7.1 "Algorithm" factor is not smaller than 0.05. Although there are differences between three algorithms in Table 7.2, these differences are not statistically significant.

Table 7.2. Means of Time for Algorithms

Estimates Dependent Variable:time	
Algorithm	Mean
1 (Apriori)	110,101
2 (AprioriTid)	143,564
3 (HEA)	132,317

Table 7.3. Means of Time for Datasets

Estimates Dependent Variable:time	
Datasetno	Mean
1	287,468
2	71,196
3	85,384
4	70,595

Table 7.4. Means of Time for Support values

Estimates Dependent Variable:time	
support	Mean
0.30	32,019
0.25	41,423
0.20	114,643
0.15	180,843
0.10	274,376

Means of time decreases as support values increases. The Figure 7.5 represents this inverse proportion.

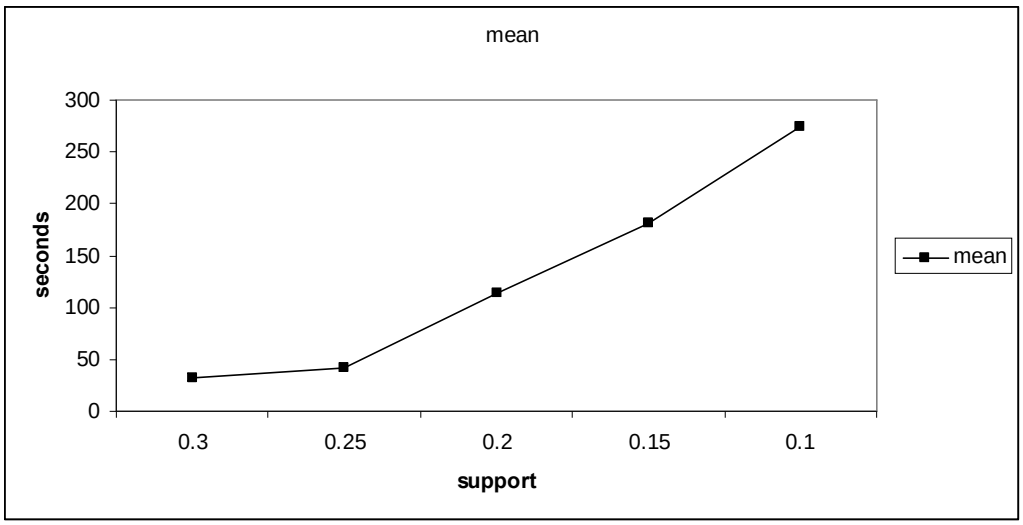

Figure 7.5. Means of Time for Support values

Table 7.5. Means of Time for Dataset size

Estimates Dependent Variable:time	
Dataset size	Mean
100	65,228
1000	105,036
5000	215,719

In order to see the difference of estimated marginal means for database size the database size values were transformed to log scale. The aim of this transformation was because of large range of the database sizes. The Figure 7.6 is created in SPSS Chart Builder module by using transformed values of database sizes. Figure 7.6 shows that means of mining times increase parallel with database size.

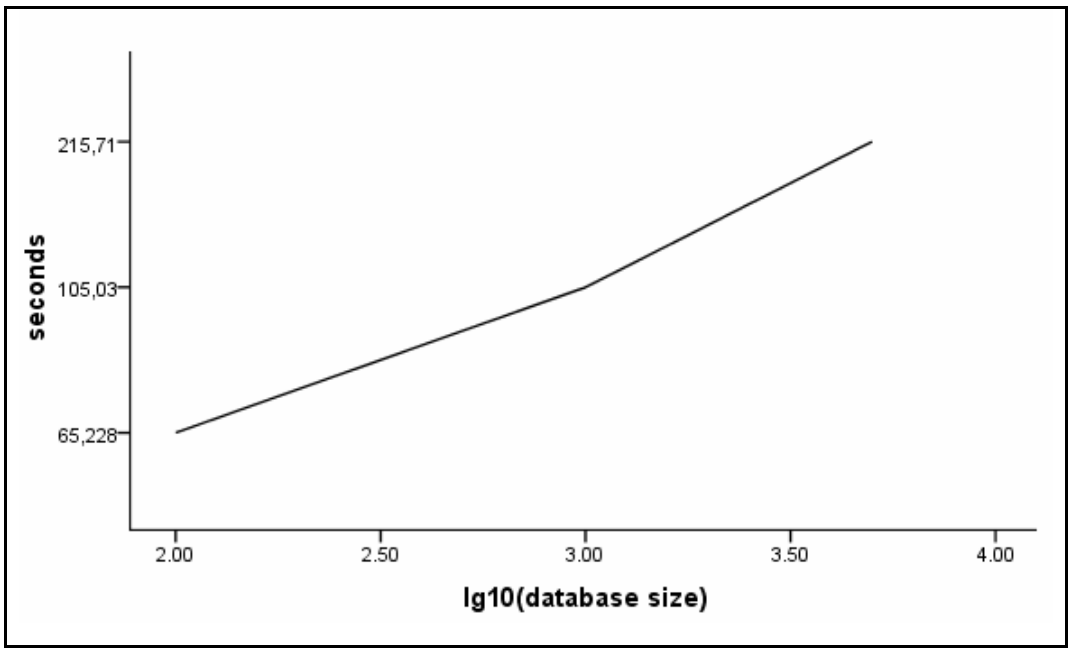

Figure 7.6. Means of Time for Dataset size

CHAPTER 8

CONCLUSION

The principal purpose of this study is applying association rule algorithms directly to the XML documents with using XQuery. Although there have been several studies about algorithms for association rules implementing on XQuery which are used in XML mining such as Apriori , the authors do not discuss the issue of large datasets and we have not found any study about applying AprioriTid and HEA with XQuery. Also there is no study about comparison of these three apriori-like algorithms on XML documents using different support levels and different database sizes. The algorithms evaluated in this study are:

- Apriori Algorithm
- AprioriTid Algorithm
- HEA Algorithm

The above algorithms were chosen for the implementation and comparison due to the fact that they serve to the same purpose and to see their effect in mining XML documents. The source codes of dataset creation process were written in JAVA.. Association rule algorithms are implemented in XQuery. The environment we worked as an editor is Eclipse with XQuery ide. The source codes of the algorithms and datasets were processed on 1.86 GHz Intel Pentium PC machine with 1 gigabyte main memory, running on Microsoft Windows XP.

In this study we see that AprioriTid and HEA algorithms can be applied on XQuery and can be used in XML mining. According to our observations, the performances of the algorithms are depending on the support levels and the size of the data sets.

Future Work

There is an increasing research efforts going on in XML data mining which is based on XML documents. Investigating the behaviors of the XQuery implementations of algorithms on larger XML datasets and their relations with memory size could be defined as a future research area.

REFERENCES

Agrawal, R., Imielinski, T. and Swami, A. 2003. Mining Association Rules between Sets of Items in Large Databases. *SIGMOD Conference* 207-216

Apte, C. and Hong, S.J 1994. Predicting equity returns from securities data. *Advances in Knowledge Discovery and Data Mining* 541-560.

Berkhin, P. 2002. Survey of clustering data mining techniques. *Technical report, Accrue Software* San Jose, CA

Berry, M. and Linoff, S. 2000. *Mastering Data Mining: The Art and Science of Customer Relationship Management*. John Wiley and Sons Inc, USA.

Braga, D., Campi, A., Klemettinen M. and Lanzi P. L. 2001. Mining Association Rules from XML data. *In Proceedings of the 4th International Conference on Data Warehousing and Knowledge Discovery* 21-30.

Braga, D. and Campi, A. 2003. XQuery by Example.*12th International World Wide Web Conference Poster Session* 291.

Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. 1984. *Classification and Regression Trees*. Wadsworth International, Belmont.

Brundage, M. 2004. *XQuery: The XML Query Language*. Addison Wesley.

Cheung, D., Vincent, T., Wai-Chee, F. and Yongjian, F. 1996. Efficient Mining of Association Rules in Distributed Databases. *IEEE Transactions on Knowledge and Data Engineering 8*(6): 911-922.

Djorgovski, S. and Weir, N. 1996. From Digitized Images to On-Line Catalogs: Data Mining a Sky Survey. *AI Magazine* 17(2): 51–66.

Dunham, M. 2002. Data Mining: Introductory and Advanced Topics. Prentice-Hall

Edelstein, H. 1998. *Introduction to Data Mining and Knowledge Discovery*. Two Crows Corporation 2nd Edition, ISBN 1892095009

Egnor, D. and Lord, R. 2000. Structured Information Retrieval using XML. In *Working Notes of the ACM SIGIR Workshop on XML and Information Retrieval* 663-667.

Facca, F., Lanzi, P., Colombetti, M. and Ceri, S. 2004. Mining Patterns from XML Data: a structure-based approach. *Technical Report Dipartimento di Elettronica e Informazione,* Politecnico di Milano.

Fayyad, U., Shapiro, G. And Smyth, P. 1996. *Advances in Knowledge Discovery and Data Mining*, MIT Press.

Fayyad, U., Shapiro, G. And Smyth, P. 1996. Data Mining and Knowledge Discovery in Databases: An overview. *Communications of ACM* 39(11):27-34.

Feng, L. and Dillon, T. 2004. Mining XML-Enabled. Association Rules with Templates. *Proc. of the 3rd. International Workshop on Knowledge Discovery in Inductive Databases* 66-88.

Feng, L. and Dillon, T. 2005. An XML-Enabled Data Mining Query Language XML-DMQL. *International Journal of Business Intelligence and Data Mining* 22-41. ISSN 1743-8187

Funderburk E., Malaika S. and Reinwald B. 2002. XML programming with SQL/XML and XQuery. *IBM Systems Journal,* 41(4):642-665.

Goldfarb, C. 1998. *The XML Handbook*. Prentice Hall PTR.

Grabusts P., 2004. Using Association Rules to Extract Regularities from Data. *Proceedings of 6.International Baltic Conference on Data Bases and Information Systems* Riga 117-126.

Han, J. and Kamber, M. 2001. *Data Mining Concepts and Techniques*, Morgan Kauffmann Publishers Inc.

Hollenstein, S. 2005. XQuery Similarity Joins. Master's Thesis, Dynamic and Distributed Information Systems, University of Zurich.

Jeong, B., Lee, D., Lee, J. and Cho, H. 2006. Towards XML Mining: The Role of Kernel Methods. *Proceedings of the 2006 Fall Data Mining Conference*, Seoul, South Korea.

Kotsiantis, S., Kanellopoulos, D. 2006. Association Rules Mining: A Recent Overview. *GESTS International Transactions on Computer Science and Engineering*, 32(1):71-82.

Kumar, V. 2000. Interestingness Measures for Association Patterns: A Perspective. *Technical Repor*t, Department of Computer Science, University of Minnesota.

Laurent, H., 1999. Describing Your Data: DTDs and XML Schemas http://www.xml.com/pub/1999/12/dtd/ (accessed December 12, 2006).

Ming, Z., JingTao, Y. 2005. XML algebras for data mining. Data Mining and Knowledge Discovery: Theory, Tools, and Technology VI. *Proceedings of the SPIE,* 5433:209-217

Mulchandani, M., Wang, L. and Rundensteiner, A. 2003. Updating XQuery Views Published over Relational Data. *Technical Report WPI-CS-TR-03-23b*, Computer Science Department, WPI.

Nayak, R., Witt, R. and Tonev, A. 2002. Data mining and XML documents. *In Proceedings International Conference on Internet Computing, IC'2002* 3:660-666.

Nayak, R. and Zaki, M. 2006. Knowledge Discovery from XML documents: PAKDD 2006 Workshop Proceedings, *Lecture Notes in Computer Science, Springer-Verlag*.

Rinta-Runsala, E. And Bounsaythip, C. 2001. *VTT Information Technology Research Report* TTE-2001 18.

Robie, J., Katz, H., Chamberlin, D., Draper, D., Fernandez, M., Kay, M. 2004. *XQuery from the Experts: A Guide to the W3C XML Query Language*, ISBN 0-321-18060-7

Rusu, L., Rahayu, W. and Taniar, D. 2006. Mining Changes from Versions of Dynamic XML Documents. *Knowledge Discovery form XML Documents 2006*, LNCS3915 3-12.

Skonnard and Gudgin 2002. *Essential XML Quick Reference*. Addison-Wesley Publishers.

Termier, A., Rousset, M. and Sebag, M. 2002. Treefinder: A First Step towards XML Data Mining, *Proceedings of the IEEE International Conference Data Mining* 450-458.

The World Wide Web Consortium (W3C) - http://www.w3.org (accessed June 12, 2007)

Wan, J. and Dobbie, G. 2003. Extracting association rules from XML documents using XQuery. *Proceedings of the 5th ACM international workshop on Web information and data management,* 94-97.

Wang, F. 2004. *XML-based Support for Database Histories and Document Versions*. Ph.D. Thesis, University of California, Los Angeles.

Weiss, S.M. and Kulikowski, C.A. 1991. *Computer systems that learn: classification and prediction methods from statistics, neural nets, machine learning, and expert systems*. New York, NY: Morgan Kaufman.

Werbos, P. 1974. *Beyond regression: New tools for predictions and analysis in the behavioural sciences*. PhD Thesis, Cambridge, MA: Harvard University.

Westphal, C. And Blaxton, T. 1998.*Data Mining Solutions: Methods and Tools for Solving Real-World Problems*, Wiley Computer Publishing.

Zaïane, R. 1999. *Principles of Knowledge Discovery in Databases*. University of Alberta, USA.

Zaki, M. and Aggarwal, C. 2003. XRULES: An Effective Structural Classifier for XML Data. *9th International Conference on Knowledge Discovery and Data Mining*, 62(1):137-170.

Zhao, Q., Chen, L., Bhowmicka, S. and Madriab, S. 2005. XML structural delta mining: Issues and challenges. *Data & Knowledge Engineering archive* 59(3):627-651.

Zhi-Chao, L., Pi-Lian H., Lei, M. 2005. A high efficient AprioriTid algorithm for mining association rule. *Machine Learning and Cybernetics*. *Proceedings of 2005 International Conference* 1812 – 1815.