DISCOVERY OF MULTIPLE-LEVEL RULES FROM LARGE DATABASES

by

Yongjian Fu

B. Sc. Zhejiang University, China 1985

M. Sc. Zhejiang University, China 1988

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY
in the School
of
Computing Science

SIMON FRASER UNIVERSITY
July 1996

APPROVAL

Name: Yongjian Fu

Degree: Doctor of Philosophy

Title of thesis: Discovery of Multiple-Level Rules from Large Databases

Examining Committee: Dr. Robert F. Hadley
Chair

Dr. Jiawei Han
Senior Supervisor

Dr. Veronica Dahl
Supervisor

Dr. Tiko Kameda
SFU Examiner

Dr. Len Shapiro
External Examiner, Portland State University

Date Approved: July 26, '96

SIMON FRASER UNIVERSITY

PARTIAL COPYRIGHT LICENSE

I hereby grant to Simon Fraser University the right to lend my thesis, project or extended essay (the title of which is shown below) to users of the Simon Fraser University Library, and to make partial or single copies only for such users or in response to a request from the library of any other university, or other educational institution, on its own behalf or for one of its users. I further agree that permission for multiple copying of this work for scholarly purposes may be granted by me or the Dean of Graduate Studies. It is understood that copying or publication of this work for financial gain shall not be allowed without my written permission.

Title of Thesis/Project/Extended Essay

Discovery of Multiple-Level Rules from Large Databases

Author: _____

(signature)

_____

(name)

August 15, 1996

_____

(date)

Abstract

With the widespread computerization in business, government, and science, the efficient and effective discovery of interesting information from large databases becomes essential. Data mining or Knowledge Discovery in Database (KDD) emerges as a solution to the data analysis problems faced by many organizations. Previous studies on data mining have been focused on the discovery of knowledge at a single conceptual level, either at the primitive level or at a rather high conceptual level. However, it is often desirable to discover knowledge at multiple conceptual levels, which will provide a spectrum of understanding, from general to specific, for the underlying data.

In this thesis, we first introduce the conceptual hierarchy, a hierarchical organization of the data in the databases. Two algorithms for dynamic adjustment of conceptual hierarchies are developed, as well as another algorithm for automatic generation of conceptual hierarchies for numerical attributes. In addition, a set of algorithms is developed for mining multiple-level characteristic, discriminant and association rules. All algorithms developed were implemented and tested in our data mining prototype system, DBMiner. The attribute-oriented induction method is extended to discover multiple-level characteristic and discriminant rules. A progressive deepening method is proposed for mining multiple-level association rules. Several variants of the method with different optimization techniques are implemented and tested. The results show the method is efficient and effective. Furthermore, a new approach to association rule mining, *meta-rule guided mining*, is proposed. The experiments show that meta-rule guided mining is powerful and efficient. Finally, an application of data mining techniques, cooperative query answering using multiple layered databases, is presented.

Our study concludes that mining knowledge at multiple levels is both practical and desirable, and thus is an interesting research direction. Some future research problems are also discussed.

Dedication

To my parents and my wife.

Acknowledgements

I would like to thank Prof. Jiawei Han, my senior supervisor, for his continuous help, encouragement, and support, during my study. Prof. Han always find time in his busy schedule for frequent discussions with me and his creative thinking and insight make our discussions fruitful and interesting. My endeavors would not have been successful without him.

I would also like to thank Prof. Veronica Dahl for serving on my supervisory committee. Prof. Dahl gave me good advice on my research.

My deepest thanks to Prof. Tiko Kameda and Prof. Len Shapiro for serving as examiners of this thesis.

I wish to express my gratitude to many people in the School of Computing Science. Prof. Lou Hafer, Prof. Arvind Gupta, and Prof. Qiang Yang provided various kinds of help when they were most needed. Mrs. Kersti Jaager and other secretaries were always available for help.

Micheline Kamber read this thesis and gave many useful comments. The discussions with Prof. David Cheung of Hong Kong University helped me understand more about the mining of association rules. My thanks also go to many fellow graduate students who made my days at SFU enjoyable: Krzysztof (Kris) Koperski, Wei Wang, Osmar Zaiane, Andrew Fall, Jiashua Liu, Tong Lu, Jie Wei, Yao Liang, Martin Vorbeck, Jenny Chiang, Hui Li, Wan Gong, Yijun Lu, Nebojsa Stefanovic, Betty Xia, Hongshen Chin, and Ye Lu.

I am very grateful to my wife, Lei Jiang, who always energizes me with her love, understanding, and support. I am also indebted to my parents for their everlasting understanding and encouragement.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

With the rapid growth in size and number of available databases in commercial, industrial, administrative and other applications, it is necessary and interesting to examine how to extract knowledge automatically from huge amounts of data [33, 72, 36]. For example, the Wal-Mart databases collect 20 million transactions every day.

Knowledge Discovery in Databases (KDD), or data mining, is the effort to understand, analyze, and eventually make use of the huge volume of data available. Through the extraction of knowledge in databases, large databases will serve as a rich, reliable source for knowledge generation and verification, and the discovered knowledge can be applied to information management, query processing, decision making, process control and many other applications. Therefore, **data mining** has been considered as one of the most important research topics in databases by many database researchers [97, 96].

Knowledge Discovery in Databases (KDD) is defined as *the nontrivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data* by Fayyad et al. [86] In their opinion, there are usually several steps in a KDD process: data selection, preprocessing, transformation, data mining, and interpretation/evaluation of the results [86], as shown in Figure 1.1. *Data mining* is only one step of the process, involving the application of discovery tools to find interesting patterns from targeted data. However, since data mining is the central part of the KDD process, the term *data mining* and the term *knowledge discovery in databases*

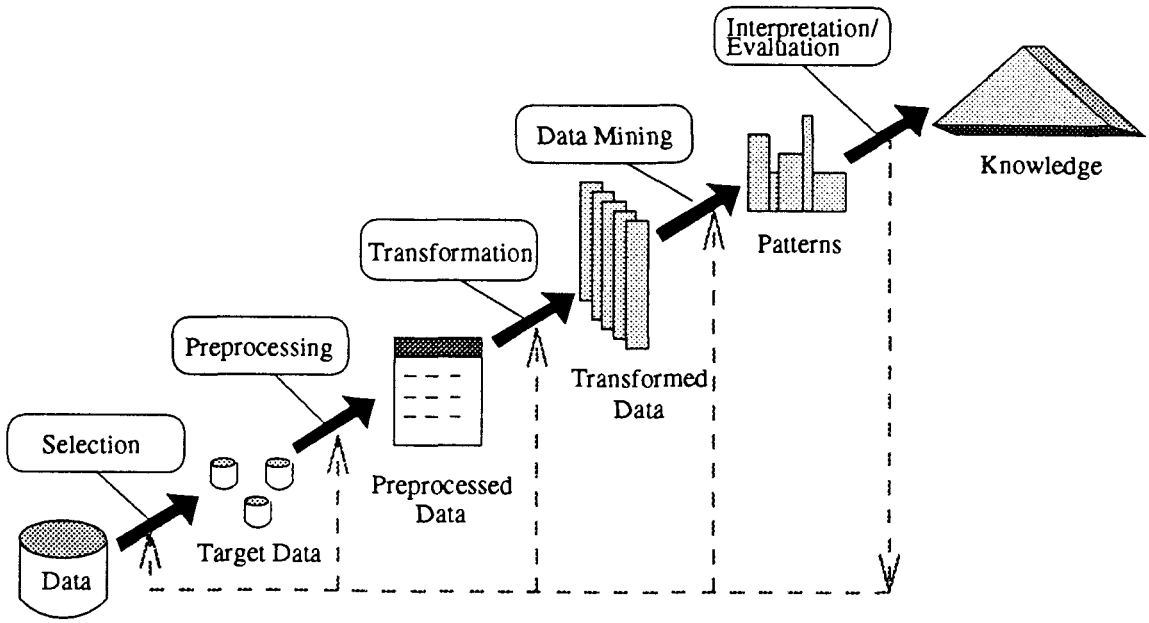

Figure 1.1: Steps of the KDD process (Fayyad et al. 1996).

have been used interchangeably by many researchers[97, 3, 50, 81, 59]. In this thesis, *data mining* and *knowledge discovery in databases* are used without distinction.

The definition of KDD in [86] gives the basic characteristics of the knowledge discovery process, but there are some points we would like to emphasize. First, data mining deals with a large amount of data, which makes the efficiencies and scale-up abilities of data mining algorithms a very important issue. Second, the data is usually stored in databases, and mature technologies of database management (such as data storage, indexing, query optimization, etc.) should be employed to deal with low end data processing. Finally, users of data mining systems are typically looking for something interesting. Their interests can determine their judgement, regarding the usefulness and novelty of the discovered knowledge, for example.

Based on the above analysis, we give our definition of data mining.

Definition 1.0.1 Data Mining (or Knowledge Discovery in Databases) is the extraction of *interesting* patterns in large databases.

Figure 1.1 gives a very good overview of the data flow in a KDD process. The

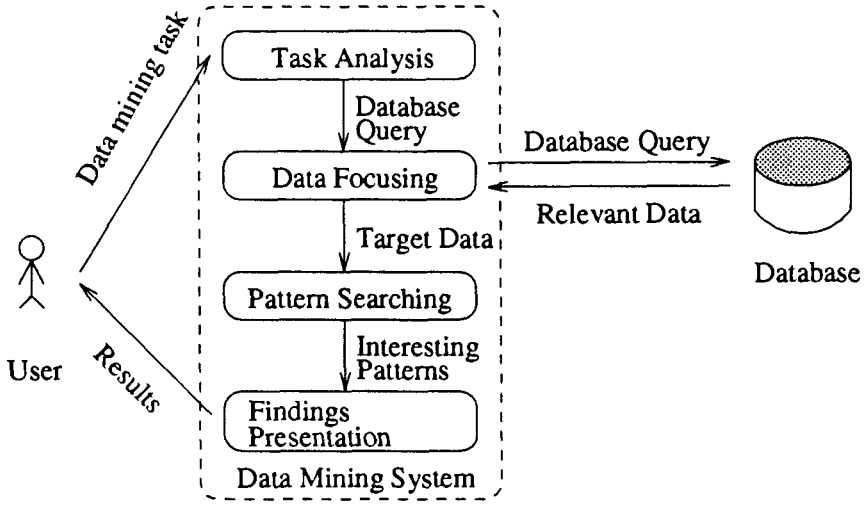

Figure 1.2: Control flow of the data mining process.

control flow of the data mining process is shown in Figure 1.2. A data mining session is usually an interactive process of data mining query submission, task analysis, data collection from the database, interesting pattern search, and findings presentation.

1.1 Data Mining Tasks

There have been many interesting studies on knowledge discovery in databases [33, 72, 87]. These studies cover a wide variety of data mining tasks and use different methodologies. The most common types of data mining tasks, classified based on the kind of knowledge they are looking for, are listed as follows. A survey of different methodological approaches to KDD, including machine learning, database-oriented, statistics, etc., is given in Chapter 2.

- Characterization is the summarization or abstraction of a set of task-relevant data into a relation, called *generalized relation*, which can then be used for extraction of characteristic rules. The characteristic rules present the characteristics of the data set, called the target class, and can be at multiple conceptual levels and viewed from different angles. For example, the symptoms of a specific disease can be summarized by a set of characteristic rules.

- Discrimination is the discovery of features or properties that distinguish the class being examined (*target class*) from other classes (called *contrasting classes*). A set of discriminant rules is discovered which summarize the features that distinguish the *target class* from *contrasting classes.* For example, to distinguish one disease from others, a discriminant rule summarizes the symptoms that differentiate this disease from others.

- Classification is labeling or categorizing of the data into a set of known classes. A set of training data (i.e., a set of objects whose class label is known) is given and analyzed, and a classification model is constructed based on the features of the data in the training set. A set of classification rules is generated from the classification model, which can be used to classify future data and develop a better understanding of each class in the database. For example, classification rules about diseases can be extracted from known cases (training set) and used to diagnose new patients based on their symptoms.

- Association rule mining is the discovery of associations or connections among objects. An association rule is in the form of "$A_1 \wedge \cdots \wedge A_i \rightarrow B_1 \wedge \cdots \wedge B_j$" which means objects $B_1, \cdots, B_j$ tend to appear with objects $A_1, \cdots, A_i$ in the target data. Association rules at multiple conceptual levels will reveal such kind of association in the relevant set(s) of data in a database. For example, one may discover that a set of symptoms often occur together with another set of symptoms, and then further study the reasons behind this association.

- Clustering is the identification of classes (clusters) for a set of unclassified objects based on their attributes. The objected are so clustered that the intraclass similarities are maximized and the interclass similarities are minimized based on some criteria. Once the clusters are decided, the objects are labeled with their corresponding clusters, and common features of the objects in a cluster are summarized to form the class description. For example, a set of new diseases can be grouped into several categories based on the similarities in their symptoms, and the common symptoms of the diseases in a category can be used to describe

that group of diseases.

- Prediction is the estimation or forecast of the possible values of some missing data or the value distribution of certain attribute(s) in a set of objects. This involves finding the set of attributes relevant to the attribute of interest (by some statistical analysis) and predicting the value distribution based on a set of data similar to the selected object(s). For example, an employee's potential salary can be predicted based on the salary distribution of similar employees in the company.

- Evolution mining is the detection and evaluation of data evolution regularities for certain objects whose behavior changes over time. This may include characterization, classification, association, or clustering of time-related data. For example, one may find the general characteristics of the companies whose stock price has gone up over 20% last year, or evaluate the trend or particular growth patterns of high-tech stocks.

- Deviation mining is the discovery and evaluation of the deviation patterns of objects in the target data in a time-related database. The expected behavior or norm of the objects is usually given by the user or computed based on some assumption, such as average, linear growth, etc. For example, one may discover and evaluate a set of stocks whose behavior deviates from the trend of the majority of stocks during a certain period of time.

1.2 Motivation

Previous research on data mining focused on discovery of knowledge at a single conceptual level, either primitive or general. The knowledge is said to be at a primitive level if the patterns involve only the raw data stored in databases. The knowledge is said to be at a general level if the patterns involve higher level concepts, usually abstraction or generalization of some primitive level concepts.

Most previous research focused on finding knowledge at the primitive level, i.e.,

rules, patterns, and regularities among the raw data stored in databases [4, 5, 15, 32, 56, 73, 85, 82, 89, 94, 115].

Example 1.2.1 The following rule from Shen et al. [94] is a primitive level rule discovered in a chemical database.

IF ingredients = "BX89" & "GF102" AND property = P_1
THEN Clusters(P_1) = {(2.4, 3.5, 0.97), (202.3, 0.5, 0.03)}

The rules says that if a compound has ingredients "BX89" and "GF102", its property P_1 falls into a cluster with a mean of 2.4 and a variance of 3.5 with 97% probability, and falls in the cluster with a mean of 202.3 and a variance of 0.5 with 3% probability. □

Han and his associates proposed the discovery of rules at the general conceptual level [48, 46]. A general level rule is a rule whose concepts (constants) can be at either nonprimitive level (the *abstract* concepts or values not in the database), or primitive level (the data stored in the database).

Example 1.2.2 The following rule is a general rule from Han et al. [46] which may be discovered in a personnel database.

IF Position is "professor" AND Department is "Applied Science"
THEN Sex is "male" AND Age is "old" AND
Birth_place is "Canada" AND Salary is "high". [0.20]

The number in the square brackets is the confidence of the rule which says that 20% of the Applied Science professors are old, male, Canadian-born, and with high salary. The rule is a general rule since it contains some non-primitive concepts like "high" and "old" which are not stored in the database. □

However, there are many cases where knowledge or rules at multiple conceptual levels, i.e., *multiple-level* knowledge or rules, are desired. Several possible scenarios are listed below, from the perspectives of users' interests, data mining processing, and discovered results, respectively.

- Users' interests may vary from person to person. For example, an executive may want to have a general view or summary of the sales of all products while a sales representative may like to see the detailed information about a particular product.

- A data mining session usually involves many interactions of "zoom-in" and "zoom-out", i.e., specialization to see details and generalization to see summaries. For example, a sales manager may browse the overall sales, then look into each product's sales.

- Strong rules are more likely to exist at high conceptual levels but may likely repeat common knowledge. For example, the strong high level association rule "80% of customers who buy milk also buy bread" may be known to store managers. On the other hand, primitive level rules may be more interesting, but are hard to find. For example, the primitive level association rule "40% of customers who buy 2% Dairyland milk also buy Oldmill whole-wheat bread" is difficult to find and could be mixed with many uninteresting rules.

It is natural to ask if we can discover both primitive level and high conceptual level rules from databases at the same time, in other words, find multiple-level rules. In this thesis, we address the issue of discovery of multiple-level rules from large databases.

1.3 Problem Specification

The problem of finding single level rules in databases has been studied extensively. However, to the best of our knowledge, the problem of discovery of multiple-level rules has not been thoroughly addressed. There are several questions that need to be answered. For example, how do we define the levels? Are the levels static or dynamic? How can we discover multiple-level rules efficiently? Can we somehow benefit from previous research results?

Before we formally define multiple-level rules, we introduce the term "conceptual hierarchy" which, roughly speaking, is a taxonomic organization for *concepts* or *objects* in a database. Conceptual hierarchies define the levels of concepts and provide

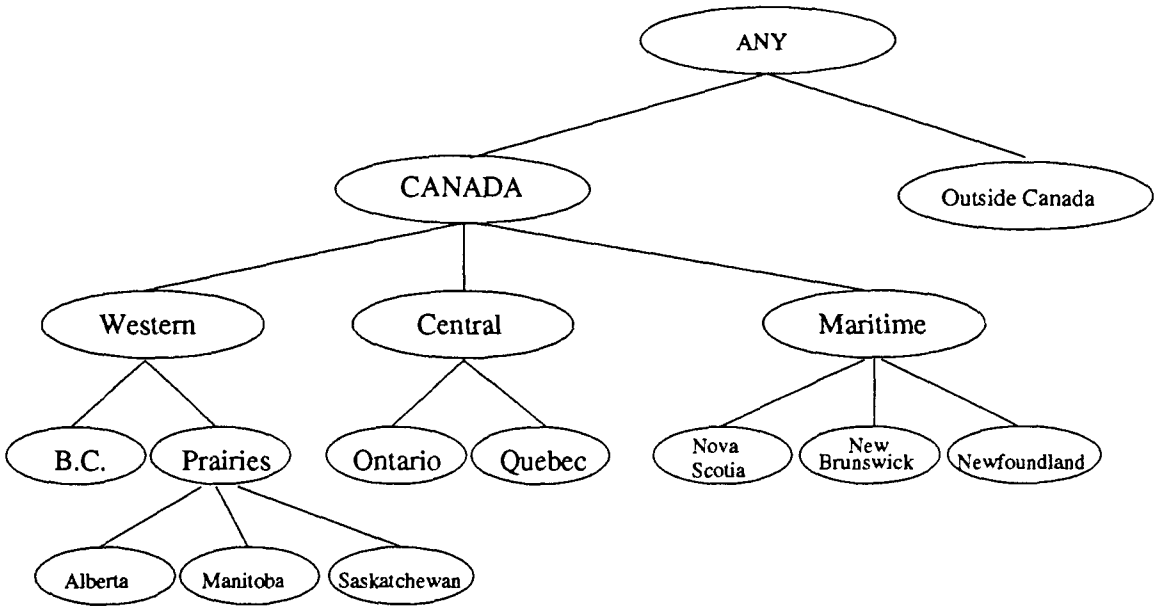

Figure 1.3: A conceptual hierarchy for the provinces of Canada.

background information for data mining. We will discuss conceptual hierarchies in detail in Chapter 3 as well as algorithms for dynamic adjustment and automatic generation of conceptual hierarchies. An example of conceptual hierarchy is shown in Figure 1.3 for the provinces of Canada.

The rules found by most data mining algorithms are **production rules** in the following form:

$$\text{IF } P_1 \wedge P_2 \cdots \wedge P_n \text{ THEN } Q_1 \wedge Q_2 \cdots \wedge Q_m$$

also written as $$P_1 \wedge P_2 \cdots \wedge P_n \rightarrow Q_1 \wedge Q_2 \cdots \wedge Q_m$$

where $P_i (i = 1, \ldots, n)$ and $Q_j (j = 1, \ldots, m)$ are predicates. A predicate can be a user-defined predicate, or an assertion of the form $A \otimes c$, where A is an attribute in the database, $\otimes$ is a binary operator, usually a comparison, and c is a constant. We will follow the traditional denotation of the *rule* in this thesis. Logic terms will have their common meanings, such as in [41], if not defined otherwise.

Definition 1.3.1 Multiple-level rules are rules in which the concepts or constants may be at multiple conceptual levels in conceptual hierarchies.

Our problem is to discover the rules at different conceptual levels, i.e., multiple-level rules, given a large database together with conceptual hierarchies about the data. Most rule discovery methods find primitive level rules which only involve concepts at the primitive level. Some studies find rules at non-primitive levels, or general rules [48]. We extend the previous studies [45, 48, 4, 94] by finding rules at different levels (as opposed to rules at one specific level).

There are many kinds of rules existing in a large database. It is practically impossible to find *all* kinds of rules which can be induced from the database. The most common kinds of rules that are interesting to data mining users are discussed in Section 1.1. We confine our study to the discovery of three types of common rules: characteristic rules, discriminant rule, and association rules. Discovery of some other kinds of multiple-level rules is discussed in Section 8.3.

There has been a great deal of research on the discovery of single-level rules. Although previous results cannot be applied directly for the discovery of multiple-level rules, they provide some useful hints. In this thesis, some previous methods for mining single-level rules are extended to discover multiple-level rules. For example, the attribute-oriented induction method [13, 45] for discovery of single-level characteristic and discriminant rules is extended to find multiple-level characteristic and discriminant rules.

1.4 The DBMiner System

A data mining prototype system, DBMiner, has been developed along with our research. The system serves several purposes:

- Our algorithms are implemented in the system and tested against real databases.
- The application of the system to real databases raises more interesting research problems and helps us to revise our original designs of the algorithms.
- As a data mining tool, DBMiner can be employed by database users to discover interesting knowledge in their databases, and to ultimately benefit society.

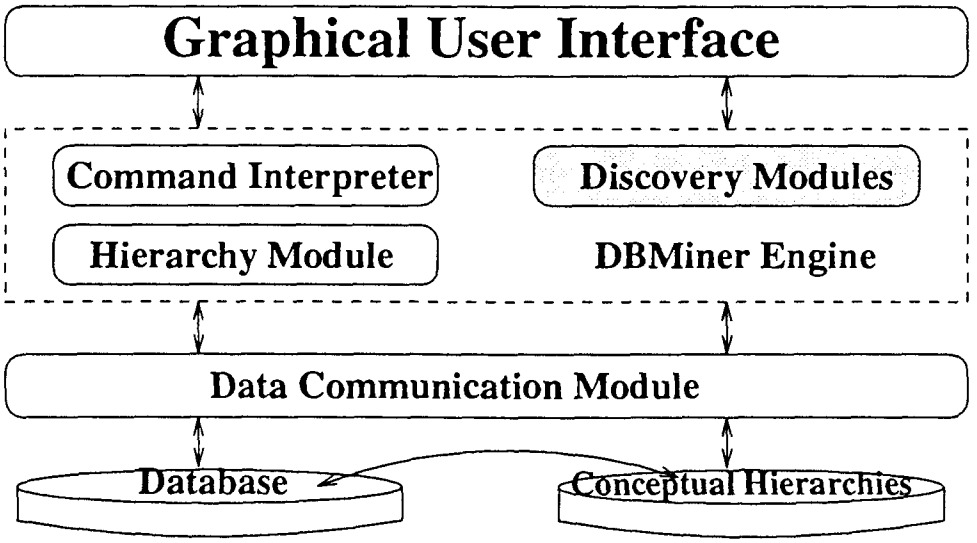

Figure 1.4: Architecture of the DBMiner system.

We briefly introduce the architecture of DBMiner and its features in the following sections. Detailed descriptions of DBMiner can be found in its user's manual [42].

1.4.1 Architecture of DBMiner

DBMiner is composed of three parts: Graphical User Interface (GUI), DBMiner Engine, and Data Communication Module, as shown in Figure 1.4. DBMiner has two versions, the Unix (SunOS) version running on Sun SparcStation and the Windows/NT version running on x86 compatible PCs. Both versions have the same architecture.

We briefly explain the functions of each part, and their implementations.

- The Graphical User Interface (GUI) of DBMiner communicates interactively with users for specifying data mining task, setting control parameters, and displaying results. The GUI of the UNIX version is implemented using XView. Visual C++ is used for the GUI of the Windows/NT version.

- The engine of DBMiner is the core component of the system. Written in ANSI C, the engine is platform independent and transportable between the Unix version and the Windows/NT version. Therefore, both versions have the same

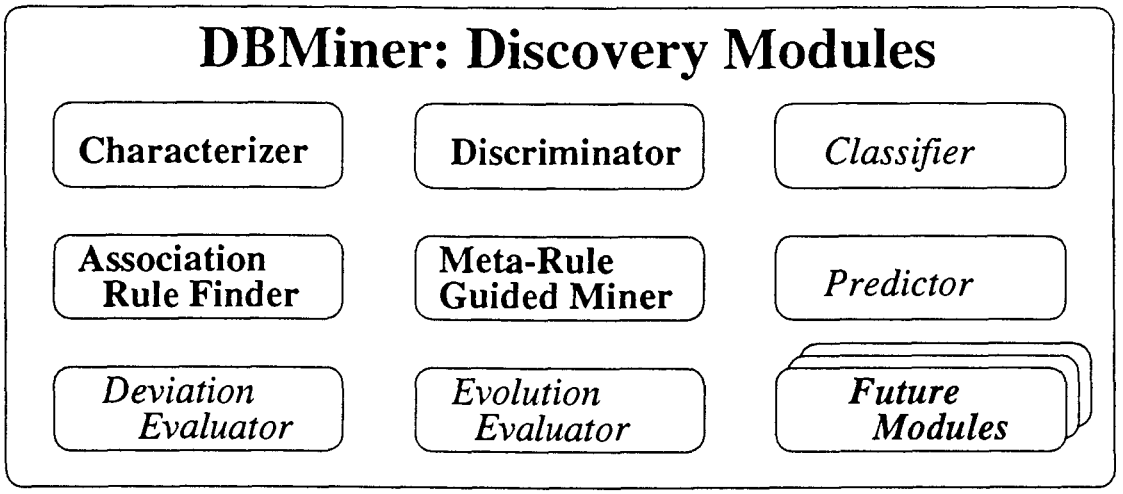

Figure 1.5: Knowledge discovery modules of DBMiner.

engine. The engine contains functional modules of DBMiner, including a data mining query parser, a conceptual hierarchy module, and the discovery modules explained in the next paragraph.

- The data communication component of DBMiner handles data transmissions between the engine and the database server (SQL server). We use Sybase as the database server on Unix and Microsoft SQL server on Windows/NT.

The discovery modules of DBMiner, shown in Figure 1.5, include characterizer (finding characteristic rules), discriminator (finding discriminant rules), classifier (finding classification rules), association rule finder, meta-rule guided miner, predictor (finding prediction rules), evolution evaluator, deviation evaluator, and some planned future modules. The modules in italic have been or are being implemented by other researchers in the DBMiner group.

Note that the DBMiner system has two implementations using different data structures: the relational table implementation and the data cube (multidimensional database) implementation. The author of this thesis implemented the relational table version and helped to implement the data cube version. Of the relational table version, the GUI on Windows/NT and several discovery modules, as mentioned above, have been implemented by other researchers.

1.4.2 Features of DBMiner

The DBMiner system has several interesting features.

- The system is built on top of database management systems (DBMSs), but is independent of the DBMSs. Many database servers can be used by DBMiner by simply providing the data communication primitives in the data communication module.
- DBMiner discovers several kinds of rules: characteristic rules, discriminant rules, association rules, classification rules, prediction rules, etc. This thesis will only discuss the modules I have designed and implemented: the characterizer, the discriminator, the association rule finder, and the meta-rule guided miner.
- A data mining language, Data Mining Query Language (DMQL), is proposed for the uniform specification of data mining tasks. The DMQL query is parsed by a parser written in YACC and LEX and the results are sent to other modules. Several examples of DMQL queries will be presented when we discuss the discovery of different kinds of rules. A full specification of DMQL can be found in [51].
- Conceptual hierarchies can be dynamically adjusted and interactively revised. For numerical attributes, conceptual hierarchies can be generated automatically.

1.4.3 Testing Databases

Several real databases are used in our experiments with DBMiner. In this thesis, the NSERC research grant database, which contains the information about the research grants awarded by the NSERC (*Natural Sciences and Engineering Research Council of Canada*) in each year, is used to report our results. Other databases are also used, including several databases provided by MPR Teltech Ltd., but not reported because of the confidential nature of the data. Nevertheless, the performances of the algorithms are consistent regardless of the data sets.

The NSERC databases of 1991, 1994, 1995, and 1996 are used. Because they have the similar structure and almost the same size (about 10% variance), we will just use one of them to present the results. The average size of NSERC databases is 12MB. There are 8 – 10 tables in NSERC databases. The largest table has 10,000 – 11,000 tuples and 14 attributes (6 categorical, 8 numerical).

On the Unix system, the databases are stored in a Sybase Server running on a Sun SparcStations 5 with 32MB memory, connected to workstations through Ethernet. Indices are built on the primary keys. On the Windows/NT system, the databases are stored in a local SQL server and no index is used.

1.5 Outline of the Thesis

The remainder of this thesis is organized as follows. In Chapter 2, a brief survey of related work in data mining is given. Conceptual hierarchies are introduced in Chapter 3, along with some algorithms developed for manipulation of conceptual hierarchies. In Chapter 4, methods for the discovery of multiple-level characteristic and discriminant rules are investigated. A set of algorithms for the discovery of multiple-level characteristic and discriminant rules is proposed, implemented, and tested. The discovery of multiple-level association rules is proposed in Chapter 5, and a progressive deepening method is presented with some experimental results. In Chapter 6, a meta-rule guided mining method is proposed for mining multiple-level association rules. Chapter 7 shows the application of data mining techniques in cooperative query answering. The study is concluded in Chapter 8, which also presents some future research problems. More detailed descriptions of the chapters are given as follows.

In Chapter 2, we briefly survey the related work in data mining. Different approaches to data mining, including machine learning, statistics, and many others, are discussed. Some representative data mining systems are also introduced.

In Chapter 3, the formal definition of conceptual hierarchy is given. Some related issues, such as motivations for using conceptual hierarchies, availability of conceptual

hierarchies, and specifications of conceptual hierarchies, are also discussed. Two algorithms for conceptual hierarchy adjustment are presented and examined. Another algorithm for the generation of conceptual hierarchies for numerical attributes is also proposed and tested.

In Chapter 4, the Attribute-Oriented Induction (AOI) method, introduced by Han et al. [44], and enhanced by Han and Fu [48], for the discovery of general level rules, is extended to discover multiple-level characteristic and discriminant rules. A set of algorithms for the discovery of multiple-level characteristic and discriminant rules is proposed. The algorithms are implemented and tested, and some results are presented.

The concept of multiple-level association rule is introduced in Chapter 5. A progressive deepening method for mining multiple level association rules is proposed. Several variants of the method are implemented and tested and the results, both on synthetic databases and real databases, are reported.

In Chapter 6, we propose a new approach — meta-rule guided mining — for mining multiple-level association rules. A meta-rule is a rule template which specifies the format of concrete rules to search for. Two algorithms for mining single-variable rules are presented and tested, and their relative performances are compared. The algorithms can also be adopted for mining multiple-variable rules with minor modification.

There are many possible applications of data mining techniques. We discuss the application of data mining techniques in cooperative query answering in Chapter 7. A multiple layered database (MLDB) model is proposed and examined in Chapter 7. An MLDB can be constructed using data mining techniques and used for cooperative query answering.

We conclude our study in Chapter 8. My major thesis work is summarized and the conclusions from our study are given. Some possible future research problems are also discussed.

Chapter 2

Related Work in KDD

There has been a great deal of research in KDD, especially in recent years. In this chapter, we will discuss the different approaches to KDD as well as some typical data mining systems. Researchers in KDD come from different backgrounds and take different approaches. Based on the basic methodology used by researchers, studies on KDD are classified into five categories: mathematical and statistical approaches, machine learning approaches, database-oriented approaches, integrated approaches, and other approaches:

- Mathematical and statistical approaches.
  Usually a mathematical or statistical model is built, and then rules, patterns, and regularities are drawn from the model. For example, a Bayesian network can be constructed from the given training data set and the implications among objects can be extracted from the parameters and linkages of the network.
- Machine learning approaches.
  A cognitive model is used by most machine learning approaches to resemble the human learning process. For example, in the learning from examples paradigm, a set of positive examples (members of the target class) and a set of negative examples (nonmembers of the class) are given, and a concept which best describes the class is learned or discovered through intelligent search in the concept space.

- Database-oriented approaches.
  Database technologies and database-specific heuristics are used to exploit the characteristics of the data in hand. For example, transactional databases are scanned iteratively to discover patterns in customer shopping practices.
- Integrated approaches.
  Several methods are integrated into a unified framework to exploit the advantages of different approaches. For example, induction from machine learning can be integrated with deduction from logical programming or deductive databases, in which the former searches for patterns in the objects collected by the latter, while the latter verifies the patterns found by the former.
- Other approaches.
  Other approaches include visual exploration, neural networks, knowledge representation and so on. Since there is relatively less research from these approaches for KDD, they are put into one category. Nevertheless, they are interesting studies and could be important to KDD.

Each category may be divided into subclasses as described in Section 2.1.

2.1 Approaches to KDD

Different approaches to KDD in each of the five categories will be discussed in this section by surveying the representative work in each category.

2.1.1 Mathematical and Statistical Approaches

Statistical Approaches

Statistics has been an important tool for data analysis for a long time. Bayesian inference is the most extensively studied statistical method for knowledge discovery.

A Bayesian classification method, AUTOCLASS was developed by Cheeseman et al. [16, 14]. Given a set of objects (evidences), $E = \{E_1, \cdots, E_I\}$, with unknown classes and simple values (logical, integer, or real), AUTOCLASS tries to find clusters

of the objects to represent the classes so that it can assign each new object to a class. This kind of problem is usually referred to as unsupervised learning or clustering.

A Bayesian classification model is specified by two sets of parameters: 1) a set of discrete parameters, T, such as correlationships among attributes, and the number of classes, which describes the general form of the model; and 2) a set of continuous parameters, V, the variables in the general forms specified by T, determining the specific model within the description of T.

In the search space S defined by all available T and possible V, AUTOCLASS searches for the pair (T,V) which maximizes the joint belief $J(EVT|S)$, i.e., the most probable (T,V) which classifies E. The problem is decomposed into two subproblems.

1. From a set of possible Ts, which have different attribute dependencies and class structures (flat or hierarchical), AUTOCLASS searches for a most probable T regardless of V, i.e., the T which maximizes $P(T|ES)$. This is done by approximating $P(T|ES)$ using a probability density function (p.d.f.) of T which can be computed from the p.d.f of each attribute. AUTOCLASS assumes Bernoulli distributions for discrete attributes and normal distributions for continuous attributes.
2. For a given T, AUTOCLASS searches for a most probable V by breaking continuous V into regions and finding the region, R, which maximizes the *marginal joint*:

$$M(ETR|S) = \int_{V \in R} dP(EVT|S)$$

Several local maxima of R will be reported.

AUTOCLASS can find hierarchical clusterings and the local maxima are usually sufficient. However, it deals only with simple values.

The use of Bayesian networks (Directed Acyclic Graph or DAG) for the discovery of causal relationships among objects is proposed for KDD by Buntine [11, 12], Spirtes et al. [103], and Hackerman et al. [54, 55]. Nodes in a Bayesian network represent variables or states, and arcs represent the dependencies between nodes, directed from the cause to the effect. Figure 2.1 gives a very simple Bayesian network for medical problems [11].

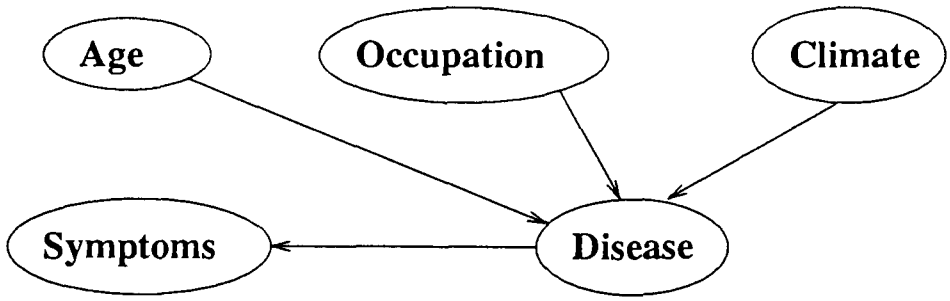

Figure 2.1: A simple Bayesian network.

There are usually three steps in constructing a Bayesian network. They involve:

- Deciding which variables to be modeled as nodes.
- Determining the structure of the DAG, for example, the connections from a node to other nodes.
- Estimating the parameters of the DAG, i.e., the dependencies among the nodes in terms of probability distributions.

Usually the first step is performed by user or domain expert. Buntine and Hackerman et al. used Bayesian metrics, such as maximum a posteriori probability, and heuristic search to find the structure and parameters [12, 54]. Starting from a given or randomly generated network, the algorithms search for a better network (based on the metrics) until a local optimum is found. Spirtes et al. [103] used conditional independence tests in their TETRED system to find the optimal network.

Bayesian networks are powerful tools for analyzing causal relationships in databases. However, the complexity of the model grows exponentially with the number of the nodes in a network. Usually, a local optimum is found, whose goodness depends highly on the heuristics and the application domain.

To summarize, statistical approaches have a solid theoretical foundation, namely the Bayesian distribution theorem. They perform well for quantitative data and are robust with noise. However, almost all of them depend on some statistical assumptions which usually do not hold in real world data. Moreover, results from statistical methods can be difficult for nonexperts in statistics to understand.

Rough Sets

Rough sets were first introduced by Pawlak [83]. They were used for knowledge discovery by Ziarko [115] as a tool to find dependencies in data and to derive decision tables. A decision table lists the conditions and the classes based on these conditions. For example, the following decision table tells how the size, transmission type, and weight of a car determines its gas mileage (class). The blank cells mean "doesn't-matter".

size	transmission	weight	mileage
medium	auto		medium
small			high
large		light	medium
	auto	medium	medium
		heavy	low
	manual	medium	high

Given a set of objects, E, called *elementary objects*, the lower approximation of a set X, $\underline{IND}(E,X)$ is the union of all elementary objects fully subsumed by X:

$$\underline{IND}(E,X) = \bigcup\{Y \in E \wedge X \supseteq Y\}$$

as illustrated in Figure 2.2, in which each basic box is an elementary object, set X is the area in the dashed line, and $\underline{IND}(X)$ is the dark area.

Let U be the set of all objects. Each object is a tuple that has values for a set of attributes A, consisting of determining attributes P, whose values determine the object's class, and determined attributes Q, the class attributes which label the object's class.

Ziarko defines the equivalence relation, $IND(X)$, of any set of attributes $X \subset P \cup Q$ as follows:

$$IND(X) = \{(a,b)|a,b \in U \wedge x(a) = x(b)\}$$

where $x(a)$ is the value of a on an attribute $x \in X$. $IND(X)$ gives a classification of objects in U based on their values of attributes in X. All classes in $Q' = IND(Q)$

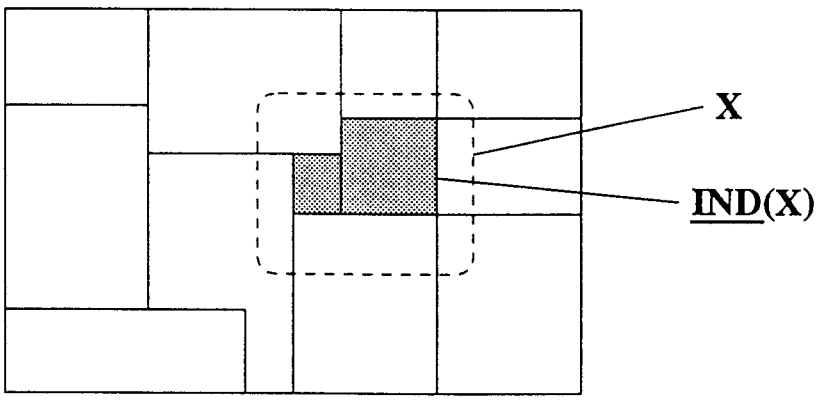

Figure 2.2: Lower approximation of a set.

can be approximated by the classes in $IND(X)$:

$$POS(X,Q) = \bigcup\{\underline{IND}(IND(X),Y)|Y \in Q'\}$$

where $\underline{IND}(IND(X),Y)$ is the lower approximation of the set Y in terms of the elementary objects — the classes in $IND(X)$. $POS(X,Q)$ gives the dependency between X and Q because the values of the attributes in Q of any object in $POS(X,Q)$ can be determined solely by the object's values of the attributes in X. A heuristic search method is used to find the minimal set $P' \in P$ which still keeps the dependency, i.e., $POS(P',Q) = POS(P,Q)$. Decision table are then derived from the minimal set. Decision rules (or classification rules) can be obtained from the decision table.

Hu and Cercone integrated rough sets with attribute-oriented induction to find high level rules [57]. A set of objects are first generalized using attribute-oriented induction [45]. The rough sets method is then applied on these generalized objects to find the decision table at a general level.

Rough sets provide a tool for KDD with a solid mathematical foundation. However, it can only discover qualitative rules, i.e., exact rules. The computational complexity of finding the best minimal set is exponential to the number of attributes.

2.1.2 Machine Learning Approaches

Learning from Examples

Given a set of positive examples (objects in the class) and a set of negative examples (objects not in the class), this approach searches for a class description (concept) that covers all positive examples and excludes all negative examples, i.e., *learning from examples*. The search space is defined by the attributes of the objects.

The AQ algorithm, proposed by Michalski [75], uses a bottom-up generalization method to search for the description, as is summarized as follows:

1. Randomly select a positive example;
2. Form a *star* which is a generalization of the example by dropping conditions, adding selectors, extending intervals, climbing conceptual trees, etc., but which does not cover any negative example.
3. Form a description for the star and remove the positive examples covered by it.
4. Continue the process until all positive example are accounted for.
5. Form the disjunction of the descriptions as the resulting class description.

The results of each iteration are examined by users and the process stops if a satisfactory description is found. Heuristics and background knowledge can be used to guide the search in Step 2.

Mitchell [79, 80] proposed a combined top-down and bottom-up approach to search for the best description. The algorithm searches for the best description in a *version space*, defined by all descriptions. From the top, a most general description is given and specialized against the negative examples. From the bottom, positive examples are generalized by similar techniques used in the AQ algorithm. When the two descriptions meet, a correct concept description is found.

Methods of learning from examples imitate the human learning process. They are better suited to handle qualitative or categorical data rather than quantitative or numerical data. Most learning from examples methods assume that the data set can be fit in main memory so that many scans of the data set are possible.

Conceptual Clustering

An interesting method for clustering of objects, *conceptual clustering*, was first introduced by Michalski and Stepp [76]. Given a set of objects, conceptual clustering finds clusters based on conceptual closeness among objects. An object is an n-ary tuple represented by a vector of values on a set of attributes. A cluster is a conjunction of predicates on the attributes and values. In their algorithm, Cluster/2, a small number of random objects, called seeds, are selected as representatives of each cluster, and cluster descriptions are derived from the seeds. Other objects are put into clusters based on closeness. A hierarchical clustering can be built by splitting clusters into sub-clusters. The search for the best clusters, based on some clustering criteria, is stopped when a local maxima is found. Heuristics are used in selecting seeds for the next round. Clustering criteria may be based on: fitness, complexity, coverage, disjointedness, etc., where:

- Fitness is ratio of the number of objects in a cluster versus the space covered by the cluster.
- Complexity of the clustering is decided by the number of clusters and the number of predicates in each conjunction.
- Coverage is the number of objects covered by the clusters.
- Disjointedness measures the number of objects that are covered by more than one cluster.

A clustering tree is built by Fisher [34] in his COBWEB algorithm. Each node in the tree is a cluster and can be split into subclusters as children. Initially, the clustering tree has one node, the root. COBWEB incrementally adds objects into the clustering tree and adjusts the tree accordingly. COBWEB uses a measure, called *category utility*, which is the increase of the number of objects whose classes (clusters) can be correctly guessed given a clustering compared to that without the clustering. Based on this measure, one or more of the following operations takes place:

- Adding the object into an existing cluster.

- Creating a new cluster to accommodate the incoming object.
- Splitting a cluster into two.
- Merging two clusters into one.

Conceptual clustering generates more understandable clusters compared to statistical approaches, such as Bayesian classification. They perform well for categorical data, but are usually poor at numerical data. Like the methods of learning from examples, conceptual clustering also assumes that the data set can be held in main memory, which is often impossible for very large databases.

Decision Tree Induction

A decision tree is a tree whose nonleaf nodes are attributes of objects and whose leaf nodes are class labels. Branches from a parent node to its children are marked by the possible values of the corresponding attribute at the parent node. An object, represented by a vector of values on a set of attributes, is classified by tracing the path from the root of the tree to a leaf node, and taking the branches (arcs) according to its values along the way. The leaf node holds the class prediction of the object. A simple decision tree for the gas mileage of cars is given in Figure 2.3. For example, the mileage of a medium-sized car with automatic transmission can be determined as "medium" from the decision tree. Like the decision table in Section 2.1.1, the decision tree classifies the tuples, but uses a different knowledge representation form.

Given a set of objects whose class labels are known, a decision tree can be induced which classifies the objects. Classification rules can be extracted from the decision tree. For example, the following classification rule can be extracted from the decision tree shown in Figure 2.3:

IF size(x) = "medium" AND transmission(x) = "automatic"
THEN mileage(x) = medium.

Quinlan uses entropy to induce decision trees in his ID3 algorithm [88]. Starting from an empty tree and a set of objects, ID3 chooses the attribute which generates maximum information gain (calculated from entropies) as the root node. A branch

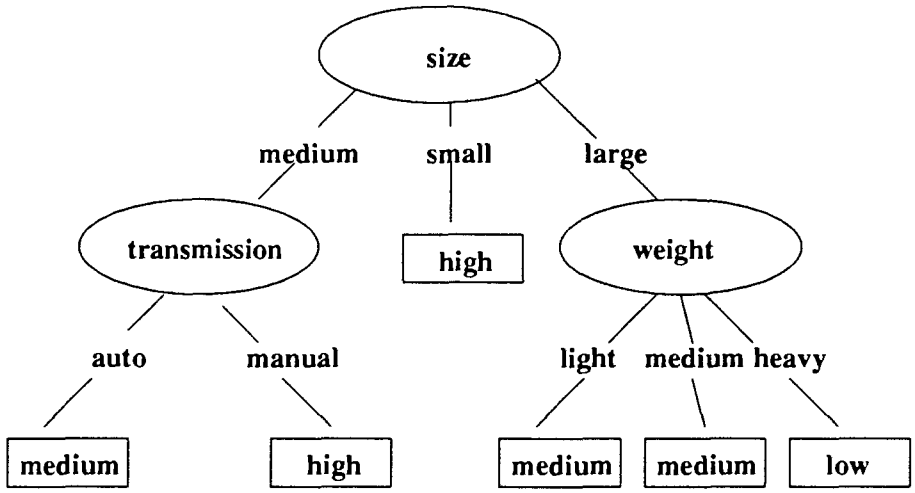

Figure 2.3: A simple decision tree.

connecting to a to-be-built subtree is created for each value of the attribute. The objects are partitioned into subtrees based on their values of the attribute. The process repeats for the subtrees until all objects in the subtree are from a single class.

Utgoff proposed an algorithm for the incremental update of the decision tree based on ID3 [107]. Cheng et al. proposed grouping some branches into one to improve the quality of the induced tree [17]. Smyth and Goodman [102] used a measurement call *J-measure* to induce classification rules directly from databases. Manago and Yodratoff induced decision trees from complex structured data [70]. Later enhancement of ID3 by Quinlan led to C4.5 which could extract compact classification rules and accommodate noise and missing data [91]. Uthurusamy et al. [108] and John [61] also discussed how to to deal with noisy or inconclusive data.

Decision tree induction is the most commonly used method for discovering classification rules. Decision trees provide a natural classification of the data, and thus are easy for human to understand. The tree induction process can usually generate a decision tree that is accurate and robust. However, the induced decision tree may be highly influenced by the bias in the training data. Sometimes the decision tree can grow too large which makes the derivation of classification rules from the decision tree computationally expensive.

To sum up, most machine learning methods have a very good cognitive model so

that the results are easy to understand for human. However, they usually assume the data set is small (a few hundred or a few thousand tuples) and can be fit into main memory.

2.1.3 Database-Oriented Approaches

An interesting method, *Attribute-Oriented Induction* (AOI), was developed by Han et al. [48, 46]. The method used data focusing and conceptual-hierarchy-based generalization to find high level rules from relational databases. AOI assumes that a set of conceptual hierarchies is available for the attributes. The basic techniques of AOI are given as follows.

1. Data focusing: The task relevant data are collected from the database. Usually, an SQL query is formed based on the mining request and sent to a DBMS.
2. Attribute generalization: If there is a large number of distinct values for an attribute, and there is a conceptual hierarchy for the attribute, the attribute can be generalized by conceptual hierarchy climbing, i.e., replacing the lower level concepts of the attribute by the corresponding higher level concepts.
3. Attribute removal: If there is a large number of distinct values for an attribute, and the attribute cannot be generalized or the higher level concepts of the attribute are stored in another attribute, the attribute should be dropped.
4. Count propagation: A new attribute, *count*, is attached to each tuple and is accumulated when merging equivalent tuples during generalization.
5. Attribute generalization control: Generalization on an attribute A_i is performed until the concepts in A_i has been generalized to a desired level, or the number of distinct values in A_i in the resulting relation is no greater than a prespecified or default *attribute threshold.*

Han et al. [46] discussed the discovery of characteristic and discriminant rules from relational databases using AOI. Han and Fu later [48] explored the use of AOI for the discovery of other kinds of rules, and in other kinds of databases, including object-oriented, deductive, and spatial databases.

The attribute-oriented induction method is efficient and effective for very large databases. AOI finds general, high level knowledge from databases, but leaves the detailed information out. It does, however, require domain knowledge in the form of conceptual hierarchies.

Agrawal et al. proposed an iterative searching method, *Apriori*, for mining association rules in transactional databases [2, 4]. Given a transactional database in which each transaction is a list of items, the *Apriori* algorithm finds association rules among items, such as "milk $\rightarrow$ bread" which says people who buy milk also buy bread. To find interesting association rules, two measures, *support* and *confidence*, were introduced. The *support* of a rule is the frequency of the item set (or *itemset*) composed by all the items in the rule, i.e., the probability a transaction contains the itemset. The confidence of a rule is the probability that a transaction contains the items in the right hand side of the rule when the transaction contains the left hand side items of the rule. The task is to find all association rules whose support and confidence are above the given support threshold and confidence threshold.

The Apriori algorithm [4] finds the association rules in two steps. An itemset is called a frequent itemset if its support is no less than the given support threshold. In the first step, Apriori finds *frequent* itemsets by iteratively scanning the database. In the second step, association rules are derived from the frequent itemsets and filtered out by the given confidence threshold.

The Apriori algorithm was later extended by Srikant and Agrawal [105] to discover association rules in relational tables. Each tuple in the table is transformed into a transaction by treating the attribute values as items. Numerical attributes are discretized in a way that the information loss is within the user given threshold.

Agrawal and Srikant used similar techniques to discover *sequential patterns* in a transactional database [5]. A sequential pattern is a series of items bought by a customer, for example, "TV followed by VCR followed by Video Camera". Sequential rules, such as "50% of people who buy TV buy VCR next within a year", can be extracted from the sequential patterns. They transformed the set of transactions into a set of customer shopping sequences. The Apriori algorithm can then be applied to the problem with little change.

The iterative search method used in Apriori is efficient and scales up well with respect to the size of the database. However, it was designed for the mining of association rules (or similar types), and can not be employed for mining other kinds of rules.

Database-oriented methods are very efficient and scale up well for large databases. They search for empirical patterns rather than models or theories, and therefore are robust and objective. However, database-oriented methods may rely on the underlying data model, which restricts their applications for general purpose discovery. Sometimes generality is compromised for efficiency as well.

2.1.4 Integrated Approaches

Shen et al. proposed using metaqueries as a way to integrate inductive learning methods and deductive database techniques [94]. A metaquery is a rule template, such as $P(X,Y) \wedge Q(Y,Z) \rightarrow R(X,Z)$, where P, Q, R, X, Y, Z are variables. A deductive database, LDL++, is used to collect data which are clustered using a Bayesian clustering method. Rules are extracted from the clustering and put into a knowledge base from which a user can form metaqueries.

Simoudis et al. introduced a framework which integrates induction and deduction [100]. A deductive database is used to manage *concepts*, the predicates on data, attributes, and relations. Users can query through the deductive database to verify their assumptions and define domain knowledge. The induction part searches for characteristic and discriminant rules in the databases by incrementally updating the existing rules.

A multistrategy approach was taken by Kaufman et al. [63]. Three sets of operators, one for data management, one for knowledge management, and one for knowledge discovery, are incorporated into a data and knowledge system, INLEN. The knowledge discovery tools include clustering, classification, characterization, discrimination, etc.

An integrated method may typically have the advantages of its component methods, but also inherit their disadvantages. How to make the best use of each method is the most important issue when several methods are integrated.

2.1.5 Other Approaches

Knowledge Representation Approaches

Gaines [40] proposed using exception directed acyclic graphs (EDAGs) to represent knowledge. An EDAG is a DAG in which nodes are premises (predicates) of rules. Some nodes have attached conclusions. As with decision trees, a path from a root to a leaf node in a EDAG gives the conclusions, usually class label(s), about the objects. In contrast to decision trees, EDAGs group common premises of rules into one node and allow some conclusions to be default conclusions.

Knowledge representations are important for KDD. However, methods of knowledge representations usually employ, explicitly or implicitly, a set of discovery tools. These tools are essential in order to take full advantage of the knowledge representations.

Visualization and Interactive Approaches

Most KDD systems involve some kind of human interactions. Zytkow and Baker showed how a scientific discovery method can be adapted to interactively mine regularities in databases [116]. Starting from the whole database, the algorithm, *FORTY-NINER*, searches for regularities among two or more attributes. The results are presented to the user who then can decide to partition the data, change parameters, and so on.

Keim et al. [64] had some interesting ideas on visualization of data. Data are transformed and presented in visually contrasting forms, such as graphs, icons, pictures, etc. Users can interactively select the interesting parts for further explorations.

Visualization of data helps to understand the data, however the choice of a proper visual form is important. Interactive approaches are suitable for data exploration, but may be too slow for large databases, and the discovered results may be incomplete.

Neural Network Approaches

Lu et al. proposed a neural network approach to data mining [69]. A neural network was built to classify a set of objects whose class labels are known. Classification rules are extracted from the neural network by discretization of the parameters.

Neural networks are accurate and robust for processing numerical data, but are poor at categorical data. Neural networks usually require a long training time, but run fast once trained. Like Bayesian inference methods, neural networks also suffer from the problem of understandability.

2.2 Typical KDD Systems

A few typical KDD systems are discussed in this section. We will focus on their structures and the data mining techniques used. More information can be found in the GTE lab's Knowledge Discovery Mine WWW page at `http://info.gte.com/~kdd/`. Several systems, such as AUTOCLASS, TETRED, ID3 and C4.5, and INLEN, etc., were introduced in Section 2.1 and will not be repeated here. DBMiner was discussed in Section 1.4.

2.2.1 The KEFIR System

KEFIR (KEy FIndings Reporter) was developed by Matheus et al. in the GTE Lab [74, 73]. The design and process flow of KEFIR are given in Figure 2.4. From a medical database, KEFIR finds the trends in the values of the attributes which deviate from the expected norm given by the experts. The interestingness of the findings are evaluated based on potential actions against the findings and benefits from the actions.

For example, KEFIR found that the "average length of stay" increased 22.6% from 1992 to 1993 and if this trend continued into 1994, it would result in $263,000 of extra expenses than the expected. KEFIR then explained that the increase was caused by the 247.9% increase of average length of stay in "Medical Nervous System". Some actions, such as enhancing "chronic care management", were recommended in order to

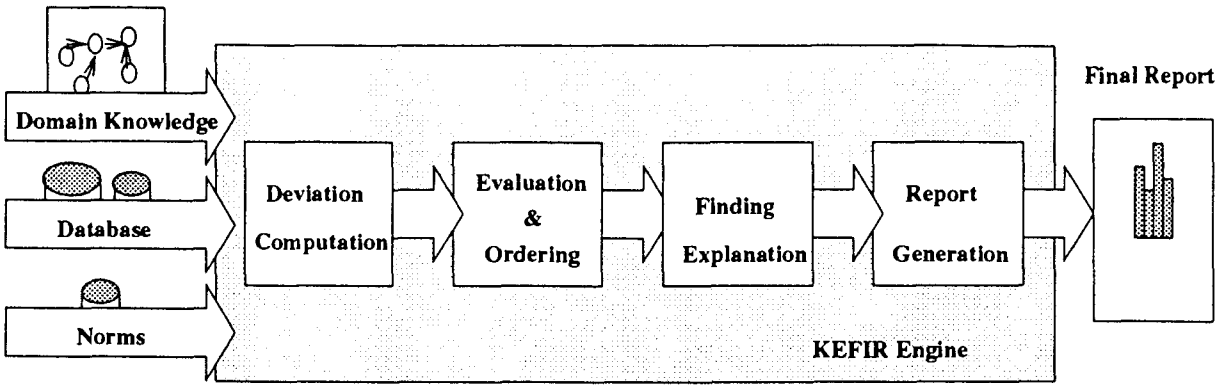

Figure 2.4: Design and process flow in KEFIR (Matheus et al. 1996).

bring the expense back to its expected value. A final report was output summarizing all the analysis, recommendations and projected savings [74].

2.2.2 The Quest System

Quest was developed by Agrawal et al. at IBM Almaden Research Center [3, 105, 4, 5]. It can find association rules, classification rules, time-series patterns, and sequential patterns from large databases. The architecture of Quest is given in Figure 2.5.

- Association rules are found using the Apriori algorithm. General level association rules can be found using taxonomy of items. This is the closest work to our multiple-level association rules. However, as explained in Chapter 5, there are a number of differences.
- Classification rules are extracted from a decision tree. The decision tree is induced using a technique, *pre-sorting of the values of attributes*, and pruned using Minimum Description Length (MDL) principle.
- Time-series patterns are discovered by searching for matching patterns in two sets of time-series data, for example, the stock prices of two companies.
- Sequential patterns are extracted using an Apriori-like algorithm by transforming transactional data into customer shopping sequence data.

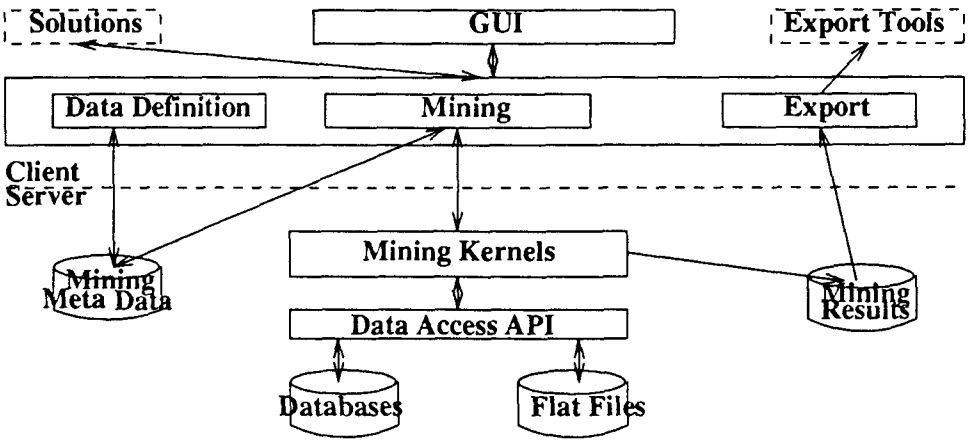

Figure 2.5: Architecture of Quest (Agrawal et al. 1996).

2.2.3 The SKICAT System

SKICAT is developed by Fayyad et al. [32]. The architecture of SKICAT is shown in Figure 2.6. The system automatically searches and classifies sky objects in digitalized sky images. These images are processed to generate image segmentations. The features of these segmentations are extracted to represent the objects. The objects are classified by a classifier and put into the catalog.

SKICAT used a generalized ID3 algorithm [17] to induce a decision tree. Some objects are classified by the experts (astronomers) and used as training data to help the induction of the decision tree. A set of rules is extracted from the decision tree to form the classifier.

2.2.4 Other KDD Systems

Many other data mining systems have been developed, such as IMACS [10, 9], Recon [99], Explora [66], Spotlight [6], and several others.

- IMACS was developed by Brachman et al. at AT&T [10, 9]. It takes a unique approach, a human centered discovery process. The system has integrated support for human problem solving by visualization, interaction, knowledge representation and data processing.

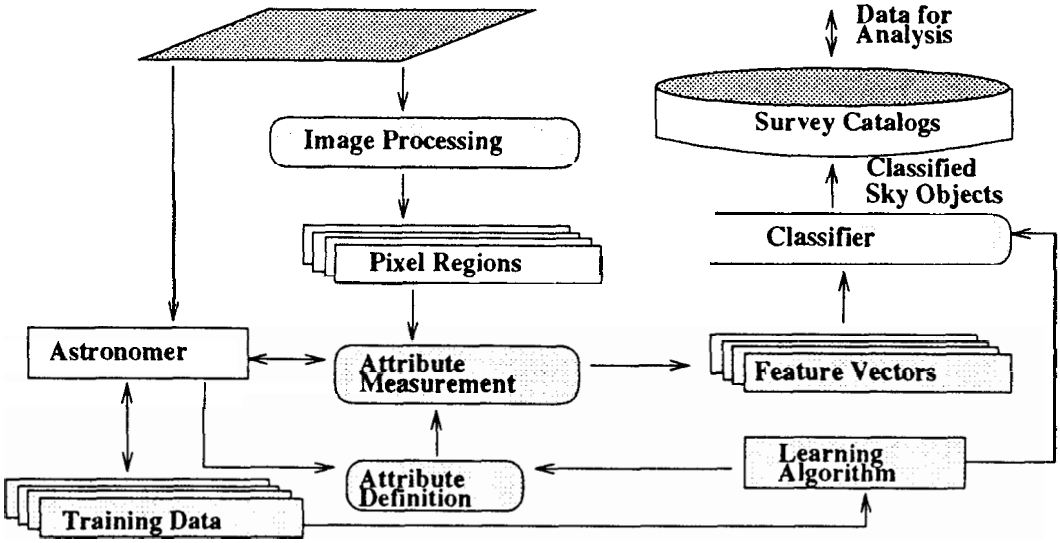

Figure 2.6: Architecture of SKICAT (Fayyad et al. 1996).

- Recon was developed by Simoudis et al. [100]. It integrates three kinds of tools, data visualization, rule induction and deductive database to support interactive data mining.
- Explora was developed by Klösgen [66] at the German National Research Center for Computer Science. It integrates a set of basic statistical, machine learning, and general artificial intelligence tools, as assistants for interactive knowledge discovery.
- Spotlight was developed by Anand et al. at AT&T [6] to navigate through very large databases (gigabytes).

2.3 Summary

In this chapter, different approaches to KDD are discussed. The studies on KDD are classified into five major categories based on their principal methodologies: mathematical and statistical approaches, machine learning approaches, database-oriented approaches, integrated approaches, and other approaches. A representative method from each category is surveyed to explain the basic ideas of each approach. We have

also analyzed the features and the limitation of each method, as well as the types of data and/or the application areas for which the methods are best suited.

Chapter 3

Conceptual Hierarchies and Their Manipulation

As mentioned in Chapter 1, concepts in databases are organized into a partial order called conceptual hierarchy. Conceptual hierarchies play an important role in the knowledge discovery process because they specify background or domain knowledge and may affect the discovery processing and the results. In this chapter, we discuss basic ideas about conceptual hierarchy and its manipulation.

This chapter is organized as follows. In Section 3.1, basic terms about conceptual hierarchies are introduced and some related issues are discussed, including the motivations for using conceptual hierarchies, the specification of conceptual hierarchies, and the availability of conceptual hierarchies. In Section 3.2, dynamic adjustment of conceptual hierarchies is discussed, with two algorithms presented. Automatic generation of conceptual hierarchies for numerical attributes is discussed in Section 3.3, with an algorithm presented. We discuss conceptual hierarchy generation for nonnumerical attributes and the use of more general forms of partial orders in Section 3.4. Finally, the chapter is summarized in Section 3.5.

3.1 Introduction

Some partial orders among data exist in a database. For example, "B.C." is a part of "Canada". Conceptual hierarchies are used to capture such a partial order.

Definition 3.1.1 A *conceptual hierarchy* consists of a set of *nodes* organized in a tree, where the nodes in the tree represent values of an attribute, called *concepts*. A special node, "ANY", is reserved for the root of the tree.

A conceptual hierarchy for an attribute, *province*, is shown in Fig. 1.3. Terms related to trees, such as a leaf node, a nonleaf node, a parent, etc., are used under their original meanings. For example, the node "Prairies" is a parent of the node "Alberta" which is a leaf node.

A number is assigned to the *level* of each node in a conceptual hierarchy. The level of the root node is one. The level of a non-root node is one plus the level of its parent. This top-down assignment of levels is adopted because it is simple and straightforward. Please note that a higher level concept has a smaller level number. Also, the level of a node may change after an adjustment of the conceptual hierarchy as shown in Section 3.2.

Since values are represented by nodes, the levels of nodes can also be used to define the levels of values. A concept is a general or higher level concept if its corresponding node is a nonleaf node in the conceptual hierarchy. A leaf node in a conceptual hierarchy usually represents a primitive concept, i.e., a value stored in a database. For numerical attributes, the leaf nodes may represent the lowest level groupings or segments, in which case primitive concepts are implicitly stored in a conceptual hierarchy. For example, a primitive concept "15" may be a child of a lowest level grouping "0-20". Primitive concepts are sometimes called "data" to emphasize the fact that they are stored in databases.

Multiple conceptual hierarchies can be specified for an attribute. However, we assume that at most one conceptual hierarchy is used for an attribute for a particular data mining task. Therefore, we may say "an attribute's conceptual hierarchy" which refers to the conceptual hierarchy of the attribute used for the current data mining

task. Of course, different conceptual hierarchies of an attribute may be used for different data mining tasks as we discussed in Section 3.4.

We confine a conceptual hierarchy to be a tree structure because it is simple, easy to use and maintain, and usually sufficient for the data mining task. The use of more general partial orders is discussed in Section 3.4.

Sometimes "hierarchy" is used for "conceptual hierarchy" when it is clear from the context.

3.1.1 Motivations for Using Conceptual Hierarchies

The introduction of conceptual hierarchies into data mining can be justified by the following discussions.

- A conceptual hierarchy provides domain knowledge about the data. Such background or domain knowledge is necessary and useful in the process of discovery. The use of background knowledge is echoed in the scientific discovery process in which scientists learn more about nature by conducting experiments designed based on a priori belief or knowledge.

- Conceptual hierarchies organize concepts in a hierarchical or tree form. Hierarchical organizations are familiar to humans and easy to understand, such as taxonomical classifications. Conceptual hierarchies make it easy for humans to understand the discovered results.

- Conceptual hierarchies define levels for concepts elegantly and concisely. This is necessary and helpful for the discovery of multiple-level rules.

- Conceptual hierarchies are often available and can be adjusted and generated automatically. The availability of conceptual hierarchies is discussed in Section 3.1.3. The dynamic adjustment and the automatic generation of conceptual hierarchies are discussed in Section 3.2 and Section 3.3.

3.1.2 Specifications of Conceptual Hierarchies

In this section, we discuss two forms of specifications of conceptual hierarchies: the instance-based specification and the schema-based specification.

- The *instance-based specification* of a conceptual hierarchies is to list all children-parent pairs. For example, the conceptual hierarchy for the attribute "province" in Figure 1.3 can be specified as `province: {B.C., Prairies}` $\subset$ `{Western}`, `{Western, Central, Maritime}` $\subset$ `{Canada}` , etc., in which the attribute name is followed by a list of children-parent pairs. The symbol $\subset$ defines the partial order. The set to the left of $\subset$ lists the children, and the set to the right of $\subset$ lists the parent.

- The *schema-based specification* of a conceptual hierarchy is to specify the hierarchy in terms of a database schema. For example, given the schema of a relation, *address(street, city, province)*, we can define the partial order that a street is part of a city, which in turn is part of a province, by stating `(street)` $\subset$ `(city)` $\subset$ `(province)`.

  A schema-based specification involves several attributes in an original schema. We introduce a "macro" attribute which "summarizes" all the original attributes involved in the specification. The original attributes are replaced by the "macro" attribute. The hierarchy is thus specified for the "macro" attribute whose values are the union of values of the original attributes. For example, we can introduce a "macro" attribute, *place*, which replaces the original attributes *street, city*, and *province*. The values of *place* are the union of the values of *street, city*, and *province*. The above specification is then for the attribute *place*.

  A schema-based specification takes the advantage of the database schema, which provides meta-data information about the data. It can specify the partial order more concisely than by listing all instances of streets, cities, and provinces.

A conceptual hierarchy can be defined by either an instance-based specification or a schema-based specification. Moreover, a schema-based specification can be mapped

into an instance-based specification by explicitly listing each tuples in the relation in a partial order form.

However, it is not always possible to map a hierarchy defined by an instance-based specification into a schema-based specification. A schema-based specification can be used to define a hierarchy whose paths from every leaf node to the root have the same length, by transformation of each path into a tuple in a relation which has a field for each level of the hierarchy. However, if a hierarchy has paths of various lengths, null values have to be used in the relation, which may cause semantical inconsistence and ambiguity.

Fortunately, a conceptual hierarchy can always be defined by an instance-based specification and the transformation of the hierarchy into a schema-based specification is not necessary. A schema-based specification may provide an alternative and concise way to specify a hierarchy, but only when such a specification is possible.

3.1.3 Availability of Conceptual Hierarchies

Because conceptual hierarchies play a central role in our approach, it is essential that conceptual hierarchies be available for most data mining tasks. In this section, we discuss how conceptual hierarchies can be derived.

In general, there are three ways to obtain conceptual hierarchies.

- Conceptual hierarchies may be provided by domain experts or users. For example, the user can specify the hierarchy in Fig. 1.3 as discussed earlier.
- Conceptual hierarchies may be derived from the schema of data relations, in the form of scheme-based specification, such as the *address* mentioned above.
- For numerical attributes, it is possible to generate hierarchies automatically. We propose an algorithm for generating hierarchies in Section 3.3.

Based on the above discussion, it is evident that conceptual hierarchies are often available. Moreover, the effort to obtain conceptual hierarchies is usually minor. First,

the basic understanding of the schema will enable the user to obtain the schema-based specifications easily. Furthermore, for a categorical (nonnumerical) attribute, the specification of hierarchies does not require much effort from users or experts, because there are usually only a small number of concepts, or values. For numerical attributes, even though there may be a large number of values, we only need to specify the ranges or groupings which are usually few.

3.1.4 Operations Using Conceptual Hierarchies

Given a task-relevant set of data and a conceptual hierarchy (which may be automactically generated) for each attribute of the data set, two operations are possible: generalization and specialization.

Definition 3.1.2 The *generalization* of a concept of an attribute is the replacement of the concept with one of its ancestors in the attribute's hierarchy. The concept is *generalized* to a higher level, l, if it is replaced by its corresponding ancestor at level l.

Definition 3.1.3 An attribute is *generalized* to a level, l, if all of its values in the data set whose level is lower than l are generalized to level l.

Definition 3.1.4 The *specialization* of a concept of an attribute is the replacement of the concept with one of its descendants in the attribute's hierarchy. The concept is *specialized* to a lower level, l, if it is replaced by its corresponding descendant at level l.

Definition 3.1.5 An attribute is *specialized* to a level, l, if its values in the data set which are nonleaf nodes and at levels higher than l are specialized to level l.

For example, using the hierarchy in Fig. 1.3, the concept "B.C." of the attribute *province* can be generalized to "Western" at level 3. If all values are generalized to "Western", "Central", "Maritime", and "Outside Canada", the attribute *province* is generalized to level 3. Similarly, the concept "Western" can be specialized to "B.C."

at level 4. If all values are specialized to individual provinces, the attribute *province* is specialized to level 5 because all provinces are leaf nodes at level 4 or 5.

To ensure the correctness of the operations, special attention should be paid to specializations. A concept should be specialized to the corresponding descendant from which the concept is generalized from. For example, if the concept "Western" is generalized from the primitive concept "B.C.", it should be specialized to "B.C.", not "Alberta" or "Manitoba". More about specialization is discussed in Chapter 4.

3.2 Dynamic Adjustment of Conceptual Hierarchies

As mentioned in section 3.1.3, conceptual hierarchies may be provided by users or may exist in some data relations. However, sometimes, the given hierarchy is not appropriate for the particular mining task. It is therefore necessary to dynamically refine or adjust an existing conceptual hierarchy based on the mining task, the set of relevant data, and data distribution statistics.

In this section, two algorithms are given for dynamic adjustment of conceptual hierarchies. The first algorithm, *prime level focusing*, balances the nodes at a level of interest to the user, called prime level, which is computed from a given threshold. The second algorithm, the v-node insertion algorithm, adjusts nodes at any level by inserting "virtual" or "dummy" nodes, called v-nodes, into the hierarchy.

3.2.1 Dynamic Conceptual Hierarchy Adjustment with Attribute Threshold

Usually, data mining finds patterns which are presented in terms of concepts of the attributes. For an attribute, its *attribute threshold* is an integer restricting the number of distinct values of the attribute in the discovered patterns.

Basic Ideas

Definition 3.2.1 The *prime level* of an attribute is the lowest level such that when the attribute is generalized to that level, the number of its distinct values in the data set is no more than the given *attribute threshold.* A *prime relation* is a relation whose attributes are all generalized to their prime levels.

Given an attribute threshold and a data set, the prime level of an attribute can be determined from the given conceptual hierarchy.

Definition 3.2.2 The prime level of a conceptual hierarchy is the prime level of its corresponding attribute.

Example 3.2.1 Suppose all the leaf nodes of the hierarchy in Fig. 1.3 appear in the data set. The prime levels of the attribute, *province*, for different attribute thresholds are shown in Fig. 3.1. The number besides each node is the level of the node.

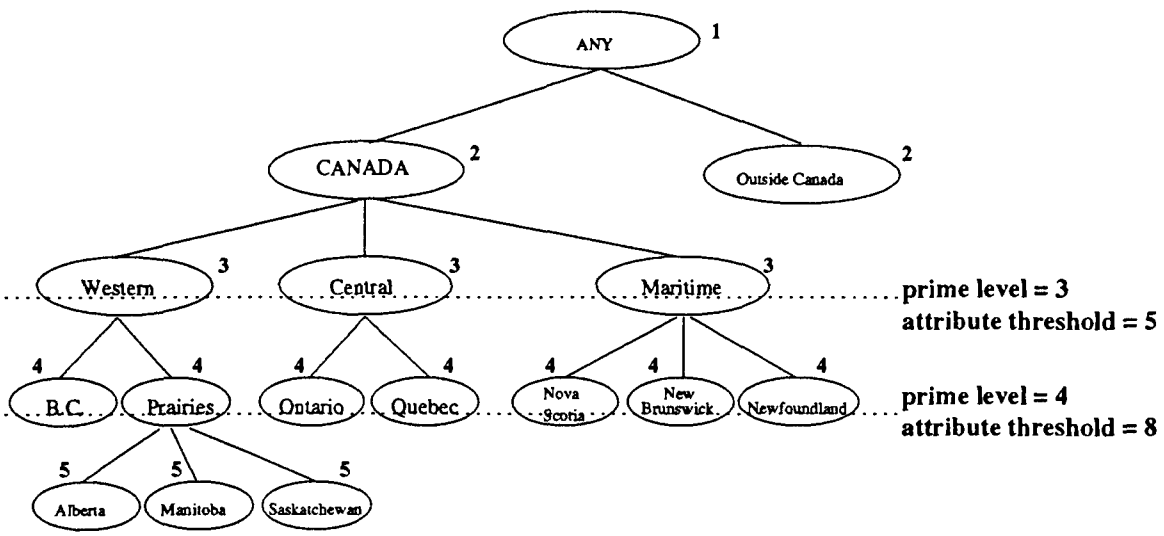

Figure 3.1: Prime levels and attribute thresholds.

□

Example 3.2.2 Suppose the hierarchy in Fig. 3.1 is given for the birth places of students of Canadian universities. Such a hierarchy may not fit all data mining tasks.

For example, to find the regularities of the birth places of the students in Simon Fraser University, it may be desirable to express the generalized level concepts as {*B.C., Other_Provinces_in_Canada, Outside_Canada*}. But if the task is to find the regularities of the birth places of the students in University of Toronto, the generalized level concepts {*Ontario, Quebec, Other_Provinces_in_Canada, Outside_Canada*} may be more suitable. Such adaptation of different data distributions can be achieved by dynamic adjustment of conceptual hierarchies based on the set of relevant data.

□

Example 3.2.2 indicates that dynamic adjustment of conceptual hierarchies according to the distributions of the relevant set of data should be used in many generalization processes. At first glance, dynamic adjustment of an existing conceptual hierarchy seems to be an overly complex process since it corresponds to dynamically regrouping the data, and its complexity grows exponentially with the size of the hierarchy. However, since the given conceptual hierarchy provides important semantic information about conceptual clustering, it is important to preserve the existing data partition as much as possible. This could be done by performing minor refinements on the existing clustering, which will substantially reduce the total number of combinations to be considered.

The following observations may lead to the design of an efficient and effective algorithm for dynamic conceptual hierarchy adjustment.

First, dynamic adjustment of conceptual hierarchies should not be performed during the collection of the set of relevant data. This is because the data retrieval process involves only the mapping of higher level concepts in the query (or mining task) to their corresponding lower level data, which should be determined by the semantics specified in the existing conceptual hierarchy.

Secondly, conceptual hierarchy adjustment is a highly dynamic process. The next mining task may have a different relevant set of data with a different data distribution, which may require the hierarchies to be adjusted differently from the current task. Therefore, an adjusted hierarchy is usually not stored for future usage.

Thirdly, it is often desirable to present the regularities by a set of nodes, which are usually generalized or high level concepts, with relatively even data distribution, i.e., not a blend of very "big" (i.e., occurring frequently in the data set) nodes and very small ones at the same level of abstraction. Thus, it is desirable to promote the big low-level nodes to higher levels, and to merge the tiny nodes when presenting final results.

Finally, by giving attribute threshold for an attribute, the users implicitly choose the attribute's prime level. Although a conceptual hierarchy could be quite deep, the users may be mostly interested in the concepts at the levels close to the prime level. Therefore, the adjustment of conceptual hierarchies can be focused at the levels close to the prime level. The adjustment of hierarchies at all levels (i.e., without a given attribute threshold), is discussed in Section 3.2.2.

Based on the above observations, we introduce some new terminology and present an algorithm, called prime level focusing, for the dynamic adjustment of conceptual hierarchies.

Definition 3.2.3 The *count* of a node is a number associated with the node, representing, if a leaf node, the number of occurrences of the value in the task-relevant data set, or if a nonleaf node, the sum of the count of its children nodes. The *total count* of an attribute is the sum of the counts of all the leaf nodes in the data set.

Definition 3.2.4 The *weight* of a node is the ratio of the node's count over the total count of the attribute. A node is *big* if its weight is larger than $1/T$ where T is the attribute threshold. Otherwise, it is a *small* node.

The total count and the attribute threshold are changed in the following algorithm to reflect the dynamic nature of big nodes. However, these changes are only effective in the scope of the algorithm.

Algorithm for Dynamic Conceptual Hierarchy Adjustment with Attribute Threshold

The prime level focusing algorithm for dynamic hierarchy adjustment is presented as follows. This algorithm tries to have evenly weighted nodes at the prime level

by top-down big nodes promotion and bottom-up small nodes merging. Other approaches such as bottom-up promotion of big-nodes are also possible. However, our approach represents one of the interesting features in human exploration in which interesting patterns (big nodes) at higher levels are explored before lower level details are examined.

Algorithm 3.2.1 (Prime Level Focusing) *Dynamic adjustment of conceptual hierarchies based on the data distribution of a given attribute in the initial relation (i.e., the set of data relevant to the data mining task) and the attribute's threshold.*

Input. (i) A mining task-relevant initial relation $\mathcal{W}_0$, (ii) an attribute A, (iii) the attribute threshold, T, for A, and (iv) a prespecified conceptual hierarchy H.

Output. An adjusted conceptual hierarchy H' of attribute A for the derivation of the prime relation.

Method. The adjustment essentially consists of two processes: top-down big nodes promotion and bottom-up small nodes merging.

1. Initialization:
   (a) Assign the levels to the nodes in the hierarchy H, i.e., 1 for the root, and 1 plus the level of the parent for all other nodes.
   (b) Scan once the corresponding attribute of each tuple in the initial relation $\mathcal{W}_0$, calculate the count $c_i.count$ for each leaf node c_i, and propagate the counts to the corresponding parents in the hierarchy H. The *total* is the sum of the counts of all the leaf nodes in the hierarchy. Notice that only the nodes with a nonzero count are considered in the following computation.
2. Top-down adjustment of conceptual hierarchy H.
   (a) Set a buffer set, *Prime*, initially empty, and another buffer set, *Buff*, to hold the root of H.
      i. Calculate the weight of each node c_i as $c_i.weight := c_i.count/total$.

ii. Set the weight threshold τ as $\tau := 1/T$.

iii. Perform node marking: A big leaf node is marked B, a big nonleaf node is marked B', a small leaf node is marked S, and a small nonleaf node is marked S'.

(b) Call *expand_buffer*, which is implemented as follows.

i. Move every B-marked node from *Buff* to *Prime*;

ii. Replace every B'-marked node by its children;

iii. Repeat this process until there is no change (i.e., only the nodes marked S or S' are left in *Buff*).

(c) Perform weight re-calculation and node re-marking again as following. If $|Prime| + |Buff| \leq T$, move all the nodes from *Buff* to *Prime*, and the process terminates. Otherwise, set T' to $T - |Prime|$, $total'$ to the sum of the counts in *Buff*, $weight'$ of each node in *Buff* to $count/total'$, and τ' to $1/T'$. Mark the nodes based on the $weight'$ and τ', and repeat the *expand_buffer* and weight re-calculation processes until there is no change.

3. If there are still nodes left in *Buff*, perform bottom-up merging of the remaining nodes in *Buff* as follows.

Starting at the bottom level, step up one level (suppose, to level i) and merge the nodes in *Buff* which share a common ancestor at level i. If the $weight'$ of the merged node is no less than τ', move it to *Prime* (and decrement T'). If the total number of nodes in *Buff* is no more than T', then move all the nodes in *Buff* to *Prime*, else perform weight re-calculation, step up a level, and repeat the process.

We have the following convention for naming a merged node. Name a node $A + B$ if it is the result of merging two nodes A and B. Otherwise, name it $E - A$ if it is equivalent to an existing node E with one child node A removed. Otherwise, name it *Other_E* if it is equivalent to an existing node E with more than one child node removed.

4. Let l be the lowest level (largest level number) of the nodes in *Prime*. If a nodes in *Prime* is not at level l, thus at level higher than l, "copies" of the node are inserted between the node and its parent to lower the node to level l. A "copy" is a node which represents the same concept as a node in *Prime*. □

Theorem 3.2.1 *There are no more than T (the attribute threshold) nodes in Prime, and there exists a (generalization) path between every node in the initial relation and a node in the prime relation after the execution of Algorithm 3.2.1.*

Rationale. According to the algorithm, every node moved into *Prime* must satisfy one of the following three conditions: (1) having a weight greater than τ or τ', (2) when $|Prime| + |Buff|$ is no more than T or T', or (3) when the remaining nodes are grouped into T' groups (i.e., T' new nodes) when there are no more levels to climb. Moreover, the computations of T', τ and τ' ensure that the number of the accumulated nodes is no more than T. Thus the algorithm cannot generate more than T nodes in *Prime*. In addition, every non-zero count node is either a leaf node moved into *Prime*, or is associated with a nonleaf (ancestor) node that is eventually moved into *Prime* according to the algorithm. There should exist a path from every node to a node in *Prime* (thus in the prime relation), after the execution of the algorithm. □

Furthermore, Algorithm 3.2.1 is designed based on the consideration that the nodes in the prime relation should carry relatively even data distribution, and that the shape of the hierarchy should be preserved when possible. Therefore, hierarchy adjustment following the algorithm should produce desirable results.

Experiments with NSERC Databases

The prime level focusing algorithm is implemented in the DBMiner system.

The mining query is to find characteristic rules (explained in Chapter 4) of the 1991 NSERC Research Grants in Computing Science in relevance to provinces. The original conceptual hierarchy for the attribute *Province* is given. Fig. 3.2 shows the

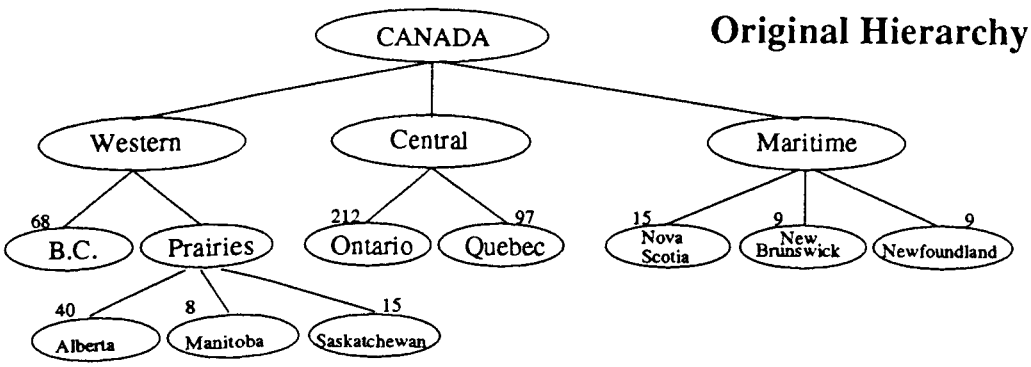

Figure 3.2: Original conceptual hierarchy for *province*.

relevant part of the hierarchy. The number besides each node is the node's count. The attribute threshold for *Province* is set to 7.

Using the original hierarchy *without* dynamic adjustment, the derived prime relation consists of 7 values in the attribute: {"British Columbia" (68), "Prairies" (63), "Ontario" (212), "Quebec" (97), "New Brunswick" (15), "Nova Scotia"(9), "Newfoundland" (9)}, (where each number in parentheses indicates the count). These correspond to the 7 nodes at level 3 in Fig. 3.2. This is undesirable since the level 4 node "Alberta" has count 40, whereas each Maritime province (at level 3) has much smaller counts. Notice that some nodes, such as "Ontario" (212), are leaf nodes which, though quite big, cannot be split further.

Following Algorithm 3.2.1, the dynamic adjustment of hierarchy is performed based on the current mining task and the counts of nodes. This results in Fig. 3.3, in which "Alberta" is promoted, and the maritime provinces are merged. Note that a "copy" of the node "Maritime" is inserted so that "Maritime" will be at the same level as others in the *Prime*. The attribute in the prime relation consists of 6 nodes: {"British Columbia" (68), "Alberta" (40), "Sas+Man" (23), "Ontario" (212), "Quebec" (97), "Maritime" (33)}, with a relatively even distribution among all the nodes at the prime level.

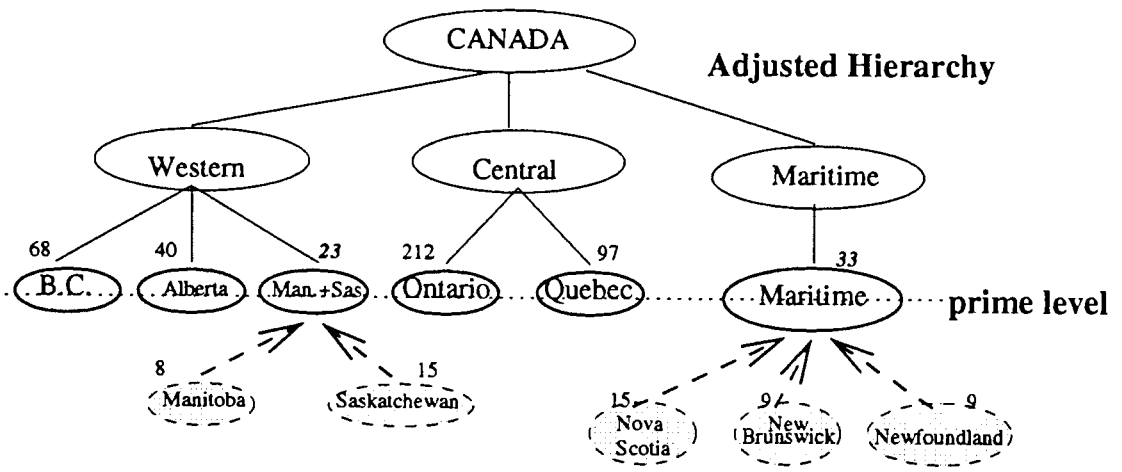

Figure 3.3: Dynamically adjusted conceptual hierarchy for *province*.

3.2.2 Dynamic Conceptual Hierarchy Adjustment without Attribute Threshold

The algorithm introduced in the previous section generates a hierarchy whose nodes are evenly distributed at the prime level. This solution may be desirable and sufficient for many cases. However, there are some restrictions which may affect its usage and result.

- Sometimes the attribute threshold is unknown or uncertain to the user. It is desirable to adjust conceptual hierarchies even if the attribute threshold is not available.
- Algorithm 3.2.1 focuses on evenly weighted nodes (even distribution of weight) at the prime level. For multiple-level rules, it would be beneficial to have a hierarchy that is evenly weighted at all its levels.
- The original nodes may be removed from the resulting conceptual hierarchy. A problem could result if the user wanted to keep all the original nodes.

Based on the same goal and observations of Section 3.2.1, we present the second algorithm for conceptual hierarchy adjustment, v-node insertion. The algorithm has

the following features compared with algorithm 3.2.1.

- It does not use the attribute threshold and focuses on all levels instead of only the prime level.
- The algorithm generates the optimal hierarchy based on an entropy measure, while keeping the original conceptual hierarchy as much as possible.
- A parameter α is used to adjust the algorithm's performance.

Basic Ideas

Assume we are given a set of task-relevant data, D, and a conceptual hierarchy, H, of an attribute A. We remove all nodes in H with count 0 with respect to D since they are irrelevant to the current data mining task.

Definition 3.2.5 A terminal set at level l, T_l, is the set of nodes at level l and the leaf nodes at higher levels, in H.

$$T_l = \{n | n \in \text{H, level}(n) = l \text{ or } n \text{ is a leaf node and } level(n) < l) \}.$$

The terminal set corresponds to the output values of the attribute A if the data are generalized to level l using H.

Definition 3.2.6 A node set is called the specialized set of another set T if it is obtained by recursively replacing one of the nodes in T by its children.

For example, in the conceptual hierarchy shown in Figure 3.2, the sets {*B.C., Prairies, Central, Maritime*} and {*B.C., Alberta, Manitoba, Saskatchewan, Central, Maritime*} are specialized sets of the set {*Western, Central, Maritime*}.

As observed in Section 3.2.1, it is desirable to have relatively even weighted nodes at each level. A measure, entropy, is introduced to quantitate the evenness of a node set.

Definition 3.2.7 The entropy of a node set $T = \{n_1, n_2, \cdots, n_k\}$, $E(T)$, is given as follows:

$$E(T) = \sum_{i=1}^{k} p(n_i) \log(1/p(n_i)).$$

where $p(n_i)$ is the weight of the node n_i as defined in Definition 3.2.4.

For simplicity, we say "entropy at level l" which actually means "entropy of the terminal set of H at level l".

Our goal is to generate a hierarchy which keeps the nodes and orders among the nodes in H, but minimizes the entropy at each level. In the meantime, we want to have as many "big" nodes as possible at each level. The two goals are conflicting since the more nodes the hierarchy has at a level, the larger the entropy is at that level, as shown in Lemma 3.2.1.

Lemma 3.2.1 *If T_1 is a specialized set of another set T_0, $E(T_1) \geq E(T_0)$.*

Proof. Since T_1 is obtained by recursively replacing a node by its children, we only need to prove the operation increases entropy. Let a set N have nodes $n_1, \cdots, n_k$, with weights $p_1, \cdots, p_k$, respectively. Suppose N' is obtained by replacing n_i with its children, $n_{i1}, \cdots, n_{im}$, with weights, $p_{i1}, \cdots, p_{im}$ respectively. Obviously, $\sum_{j=1}^{m} p_{ij} = p_i$.

$$\begin{aligned} E(N') &= \sum_{j=1}^{i-1} p_j \log(1/p_j) + \sum_{j=1}^{m} p_{ij} \log(1/p_{ij}) + \sum_{j=i+1}^{k} p_j \log(1/p_j) \\ &\geq \sum_{j=1}^{i-1} p_j \log(1/p_j) + \sum_{j=1}^{m} p_{ij} \log(1/p_i) + \sum_{j=i+1}^{k} p_j \log(1/p_j) \\ &= \sum_{j=1}^{k} p_j \log(1/p_j) \\ &= E(N) \end{aligned} \tag{3.2.1}$$

□

To compromise the two conflicting goals, a special kind of nodes, called *v-nodes*, is introduced, in contrast to the *concept node* which is a node in the original conceptual hierarchy H. A v-node can be inserted between a "small" node and its parent so that the original nodes and orders are kept and each level has as many and as evenly weighted "big" nodes as possible. To add more flexibility, a real number, α ($0 < \alpha \leq 1$), is introduced to refine the definition of "big" node in algorithm 3.2.1.

Definition 3.2.8 A node n at level l is called α-**big** if $p(n) \geq \alpha \times 1/|T_l|$. Otherwise, it is called α-**small**.

Definition 3.2.9 A virtual node, or v-node, is an α-small node which shares the name of its only child, called its concrete node, which itself can be a v-node. A v-node, v, can only be inserted into a hierarchy by making its concrete node, n_c, the child of v and by letting the parent of n_c become the parent of v.

The situations before and after the insertion of a v-node are demonstrated in Figure 3.4.

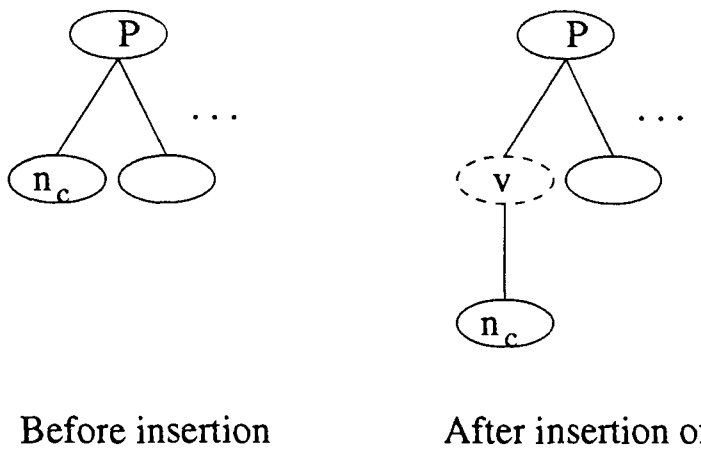

Figure 3.4: Insertion of a v-node.

A v-node is different from a "copy" in Section 3.2.1. A v-node can represent an α-small node at any level, but a "copy" can only represent a node at prime level which can be α-big or α-small.

Definition 3.2.10 A conceptual hierarchy is α-compatible to H if and only if it is obtained by inserting virtual nodes into H.

Our goal is to generate a hierarchy that is α-compatible to H and has minimum entropy at each level. An algorithm is presented next.

Algorithm for Dynamic Conceptual Hierarchy Adjustment without Attribute Threshold

The basic idea of the algorithm is to push α-small nodes to lower levels by inserting v-nodes. Thus, at each level, we will have the relatively even distributed α-big nodes. In other words, the algorithm balances the need for minimum entropy at each level and the need to have as many nodes as possible.

The algorithm is given as follows. For simplicity, we assume the preprocessing is done, that is, the level number and weight of each node is pre-computed, otherwise, step 1 of the algorithm 3.2.1 should be executed.

If all nonleaf nodes are α-small or there is no nonleaf node at level l, the insertion of v-node does not change T_i for $i = 1, \cdots, max\_level(H)$, and so the process stops.

Algorithm 3.2.2 (v-node Insertion) *Dynamic adjustment of conceptual hierarchies by the insertion of v-nodes.*

Input: (i) a pre-computed, given conceptual hierarchy, H, (ii) a number $0 < \alpha \leq 1$.

Output: An adjusted hierarchy.

Method. The output hierarchy is obtained by top-down iterative inserting of v-node into H.

1. Remove the irrelevant nodes (those with count 0).
2. From top to bottom, for all levels do:
   (a) Collect all terminal nodes into a set.
   (b) If at least one nonleaf node is α-large
   insert a v-node for each α-small nonleaf node in the set.
   (c) Otherwise, the process stops.
3. Output the adjusted hierarchy.

Using C-like syntax, the algorithm can be written as follows.

```
remove nodes with count 0;
l = 2; /* start from second level */
```

$T = \{n | level(n) = l$ or n is a leaf node and $level(n) < l\}$; /* terminal set at level l */
while TRUE do {
    $NL = \emptyset$; /* α-small nonleaf nodes */
    stop = TRUE;
    for each nonleaf node n in T do
        if n is α-small then
            $NL = NL \cup \{n\}$;
        else stop = FALSE; /* at least one nonleaf node is α-large */
    if stop then break;
    for each node n in NL do
        insert a virtual node between n and n's parent.
    $l = l + 1$; /* next level */
    $T = \{n | level(n) = l$ or n is a leaf node and $level(n) < l\}$;
    }
Output the adjusted hierarchy.

□

Proposition 3.2.1 *The algorithm 3.2.2 terminates.*

Rationale. Consider the nonleaf nodes of H. The *while* loop processes at least one nonleaf node if there is an α-big nonleaf node. Otherwise, it stops. A brief analysis will show that the time complexity of the algorithm 3.2.2 is $O(N \times D)$ where N is the number of nonleaf nodes in H and D is the depth (number of levels) of the adjusted hierarchy. □

Theorem 3.2.2 *Among all the α-compatible conceptual hierarchies of H, the conceptual hierarchy generated by algorithm 3.2.2, H_0, has the minimum entropy at each level.*

Rationale. Suppose there is another α-compatible hierarchy, H_1. Let T_i^0 and T_i^1 be the terminal set at level i of H_0 and H_1, respectively. We compare T_i^0 and T_i^1 for $i = 1, \ldots, max\_level(H_0)$. Let l be the highest level that the two sets disagree. That is, $T_i^0 = T_i^1$ for $i = 1, \ldots, l-1$, and $T_l^0 \neq T_l^1$.

The discrepancy between T_l^0 and T_l^1 can only be caused by inserting v-node at level $l-1$ in one of H_0 and H_1, but not the other. We consider the two cases.

Case 1. A v-node is inserted in H_0 but not in H_1. Clearly, T_l^1 is a specialized set of T_l^0. Based on Lemma 3.2.1, $E(T_l^0) \geq E(T_l^1)$. Due to the nature of terminal sets, T_i^1 is a specialized set of T_i^0 for all $i \geq l$. This means that $E(T_i^0) \geq E(T_l^1)$ and $|T_i^0| \leq |T_l^1|$ for $i \geq l$.

Case 2. A v-node, v_1, is inserted in H_1 but not in H_0. Let v_0 be the counterpart of v_1 in H_0. Since v_0 is a concept node (i.e., not a v-node), v_0 must be a α-big node and so is v_1 because $T_{(l-1)}^0$ and $T_{(l-1)}^1$ are the same. This contradicts our definition of v-node. Using similar analysis and the observation in Case 1, we conclude that this case can never occur.

Therefore, the only possible difference could be inserting a v-node in H_0 but not in H_1, which decreases the entropy. □

Experiments with NSERC Databases

The v-node insertion algorithm is implemented in the DBMiner system.

Given the same data mining task and conceptual hierarchy as in Section 3.2.1, the v-node insertion algorithm will generate different hierarchies given different α values. The adjusted hierarchy for $\alpha = 0.6$ is shown in Figure 3.5 and another is shown in Figure 3.6 for $\alpha = 0.9$. The v-nodes are in dashed-line circles. The count, rather than the weight, is displayed beside each node.

As shown in Fig. 3.5 and Fig. 3.6, the value of α affects the resulting hierarchy. On the one hand, the larger the value of α is, the more v-nodes are to be inserted and the resulting hierarchy is more even at each level. On the other hand, the smaller the value of α is, the fewer v-nodes are to be inserted and the resulting hierarchy has more nodes at each level. The selection of α is usually task dependent and should be controlled by the user. In our experiments with NSERC databases, we found that suitable values of α were usually from 0.5 to 1.0.

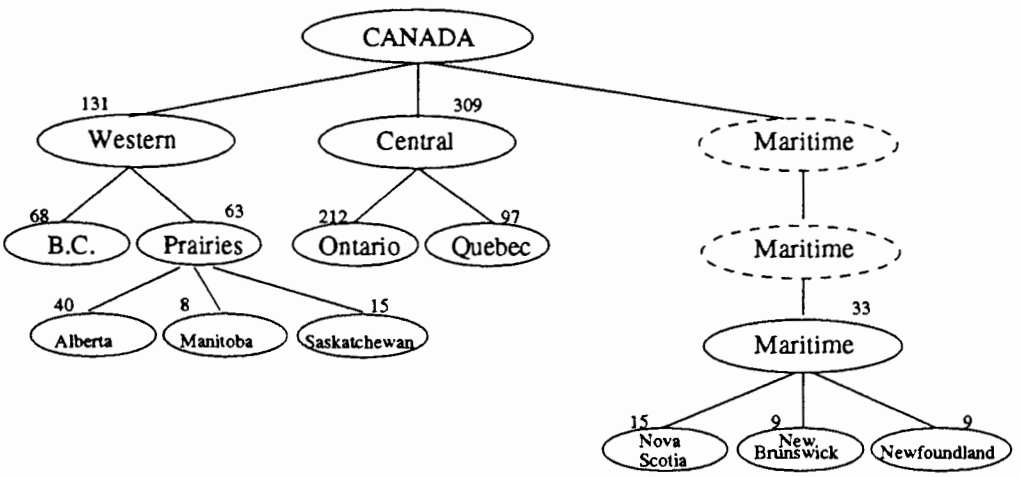

Figure 3.5: Adjusted conceptual hierarchy for $\alpha = 0.6$.

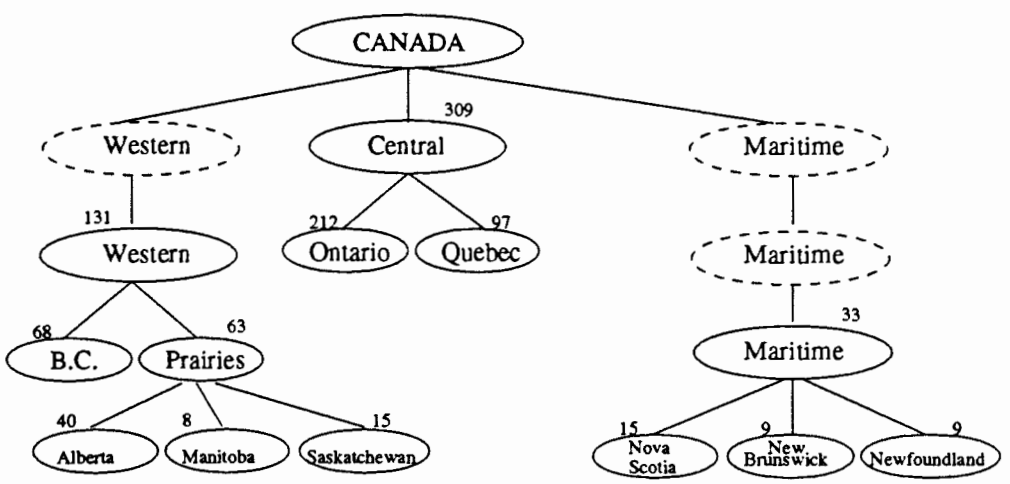

Figure 3.6: Adjusted conceptual hierarchy for $\alpha = 0.9$.

3.3 Automatic Generation of Conceptual Hierarchies for Numerical Attributes

As we mentioned in Section 3.2, given conceptual hierarchy may not fit for the current task, or conceptual hierarchies may not exist for some attributes. Adjustment of given conceptual hierarchies based on the current situation is discussed in Section 3.2. Moreover, for numerical attributes, such as integer, real, etc., which occur frequently in databases, conceptual hierarchies can be generated automatically by the examination of data distribution characteristics. The automatically generated hierarchies may reflect the current data distributions and deal with the problem when the hierarchies are not available. The automatic generation of hierarchies for nonnumerical (categorical) attributes is discussed in Section 3.4.

3.3.1 Basic Ideas

The following two standards are used for the automatic generation of conceptual hierarchies for numerical attributes.

1. *Completeness*: The value ranges of the hierarchy of a numerical attribute should cover all of its values in the set of data relevant to the current mining task.

2. *Uniformity*: The set of ranges presented in the prime relation should have relatively even distribution based on the frequency of counts of the attribute values in the set of data relevant to the current mining task.

Example 3.3.1 Suppose the mining task is to study the regularities of NSERC research grants for Computing Science in terms of the recipient's provinces and the amount of grants received. The attribute *amount* is a numerical attribute. For automatic construction of hierarchy for the attribute *amount*, the completeness requirement implies that the hierarchy constructed should cover all the amounts in the relevant data set, which could be in the range of {\$2,000 - \$97,000}, i.e., *2k - 97k*. The uniformity requirement implies that the ranges of the amounts of the grants in the prime relation should be relatively evenly distributed across the whole range. If the

attribute threshold value is 4, and more people receive grants in the amount of low and medium ranges, the desired distribution could be $\{[2-12k), [12-16k), [16-23k), [23-90k)\}$. Such a set of ranges has been generated automatically. □

Algorithm for Automatic Generation of Hierarchies for Numerical Attributes

Based on the similar observations analyzed in Section 3.2.1, the algorithm for automatic generation of conceptual hierarchies for numerical attributes of an data set is presented as follows.

Algorithm 3.3.1 (Conceptual hierarchy generation for a numerical attribute) *Automatic generation of conceptual hierarchy for a numerical attribute based on its data distribution in the initial relation.*

Input: An initial relation that contains a numerical attribute A with an attribute threshold T.

Output: A conceptual hierarchy H_A on A for the presentation of the prime relation.

Method. The hierarchy H_A is constructed as follows.

1. *Estimation of the total value range by data sampling.* Sample a set of values of A in the initial relation. Let *low* and *high* be, respectively, the smallest and the largest values of the sampled data.
2. *Derivation of interval value.* Let $interval = (high - low)/(k \times T)$, where k is a constant reflecting the fineness of the segmentation. Usually, k is set between 5 to 10. Rounding or truncating is performed on *interval* to make it customized for human. For example, an *interval* of 474 is rounded up to 500. The range *low*/*high* is truncated/rounded accordingly.
3. *Creation of segments.* A set of segments are created based on the range and interval. $[low, low + interval]$, $[low + interval, low + 2 \times interval]$, ..., $[low + (k \times T - 1) \times interval, high]$.

4. *Merge of segments based on data distribution.* Segments are merged into nodes based on their count frequency distribution.

   First, a histogram (count frequency) is computed based on the data set of the attribute in the initial relation. Each segment is attached a *count* which is initialized to 0. The computation is performed as follows.

   For each tuple t in the initial relation

   if there is a segment $s = [l, h)$ such that $l \leq t[A] < h$

   then $count[s] := count[s] + 1$;

   else { create a segment new: $[low + k \times interval, low + (k+1) \times interval)$

   where $k = (t[A] - low)/interval$;

   $count[new] := 1$;}

   Segments are then merged into nodes so that these nodes will have relatively even distribution of count frequencies. This is implemented as follows. Arrange segments in ascending order based on their range values. Merge the sequence (of segments) whose sum of the counts reaches the closest to $total\_count/T$ into one node, with its *low* range set to the *low* of the first segment, and *high* set to the *high* of the last segment. Repeat the process for the remaining segments until there are no segments left.

```
sum := 0;
first := 1;
node_count := 0;
for i := 1 to n do {
    sum_sav := sum;
    sum := sum + count[s[i]];
    if (sum ≥ total/T) or (i = n) then {
        if node_count = T − 1 /* This is the last node. */
        then i := n
        else if sum − total/T > total/T − sum_sav
        then i := i − 1;
```

merge segments from $first$ to i into a new node;
$sum := 0$;
$node\_count := node\_count + 1$;
$first := i + 1$; } }

The above piece of code shows the segment merging process. Note that to create more levels in the resulting hierarchy, the segments can be merged into subgroups which in turn are merged into a node. Also, the prime level nodes can be further merged into higher level nodes. □

Theorem 3.3.1 *The worst-case time complexity of Algorithm 3.3.1 is $O(n)$, where n is the number of tuples in the initial relation.*

Rationale. Step 1 (data sampling) costs less than n since it only takes a proper subset of the initial relation and linear time to find *high* and *low*. Steps 2 & 3 work on the creation of intervals and segments using *low*, *high* and T, which is much smaller than n. In Step 4, the computation of the histogram takes $O(n)$ time since it scans the initial relation once in the computation, where the merge of segment takes the time proportional to the number of segments, which is smaller than n. Obviously, adding all the steps together, the worst-case time complexity of the algorithm is $O(n)$. □

Notice that when the size of the relevant data set is huge, it may still be costly to calculate the histogram, and the histogram of the reasonably-sized *sampled* data may be used instead. Also, if the distribution is known beforehand, nodes can be built based on the known distribution.

Other techniques exist for the automatic generation of conceptual hierarchies for numerical attributes. For example, Chiu et al. proposed an algorithm for discretization of data using hierarchical maximum entropy [18]. In their method, the initial node is the entire data set. The node is split into several subnodes based on the hierarchical maximum entropy. The expected frequencies of the subnodes are computed based on the given statistical assumptions, and are compared with the real frequencies. If the difference is larger than a threshold, the subnode is split further and the

process is called recursively. Our method provides a simpler and more efficient way of computation for large data sets and still achieves elegant results.

3.3.2 Experiments with NSERC Databases

Algorithm 3.3.1 is implemented in the DBMiner system.

For the mining task in Example 3.3.1, the algorithm generate the hierarchy for the attribute *amount* as follows. First, data sampling results in "*high = 62,350*", "*low = 5,468*" and "*interval = 1,000*". Segments are then created, and a histogram is calculated for the current task following the Algorithm 3.3.1, resulting in Fig. 3.7. Then, the hierarchy is built using the histogram, following the segment merge method presented in Algorithm 3.3.1. The result is shown in Fig. 3.8.

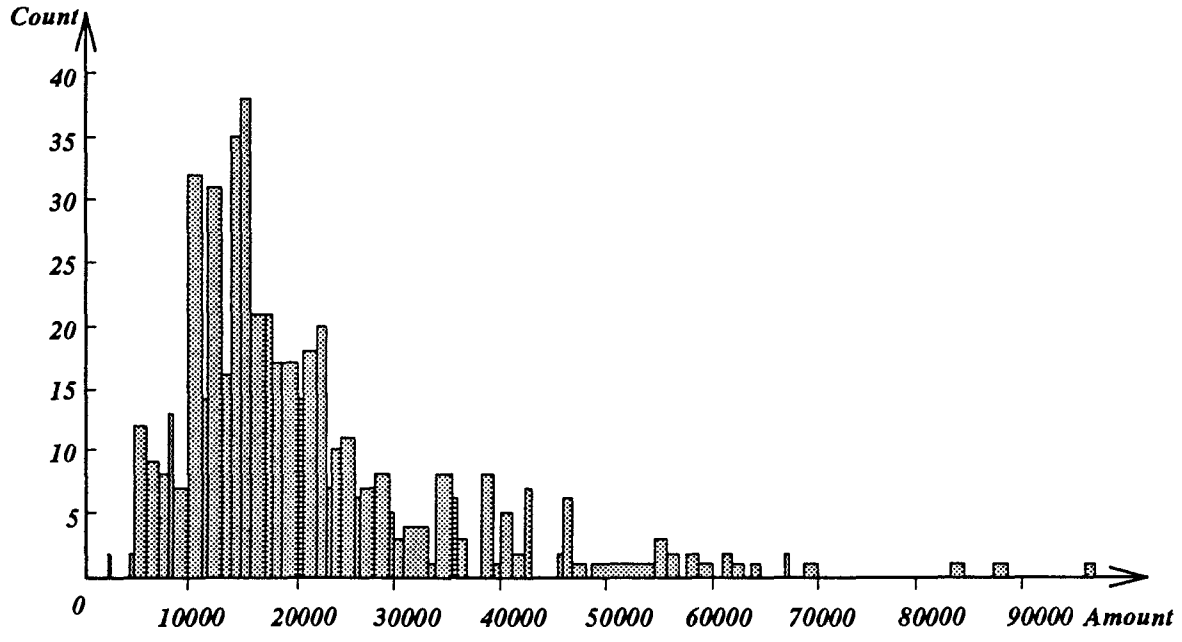

Figure 3.7: Histogram of *Amount* for the current task.

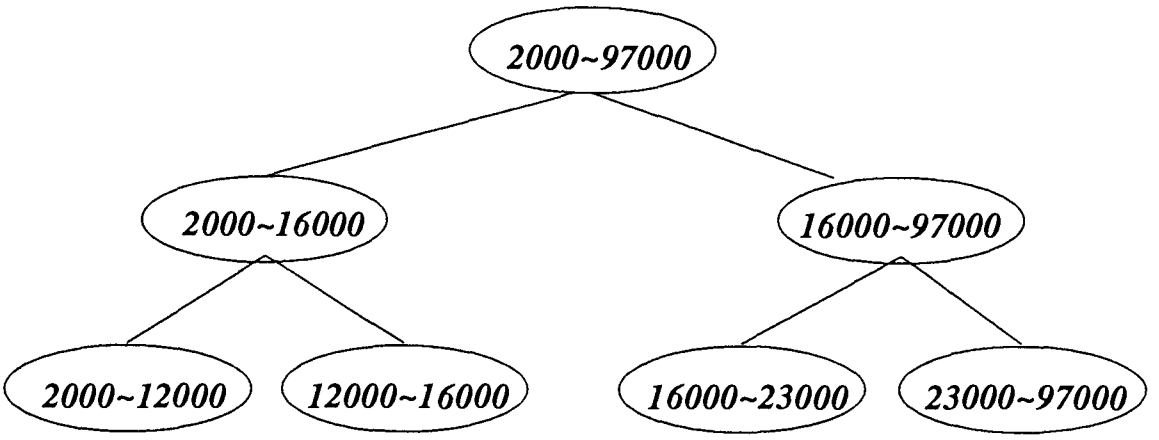

Figure 3.8: Conceptual hierarchy generated for the attribute *Amount*.

3.4 Discussions

We discuss the use of general forms of partial orders and the automatic generation of conceptual hierarchies for nonnumerical attributes in this section.

3.4.1 Use of General Forms of Partial Orders

Conceptual hierarchies are restricted to tree structures which may represent a large amount of partial orders existing in data. However, there are some partial orders in databases which do not form a tree structure. Many studies on general forms of partial orders exist [111, 30]. Our methods can be extended in two ways to deal with general partial orders.

- A general partial order can be split into several conceptual hierarchies so that one of them is used in a particular data mining session. This is based on the assumption that there are many ways to organize the concepts, but usually only one of them is used for a particular data mining task. For example, given a partial order for *time* in Fig. 3.9 which is not a conceptual hierarchy, it can be split into two conceptual hierarchies as shown in Fig. 3.10. A company can calculate its weekly or monthly sales by choosing the corresponding hierarchy.

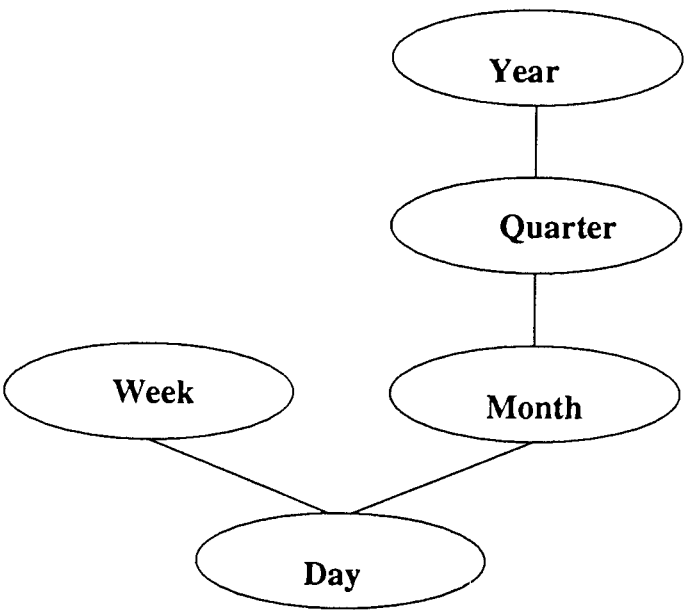

Figure 3.9: A partial order for *time*.

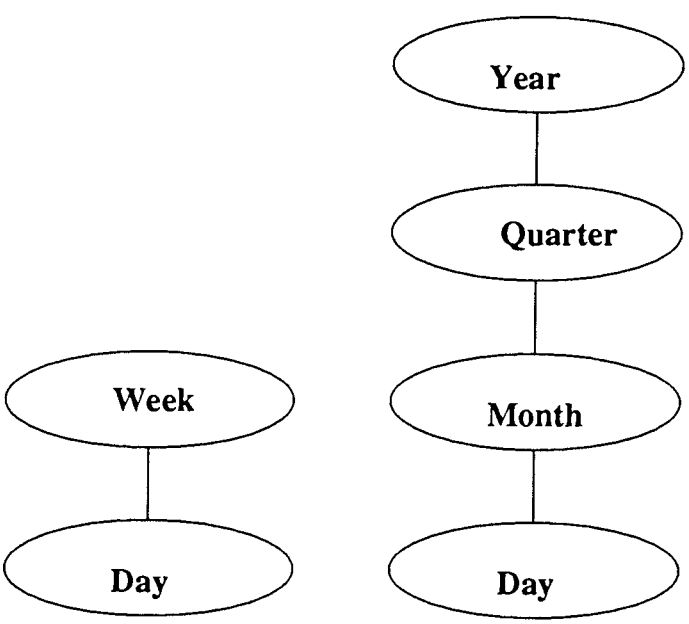

Figure 3.10: Split of a general partial order into hierarchies.

- If the arcs or edges of a general partial order are attached with probabilities, the generalization or specialization can be done by splitting the counts of nodes. For example, using the partial order for *age* as shown in Fig. 3.11, we can generalize 100 young people into 80 adolescents and 20 adults.

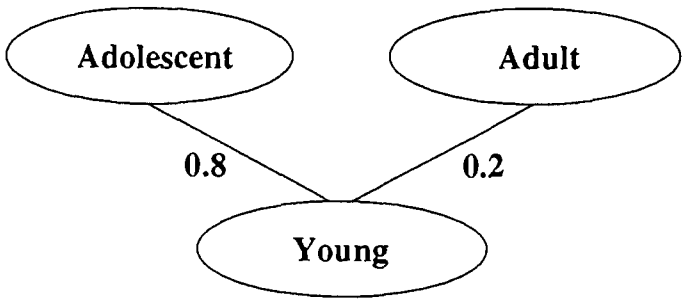

Figure 3.11: A general partial order with probabilities.

However, more research is needed to use general partial orders for data mining. For example, how to split the count when the probabilities are unknown? How to handle the generalization when a node has parents who hold parent-child relationship themselves? For example, given the partial order for clothing in Fig. 3.12, if we generalize "T-Shirt" to both "Garments" and "Shirts", we end up presenting both concepts at the same time in the patterns. This situation is not desirable because "Shirts" are already covered by "Garments".

3.4.2 Automatic Generation of Conceptual Hierarchies for Nonnumerical Attributes

The algorithm proposed in the previous section can automatically generate conceptual hierarchies for numerical attributes. Nevertheless, automatic generation of conceptual hierarchies for nonnumerical attributes still remains an attractive goal because of the substantial efforts for construction and maintenance of conceptual hierarchies in large databases.

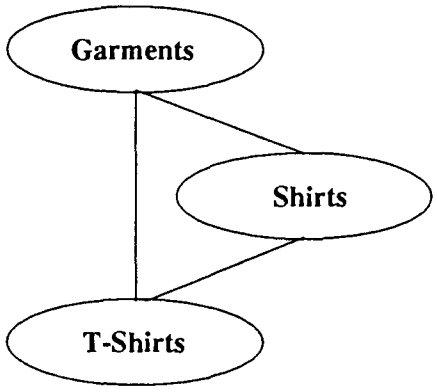

Figure 3.12: A general partial order for clothing.

There have been many interesting studies on automatic generation of hierarchies for nonnumerical data, which can be categorized into different approaches: machine mining approaches [76, 34], statistical approaches [27], visual feedback approaches [64], algebraic (lattice) approaches [78], etc.

The machine mining approach to conceptual hierarchy generation is the most closely related work to our problem. Many influential studies have been performed in this, including Cluster/2 by Michalski and Stepp [76], COBWEB by Fisher [34, 35], hierarchical and parallel clustering by Hong and Mao [56], and many others.

These approaches have their own strengths and weaknesses for different applications. However, they all use some kind of relevance among attributes to search for the best hierarchical clusterings in the representation space defined by related attributes. How to develop a conceptual hierarchy without reference to other attributes is not evident. Our goal is to develop an *efficient* algorithm to *maximize the automatic data clustering capability* for *large databases.* Careful examination and experimentation is required in order to develop the best algorithm. This is an interesting problem for future research.

3.5 Summary

The definition of conceptual hierarchy, as well as related terms, is given in this chapter. Related issues to conceptual hierarchies are then discussed, including the motivations behind the use of conceptual hierarchies, the specifications of conceptual hierarchies, the availability of conceptual hierarchies, and the operations using conceptual hierarchy. Moreover, two algorithms have been proposed for the dynamic adjustment of conceptual hierarchies: the prime level focusing algorithm which balances nodes at an interesting level determined by the attribute threshold, and the v-node insertion algorithm which balances nodes at all levels by inserting "virtual" or "dummy" nodes. Another algorithm has been proposed for the automatic generation of conceptual hierarchies for numerical attributes. All three algorithms have been implemented and tested, and they have demonstrated desirable performance and efficiency. Two related issues, the automatical generation of hierarchies for nonnumerical attributes and the use of more general forms of partial orders, are also discussed.

Conceptual hierarchies are used in the later chapters, including Chapter 4, 5, 6, and 7.

Chapter 4

Mining Multiple-Level Characteristic and Discriminant Rules

Many kinds of rules exist in a large database. It is impractical to find all kinds of possible rules existing in large databases. Our study focuses on the discovery of three kinds of common and useful rules: characteristic rules, discriminant rules and association rules. In this chapter, we investigate the discovery of multiple-level characteristic and discriminant rules. The discovery of multiple-level association rules is discussed in Chapter 5.

4.1 Multiple-Level Characteristic Rules

Characteristic rules are descriptions of characteristics or properties of the data set under study. Usually, the descriptions are in the form of abstractions or summarizations of the current data set. Some examples of the characteristic rule are: customer profiles in a sales database, calling patterns in a telecommunication database, and grant distributions in a research grant database.

Definition 4.1.1 A characteristic rule is a rule whose left hand side is the query condition of the interested data set and right hand side is a generalized tuple, i.e., a conjunction of (attribute, value) pairs. The confidence of a characteristic rule is the ratio of the number of primitive tuples (i.e., tuples in the original relation) the rule covers over the total number of primitive tuples in the data set.

A characteristic rule can be presented in logical form as shown in Example 1.2.2 or in tabular form as in Example 4.1.1. In the tabular form, since the rule antecedent (left hand side) is the same for all the rules found, it is omitted and only the right hand sides of the rules are presented in a table called the *generalized relation* [44]. Each line (tuple) in the table represents a *generalized tuple.*

Example 4.1.1 The follow table, adopted from [46], lists the characteristic rules describing professors in Applied Science. The rules are extracted from a university employee database. Note that a special attribute, *count* (called *vote* in the original paper) is added to record the number of primitive tuples covered by the generalized tuple. The count of each generalized tuple can be interpreted as the confidence of the corresponding rule.

Sex	Age	Birth_place	Salary	Count
male	old	Canada	high	20
male	mid-age	Canada	medium	50
female	mid-age	Canada	medium	8
male	mid-age	foreign	medium	21
female	mid-age	foreign	medium	1

□

Han et al. [44] developed a method called Attribute-Oriented Induction to discover high level characteristic rules from large databases. In their work, primitive level data are generalized into high level tuples using Attribute-Oriented Induction (AOI). The method is summarized as follows [13, 44].

1. Initial data collection: The data mining request is transformed into an SQL query and executed to collect the set of data relevant to the mining task (as the initial relation).

2. Derivation of the generalization plan for each attribute: If there is a large set of distinct values for an attribute of the initial relation, the attribute should be generalized by either attribute removal or attribute generalization. The former is performed when there is no generalization operator on the attribute, or its higher-level concepts are expressed in another attribute. The latter is performed otherwise by (1) determining the *prime level* (generalized) concepts for each attribute, (possibly after conceptual hierarchy adjustment), and (2) linking them with the data in the initial relation to form generalization pairs.

3. Prime relation derivation: Perform attribute-oriented generalization by substituting lower level concepts with their corresponding prime level concepts, eliminating duplicating tuples, and accumulating the counts in the retained generalized tuples. This leads to the *prime relation.*

4. Rule generation: Presentation of the generalized rules in prime relation (i.e., tabular form) or logical form.

Example 4.1.2 Suppose the original relation contains the following primitive tuples. During AOI, the attribute *Name* is removed because there is no generalization operator on it. The attribute *Sex* is untouched because it has only two values, and thus is already at the prime level. Other attributes are generalized. For example, the attribute *Birth_place* is generalized by replacing the primitive level concepts, such as "Toronto", with the prime level concepts, such as "Canada", using the conceptual hierarchy in Figure 1.3. The AOI results in the prime relation in Example 4.1.1.

Name	Sex	Age	Birth_place	Salary
F. Johnson	male	50	Toronto	$70,000
S. Smith	male	46	Vancouver	$65,000
D. Clark	female	39	Montreal	$45,000
...	...	...	...	...

□

Although AOI is a powerful tool for mining high level characteristic rules [48], direct application of the method for mining multiple-level characteristic rules, by multiple scanning with various attribute thresholds, is not very efficient. We extend the basic algorithm so that it can be used to discover multiple-level characteristic rules efficiently.

4.1.1 Methods for Mining Multiple-Level Characteristic Rules

There are two basic strategies for mining multiple-level characteristic rules: progressive specialization and progressive generalization.

- Progressive generalization starts with a conservative generalization process which first generalizes the data in the initial relation to slightly higher conceptual levels than the primitive data in the relation. Further generalizations can be performed on it progressively by selecting appropriate attributes for step-by-step generalization. The selection of the attributes for generalization can be based on some selection standards, such as the attributes with a large number of distinct values, the attributes with large compression ratios of distinct values, the attributes at deep levels in their hierarchies, the attributes with small information loss [90], etc.

- Progressive specialization starts with a relatively high-level generalized relation, then selectively and progressively specializes some of the generalized tuples or attributes to lower conceptual levels. The selection of the generalized tuples or attributes for specialization may depend on some selection standards, such as splitting the tuples with large counts, specializing the attributes with a small number of values, specializing the attributes with many levels in the hierarchy, specializing the attributes with large information gain [90], etc.

From the conceptual point of view, it is often desirable to adopt a top-down, progressive specialization process since it is natural to first find general data characteristics at a high conceptual level and then follow certain interesting paths to step

down and study specialized cases. However, from the implementation point of view, it is easy to perform generalization rather than specialization because generalization replaces low level tuples with high ones by ascension of a conceptual hierarchy. On the other hand, since generalized tuples do not register the detailed original information, it is difficult to get such information back when specialization is required later.

Besides the basic strategy, the other two major decisions that have to be made before an algorithm is possible are, what process control should be used during the mining session, i.e., automatic or manual (interactive), and what rule filtering mechanism should be adopted.

- *Automatic mining versus interactive mining.* Automatic mining means that generalization or specialization is performed automatically, and the results from all levels are presented all at once. Interactive mining means that the user interactively controls the generalization and specialization during the mining session. Interactive mining is suitable if the user is only interested in some of the rules while automatic mining is desirable if most or all the rules are interesting. Interactive mining is also desirable if the user does not know exactly what he or she is looking for and wants to explore the data.

- Usually many rules will be discovered in a data mining session, several of which may not be of interest to the user. To ensure that only strong or interesting rules will be presented, a rule filtering mechanism can be adopted. For example, strong characteristic rules can be discovered at multiple conceptual levels by filtering out generalized tuples having small counts in the rule generation process. Interestingness measurements, such as the IC^{++} [62], may be used to filter out uninteresting rules.

4.1.2 Minimally Generalized Relation

Generalization of a relation can be performed by conceptual hierarchy climbing (i.e., the replacement of lower level concepts by their corresponding higher level concepts) or attribute removal. Since these only involve conceptual hierarchies and the data in

the relation, generalization is thus easy to implement. The specialization of a relation is more complicated because, for a higher level concept, we cannot decide from which lower level concept it was generalized. For example, a high level concept in Figure 1.3, "Canada", may be generalized from anyone of the three lower level concepts: "Central Canada", "Western Canada", or "Maritime".

In this section, we illustrate a technique which facilitates specializations of generalized relations. The key point is to save a *minimally generalized relation*, which is derived from the initial relation by minimal commitment. That is, each attribute in the initial relation is generalized to minimally generalized levels (leaf nodes in the conceptual hierarchies) and then identical tuples in such a generalized relation are merged together [50].

Using the minimally generalized relation, both specialization and generalization can be performed with reasonable efficiency: if the current generalized relation R is to be further generalized, generalization can be performed directly on R; on the other hand, if R is to be specialized (e.g., by progressive specialization), it can be performed by generalizing the minimally generalized relation to the appropriate conceptual level(s) in order to derive the desired generalized relation.

Example 4.1.3 A possible generalization path for a relation with two attributes, *province* and *amount*, is shown in figure 4.1. The minimally generalized relation is derived from the initial relation by generalizing the attribute *amount* into the lowest level groupings (concepts). Further generalizations of *amount* will produce higher level relations, G_1, G_2, etc. The specialization of G_2, for example, can be realized by generalizing the minimally generalized relation into appropriate level (G_1).

□

By generalizing the concepts to the minimal level in the hierarchies, many tuples in the original relation will be merged into one in the resulting minimally generalized relation. For example, in Example 4.1.3, two tuples in the original relation, ("BC", 25033) and ("BC", 25500), will be merged into one tuple in the minimally generalized relation, ("BC", 25000–26000), as shown in Figure 4.1. The minimally generalized relation is usually much smaller than the original relation and can be stored in main

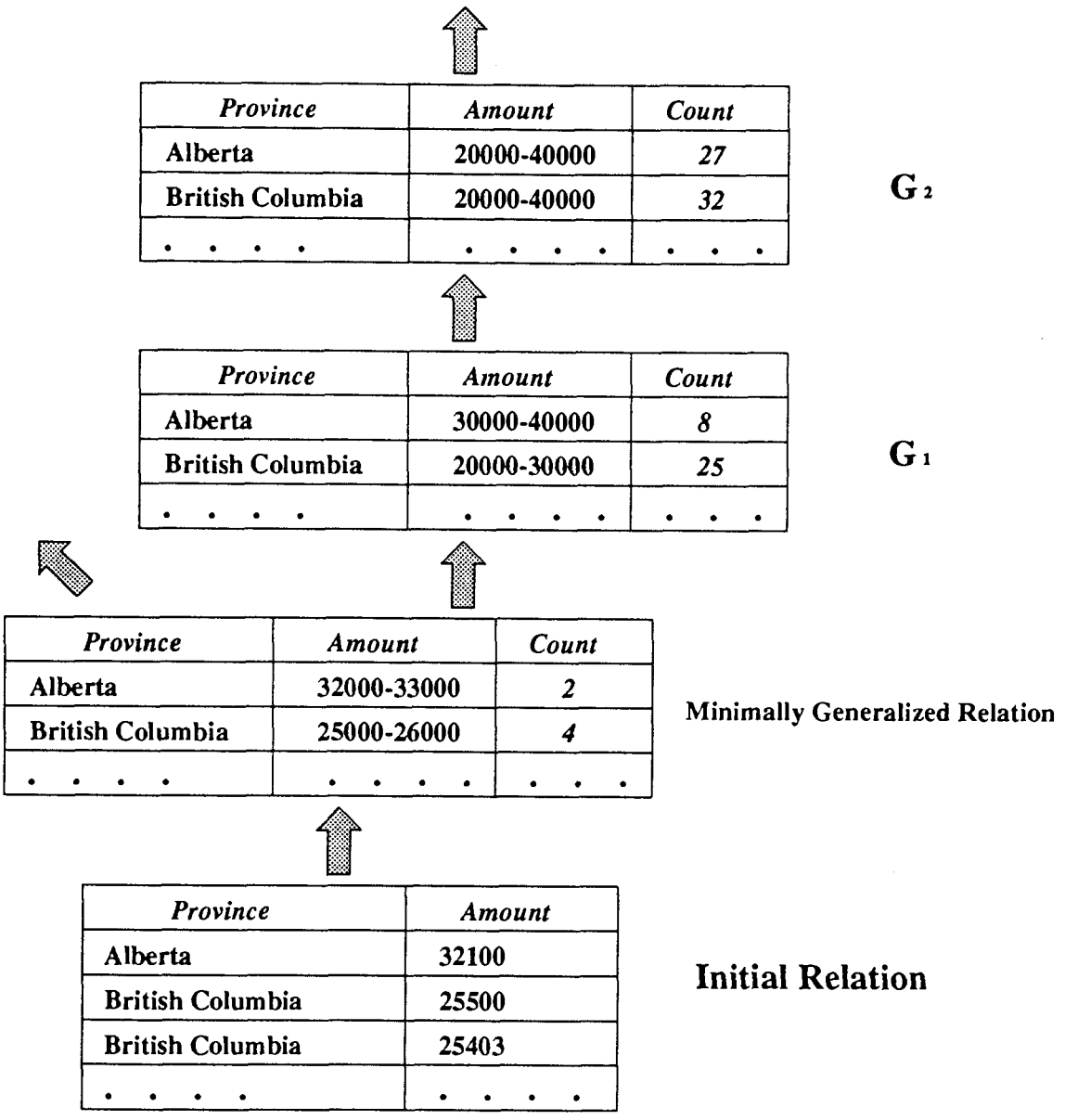

Figure 4.1: An example of the minimally generalized relation.

memory. Large minimally generalized relations can be put on disk. The derivation of a large minimally generalized relation is still beneficial because, for non-primitive characteristic rules, it avoids the query processing (joins, selections, etc.) required to obtain the initial relation.

4.1.3 An Algorithm for Mining Multiple-Level Characteristic Rules

Based on the discussion in the previous sections, an algorithm for mining multiple-level characteristic rules is presented here.

Algorithm 4.1.1 (Mining Multiple-Level Characteristic Rules) *Interactive discovery of multiple-level characteristic rules in the initial data relation.*

Input: (1) an initial relation, R_0; (2) a set of conceptual hierarchies, CH, for the attributes in R_0; (3) a set of attribute thresholds, T, for the attributes in R_0; (4) thresholds, I, for rule filtering.

Output: characteristic rules at different conceptual levels.

Method: An interactive progression method using a minimally generalized relation.

1. A minimally generalized relation, R_1, is derived from R_0 by generalizing the attributes to the minimal levels in the hierarchies and removing duplicating tuples.
2. The prime level relation is derived by attribute-oriented induction using T and CH. It is saved as the current relation.
3. Repeat the following steps until the user chooses to exit.
4. Characteristic rules are extracted from the current relation. Strong characteristic rules are filtered out using I, and presented.
5. Further progression instructions are accepted from the user. The user may select an attribute to generalize or specialize, or may end the session.

- If further generalization is desired, the selected attribute in the current relations is generalized.
- If further specialization is desired and the current relation is not at the minimal level, the attributes in the minimally generalized relation are generalized to the proper levels so as to obtain the new current relation.
- If further specialization is desired and the current relation is at the minimal level, the initial relation is fetched and generalized to obtain the new current relation.
- If exit is desired, the process stops.

□

Theorem 4.1.1 *Algorithm 4.1.1 finds characteristic rules at any conceptual level in* $O(n \log(n))$ *time, where* n *is the number of tuples in the original relation.*

Rationale. The generalizations use attribute-oriented induction whose time complexity is $O(m \log(p))$ where m is the number of tuples in the current relation and p is the number of tuples in the generalized relation [48]. The specializations are done by generalizations of the minimally generalized relation or of the initial relation. Since current relations and the minimally generalized relation have less than n tuples, the discovery of characteristic rules at any level takes at most $O(n \log(n))$ time. □

4.1.4 Experimental Results

Algorithm 4.1.1 has been implemented in DBMiner and used for the discovery of multiple-level characteristic rules. Users can interactively control the prime levels (by setting attribute thresholds) and select specialization or generalization of the current relation, and thus can find rules at multiple levels. Several attribute selection criteria are available: most distinct values, least distinct values, and information gain/loss. Users can choose the attribute to be generalized/specialized based on one of the criteria or from the attribute list. A simple policy of rule filtering using count

threshold is adopted, i.e., a generalized tuple is regarded as noise and discarded if its count is less than the given threshold.

Example 4.1.4 The following data mining query, in DMQL [51], finds multiple-level characteristic rules about 1994 NSERC research grants in "Computer Science" in terms of the recipients' province, amount of the grant, percentage of the count (number of grants), and percentage of the amount.

```
use NSERC94
find characteristic rule for 'CS_Grants'
from award A, organization O
where A.org_code = O.org_code and A.disc_code = 'Computer Science'
in relevance to province, amount, percentage(count), percentage(amount)
```

Using the hierarchy in Figure 1.3 for *province*, the rules found at second and third levels are shown as follows. Default count threshold, 0%, is used.

```
        ****************************************
        *   Characteristic Rules at Level 2    *
        ****************************************
-------------------------------------------------------------
amount              province              amount%    count%
-------------------------------------------------------------
60Ks-               Central Canada         9.39%      2.40%
60Ks-               Western Canada         5.53%      1.71%
40Ks-60Ks           Central Canada        10.76%      5.14%
40Ks-60Ks           Western Canada         3.08%      1.54%
20Ks-40Ks           Central Canada        23.31%     20.72%
20Ks-40Ks           Maritime               1.17%      1.03%
20Ks-40Ks           Western Canada        14.69%     12.84%
0-20Ks              Central Canada        19.89%     33.56%
0-20Ks              Maritime               3.21%      5.99%
0-20Ks              Western Canada         8.97%     15.07%
-------------------------------------------------------------
                                         100.00%    100.00%
Total number of primitive tuples: 584

        ****************************************
        *   Characteristic Rules at Level 3    *
```

****

amount	province	amount%	count%
60Ks-	Quebec	2.26%	0.68%
60Ks-	Ontario	7.13%	1.71%
60Ks-	Prairies	3.59%	1.03%
60Ks-	British Columbia	1.94%	0.68%
40Ks-60Ks	Quebec	1.88%	0.86%
40Ks-60Ks	Ontario	8.88%	4.28%
40Ks-60Ks	Prairies	0.65%	0.34%
40Ks-60Ks	British Columbia	2.43%	1.20%
20Ks-40Ks	Quebec	6.43%	5.65%
20Ks-40Ks	Ontario	16.88%	15.07%
20Ks-40Ks	Newfoundland	0.37%	0.34%
20Ks-40Ks	Nova Scotia	0.57%	0.51%
20Ks-40Ks	New Brunswick	0.23%	0.17%
20Ks-40Ks	Prairies	8.26%	7.36%
20Ks-40Ks	British Columbia	6.43%	5.48%
0-20Ks	Quebec	8.03%	14.04%
0-20Ks	Ontario	11.86%	19.52%
0-20Ks	Prince Edward Isl	0.07%	0.17%
0-20Ks	Newfoundland	0.80%	1.37%
0-20Ks	Nova Scotia	1.38%	2.57%
0-20Ks	New Brunswick	0.95%	1.88%
0-20Ks	Prairies	4.95%	8.39%
0-20Ks	British Columbia	4.01%	6.68%
		100.00%	100.00%

Total number of primitive tuples: 584

□

4.2 Multiple-Level Discriminant Rules

Discriminant rules present the properties that make one data set (the target class) distinct from the other(s) (the contrasting class(es)). As for characteristic rules, the confidence of a discriminant rule, called t-weight, tells how significant the rule is. Unlike the characteristic rule, a discriminant rule has another interestingness

measure, discriminant weight or d-weight [44], which tells how good the rule is as a discriminator.

Definition 4.2.1 A discriminant rule is a rule whose left hand side is a generalized tuple, i.e., a conjunctions of (attribute, value) pairs, and whose right hand side is the query condition for the target class.

The *t-weight* of a discriminant rule is the ratio of the number of primitive tuples covered by the generalized tuple over the total number of tuples, within a class. The *d-weight* of a discriminant rule is the ratio the number of primitive tuples covered by the generalized tuple in one class versus the number of tuples covered by the generalized tuple in both classes [44].

The attribute-oriented induction method proposed by Han et al. also discovers high level discriminant rules from large databases [44], as summarized below.

1. Collect the relevant set of data respectively into the target class and the contrasting class(es).

2. Extract the *prime target relation* (the prime relation corresponding to the initial relation in the target class) in a similar way as the attribute-oriented induction in mining characteristic rules. Generalize the concepts of the initial relation(s) in the contrasting class(es) to the same level as those in the prime target relation, which results in the *prime contrasting relation(s)*.

3. To generate discriminant rules, compute the d-weight for each generalized tuple in the target class and output these tuples whose d-weight is close to 100%, along with the d-weight.

Similar to characteristic rules, discriminant rules can also be presented in logical form or in tabular form by listing the left hand side of rules in a table.

Example 4.2.1 The following table lists discriminant rules about professors versus instructors in Applied Science, discovered from a university employee database. It is excerpted from [44] with the transformation of *Vote* into t-weight and d-weight.

Tuples appearing in both classes are marked with "*".

Class	Sex	Age	Birth_place	Salary	t-weight	d-weight	Mark
target class: professors	male	old	Canada	high	20%	100%	
	male	mid_age	Canada	medium	50%	77%	*
	female	mid_age	Canada	medium	8%	100%	
	male	mid_age	foreign	medium	21%	95%	*
	female	mid_age	Canada	medium	1%	100%	
contrasting class: instructors	male	young	Canada	low	60%	100%	
	male	mid-age	Canada	medium	30%	23%	*
	female	young	Canada	low	8%	100%	
	male	mid-age	foreign	medium	2%	5%	*

These tuples in the table can be easily transformed into logical form. For example, the first generalized tuple can be transformed into:

IF Sex(x) is "male" AND Age(x) is "old" AND
Birth_place(x) is "Canada" AND Salary(x) is "high"
THEN professor(x). (t=20%, d=100%)

□

Similar extensions to AOI for mining multiple-level characteristic rules will enable AOI to discover multiple-level discriminant rules.

4.2.1 Methods for Mining Multiple-Level Discriminant Rules

Like characteristic rules, we can use progressive specialization and progressive generalization to find multiple-level discriminant rules.

- Progressive specialization first generalizes the data in both classes to a rather high level and finds the discriminant rules at that level. Some attribute selection criteria, such as the ones used for the mining of characteristic rules, can then be used to specialize both classes and discover discriminant rules at lower levels.

- Progressive generalization finds discriminant rules at a very low level first. Attributes are selectively generalized for both classes, and higher level discriminant rules are found.

Similarly, the generalization/specialization process can be performed automatically, or be controlled by users. Strong discriminant rules can be obtained by filtering out those rules based on t-weight, d-weight, or some interestingness measurement [98, 84].

4.2.2 An Algorithm for Mining Multiple-Level Discriminant Rules

Based on the discussion in the previous section, an algorithm for mining multiple-level discriminant rules is presented. For simplicity, we assume there is only one contrasting class. However, the algorithm can easily be extended for multiple contrasting classes. Furthermore, we assume the target class and the contrasting class have the same set of attributes and use the same conceptual hierarchies.

Algorithm 4.2.1 (Mining Multiple-Level Discriminant Rules) *Interactive discovery of discriminant rules at different conceptual levels from the initial relations.*

Input: (1) an initial target relation, R_t; (2) an initial contracting relation, R_c; (3) a set of conceptual hierarchies, CH, for the attributes; (4) a set of attribute thresholds, T; (5) thresholds, I, for rule filtering.

Output: discriminant rules at different conceptual levels.

Method: an interactive progression method using minimally generalized relations.

1. A minimally generalized relation for the target class, R'_t, is derived from R_t by generalizing the attributes to the minimal levels in the hierarchies and removing duplicating tuples.
2. A minimally generalized relation for the contrasting class, R'_c, is derived from R_c similarly.

3. The prime level target relation and contrasting relation are derived by attribute-oriented induction using CH and T. The relations are saved as the current target relation and the current contrasting relation, respectively.
4. Repeat the following steps until the user chooses to exit.
5. Discriminant tuples are extracted by intersecting the current target relation and the current contrasting relation. They are passed to a filter which uses I. Interesting discriminant rules are presented to the user.
6. Further progression instructions are accepted from the user. The user may select an attribute to generalize or specialize, or may end the session.
   - If further generalization is desired, the selected attribute in both current relations is generalized.
   - If further specialization is desired and the current relations are not at the minimal level, the attributes in the minimally generalized relations are generalized to the proper levels in order to obtain the new current relations.
   - If further specialization is desired and the current relations are at the minimal level, the initial relations are fetched and generalized in order to obtain the new current relations.
   - If exit is desired, the process stops.

□

Theorem 4.2.1 *Algorithm 4.2.1 can find discriminant rules at any conceptual level in $O(n_t \log(n_t) + n_c \log(n_c))$ time, where n_t is the number of tuples in the original target relation and n_c is the number of tuples in the original contrasting relation.*

Rationale. Step 1 takes $O(n_t \log(l_t))$ time where l_t is the number of tuples in the minimally generalized target relation. Step 2 takes $O(n_c \log(l_c))$ time where l_c is the number of tuples in the minimally generalized contrasting relation. Step 3 takes $O(l_t \log(p_t) + l_c \log(p_c))$ where p_t and p_c are the number of tuples in the current target

and contrasting relations, respectively. Since intersecting, filtering, and presentation are linear to the number of tuples, Step 4 needs $O(p_t + p_c)$ time. Step 5 takes at most $O(n_t \log(p_t) + n_c \log(p_c))$ time. Since $l_t < n_t$, $p_t < n_t$, $l_c < n_c$, $p_c < n_c$, the total time complexity is $O(n_t \log(n_t) + n_c \log(n_c))$.

□

It should be pointed out that direct generalization and specialization of discriminant tuples may not give correct discriminant rules. For example, a discriminant tuple (age:"25–30", gpa:"3.5–3.7") may be unique for graduate students, but probably not the further generalized tuple, (age:"20–30", gpa:"3.5–3.7"), or the further specialized tuple, (age:"28–29", gpa:"3.5–3.7") – because there is simply no such kind of graduate students. Therefore, generalization and specialization of both classes in step 6 of the algorithm 4.2.1 are necessary.

4.2.3 Experimental Results

Algorithm 4.2.1 was implemented in the DBMiner system to find multiple-level discriminant rules. Users select the first generalization level by specifying attribute thresholds, and interactively generalize or specialize the current relations to discover discriminant rules at different levels. A difference with respect to the implementation for mining multiple-level characteristic rules is that a minimally generalized relation is saved for the contrasting class as well. A simple rule filtering mechanism using a t-weight threshold and a d-weight threshold is adopted to filter out the uninteresting rules.

Example 4.2.2 The following data mining query, in DMQL [51], finds discriminant rules that distinguish 1994 computer science research grants in "Alberta" from those in "Newfoundland", in terms of discipline, type of grant, and amount of grant.

```
use NSERC94
find discriminant rule for ''Alberta_CS_Grants''
where 0.province = ''Alberta''
```

```
in contrast to ''Newfoundland_CS_Grants''
where O.province = ''Newfoundland''
from award A, organization O, grant_type G
where A.grant_code = G.grant_code and
      A.org_code = O.org_code and A.disc_code = ''Computer''
in relevance to disc_code, amount, grant_order
```

Default t-weight threshold, 0%, and d-weight threshold, 100% are used. The results are shown as follows. The prime level target relation and contrasting relation are shown as well as the discriminant tuples. The attribute threshold is set to the default, five, for all attributes. Since the lower level of the attribute *disc_code* (discipline) has more than five values, the attribute is generalized to the current level which has no more than five values. The overlapping tuples (tuples appeared in both classes) are marked with "*".

```
            ****************************************
            *        Alberta_CS_Grants             *
            ****************************************
-------------------------------------------------------------------
disc_code    grant_order       amount     t-weight  d-weight  mark
-------------------------------------------------------------------
Computer   Operating Grant     0-20Ks      40.74%    73.33%    *
Computer   Operating Grant     20Ks-40Ks   44.44%    96.00%    *
Computer   Operating Grant     40Ks-60Ks    1.85%   100.00%
Computer   Structure Grant     0-20Ks       1.85%   100.00%
Computer   Structure Grant     20Ks-40Ks    5.56%    75.00%    *
Computer   Structure Grant     60Ks-        5.56%   100.00%

            ****************************************
            *        Newfoundland_CS_Grants        *
            ****************************************
-------------------------------------------------------------------
disc_code    grant_order       amount     t-weight  d-weight  mark
-------------------------------------------------------------------
Computer   Operating Grant     0-20Ks      80.00%    26.67%    *
Computer   Operating Grant     20Ks-40Ks   10.00%     4.00%    *
Computer   Structure Grant     20Ks-40Ks   10.00%    25.00%    *
```

Discriminant Tuples

disc_code	grant_order	amount	t-weight	d-weight
Computer	Operating Grant	40Ks-60Ks	20.00%	100.00%
Computer	Structure Grant	0-20Ks	20.00%	100.00%
Computer	Structure Grant	60Ks-	60.00%	100.00%

From the tables, we can see that Operating Grants from $40,000 to $60,000 are awarded to Alberta only. Some tuples, like Operating Grants from $20,000 to $40,000, are most probably in Alberta (96%). Lowering the *d*-weight threshold will qualify this kind of tuples as discriminant tuples.

If the user wants to see more detailed discriminant tuples, he/she can specialize an attribute, for example, *disc_code* (having least distinct values). Lower level target relation and contrasting relation are displayed below, together with the discriminant tuples at the lower level. As we can see from the table, the specialization brings out some new discriminant tuples, for example, the tuple ("DATABASES", "Operating Grant", "0-20Ks"), and breaks down the high level discriminant tuples. For example, the tuple ("Computer", "Structure Grant", "0-20Ks") is replaced by the lower level tuple ("SOFTWARE", "Structure Grant", "0-20Ks").

Alberta_CS_Grants

disc_code	grant_order	amount	t-weight	d-weight	mark
HARDWARE	Operating Grant	0-20Ks	1.85%	50.00%	*
SYS_ORGANIZATION	Operating Grant	0-20Ks	1.85%	100.00%	
SYS_ORGANIZATION	Operating Grant	20Ks-40Ks	3.70%	100.00%	
SOFTWARE	Operating Grant	0-20Ks	3.70%	66.67%	*
SOFTWARE	Operating Grant	20Ks-40Ks	7.41%	100.00%	
SOFTWARE	Operating Grant	40Ks-60Ks	1.85%	100.00%	
SOFTWARE	Structure Grant	0-20Ks	1.85%	100.00%	
SOFTWARE	Structure Grant	20Ks-40Ks	1.85%	100.00%	
THEORY	Operating Grant	0-20Ks	12.96%	77.78%	*

THEORY	Operating Grant	20Ks-40Ks	7.41%	80.00%	*
DATABASES	Operating Grant	0-20Ks	3.70%	100.00%	
DATABASES	Operating Grant	20Ks-40Ks	1.85%	100.00%	
DATABASES	Structure Grant	20Ks-40Ks	1.85%	100.00%	
AI	Operating Grant	0-20Ks	11.11%	75.00%	*
AI	Operating Grant	20Ks-40Ks	9.26%	100.00%	
AI	Structure Grant	20Ks-40Ks	1.85%	100.00%	
COMP_METHODS	Operating Grant	0-20Ks	5.56%	60.00%	*
COMP_METHODS	Operating Grant	20Ks-40Ks	14.81%	100.00%	
COMP_METHODS	Structure Grant	60Ks-	5.56%	100.00%	

****
* Newfoundland_CS_Grants *
****

disc_code	grant_order	amount	t-weight	d-weight	mark
HARDWARE	Operating Grant	0-20Ks	10.00%	50.00%	*
SOFTWARE	Operating Grant	0-20Ks	10.00%	33.33%	*
THEORY	Operating Grant	0-20Ks	20.00%	22.22%	*
THEORY	Operating Grant	20Ks-40Ks	10.00%	20.00%	*
AI	Operating Grant	0-20Ks	20.00%	25.00%	*
COMP_METHODS	Operating Grant	0-20Ks	20.00%	40.00%	*
COMP_METHODS	Structure Grant	20Ks-40Ks	10.00%	100.00%	

****
* Discriminant Tuples *
****

disc_code	grant_order	amount	t-weight	d-weight
SYS_ORGANIZATION	Operating Grant	0-20Ks	3.23%	100.00%
SYS_ORGANIZATION	Operating Grant	20Ks-40Ks	6.45%	100.00%
SOFTWARE	Operating Grant	20Ks-40Ks	12.90%	100.00%
SOFTWARE	Operating Grant	40Ks-60Ks	3.23%	100.00%
SOFTWARE	Structure Grant	0-20Ks	3.23%	100.00%
SOFTWARE	Structure Grant	20Ks-40Ks	3.23%	100.00%
DATABASES	Operating Grant	0-20Ks	6.45%	100.00%
DATABASES	Operating Grant	20Ks-40Ks	3.23%	100.00%
DATABASES	Structure Grant	20Ks-40Ks	3.23%	100.00%
AI	Operating Grant	20Ks-40Ks	16.13%	100.00%

AI	Structure Grant	20Ks-40Ks	3.23%	100.00%
COMP_METHODS	Operating Grant	20Ks-40Ks	25.81%	100.00%
COMP_METHODS	Structure Grant	60Ks-	9.68%	100.00%

4.3 Summary and Discussion

Issues on the discovery of multiple-level characteristic and discriminant rules have been examined and a set of algorithms has been developed and tested in our DB-Miner system. The experiments show that they can discover interesting multiple-level characteristic and discriminant rules effectively and efficiently.

- An interactive progression method has be presented for mining multiple-level characteristic and discriminant rules. It uses attribute-oriented induction as the basic tool for data generalization. A data structure, *minimally generalized relation*, is used to implement the specialization efficiently.
- Interstingness measures, such as the confidence of the characteristic rules, and the t-weight and d-weight of the discriminant rules, are employed to filter out uninteresting rules.

The progressive specialization (deepening) method is further studied in the next chapter when we discuss the mining of multiple-level association rules.

4.3.1 Characterization and On-Line Analytical Processing

On-Line Analytical Processing (OLAP) was introduced by Codd [21] to characterize the dynamic enterprise data analysis, such as summarization, grouping, synthesis, consolidation, etc. Multidimensional databases are the common approach to support OLAP [53, 1, 60, 43]. A multidimensional database can be viewed as a hypercube, in which each dimension of the hypercube is an attribute and some attributes are treated as measures, whose values compose the cell of the hypercube.

In this section, we discuss the relationships between data mining, especially the characterization, and OLAP.

Similarities and Differences between Characterization and OLAP

Characterization and OLAP have several features in common as described below.

- They both perform multi-dimensional analysis on large databases. Characterization may find characteristic rules with multiple attributes, and OLAP are usually performed on multidimensional databases.
- Several data analysis operations are employed by both, including viewing data from different angles ("pivoting"), presenting multiple-level prospects of the data ("drill-down" and "roll-up"), selecting interesting subset of the data ("slicing" and "dicing"), and grouping and aggregation.
- Conceptual hierarchies for the attributes (i.e., dimensions) are used to group the data and define the levels of the concepts.

However, characterization and OLAP are quite different as described below.

- Multidimensional databases are often built from the entire database and is stored for all the OLAP operations, where as characterization mostly works on a dynamically collected initial relation.
- Multidimensional databases are usually implemented using materialized views [53, 93, 114], and/or multidimensional indexing structures [68]. Characterization, on the other hand, does not use materialized views or special indexing structures.
- Conceptual hierarchies can be dynamically adjusted or automatically generated for characterization, whereas multidimensional databases usually use given, fixed conceptual hierarchies.

Interactions between Data Mining and OLAP

It is interesting to study what data mining and OLAP can do for each other.

- Algorithms developed for manipulations of conceptual hierarchies can be used to enhance the flexibilities of hierarchy processing in OLAP. For example, the conceptual hierarchy of a dimension, *country*, can be dynamic adjusted based

on the current sales figures, so that the groupings of the countries reflect the current sales distributions.

- Some techniques developed for OLAP can be used for data mining. For example, the materialized view techniques can be used to speed up generalization and specialization processes by actually storing some intermediate relations.

4.3.2 More about Discriminant Rules

In this section, two issues relating to discriminant rules are discussed: the relationships between discriminant rules and classification rules, and the visualization of discriminant rules.

- Relationships between discriminant rules and classification rules.
  Discriminant rules and classification rules are similar in that they both deal with tuples (objects) from different classes and try to model the differences between the classes using the attributes of the tuples. However, they are different in their purposes and approaches. Their relationships can be further explained as follows.

  - The purpose of discriminant rules is to identify the features that distinguish the target class from the contrasting classes. A tabular form (relation) usually is preferred to give a clear and uniform view. The purpose of classification rules is to classify the objects. A classifier consisting of a set of classification rules is built from the given objects. The classifier is used later to classify future objects.
  - Discriminant rules are centered around the target class whereas classification rules treat all classes equally. To make a full classification, discriminant rule should be extracted for each of the classes.
  - Compact classification rules can be extracted by further refinement of discriminant rules. For example, the contrasting class in Example 4.2.2, "Newfoundland_CS_Grants", does not have any grants over $40,000. A compact rule,

IF amount(x) > 40,000 THEN x is a grant in Alberta_CS_Grants.

can be induced which abstracts the two discriminant tuples at the prime level. This can be done using rule reduction techniques [91].

- Classification rules cannot replace discriminant rules. For example, the above rule covers two tuples which are not presented in the discriminant tuples: ("Computer", "Operating Grant", "60Ks-") and ("Computer", "Structure Grant", "40Ks-60Ks"). For the same reason, multiple-level discriminant rules are necessary because rules at different levels give unique discriminant tuples which cannot be derived from discriminant rules at other levels.

- Visualization of discriminant rules

  Characteristic rules can be presented in textual forms, such as prime relations and feature tables [45], and graphical forms, such as bar charts, pie charts, etc [48]. For discriminant rules, we can use similar techniques. For example, three relations or charts can be presented, one for the target class, one for the contrasting class, and one for the discriminant tuples [51]. In the textual forms, overlapping tuples in both classes are marked with "*", as shown in Example 4.2.2. However, it would be interesting to see that tuples in both classes are presented in one graph, with the overlapping tuples expressed in a graphical form, such as different colors, textures, etc.

Chapter 5

Mining Multiple-Level Association Rules

5.1 Introduction

With widespread applications of computers and automated data collection tools, massive amounts of transaction data have been collected and stored in databases. Discovery of interesting association relationships among huge amounts of data will help marketing, decision making, and business management. Therefore, mining association rules from large data sets has been a focused topic in recent research into knowledge discovery in databases [2, 4, 5, 65, 82, 85].

Studies on mining association rules have evolved from techniques for discovery of functional dependencies [71], strong rules [85], classification rules [46, 91], causal rules [77], clustering [34], etc. to disk-based, efficient methods for mining association rules in large sets of transaction data [2, 4, 5, 82]. However, previous work has been focused on mining association rules at a single conceptual level. There are applications which need to find associations at multiple conceptual levels. For example, besides finding that *80% of customers that purchase* milk *may also purchase* bread, it could be informative to also show that *75% of customers that purchase* wheat bread *may also purchase* 2% milk. The association relationship in the latter statement is expressed at a lower conceptual level but often carries more specific and concrete information than

that in the former. This requires progressively "deepening" the knowledge mining process in the search for refined knowledge from data. The necessity for mining multiple-level association rules or for using taxonomy information to aid the mining has also been observed by other researchers, e.g., [104, 4].

In order to confine the association rules discovered to *strong* ones, that is, patterns (or *itemsets*) which occur relatively frequently and rules which demonstrate relatively strong implication relationships, the concepts of *minimum support* and *minimum confidence* have been introduced [2, 4]. Informally, the support of an itemset A in a set of transactions S is the probability that a transaction in S contains itemset A; and the confidence of $A \rightarrow B$ in S is the probability that itemset B occurs in S if itemset A occurs in S.

For the mining of multiple-level association rules, a concept taxonomy should be provided to allow the generalization of primitive level concepts to high level concepts. In many applications, the taxonomy information is either stored implicitly in the database, such as "*Wonder wheat bread is a wheat bread which is in turn a bread*", or obtained elsewhere as discussed in Section 3.1.3. Thus, data items can be easily generalized to multiple conceptual levels. However, direct application of the existing association rule mining methods to mining multiple-level associations may lead to some undesirable results as described below.

First, large support is more likely to exist at high conceptual levels, such as *milk* and *bread*, rather than at low conceptual levels, such as a *particular brand* of milk and bread. Therefore, if one wants to find strong associations at relatively low conceptual levels, the minimum support threshold must be substantially reduced. However, this may lead to the generation of many uninteresting associations, such as "toy $\rightarrow$ milk" before the discovery of some interesting ones, such as "Dairyland 2% milk $\rightarrow$ Wonder wheat bread", because the former may occur more frequently and thus have larger support than the latter.

Second, it is unlikely to find many strong association rules at a primitive conceptual level, such as the associations among particular bar codes, because of the tiny average support for each primitive data item in a very large item set. However, mining association rules at high conceptual levels may often lead to the rules corresponding

to prior knowledge and expectations [65], such as "milk $\rightarrow$ bread", or lead to some uninteresting attribute combinations, such as "toy $\rightarrow$ milk".

In order to remove uninteresting rules generated in knowledge mining processes, researchers have proposed some measurements to quantify the "usefulness" or "interestingness" of a rule [84] and have suggested to "put a human in the loop" by providing tools to allow human guidance of the knowledge discovery process [7]. Nevertheless, automatic generation of relatively focused, informative association rules is obviously more efficient than first generating a large mixture of rules and then having to distinguish the interesting rules from the uninteresting ones.

These observations lead us to examine the methods for mining association rules at multiple conceptual levels, which may not only discover rules at different levels but also have high potential to find nontrivial, informative association rules because of its flexibility at focusing the attention to different sets of data and applying different thresholds at different levels.

Srikant and Agrawal [104] proposed using a taxonomy of items to find general level association rules. However, they used the same thresholds for all of the levels, allowing many uninteresting rules to be found with interesting ones. On the contrary, we use different thresholds at different levels and thus are more effective and flexible in finding interesting rules. Furthermore, we investigate data focusing and different optimization techniques for mining multiple-level association rules in our approach.

In this chapter, issues for mining multiple-level association rules from large databases are examined, with a top-down, progressive deepening method developed by extension of some existing algorithms for mining single-level association rules. The method first finds large (i.e., frequent) itemsets at the top-most level and then progressively deepens the mining process into their large descendants at lower conceptual levels. Some data structures and intermediate results generated while mining high level associations can be shared for the mining of lower level ones, and different sharing schemes lead to different variant algorithms. Algorithm performance identifies the conditions that each algorithm is best suited, with regard to different kinds of data distributions and thresholds.

The chapter is organized as follows. In Section 5.2, the concepts related to

multiple-level association rules are introduced. In Section 5.3, a method for mining multiple-level association rules in large data sets is studied. In Section 5.4, a set of variant algorithms for mining multiple-level association rules are introduced, with their relative efficiency analyzed. In Section 5.5, a performance study is conducted on different kinds of data distributions and thresholds, which identifies the conditions for algorithm selection. Section 5.6 extends the basic ideas to the mining of flexible association rules. Section 5.7 discusses related issues on mining multiple-level association rules, such as using multiple hierarchies, interestingness of rules, and several other issues. The study on mining multiple-level association rules is summarized in Section 5.8.

5.2 Multiple-Level Association Rules

To study the mining of association rules from a large set of transaction data, we assume that the database contains (1) a transaction data set, $\mathcal{T}$, which consists of a set of transactions $\langle T_i, \{A_p, \ldots, A_q\}\rangle$, where T_i is a transaction identifier, $A_i \in \mathcal{I}$ (for $i = p, \ldots, q$), and $\mathcal{I}$ is the set of all the data items in the item data set; and (2) the description of the item data set, $\mathcal{D}$, which contains the description of each item in $\mathcal{I}$ in the form of $\langle A_i, description_i\rangle$, where $A_i \in \mathcal{I}$.

Furthermore, to facilitate the management of large sets of transaction data, our discussion adopts an extended relational model which allows an attribute value to be either a single or a set of values (i.e., in non-first-normal form). Nevertheless, the method developed here is applicable (with minor modifications) to other representations of data, such as a data file, a relational table, or the result of a relational expression.

Definition 5.2.1 An itemset, A, is a set of data items $\{A_i, \cdots, A_j\}$, where A_i, ..., $A_j \in \mathcal{I}$. The **support** of an itemset A in a set S, $\sigma(A/S)$, is the number of transactions (in S) which contain A versus the total number of transactions in S. The **confidence** of $A \rightarrow B$ in S, $\varphi(A \rightarrow B/S)$, is the ratio of $\sigma(A \cup B/S)$ versus $\sigma(A/S)$, i.e., the probability that itemset B occurs in S when itemset A occurs in S.

If an itemset contains only one item, the item is sometimes used to represent the itemset. It is obvious from the definition that the support of an itemset cannot be larger than any of its subsets.

To find relatively frequently occurring itemsets and reasonably strong rule implications, a user or an expert may specify two thresholds: *minimum support*, σ', and *minimum confidence*, φ'. Notice that for finding multiple-level association rules, different minimum support and/or minimum confidence thresholds can be specified at different levels.

Definition 5.2.2 An itemset A is large in set S at level l if the support of A is no less than its corresponding minimum support threshold σ'_l. The confidence of a rule "$A \rightarrow B/S$" is high at level l if its confidence is no less than its corresponding minimum confidence threshold φ'_l.

Definition 5.2.3 A rule "$A \rightarrow B/S$" is strong if, for a set S, each ancestor (i.e., the corresponding high level item in the taxonomy) of every item in A and B, if any, is large at its corresponding level, "$A \cup B/S$" is large (at the current level), and the confidence of "$A \rightarrow B/S$" is high (at the current level).

The definition indicates that if "$A \rightarrow B/S$" is strong, then (1) $\sigma(A \cup B/S) \geq \sigma'$, (and thus, $\sigma(A/S) \geq \sigma'$, and $\sigma(B/S) \geq \sigma'$), and (2) $\varphi(A \rightarrow B/S) \geq \varphi'$, at its corresponding level. It also represents a filtering process which confines the itemsets to be examined at lower levels to be only those with large supports at their corresponding high levels (and thus avoids the generation of many meaningless combinations formed by the descendants of the small (rare) itemsets). For example, in a sales_transaction data set, if milk is a large item, its lower level items such as 2% milk will be examined; whereas if fish is a small item, its descendants such as salmon will not be examined further.

Based on this definition, the idea of mining multiple-level association rules is illustrated below.

Example 5.2.1 Suppose that a shopping transaction database consists of two relations: (1) a *sales_item* (description) relation (Table 5.1), which consists of a set of

attributes: *bar_code, category, brand, content, size, storage_period, price*, and (2) a *sales_transaction* table (Table 5.2), which registers for each transaction, the transaction number and the set of items purchased.

Let the query be to find multiple-level strong associations in the database for the purchase patterns related to the foods which can only be stored for less than three weeks. The query can be expressed as follows in DMQL [51].

find association rules
from sales_transactions T, sales_item I
where T.bar_code = I.bar_code and I.category = "food" and I.storage_period < 21
with interested attributes category, content, brand

bar_code	category	brand	content	size	storage_pd	price
17325	milk	Foremost	2%	1 (ga.)	14 (days)	$3.89
...	...	...	...	...	...	...

Table 5.1: A sales_item (description) relation.

transaction_id	bar_code_set
351428	{17325, 92108, 55349, 88157, ...}
982510	{92458, 77451, 60395, ...}
...	{..., ...}

Table 5.2: A sales_transaction table.

The query is first transformed into a standard SQL query which retrieves all the data items within the "food" category (covers high level concepts: *beverage, fruit, vegetable, bread, milk, meat, fish, cereal*, etc.) and with the storage period of less than 21 days.

Since there are only three interested attributes, *category, content*, and *brand* in the query, the sales_item description relation is generalized into a generalized sales_item description table, as shown in Table 5.3, in which each tuple represents a generalized

GID	bar_code_set	category	content	brand
112	{17325, 31414, 91265}	milk	2%	Foremost
141	{29563, 77454, 89157}	milk	skim	Dairyland
171	{73295, 99184, 79520}	milk	chocolate	Dairyland
212	{88452, 35672, 31205}	bread	wheat	Wonder
...	{..., ...}	...	...	...
711	{32514, 78152}	fruit_juice	orange	Minute_maid

Table 5.3: A generalized sales_item description table.

item which is the merge of a group of tuples which share the same values in the interested attributes. For example, the tuples with the same *category, content* and *brand* in Table 5.1 are merged into one, with their *bar codes* replaced by a bar_code set. Each group is then treated as an atomic item in the generation of the lowest level association rules. For example, the association rule generated regarding milk will only be in relevance to (at the low conceptual levels) *brand* (such as Dairyland) and *content* (such as 2%) but not to *size, producer*, etc.

The taxonomy information is provided implicitly in Table 5.3. Let *category* (such as "milk") represent the first-level concept, *content* (such as "2%") for the second level one, and *brand* (such as "Foremost") for the third level one. The table implies a conceptual hierarchy like Figure 5.1. We use a simple coding scheme for fast generalization and specialization. Other concise representations for hierarchies are also possible [31].

The process of mining association rules is expected to first discover large itemsets and strong association rules at the top-most conceptual level. Let the minimum support at this level be 5% and the minimum confidence be 50%. One may find the following: a set of single large items (each called a large 1-itemset, with the support ratio in parentheses): "bread (25%), meat (10%), milk (20%), ..., vegetable (30%)", a set of pair-wised large items (each called a large 2-itemset): "{vegetable, bread (19%)}, {vegetable, milk (15%)}, ..., {milk, bread (17%)}", etc. and a set of strong association rules, such as "bread $\rightarrow$ vegetable (76%), ..., milk $\rightarrow$ bread (85%)".

At the second level, only the transactions which contain the large items at the first

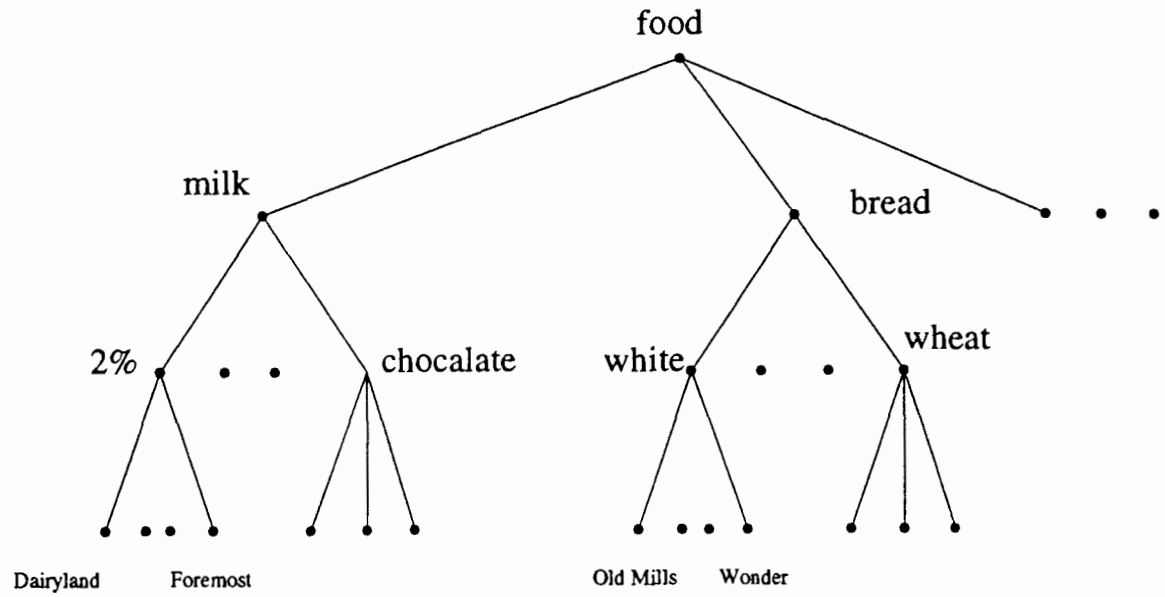

Figure 5.1: A taxonomy for the relevant data items.

level are examined. Let the minimum support at this level be 2% and the minimum confidence be 40%. One may find the following large 1-itemsets: "lettuce (10%), wheat bread (15%), white bread (10%), 2% milk (10%), chicken (5%), ..., beef (5%)", and the following large 2-itemsets: "{2% milk, wheat bread (6%)}, {lettuce, 2% milk (4%)}, {chicken, beef (2.1%)}", and the strong association rules: "2% milk $\rightarrow$ wheat bread (60%), ..., beef $\rightarrow$ chicken (42%)", etc.

The process repeats at even lower conceptual levels until no more large itemsets can be found. □

5.3 A Method for Mining Multiple-Level Association Rules

A method for mining multiple-level association rules is introduced in this section. The method uses a hierarchy-information encoded transaction table, instead of the original transaction table, in iterative data mining. This is based on the following considerations. First, a data mining query is usually in relevance to only a portion of

the transaction database, such as *food*, instead of all the items. It is beneficial to first collect the relevant set of data and then work repeatedly on the task-relevant set. Second, encoding can be performed during the collection of task-relevant data, and thus there is no extra "encoding pass" required. Third, an encoded string, which represents a position in a hierarchy, requires less bits than the corresponding object-identifier or bar-code. Moreover, encoding allows more items to be merged (or removed) due to their identical encoding, which further reduces the size of the encoded transaction table. Thus it is often beneficial to use an encoded table although our method does not rely on the derivation of such an encoded table because the encoding can always be performed on the fly.

To simplify our discussion, an abstract example which simulates the real life example of Example 5.2.1 is analyzed as follows.

Example 5.3.1 As stated above, the taxonomy information for each (grouped) item in Example 5.2.1 is encoded as a sequence of digits in the transaction table $\mathcal{T}[1]$ (Table 5.4). For example, the item '2% Foremost milk' is encoded as '112' in which the first digit, '1', represents 'milk' at level-1, the second, '1', for '2% (milk)' at level-2, and the third, '2', for the brand 'Foremost' at level-3. Similar to [4], repeated items (i.e., items with the same encoding) at any level will be treated as one item in one transaction.

TID	Items
T_1	{111, 121, 211, 221}
T_2	{111, 211, 222, 323}
T_3	{112, 122, 221, 411}
T_4	{111, 121}
T_5	{111, 122, 211, 221, 413}
T_6	{211, 323, 524}
T_7	{323, 411, 524, 713}

Table 5.4: Encoded transaction table: $\mathcal{T}[1]$.

The derivation of the large itemsets at level 1 proceeds as follows. Let the minimum support at level 1 be 4 transactions (i.e., $minsup[1] = 4$). Notice that since the total

number of transactions is fixed, the support is expressed in an absolute value rather than a relative percentage, for simplicity. The level-1 large 1-itemset table $\mathcal{L}[1,1]$ can be derived by scanning $\mathcal{T}[1]$, registering support of each generalized item, such as 1**, ..., 4**, if a transaction contains such an item (i.e., the item in the transaction belongs to the generalized item 1**, ..., 4**, respectively), and filtering out those whose accumulated support count is lower than the minimum support. $\mathcal{L}[1,1]$ is then used to filter out (1) any item which is not large in a transaction, and (2) the transactions in $\mathcal{T}[1]$ that contain only small items. This results in the filtered transaction table $\mathcal{T}[2]$ of Figure 5.2. Moreover, since there are only two entries in $\mathcal{L}$[1,1], the level-1 large-2 itemset table $\mathcal{L}$[1,2] may contain only 1 candidate item {1**, 2**}, which is supported by 4 transactions in $\mathcal{T}[2]$.

Level-1 minsup = 4
Level-1 large 1-itemsets:

$\mathcal{L}$[1,1]

Itemset	Support
{1**}	5
{2**}	5

Level-1 large 2-itemsets:

$\mathcal{L}$[1,2]

Itemset	Support
{1**, 2**}	4

Filtered transaction table:
$\mathcal{T}[2]$

TID	Items
T_1	{111, 121, 211, 221}
T_2	{111, 211, 222}
T_3	{112, 122, 221}
T_4	{111, 121}
T_5	{111, 122, 211, 221}
T_6	{211}

Figure 5.2: Large itemsets at level 1 and filtered transaction table: $\mathcal{T}[2]$.

According to the definition of multiple-level association rules (hereafter referred to as ML-association rules), only the descendants of the large items at level-1 (i.e., in $\mathcal{L}$[1,1]) are considered as candidates for the level-2 large 1-itemsets. Let $minsup[2] = 3$. The level-2 large 1-itemsets $\mathcal{L}$[2,1] can be derived from the filtered transaction table $\mathcal{T}[2]$ by accumulating the support count and removing those whose support is smaller than the minimum support, which results in $\mathcal{L}$[2,1] of Figure 5.3. Similarly, the large 2-itemset table $\mathcal{L}$[2,2] is formed by the combinations of the entries in $\mathcal{L}$[2,1],

Level-2 minsup = 3
Level-2 large 1-itemsets:

$\mathcal{L}$[2,1]

Itemset	Support
{11*}	5
{12*}	4
{21*}	4
{22*}	4

Level-2 large 2-itemsets:

$\mathcal{L}$[2,2]

Itemset	Support
{11*, 12*}	4
{11*, 21*}	3
{11*, 22*}	4
{12*, 22*}	3
{21*, 22*}	3

Level-2 large 3-itemsets:

$\mathcal{L}$[2,3]

Itemset	Support
{11*, 12*, 22*}	3
{11*, 21*, 22*}	3

Level-3 minsup = 3
Level-3 large 1-itemsets:

$\mathcal{L}$[3,1]

Itemset	Support
{111}	4
{211}	4
{221}	3

Level-3 large 2-itemsets:

$\mathcal{L}$[3,2]

Itemset	Support
{111, 211}	3

Figure 5.3: Large itemsets at levels 2 and 3.

together with the support derived from $\mathcal{T}$[2], filtered using the corresponding threshold. Likewise, the large 3-itemset table $\mathcal{L}$[2,3] is formed by the combinations of the entries in $\mathcal{L}$[2,2].

Finally, $\mathcal{L}$[3,1] and $\mathcal{L}$[3,2] at level 3 are computed similarly, with the results shown in Figure 5.3. The computation terminates since there is no deeper level in the hierarchy. Note that the derivation also terminates when an empty large 1-itemset table is generated at any level. □

The above discussion leads to the following algorithm for mining strong ML-association rules.

Algorithm 5.3.1 (ML_T2L1) *Find multiple-level large itemsets for mining strong ML-association rules in a transaction database.*

Input: (1) $\mathcal{T}[1]$, a hierarchy-information-encoded and task-relevant set of a transaction database, in the format of $\langle TID, Itemset \rangle$, in which each item in the *Itemset* contains encoded conceptual hierarchy information, and (2) the minimum support threshold ($minsup[l]$) for each conceptual level l.

Output: Multiple-level large itemsets.

Method: A top-down, progressively deepening process which collects large itemsets at different conceptual levels as follows.

Starting at level 1, derive for each level l, the large k-items sets, $\mathcal{L}[l,k]$, for each k, and the set of large itemsets, $\mathcal{LL}[l]$ (for all k's), as follows (presented in a syntax similar to C and Pascal, which should be self-explanatory).

(1) for ($l := 1$; $\mathcal{L}[l,1] \neq \emptyset$ and $l < max\_level$; l++) do {
(2) if $l = 1$ then {
(3) $\mathcal{L}[l,1] := get\_large\_1\_itemsets(\mathcal{T}[1], l)$;
(4) $\mathcal{T}[2] := get\_filtered\_t\_table(\mathcal{T}[1], \mathcal{L}[1,1])$;
(5) }
(6) else $\mathcal{L}[l,1] := get\_large\_1\_itemsets(\mathcal{T}[2], l)$;
(7) for ($k := 2$; $\mathcal{L}[l,k-1] \neq \emptyset$; k++) do {
(8) $C_k := get\_candidate\_set(\mathcal{L}[l,k-1])$;
(9) foreach transaction $t \in \mathcal{T}[2]$ do {
(10) $C_t := get\_subsets(C_k, t)$;
(11) foreach candidate $c \in C_t$ do c.support++;
(12) }
(13) $\mathcal{L}[l,k] := \{c \in C_k | c.support \geq minsup[l]\}$
(14) }
(15) $\mathcal{LL}[l] := \bigcup_k \mathcal{L}[l,k]$;
(16) } □

Explanation of Algorithm 5.3.1.

According to Algorithm 5.3.1, the discovery of large support items at each level l proceeds as follows.

1. At level 1, the large 1-itemsets $\mathcal{L}[l,1]$ is derived from $\mathcal{T}[1]$ by "*get_large_1_itemsets*($\mathcal{T}[1], l$)". At any other level l, $\mathcal{L}[l,1]$ is derived from $\mathcal{T}[2]$ by "*get_large_1_itemsets*($\mathcal{T}[2], l$)". Notice that when $l > 2$, only the item in $\mathcal{L}[l-1,1]$ will be considered when examining $\mathcal{T}[2]$ in the derivation of the large 1-itemsets $\mathcal{L}[l,1]$. This is implemented by scanning the items of each transaction t in $\mathcal{T}[1]$ (or $\mathcal{T}[2]$), incrementing the support count of an item i in the itemset if i's count has not been incremented by t. After scanning the transaction table, filter out those items whose support is smaller than $minsup[l]$.

2. The filtered transaction table $\mathcal{T}[2]$ is derived by "*get_filtered_t_table*($\mathcal{T}[1], \mathcal{L}[1,1]$)", which uses $\mathcal{L}[1,1]$ as a filter to filter out (1) any item which is not large at level 1, and (2) the transactions which contain no large items.

3. The large k (for $k > 1$) itemset table at level l is derived in two steps:

   (a) Compute the candidate set from $\mathcal{L}[l,k-1]$, as done in the *a priori candidate generation algorithm* [4], apriori-gen, i.e., it first generates a set C_k in which each itemset consists of k items, derived by joining two $(k-1)$ items in $\mathcal{L}[l,k]$ which share $(k-2)$ items, and then removes a k-itemset c from C_k if there exists a c's $(k-1)$ subset which is not in $\mathcal{L}[l,k-1]$.

   (b) For each transaction t in $\mathcal{T}[2]$, for each of t's k-item subset c, increment c's support count if c is in the candidate set C_k. Then collect into $\mathcal{L}[l,k]$ each c (together with its support) if its support is no less than $minsup[l]$.

4. The large itemsets at level l, $\mathcal{LL}[l]$, is the union of $\mathcal{L}[l,k]$ for all the k's. □

After finding the large itemsets, the set of association rules for each level l can be derived from the large itemsets $\mathcal{LL}[l]$ based on the minimum confidence at this level, $minconf[l]$. This is performed as follows [4]. For every large itemset r, if a is a nonempty subset of r, the rule "$a \rightarrow r - a$" is inserted into $rule\_set[l]$ if $support(r)/support(a) \geq minconf[l]$, where $minconf[l]$ is the minimum confidence at level l.

Algorithm ML_T2L1 inherits several important optimization techniques developed in previous studies at finding association rules [2, 4]. For example, *get_candidate_set* of the large k-itemsets from the known large $(k-1)$-itemsets follows *apriori-gen* of Algorithm Apriori [4]. Function $get\_subsets(C_k, t)$ is implemented by a hashing technique from [4]. Moreover, to accomplish the new task of mining multiple-level association rules, some interesting optimization techniques have been developed, as illustrated below.

1. Generalization is first performed on a given item description relation to derive a generalized item table in which each tuple contains a set of item identifiers (such as bar_codes) and is encoded with conceptual hierarchy information.

2. The transaction table $\mathcal{T}$ is transformed into $\mathcal{T}[1]$ with each item in the itemset replaced by its corresponding encoded hierarchy information.

3. A filtered transaction $\mathcal{T}[2]$, which filters out small items at the top level of $\mathcal{T}[1]$ using the large 1-itemsets $\mathcal{L}[1,1]$, is derived and used in the derivation of large k-items for any k $(k > 1)$ at level-1 and for any k $(k \geq 1)$ for level l $(l > 1)$.

4. From level l to level $(l+1)$, only large items at $\mathcal{L}[l,1]$ are checked against $\mathcal{T}[2]$ for $\mathcal{L}[l+1,1]$.

Notice that in the processing, $\mathcal{T}[1]$ needs to be scanned twice, whereas $\mathcal{T}[2]$ needs to be scanned p times where $p = \sum_l k_l - 1$, and k_l is the maximum k such that the k-itemset table is nonempty at level l.

5.4 Variations of the Algorithm for Potential Performance Improvement

Potential performance improvements of Algorithm ML_T2L1 are considered by exploration of the sharing of data structures and intermediate results and generating maximal results at each database scan, etc., leading to the following variations of the algorithm: (1) ML_T1LA: using only *one* encoded transaction *table* (thus T1) and

generating $\mathcal{L}[l,1]$ for *all* the *levels* in one database scan (thus LA), (2) ML_TML1: using *multiple* encoded transaction *tables* and generating $\mathcal{L}[l,1]$ for *one* corresponding conceptual *level*, and (3) ML_T2LA: using *two* encoded transaction *tables* ($\mathcal{T}[1]$ and $\mathcal{T}[2]$) and generating $\mathcal{L}[l,1]$ for *all* the *levels* in one database scan.

5.4.1 Using Single Encoded Transaction Table: Algorithm ML_T1LA

The first variation is to use only one encoded transaction table $\mathcal{T}[1]$, that is, no filtered encoded transaction table $\mathcal{T}[2]$ will be generated in the processing.

At the first scan of $\mathcal{T}[1]$, large 1-itemsets $\mathcal{L}[l,1]$ for every level l can be generated in parallel, because the scan of an item i in each transaction t may increase the count of the item in every $\mathcal{L}[l,1]$ if its has not been incremented by t. After the scanning of $\mathcal{T}[1]$, each item in $\mathcal{L}[l,1]$ whose parent (if $l > 1$) is not a large item in the higher level large 1-itemsets or whose support is lower than $minsup[l]$ will be removed from $\mathcal{L}[l,1]$.

After the generation of large 1-itemsets for each level l, the candidate set for large 2-itemsets for each level l can be generated by the *apriori-gen* algorithm [4]. The *get_subsets* function will be processed against the candidate sets at all the levels at the same time by scanning $\mathcal{T}[1]$ once, which calculates the support for each candidate itemset and generates large 2-itemsets $\mathcal{L}[l,2]$. Similar processes can be processed for step-by-step generation of large k-item-sets $\mathcal{L}[l,k]$ for $k > 2$.

This algorithm avoids the generation of a new encoded transaction table. Moreover, it needs to scan $\mathcal{T}[1]$ once for generation of each large k-itemset table. Since the total number of scanning of $\mathcal{T}[1]$ will be k times for the largest k-itemsets, it is a potentially efficient algorithm. However, $\mathcal{T}[1]$ may consist of many small items which could be wasteful to be scanned or examined. Also, it needs a large space to keep all $C[l]$ which may cause some page swapping.

The algorithm is briefly summarized as follows.

Algorithm 5.4.1 (ML_T1LA) *A variation to Algorithm ML_T2L1: using only one encoded transaction table $\mathcal{T}[1]$.*

The input and output specifications are the same as Algorithm ML_T2L1. The procedure is described as follows.

(1) $\{\mathcal{L}[1,1],\ldots,\mathcal{L}[max\_l,1]\} := get\_all\_large\_1\_itemsets(\mathcal{T}[1])$;
(2) more_results := true;
(3) for ($k := 2$; more_results; k++) do begin
(4) more_results := false;
(5) for ($l := 1$; $l < max\_l$; l++) do
(6) if $\mathcal{L}[l,k] \neq \emptyset$ then begin
(7) $C[l] := get\_candidate\_set(\mathcal{L}[l,k-1])$;
(8) foreach transaction $t \in \mathcal{T}[1]$ do begin
(9) $D[l] := get\_subsets(C[l],t)$; // Candidates contained in t
(10) foreach candidate $c \in D[l]$ do c.support++;
(11) end
(12) $\mathcal{L}[l,k] := \{c \in C[l] | c.support \geq minsup[l]\}$
(13) more_results := true;
(14) end
(15) end
(16) for ($l := 1$; $l < max\_l$; l++) do $\mathcal{LL}[l] := \bigcup_k \mathcal{L}[l,k]$; □

Example 5.4.1 The execution of the same task as Example 5.3.1 using Algorithm ML_T1LA will generate the same large item sets $\mathcal{L}[l,k]$ for all the l's and k's but in difference sequences (without generating and using $\mathcal{T}[2]$). It first generates large 1-itemsets $\mathcal{L}[l,1]$ for all the l's from $\mathcal{T}[1]$. Then it generates the candidate sets from $\mathcal{L}[l,1]$, and then derives large 2-itemsets $\mathcal{L}[l,2]$ by passing the candidate sets through $\mathcal{T}[1]$ to obtain the support count and filter those smaller than $minsup[l]$. This process repeats to find k-itemsets for larger k until all the large k-itemsets have been derived.

□

5.4.2 Using Multiple Encoded Transaction Tables: Algorithm ML_TML1

The second variation is to generate multiple encoded transaction tables $\mathcal{T}[1]$, $\mathcal{T}[2]$, ..., $\mathcal{T}[max\_l + 1]$, where max_l is the maximal level number to be examined in the processing.

Similar to Algorithm ML_T2L1, the first scan of $\mathcal{T}[1]$ generates the large 1-itemsets $\mathcal{L}[1,1]$ which then serves as a filter to filter out from $\mathcal{T}[1]$ any small items or transactions containing only small items. $\mathcal{T}[2]$ results from this filtering process and is used in the generation of large k-itemsets at level 1.

Different from Algorithm ML_T2L1, $\mathcal{T}[2]$ is not repeatedly used in the processing of the lower levels. Instead, a new table $\mathcal{T}[l+1]$ is generated at the processing of each level l, for $l > 1$. This is done by scanning $\mathcal{T}[l]$ to generate the large 1-itemsets $\mathcal{L}[l,1]$ which serves as a filter to remove from $\mathcal{T}[l]$ any small items or transactions containing only small items and results in $\mathcal{T}[l+1]$, which will be used for the generation of large k-itemsets (for $k > 1$) at level l and table $\mathcal{T}[l+2]$ at the next lower level. Notice that as an optimization, for each level $l > 1$, $\mathcal{T}[l]$ and $\mathcal{L}[l,1]$ can be generated in parallel (i.e., at the same scan).

The algorithm derives a new filtered transaction table, $\mathcal{T}[l+1]$, at the processing of each level l. Although the generation of several transaction tables may seem costly, it can save a substantial amount of processing if only a small portion of data are large items at each level. Thus it may be a promising algorithm in this circumstance. However, it may not be so effective if only a small number of the items will be filtered out at the processing of each level.

The algorithm is briefly summarized as follows.

Algorithm 5.4.2 (ML_TML1) *A variation to Algorithm ML_T2L1: using multiple encoded transaction tables.*

The input and output specifications are the same as Algorithm ML_T2L1. The procedure is described as follows.

(1) for ($l := 1$; $\mathcal{L}[l,1] \neq \emptyset$ and $l < max\_level$; l++) do begin

(2) if $l = 1$ then $\mathcal{L}[l,1] := get\_large\_1\_itemsets(\mathcal{T}[1], l)$;
(3) $\{\mathcal{T}[l+1], \mathcal{L}[l+1,1]\} := get\_filtered\_\mathcal{T}\_table\_and\_large\_1\_itemsets(\mathcal{T}[l], \mathcal{L}[l,1])$;
(4) for ($k := 2$; $\mathcal{L}[l,k-1] \neq \emptyset$; k++) do begin
(5) $C_k := get\_candidate\_set(\mathcal{L}[l,k-1])$;
(6) foreach transaction $t \in \mathcal{T}[l+1]$ do begin
(7) $C_t := get\_subsets(C_k, t)$; // Candidates contained in t
(8) foreach candidate $c \in C_t$ do c.support++;
(9) end
(10) $\mathcal{L}[l,k] := \{c \in C_k | c.support \geq minsup[l]\}$
(11) end
(12) $\mathcal{LL}[l] := \bigcup_k \mathcal{L}[l,k]$;
(13) end

Notice that on line 3, the procedure "$get\_filtered\_\mathcal{T}\_table\_and\_large\_1\_itemsets(\mathcal{T}[l], \mathcal{L}[l,1])$" scans $\mathcal{T}[l]$, collects only the large items for each transaction containing large items, which generates $\mathcal{T}[l+1]$, and accumulates the support count for each item for the preparation of $\mathcal{L}[l+1,1]$. After the scan, it removes small items from the prepared $\mathcal{L}[l+1,1]$ based on $minsup[l+1]$. Thus it generates both $\mathcal{T}[l+1]$ and $\mathcal{L}[l+1,1]$ in the same scan of $\mathcal{T}[l]$. □

Example 5.4.2 The execution of the same task as Example 5.3.1 using Algorithm ML_TML1 will generate the same large itemsets $\mathcal{L}[l,k]$ for all the l's and k's but in difference sequences, with the generation and help of the filtered transaction tables $\mathcal{T}[2], \ldots, \mathcal{T}[max\_l+1]$, where max_l is the maximum level explored in the algorithm. It first generates the large 1-itemsets $\mathcal{L}[1,1]$ for level 1. Then for each level l (initially $l = 1$), it generates the filtered transaction table $\mathcal{T}[l+1]$ and the level-$(l+1)$ large 1-itemsets $\mathcal{L}[l+1,1]$ by scanning $\mathcal{T}[l]$ using $\mathcal{L}[l,1]$, and then generates the candidate 2-itemsets from $\mathcal{L}[l,1]$, calculates the supports using $\mathcal{T}[l+1]$, filters those with support less than $minsup[l]$, and derives $\mathcal{L}[l,2]$. The process repeats for the derivation of $\mathcal{L}[l,3], \ldots, \mathcal{L}[l,k]$. □

5.4.3 Refined Technique Using Two Encoded Transaction Tables: Algorithm ML_T2LA

The third variation uses the same two encoded transaction tables $\mathcal{T}[1]$ and $\mathcal{T}[2]$ as in Algorithm ML_T2L1, but it integrates some optimization techniques considered in the algorithm ML_T1LA.

The scan of $\mathcal{T}[1]$ first generates large 1-itemsets $\mathcal{L}[1,1]$. An additional scan of $\mathcal{T}[1]$ using $\mathcal{L}[1,1]$ will generate a filtered transaction table $\mathcal{T}[2]$ and all the large 1-itemset tables for all the remaining levels, i.e., $\mathcal{L}[l,1]$ for $1 < l \leq max\_l$ by incrementing the count of every $\mathcal{L}[l,1]$ at the scan of each transaction and removing small items and the items whose parent is small from $\mathcal{L}[l,1]$ at the end of the scan of $\mathcal{T}[1]$.

The candidate set for the large 2-itemsets at each level l can then be generated by the *apriori-gen* algorithm [4], and the *get_subsets* routine will extract the candidate sets for all the level l ($l \geq 1$) at the same time by scanning $\mathcal{T}[2]$ once. This will calculate the support for each candidate itemset and generate large 2-item-sets $\mathcal{L}[l,2]$ for $l \geq 1$.

Similar processes proceed step-by-step which generates large k-item-sets $\mathcal{L}[l,k]$ for $k > 2$ using the same $\mathcal{T}[2]$.

This algorithm avoids the generation of a group of new filtered transaction tables. It scans $\mathcal{T}[1]$ twice to generate $\mathcal{T}[2]$ and the large 1-itemset tables for all the levels. It then scans $\mathcal{T}[2]$ once for the generation of each large k-itemset, and thus scans $\mathcal{T}[2]$ in total $k-1$ times for the generation of all the k-itemsets, where k is the largest such k-itemsets available. Since k-itemsets generation for $k > 1$ is performed on $\mathcal{T}[2]$ which may consist of much less items than $\mathcal{T}[1]$, the algorithm could be a potentially efficient one.

The algorithm is briefly summarized as follows.

Algorithm 5.4.3 (ML_T2LA) *A variation to Algorithm ML_T2L1: refined technique using two encoded transaction tables.*

The input and output specifications are the same as Algorithm ML_T2L1. The procedure is described as follows.

(1) $\mathcal{L}[1,1] := get\_large\_1\_itemsets(\mathcal{T}[1], 1)$;
(2) $\{\mathcal{T}[2], \mathcal{L}[2,1], \ldots, \mathcal{L}[max\_l, 1]\} := get\_filtered\_t\_table\_and\_large\_1\_itemsets(\mathcal{T}[1], \mathcal{L}[1,1])$;
(3) more_results := true;
(4) for ($k := 2$; more_results; k++) do begin
(5) more_results := false;
(6) for ($l := 1$; $l < max\_l$; l++) do
(7) if $\mathcal{L}[l, k-1] \neq \emptyset$ then begin
(8) $C[l] := get\_candidate\_set(\mathcal{L}[l, k-1])$;
(9) foreach transaction $t \in \mathcal{T}[2]$ do begin
(10) $D[l] := get\_subsets(C[l], t)$; // Candidates contained in t
(11) foreach candidate $c \in D[l]$ do c.support++;
(12) more_results := true;
(13) end
(14) end
(15) $\mathcal{L}[l,k] := \{c \in C[l] | c.support \geq minsup[l]\}$
(16) end
(17) for ($l := 1$; $l < max\_l$; l++) do $\mathcal{LL}[l] := \bigcup_k \mathcal{L}[l,k]$; □

Example 5.4.3 The execution of the same task as Example 5.3.1 using Algorithm ML_T2LA will generate the same large itemsets $\mathcal{L}[l,k]$ for all the l's and k's. It first generates large 1-itemsets $\mathcal{L}[l,1]$ from $\mathcal{T}[1]$, then T[2] and all the large 1-itemsets $\mathcal{L}[2,1]$, ..., $\mathcal{L}[max\_l, 1]$, where max_l is the maximum level to be explored. It then generates the candidate sets from $\mathcal{L}[l,1]$, and derives large 2-itemsets $\mathcal{L}[l,2]$ by testing the candidate sets against $\mathcal{T}[2]$ to obtain the support count, and filters those with count smaller than $minsup[l]$. This process repeats so as to find k-itemsets for larger k until all the large k-itemsets have been derived. □

5.5 Performance Study

To study the performance of the proposed algorithms, all four algorithms: ML_T2L1, ML_T1LA, ML_TML1, and ML_T2LA, were implemented and tested on a SUN

Sparc-2 workstation with 16 megabytes of main memory.

The testbed consists of a set of synthetic transaction databases generated using a randomized item set generation algorithm similar to that described in [4].

The following are the basic parameters of the generated synthetic transaction databases: (1) the total number of items, I, is 1000; (2) the total number of transactions is 100,000; and (3) 2000 potentially large itemsets are generated and put into the transactions based on an exponential distribution. Table 5.5 shows the database used, in which S is the average size (# of items in a potential large itemset) of these itemsets, and T is the average size (# of items in a transaction) of a transaction.

Database	S	T	# of transactions	Size(MBytes)
DB1	2	5	100,000	2.7MB
DB2	4	10	100,000	4.7MB

Table 5.5: Parameters used to generate the transaction tables.

Each transaction database is converted into an encoded transaction table, denoted as $\mathcal{T}[1]$, according to the information about the generalized items in the item description (hierarchy) table. The maximal level of the conceptual hierarchy in the item table is set to 4. The number of the top level nodes keeps increasing until the total number of items reaches 1000. The fan-outs at the lower levels are selected based on the normal distribution with the mean value being $M2$, $M3$, and $M4$ for the levels 2, 3, and 4 respectively, and a variance of 2.0. These parameters are summarized in Table 5.6.

Item Table	#_nodes at level-1	M2	M3	M4
$I1$	8	5	5	5
$I2$	15	6	3	4

Table 5.6: Parameters settings of the item description (hierarchy) tables.

The testing results presented in this section are on two synthetic transaction

databases: one, $T10$ ($DB2$), has an average transaction size (# of item in a transaction) of 10; while the other, $T5$ ($DB1$), has an average transaction size of 5.

Two item tables are used in the testing: the first one, $I1$, has 8, 5, 5 and 5 branches at the levels 1, 2, 3, and 4, respectively; the second, $I2$, has 15, 6, 3 and 4 branches at the corresponding levels.

5.5.1 Scale Up Experiments

Figure 5.4 shows the running time of the four algorithms in relevance to the number of transactions in the database. The test uses the database $T10$ and the item set $I1$, with the minimum support thresholds being $(50, 10, 4, 2)$, which indicates that the minimum support of level 1 is 50%, and that of levels 2, 3 and 4 are respectively 10%, 4%, and 2%.

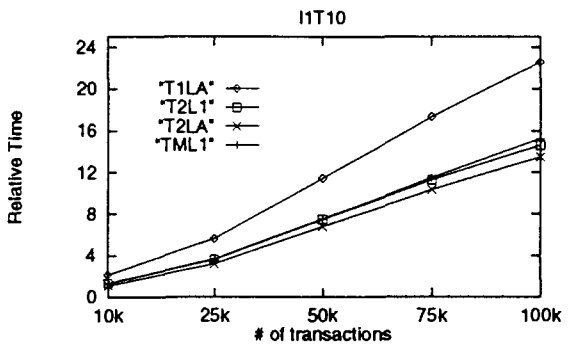

Figure 5.4: Performances with thresholds (50, 10, 4, 2).

The four curves in Figure 5.4 show that ML_T2LA has the best performance, while the ML_T1LA has the worst among the four algorithms under the current threshold setting. This can be explained as follows. The first threshold filters out many small 1-itemsets at level 1 which results in a much smaller filtered transaction table $\mathcal{T}[2]$. Moreover, the later filters are not so strong, and parallel derivation of $\mathcal{L}[l, k]$ without derivation of $\mathcal{T}[3]$ and $\mathcal{T}[4]$ is more beneficial. These lead ML_T2LA to be the best algorithm. On the other hand, ML_T1LA is the worst algorithm since it consults a large $\mathcal{T}[1]$ at every level.

Figure 5.5 shows that ML_T1LA is the best whereas ML_TML1 is the worst among the four algorithms under the following setting: a different test database $T5$, the same item set $I1$, and with the minimum support thresholds: $(20, 8, 2, 1)$. This is because the first threshold filters out few small 1-itemsets at level 1 which results in almost the same sized transaction table $\mathcal{T}[2]$. The generation of multiple filtered transaction tables is largely wasted, which leads to the worst performance of ML_TML1. Thus parallel derivation of $\mathcal{L}[l, k]$ without derivation of any filtered transaction tables applied in ML_T1LA leads to the best performance.

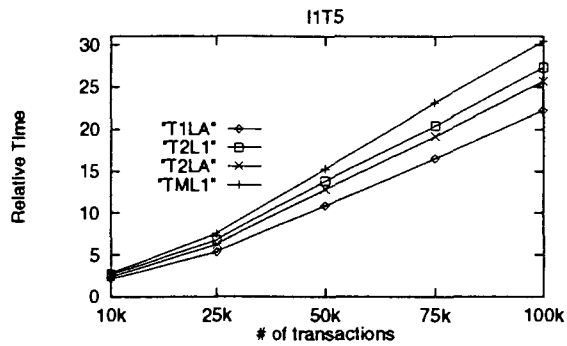

Figure 5.5: Performances with thresholds (20, 8, 2, 1).

Figure 5.6 shows that ML_T2L1 and ML_TML1 are closely the best whereas ML_T2LA and ML_T1LA are the worst under the setting: a test database $T10$, an item set $I2$, and with the minimum support thresholds: $(50, 10, 5, 2)$. This is because the first threshold filters out relatively more 1-itemsets at level 1 which results in a small transaction table $\mathcal{T}[2]$. Thus the generation of multiple filtered transaction tables is relatively beneficial. Meanwhile, the generation of multiple level large 1-itemsets may not save much because one may still obtain reasonably good sized itemsets in the current setting, which leads ML_T2L1 to be the best algorithm in terms of performance.

Figure 5.7 shows that ML_TML1 is the best whereas ML_T1LA is the worst under the setting: a test database $T5$, an item set $I2$, and with the minimum support thresholds: $(30, 15, 5, 2)$. This is because every threshold filters out relatively many

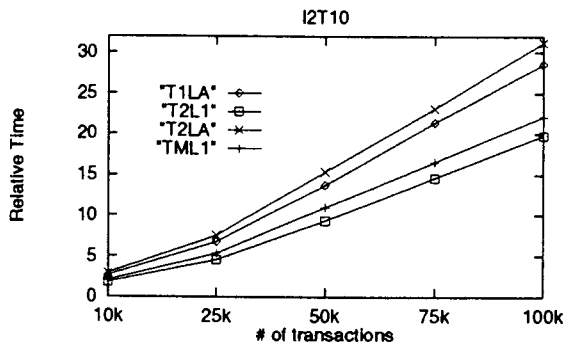

Figure 5.6: Performances with thresholds (50, 10, 5, 2).

1-itemsets at each level, resulting in much smaller transaction tables at each level. Thus the generation of multiple filtered transaction tables is beneficial, which leads to ML_TML1 as the best algorithm, followed by ML_T2L1, ML_T2LA and ML_T1LA in sequence.

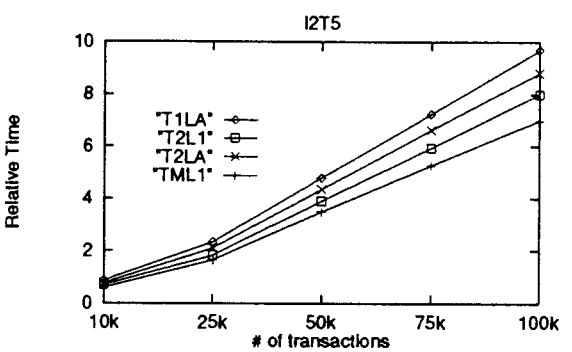

Figure 5.7: Performances with thresholds (30, 15, 5, 2).

The above four figures show two interesting features. First, the relative performance of the four algorithms under any setting is relatively independent of the number of transactions used in the testing, which indicates that the performance is highly relevant to the threshold setting (i.e., the power of a filter at each level). Thus based on

the effectiveness of a threshold, a good algorithm can be selected to achieve good performance. Second, all the algorithms have relatively good "scale-up" behavior since the increase of the number of transactions in the database will lead to approximately the linear growth of the processing time, which is desirable in the processing of large transaction databases.

5.5.2 Comparisons of Relative Performances

Figure 5.8 shows the running time of the four algorithms in relevance to the minimum support thresholds. The test uses the database $T10$ and the item set $I2$, with a sequence of threshold settings: *thre1*, ..., *thre6*. The setting of *thre1* is $(60, 15, 5, 2)$ (with the same notational convention). The remaining threshold settings are as follows: *thre2*: $(55, 15, 5, 2)$, *thre3*: $(55, 10, 5, 2)$, *thre4*: $(50, 10, 5, 2)$, *thre5*: $(50, 10, 5, 1)$, *thre6*: $(50, 5, 2, 1)$. The value-decreasing sequence of minimum support thresholds indicates that a weaker filtering mechanism is applied to the later portion of the sequence.

The relative performance of the four algorithms shows interesting trends of growth as indicated by the four curves in Figure 5.8. The stronger the filtering mechanism, the more 1-itemsets are filtered out at each level, and the smaller large 1-itemsets are resulted in. Thus ML_TML1, which generates a sequence of filtered transaction tables, has the lowest cost at *thre1*, *thre2* and also (but marginally) *thre3*, but the highest cost at *thre5* and *thre6* (since few items are filtered out). On the contrary, ML_T1LA, which uses only one encoded transaction table but generates the large 1-itemsets for each level at the beginning has the highest cost at *thre1*, *thre2* and *thre3*, but the lowest cost at *thre6*. The other two algorithms stand in the middle with ML_T2LA performing the best at *thre5* when the threshold is reasonable small, especially at the lower levels, and ML_T2L1 performing the best at *thre4* when the threshold is reasonable small, but the lowest level is not as small as *thre5*. Since ML_T2LA scans $\mathcal{T}[1]$ twice and needs to maintain all large itemsets $\mathcal{L}[l, k]$ at the same time, it is outperformed by ML_T2L1 when the thresholds are big enough so that a substantial amount of $\mathcal{T}[1]$ is cut and the maximal length of large itemsets at

each level is small. Moreover, one may observe the significant performance degradation from *thre4* to *thre5*. This, based on our speculation, is because of the limited size of main memory. We observed a surge of page swapping when the support threshold is dropped significantly.

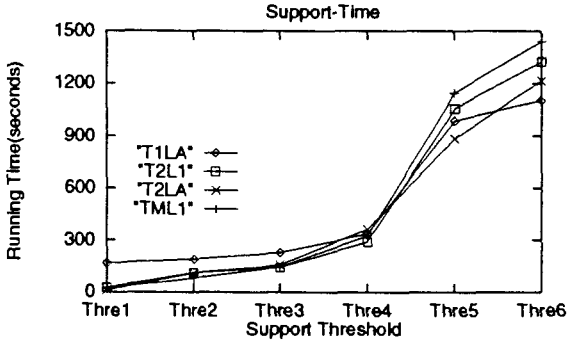

Figure 5.8: Relative performances with different thresholds.

5.5.3 Experiments on NSERC Databases

The algorithm *ML_T2L1* was implemented in our DBMiner system. To apply the algorithm which assumes each tuple is a set of items, each attribute in the initial relation is generalized to the lowest level in its hierarchy, so that a tuple can be treated as a transaction with each attribute value viewed as an item.

Example 5.5.1 The following data mining query, in DMQL, finds association rules in research grants in Computer Science in the 1994 NSERC database, with respect to the recipients' discipline, organization, and the amount of the grant. The minimal support and minimal confidence are set to (20%, 50%), (10%, 40%), and (1%, 30%) at level 1, 2, and 3 respectively.

use NSERC94
find association rules for "CS_Grants"
from award A, organization O

where O.org_code = A.org_code and A.disc_code = "Computer"
with interested attributes disc_code, org_name, amount
set minimum support 20%, 10%, 1%
set minimum confidence 50%, 40%, 30%

Since the level 1 concepts are always "ANY", the association rules at that level are trivial and thus are not shown here. The following are the association rules found at level 2. The organizations (org_name) are at the provincial level, and the amounts are categorized into $20,000 intervals. For example, rule R1 says that 47% of Ontario's grants are less than $20,000, and 14% of all grants (in Computer Science) consist of these small Ontario grants.

```
R1 : org_name in "ONTARIO" |=> amount in "0-20Ks"[0.14,0.47]
R2 : amount in "20Ks-40Ks" |=> disc_code in "Software"[0.15,0.60]
R3 : org_name in "ONTARIO" |=> disc_code in "Software"[0.18,0.61]
R4 : disc_code in "Software" |=> org_name in "ONTARIO"[0.18,0.43]
R5 : amount in "0-20Ks" |=> disc_code in "Software"[0.22,0.61]
R6 : disc_code in "Software" |=> amount in "0-20Ks"[0.22,0.52]
R7 : amount in "20Ks-40Ks" |=> org_name in "ONTARIO"[0.11,0.43]
```

The following are the association rules found at level 3. At this level, the organizations are specialized into institutions and the amount is categorized into $5,000 intervals. For example, rule R8 says 43% of the University of Ottawa's grants are within $15,000 to $20,000, which constitute 1% of all grants (in Computer Science).

```
R8 : org_name is "Ottawa" |=> amount in "15Ks-20Ks"[0.01,0.43]
R9 : org_name is "Concordia" |=> amount in "15Ks-20Ks"[0.01,0.37]
R10: disc_code in "Architecture" |=> amount in "15Ks-20Ks"[0.01,0.30]
R11: disc_code in "System Design" |=> amount in "15Ks-20Ks"[0.02,0.34]
```

□

5.6 Generation of Flexible Association Rules

Our study has been confined to mining association relationships level-by-level in a fixed hierarchy. It is often necessary or desirable to find flexible association rules which are not confined to strict, pre-arranged conceptual hierarchies.

5.6.1 Mining Cross-Level Association Rules

We may relax the restriction of mining strong associations among the concepts at the *same level* of a hierarchy to allow the exploration of "*level-crossing*" association relationships. This relaxation may lead to the discovery of associations like "2% Foremost milk $\rightarrow$ Wonder bread" in which the two concepts are at different levels of a hierarchy. This can be achieved by making minor modifications to our algorithms since the new requirement associates the itemsets like $(\{112, 2*1\})$, as demonstrated in the example below.

Example 5.6.1 For the same transaction tables and conceptual hierarchies as given in Example 5.3.1, we examine the mining of strong multiple-level association rules which include nodes at different levels in a hierarchy.

Let minimum support at each level be: minsup = 4 at level-1, and minsup = 3 at levels 2 and 3.

The derivation of the large itemsets at level 1 proceeds in the same way as in Example 5.3.1, which generates the same large itemsets tables $\mathcal{L}[1,1]$ and $\mathcal{L}[1,2]$ at level 1, and the same filtered transaction table $\mathcal{T}[2]$, as shown in Figure 5.2.

The derivation of level-2 large itemsets generates the same large 1-itemsets $\mathcal{L}[2,1]$ as shown in Figure 5.9. However, the candidate items are not confined to the pairing of only those in $\mathcal{L}[2,1]$ because the items in $\mathcal{L}[2,1]$ can be paired with those in $\mathcal{L}[1,1]$ as well, such as $\{11*, 1**\}$ (for potential associations like "milk $\rightarrow$ 2% milk"), or $\{11*, 2**\}$ (for potential associations like "2% milk $\rightarrow$ bread"). These candidate large 2-itemsets will be checked against $\mathcal{T}[2]$ to find large items (for the level-mixed nodes, the minimum support at a lower level, i.e., minsup[2], can be used as a default). Such a process generates the large 2-itemsets table $\mathcal{L}[2,2]$ as shown in Figure 5.9.

Notice that the table does not include the 2-item pairs formed by an item with its own ancestor such as $\langle\{11*, 1**\}, 5\rangle$ since its support must be the same as its corresponding large 1-itemset in $\mathcal{L}[2,1]$, i.e., $\langle\{11*\}, 5\rangle$, based on the set containment relationship: any transaction that contains $\{11*\}$ must contain $\{1**\}$ as well.

Similarly, the level 2 large 3-itemsets $\mathcal{L}[2,3]$ can be computed, with the results shown in Figure 5.9. Also, the entries which pair with their own ancestors are not listed here since it is contained implicitly in their corresponding 2-itemsets. For example, $\langle\{11*, 12*\}, 4\rangle$ in $\mathcal{L}[2,2]$ implies $\langle\{11*, 12*, 1**\}, 4\rangle$ in $\mathcal{L}[2,3]$.

Level-2 minsup = 3

Level-2 large 1-itemset:
$\mathcal{L}[2,1]$

Itemset	Support
{11*}	5
{12*}	4
{21*}	4
{22*}	4

Level-2 large 2-itemset:
$\mathcal{L}[2,2]$

Itemset	Support
{11*, 12*}	4
{11*, 21*}	3
{11*, 22*}	4
{12*, 22*}	3
{21*, 22*}	3
{11*, 2**}	4
{12*, 2**}	3
{21*, 1**}	3
{22*, 1**}	4

Level-2 large 3-itemset:
$\mathcal{L}[2,3]$

Itemset	Support
{11*, 12*, 22*}	3
{21*, 22*, 1**}	3

Figure 5.9: Cross-level large itemsets at level 2.

Finally, the large 1-itemset table at level 3, $\mathcal{L}[3,1]$, should be the same as Figure 5.3. The large 2-itemset table includes more itemsets since these items can be paired with higher level large items, which leads to the large 2-itemsets $\mathcal{L}[3, 2]$ and large 3-itemsets $\mathcal{L}[3, 3]$ as shown in Figure 5.10. Similarly, the itemsets {111, 11*} and {111, 1**} have the same support as {111} in $\mathcal{L}[3, 1]$ and are thus not included in $\mathcal{L}[3,2]$.

Since the large k-itemset (for $k > 1$) tables do not explicitly include the pairs of items with their own ancestors, attention should be paid to include them at the generation of the association rules. However, since the existence of a specialized item always indicates the existence of an item in that class, such as "2% milk $\rightarrow$ milk

(100%)", such trivial rules should be eliminated. Thus, only nontrivial implications, such as "milk → 2% milk (70%)", will be considered in the rule generation. □

Level-3 minsup = 3

Level-3 large 1-itemset:

$\mathcal{L}$[3,1]

Itemset	Support
{111}	4
{211}	4
{221}	3

Level-3 large 2-itemset:

$\mathcal{L}$[3,2]

Itemset	Support
{111, 211}	3
{111, 21*}	3
{111, 22*}	3
{111, 2**}	4
{11*, 211}	3
{1**, 211}	3

Level-3 large 3-itemset:

$\mathcal{L}$[3,3]

Itemset	Support
{111, 21*, 22*}	3

Figure 5.10: Cross-level large itemsets at level 3.

5.6.2 Mining Association Rules in Mixed Hierarchies

Sometimes, it is necessary or desirable to find associations among the concepts associated with alternative, multiple hierarchies. For example, following the hierarchy given in Example 5.2.1, one may find relationships like "2% milk → wheat bread". Alternatively, one may like to find "Foremost milk → Wonder bread" or "2% milk → Wonder bread", which may require an alternative conceptual hierarchy, i.e., the hierarchy {*content, brand, category*} ⊂ {*brand, category*} ⊂ {*category*}, where *brand* is taken as a higher level structure than *content*, between *category* and *content*. It seems to be challenging to explore so many alternatives since there may exist only a small number of fixed hierarchies in a database. However, the algorithms presented in this chapter can be adapted with slight modification in order to meet the challenge since the new requirement essentially associates the itemsets in some alternative generalized forms, such as ⟨{1*2}, {2*1}⟩, ⟨{12*}, {2*1}⟩, etc.

Example 5.6.2 (Mining association rules in mixed hierarchies) For the same transaction and item databases as that of Example 5.3.1, find multiple-level association rules

between brands and content specifications of different categories.

Let the minimum support at each level be the same as in Example 5.3.1, i.e., minsup = 4 at level-1, and minsup = 3 at levels 2 and 3.

The derivation of the large itemsets at level 1 proceeds in the same way as Example 5.3.1, which generates the same large itemsets tables $\mathcal{L}[1,1]$ and $\mathcal{L}[1,2]$ at level 1 and the same filtered transaction table $\mathcal{T}[2]$, as shown in Figure 5.2.

However, the level-2 large itemsets are different from those in Example 5.3.1 because our method first generates large 1-itemsets in the forms of both 11*and 1*1, i.e., including both hierarchies, then pairs the large 1-items for those from different categories, such as {11*, 2*1}, and then finds large 3-itemsets with such properties, etc. Therefore, the large itemset tables at level-2 are $\mathcal{L}[2,1]$, $\mathcal{L}[2,2]$ and $\mathcal{L}[2,3]$, as shown in Figure 5.11.

Finally, large itemset tables at level 3 should be the same since two hierarchies share the same leaf nodes at level 3. Thus it will generate the same tables $\mathcal{L}$[3,1] and $\mathcal{L}$[3,2] as shown in Figure 5.3. Notice that since the two hierarchies (*category-brand* and *category-content*) share the same level-3 leaf nodes (*brand-content*), the expansions following each hierarchy may lead to redundancy. One may mark the lower level nodes once explored, and no marked nodes will be checked again, which avoids redundant exploration.

Notice also that the query is to find associations between *different* categories. If it were to include associations among the items in the same category, such as "2% milk → Foremost milk", more large 2-itemsets would have been found in $\mathcal{L}$[2,2] because the 2-itemset {11*, 1*1} would also form a large 2-itemset as well. Note also that the rule "2% milk → Foremost milk" indicates that a person who buys 2% milk will also buy Foremost milk (which, however, may not necessarily be 2% Foremost milk!). □

Level-2 minsup = 3

Level-2 large 1-itemsets: $\mathcal{L}$[2,1]

Itemset	Support
{11*}	5
{12*}	4
{1*1}	4
{21*}	4
{22*}	4
{2*1}	4

Level-2 large 2-itemsets: $\mathcal{L}$[2,2]

Itemset	Support
{11*, 12*}	4
{11*, 21*}	3
{11*, 22*}	4
{11*, 2*1}	4
{12*, 22*}	3
{12*, 2*1}	4
{21*, 22*}	3
{21*, 1*1}	3

Level-2 large 3-itemsets: $\mathcal{L}$[2,3]

Itemset	Support
{11*, 12*, 22*}	3
{11*, 12*, 2*1 }	3

Figure 5.11: Mixed-hierarchy large itemsets at level 2.

5.7 Discussion

5.7.1 More about Conceptual Hierarchies

In our discussion, we have assumed desired conceptual hierarchies exist and are presented in the form of relational tables (e.g., *sales_item* in Table 5.1). However, there are often cases where portions of conceptual hierarchies do not exist. For example, the hierarchy relationships, such as "peanuts, pistachios, ..., walnuts $\subset$ nuts", may not be stored in the sales_item relation. Therefore, it is often necessary for experts or users to specify portions of hierarchies to facilitate mining multiple-level association rules. Specified hierarchies can be mapped into relations with the paths from high-level general concepts to low-level specific ones registered in tuples. Null values should be allowed in the mapped relational entries if there exist unbalanced nodes in a hierarchy.

Note that there may often exist more than one possible way of mapping a relation into a conceptual hierarchy. For example, "2% Foremost milk $\subset$ 2% milk $\subset$ milk" and "2% Foremost milk $\subset$ Foremost milk $\subset$ milk" are both meaningful hierarchies, but "2% Foremost milk $\subset$ 2% Foremost $\subset$ Foremost" may not be. An expert or a user may

provide mapping rules at the schema level (i.e., schema hierarchies) to indicate meaningful or desired mappings, such as "{content, brand, category} $\subset$ {content, category} $\subset$ category", etc.

Conceptual hierarchies may not exist for numerical valued attributes but can be automatically generated according to data distribution statistics, as described in Section 3.3. For example, a hierarchy for the price range of sales items can be generated based on the distribution of price values. Moreover, a given conceptual hierarchy for numerical or nonnumerical data can be dynamically adjusted based on data distribution as discussed in Section 3.2. For example, if there are many distinct country names in the attribute "place_made", countries can be grouped into continents, such as *Asia, Europe, South_America*, etc. Moreover, if most fresh food products are from *B.C.* and *Northwest America*, the geographic hierarchy can be automatically adjusted to reflect this distribution when studying fresh food products.

5.7.2 Interestingness Measure

Many multiple-level association rules may be found from the database using the progressive deepening method. Sometimes a low level rule can be expected or estimated from a high level rule, and thus is not interesting to user. For example, a rule "1% milk $\rightarrow$ whole-wheat bread [2%, 20%]" is not interesting if we have a higher level rule "milk $\rightarrow$ bread [40%, 80%]" and 25% of all the milk is the 1% milk and 20% of all the bread is the whole-wheat bread.

Srikant and Agrawal [104] proposed an interestingness measure to filter out uninteresting rules. A low level rule is interesting if its support or confidence is different by a factor of k from the expectations, computed from the support and confidence of a high level rule assuming even distributions among children. Some other interestingness measures, which detect the deviations from the given norms [84], for example, can be adapted as well for mining multiple-level association rules.

These interestingness measures can be integrated in our method. After the large itemsets are found, the rule-generation module will generate rules from top level to primitive level. When a strong rule is found, it is output if it passes the interestingness

test; otherwise, it is discarded.

Another direction to focus on interesting rules is using meta-rules which are rule templates specifying the format of the rules to be found. The meta-rule guided mining of association rules is studied in Chapter 6.

5.7.3 Re-examination of the Definition of Strong Multiple-Level Association Rule

Strong multiple-level association rules were introduced in Definition 5.2.3 for a large class of applications. Algorithms studied in Sections 5.3 and 5.4 follow this definition. However, different applications may require finding different kinds of multiple-level association rules. We examine how the variations of the rule definition may influence the rule mining algorithms.

First, the multiple-level association rules may include multiple conceptual hierarchies, their mixtures, and the associations among the items at different levels of a hierarchy, etc. Such variations have been examined in Section 5.6.

Second, our definition examines an item at level l if its parent is a large 1-item at level $l-1$. An alternative is to examine the associations among k items at level l only if the (k-arity) associations of their k parents are in the large k-itemsets at level $l-1$. For example, only if "{bread, milk} " are large 2-itemsets, will their lower level combinations of different kinds of milk and bread be examined. This definition may exclude many itemsets that have been previously considered and reduce the set of candidate itemsets to be examined at lower levels. Its efficient rule mining algorithms can be worked out accordingly. However, since large single items are usually interesting enough to warrant detailed examinations, a strict requirement of examining only those itemsets whose parents are large k-itemsets (for $k > 1$) may miss many potentially interesting associations.

Third, our definition concerns a minimum support threshold in relevance to a specified set of data instead of to the *whole* database. The minimum support can be specified as a ratio, such as the number of transactions containing particular itemsets versus the total number of transactions within a specified domain. The flexible

definition of domains at different levels, especially the confinement of the domains be smaller at lower levels, not only clarifies the concept of a rule but also reduces the search effort. For example, for the top-level, the support of an itemset could be the ratio of the set of transactions containing the itemset versus either the whole set of transactions in the transaction database or the set of transactions in relevance only to the data mining query (e.g., the transactions containing *fresh food* items). For level two, the support of an itemset could be the ratio of set of the transactions containing the itemset versus the set of transactions containing large items (instead of the whole set of transactions), etc. As long as the support is well defined and fixed at each level (for different large k-itemsets), the computation will be the same as those outlined in the algorithms.

Notice that it is natural to consider using a larger minimum support when deriving large 1-itemsets and substantially reduce the minimum support at the derivation of large 2-itemsets, etc. However, based on our observation, the restriction on the fixed minimum support threshold at a level for k-itemsets (for all k's) may not be easily relaxed. This is because a key optimization technique applied in both single-level and multiple-level association rule mining algorithms is to use only the entries in the large k-itemsets to derive the candidate large $(k+1)$-itemsets. This optimization is not applicable if the minimum support changes on different k's. A compromise is to derive intermediate large k-itemsets for all the k's by first taking the smallest minimum support among the k-itemsets (if they are allowed to be different) and then filtering out those which are not large for the current k. By doing so, the current large-itemsets-mining algorithms are still applicable by augmenting an additional filtering process. Whether there may exist more efficient algorithms under this restriction remains a research issue.

5.8 Summary

We have extended the scope of the study of mining association rules from single level to multiple conceptual levels and have studied methods for mining multiple-level association rules from large transaction databases. A top-down progressive deepening

technique is developed for mining multiple-level association rules, which extends the existing single-level association rule mining algorithms and explores techniques for sharing data structures and intermediate results across levels. Based on different sharing techniques, a group of algorithms, notably, ML_T2L1, ML_T1LA, ML_TML1 and ML_T2LA, have been developed. Our performance study shows that different algorithms may have the best performance for different distributions of data and different thresholds.

Related issues, including methods for mining flexible multiple-level association rules, conceptual hierarchy handling, interestingness measures to filter out uninteresting rules, and the adaptation to different mining requests are also discussed in the paper. Our study shows that mining multiple-level association rules from databases has wide applications, and efficient algorithms can be developed for the discovery of interesting and strong such rules in large databases.

Chapter 6

Meta-Rule Guided Mining of Multiple-Level Association Rules

6.1 Introduction

A frequently encountered phenomenon in data mining is that although a mining system may discover a quite large number of rules, many of them could be poorly focused or uninteresting to users. Two major factors may contribute to this phenomenon: (1) lack of focus on the set of data to be studied, and (2) lack of constraints on the forms and/or kinds of rules or knowledge to be discovered.

The first problem, *the lack of focus on the set of data to be studied*, can be handled by introducing a data mining interface which specifies the set of data relevant to a particular mining task. For example, the DBMiner system uses an SQL-like interface [51] to specify the task-relevant set of data for a data mining query. Thus, in order to find the general characteristics of computer science graduate students in Canada, a where-clause is used to retrieve only those students of interest.

However, the second problem, *the lack of constraints on the forms and/or kinds of rules or knowledge to be discovered*, is not so straightforward to solve. There are many ways to specify the kinds of knowledge or the forms of rules to be discovered. For example, one may specify the types of knowledge to be discovered, such as characteristic rules, classification rules, association rules, and so on [48], or specify the

number of disjuncts in a generalized rule, i.e., the expected (or maximum) number of distinct values of each generalized attribute or the number of tuples in the generalized relation [46]. Moreover, one may also specify some syntactic or semantic constraints on the forms of discovered rules [4, 65].

Recently, Shen et al. [94] proposed an interesting technique to specify the form of rules to be discovered in data mining, called *metaquery*, which presents a desired logical form for the rules to be discovered and serves as an important interface between human discoverers and the discovery system. The *metaquery* approach confines the rules to be discovered to be in a specified form, such as "$P(x, y) \wedge Q(y, z) \Rightarrow R(x, z)$", where P, Q and R are predicate variables that can be bound to any concrete predicates, and x, y, and z are variables that can be bound to some data in the database. Such kind of metaqueries can be given by users or revised interactively from initial metaqueries formed by the system based on the schema information. Such rule forms can also serve as a linkage between deductive and inductive aspects of knowledge discovery and facilitates a deductive-inductive-human discovery loop. Thus, it represents an interesting direction to pursue.

In their initial study of metaquery-directed data mining [94], the rules to be discovered are confined to single conceptual level, whereas the knowledge discovery method is confined to Bayesian Data Cluster linked with a deductive database system $\mathcal{LDL}++$. Based on our observation, the scope of metaquery-directed mining could be substantially extended if the discovery of rules at multiple conceptual levels is explored [37, 47]. Moreover, since a metaquery and its instantiated rules are in the form of association rules, the performance could be substantially enhanced if the database-oriented association rule mining algorithms [4] are adopted in the data mining process.

As we mentioned in the previous chapter, one problem of the current rule mining methods is that they often discover a large number of association rules, and some of such rules may not be desirable to users. While some interestingness measures like the *R-interestingness* [104] can be used to prune some uninteresting rules, it is more desirable and effective to use a rule template, or meta-rule.

In this chapter, issues for meta-rule guided mining of multiple-level association

rules are studied and a set of efficient mining algorithms is developed and tested. The study shows that the integration of meta-rule guided knowledge mining with the mining of multiple-level association rules enhances both the power and performance of a data mining system and thus is an interesting direction to pursue [37].

The remainder of the chapter is organized as follows. In Section 6.2, preliminary concepts about meta-rule guided mining of multiple-level association rules are introduced, starting with some motivating examples. In Section 6.3, methods for mining meta-rule-guided single-variable rules are studied. In Section 6.4, methods for mining meta-rule-guided multiple-variable rules are examined. Variation of methods and other relevant issues on meta-rule-guided data mining are discussed in Section 6.5, and the study is summarized in Section 6.6.

6.2 Preliminary Concepts

To simplify our discussion, a relational model is adopted in our study, however, the methods developed here can be applied with some modifications to other data models, including extended-relational and object-oriented ones.

For effective data mining, a particular user is usually interested in only a subset of the data stored in a large database. A DMQL data mining query [51] submitted to a data mining system should first be transformed into two portions: a *data collection* portion and a *knowledge discovery* portion. The former is essentially an SQL-query which will be executed against the database to collect the interested set of data. The latter, i.e., the knowledge discovery portion, will be examined in detail.

Example 6.2.1 Suppose that a portion of the relational schema of a university database is presented as follows.

student(name, sno, status, major, gpa, birth_date, birth_place, address)
course(cno, title, dept)
grading(sno, cno, instructor, semester, grade)

Let a data mining query (q_1) be presented as follows, which is to find the relationships between the attributes *status, gpa, birth_place*, and *address*, in relevance to

major, for the students born in Canada.

(q_1) : discover rules in the form of

$major(s : student, x) \wedge Q(s, y) \rightarrow R(s, z)$

from student

where birth_place = "Canada"

in relevance to major, gpa, status, birth_place, address

The meta-rule of (q_1), "$major(s : student, x) \wedge Q(s, y) \rightarrow R(s, z)$", specifies the form of the rules to be discovered, that is, each rule to be discovered is a logic rule containing two binary predicates, $major(s, x)$ and $Q(s, y)$, serving as the antecedent and one binary predicate, $R(s, z)$, as the consequent, with all the predicates sharing the first variable s which is the key of the relation *student*. Q and R are two predicate variables which can be instantiated by a list of relevant attributes: *gpa, status, birth_place,* and *address.*

By data mining techniques, the following rules may be discovered from the database.

$$major(s, \text{"}Science\text{"}) \wedge gpa(s, \text{"}Excellent\text{"}) \rightarrow status(s, \text{"}Graduate\text{"}) \ (60\%) \quad (6.2.1)$$

$$major(s, *) \wedge birth\_place(s, \text{"}B.C.\text{"}) \rightarrow address(s, \text{"}Burnaby\text{"}) \ (55\%) \quad (6.2.2)$$

Rule (6.2.1) indicates that 60% of the students majoring in science and having excellent gpa are graduate students and rule (6.2.2) indicates that 55% of the students majoring in anything and born in B.C. are living in the city of Burnaby.

The rules expressed by even lower level concepts, such as rules (6.2.3) to (6.2.4), can be further discovered if multiple-level information can be mined from the database. The semantic meaning of these rules is self-explanatory.

$$major(s, \text{"}Physics\text{"}) \wedge gpa(s, \text{"}3.8\_4.0\text{"}) \rightarrow status(s, \text{"}M.Sc\text{"}) \quad (76\%) \quad (6.2.3)$$

$$major(s, \text{"}CS\text{"}) \wedge birth\_place(s, \text{"}Vancouver\text{"}) \rightarrow address(s, \text{"}NorthBurnaby\text{"}) \quad (85\%) \quad (6.2.4)$$

Moreover, the associations among several relations can be discovered by joining these relations together. The relational joins can be explicitly expressed in the meta-rules as presented in the following data mining query (q_2).

(q_2) : discover rules in the form of

$$major(s, x) \wedge P(c, y) \rightarrow Q(s : S, c : C, z).$$

from student S, grading G, course C

where S.birth_place = "Foreign"

The query is to find the relationships among three predicates, one of which is instantiated to $major(s, x)$, the second contains the key of the *course* relation, and the third one, the consequent predicate, contains two key components from two relations: *student* and *course*, for the relevant set of the data: the students born in foreign countries.

By mining rules from multiple conceptual levels, the following rules may be discovered from the database.

$$major(s, \text{"}Science\text{"}) \wedge dept(c, \text{"}CS\text{"}) \rightarrow grade(s, c, \text{"}Good\text{"}) \quad (60\%) \ (6.2.5)$$

$$major(s, \text{"}Math\text{"}) \wedge cno(c, \text{"}CS\_400\_level\text{"}) \rightarrow grade(s, c, \text{"}A-\text{"}) \quad (42\%) \ (6.2.6)$$

In Example 6.2.1, both the data mining queries and the discovered rules contain concepts at nonprimitive levels, i.e., levels higher than those stored in databases, such as "Science", "Graduate", "Excellent", etc. The high level concepts appearing in the query help the collection of the relevant set of data, whereas the concepts organized at different levels help in progressively deeping the data mining process by first browsing the high-level data and then mining detailed regularities at low levels.

In this chapter, we assume conceptual hierarchies are provided, which organize multiple levels of concepts for mining rules at multiple conceptual levels. However, the conceptual hierarchies can also be dynamically adjusted and/or automatically generated for flexible data mining as discussed in Chapter 3.

To confine our study, we assume the rules to be discovered are conjunctive rules, i.e., a set of conjuncts in both the rule head and body. Moreover, the predicate variable in the meta-rules can only be instantiated against database schema (attributes). Furthermore, each predicate variable in a meta-rule is different from others and is instantiated to a distinct and different predicate name. Some relaxations of these restrictions will be discussed in Section 5.

As a notational convention, a predicate name starting with an upper-case letter represents a predicate variable. It can be instantiated by binding it to a concrete attribute name (which starts with a lower-case letter) in the schema. For example, a predicate variable $P(x, y)$ can be instantiated to $status(x, \text{"}Graduate\text{"})$ in Example 6.2.1.

Definition 6.2.1 A meta-rule is a rule template in the form of

$$P_1 \wedge P_2 \cdots \wedge P_m \rightarrow Q_1 \wedge Q_2 \wedge \cdots \wedge Q_n. \tag{6.2.7}$$

where P_i (for $i = 1, \ldots, m$) and Q_j (for $j = 1, \ldots, n$) are either instantiated predicates or predicate variables.

The rule "$major(s, x) \wedge P(c, y) \rightarrow Q(s : S, c : C, z)$" in Example 6.2.1 is a meta-rule.

Definition 6.2.2 A rule, R_c, complies with a meta-rule, R_M, if and only if it can be unified with R_M.

For example, rule (6.2.5) complies with the meta-rule "$major(s, x) \wedge P(c, y) \rightarrow Q(s : S, c : C, z)$" in Example 6.2.1.

Definition 6.2.3 A pattern, p, is one predicate p_i or a set of conjunctive predicate $p_i \wedge \cdots \wedge p_j$, where $p_i, \ldots, p_j$ are predicates instantiated against the database schema. The support of a pattern p in a set S, $\sigma(p/S)$, is the number of the tuples in S which contain p versus the total number of tuples in S. The confidence of $p \rightarrow q$ in S, $\varphi(p \rightarrow q/S)$, is the ratio of $\sigma(p \wedge q/S)$ versus $\sigma(p/S)$, i.e., the probability that pattern q also occurs in S when pattern p occurs in S.

As in Chapter 5, a *minimum support*, σ', and a *minimum confidence*, φ' are specified for each level.

Definition 6.2.4 A pattern p is large in set S at level l if the support of p is no less than its corresponding minimum support threshold σ'_l. The confidence of a rule "$p \rightarrow q/S$" is high at level l if its confidence is no less than its corresponding minimum confidence threshold φ'_l.

Definition 6.2.5 A rule "$p \rightarrow q/S$" is **strong** if, for a set S, each ancestor (i.e., the corresponding high level predicate) of every predicate in p and q, if any, is large at its corresponding level, "$p \wedge q/S$" is large (at the current level), and the confidence of "$p \rightarrow q/S$" is high (at the current level).

Roughly, a predicate is like an itemset in Chapter 5 The definitions for **support, confidence, large, and strong** are also similar.

Based on the two mining queries presented in Example 6.2.1, meta-rule guided mining of multiple-level association rules can be classified into two categories: (1) mining single-variable association rules, and (2) mining multiple-variable association rules. The former discovers association rules in the form like (6.2.3), in which each predicate contains only one and the same variable; whereas the latter discovers rules in the form like (6.2.5), in which some predicate(s) may contain more than one variable, which may often involve join(s) of more than one relation.

6.3 Meta-Rule-Guided Mining of Single-Variable Rules

In this section, we examine the methods for meta-rule guided mining of single-variable association rules. A single-variable association rule represents an association relationship among a set of properties in a data relation at different conceptual levels.

Definition 6.3.1 A **single-variable meta-rule** is in the form of

$$P_1(t : rel, x_1) \wedge \cdots \wedge P_m(t, x_m) \rightarrow Q_1(t, y_1) \wedge \cdots \wedge Q_n(t, x_n) \qquad (6.3.8)$$

where P_i (for $i = 1, \ldots, n$) and Q_j (for $j = 1, \ldots, m$) are either instantiated predicates or predicate variables, and the common variable t represents the key of a relation *rel.* □

By data mining, each predicate variable in a discovered rule will be instantiated to a concrete predicate name which is an attribute name of the relation *rel*, the common variable t will remain as a variable that is an abstraction of the key or key component

of the relation, and other variables in the predicates will be instantiated to the high-level or primitive level constants (i.e., properties) of the corresponding predicates (attributes).

For example, the meta-rule "$major(s : student, x) \wedge Q(s, y) \rightarrow R(s, z)$" in (q_1) of Example 6.2.1 is a single-variable meta-rule, and the discovered rule (6.2.1) indicates that the common variable s remains to be a variable which is an abstraction of the key of the relation *student*, and other variables in the predicates are instantiated to constants, such as *Science, Excellent*, and *Graduate* in the corresponding predicates, such as *major, gpa*, and *status*, respectively.

For efficient mining of multiple-level single-variable association rules, two techniques: a *large-predicate growing technique* and a *p-predicate testing technique*, are proposed and examined in the next two subsections.

6.3.1 A Large-Predicate Growing Technique

Following our previous study on mining multiple-level association rules [47], a large-predicate growing technique is proposed as follows.

First, the set of relevant data is collected into an initial data relation by executing an SQL query specified by the data mining query. Second, large 1-predicate-sets, $\mathcal{L}[1,1]$, $\mathcal{L}[2,1]$, ..., $\mathcal{L}[max\_l, 1]$, are derived at each conceptual level (from the top-most desired conceptual level, level 1 down to level max_l) by scanning the initial data relation once, where level max_l is the lowest level where a non-empty large 1-predicate-set can be derived. Third, large 2-predicate-sets are derived at each conceptual level by first generating the candidate large 2-predicate-sets and then scanning the initial data relation to compute the large 2-predicate-sets. Fourth, this process continues until the large p-predicate-sets are derived at each conceptual level, where p is the total number of predicates in the meta-rule, i.e., $p = m + n$ in rule (6.3.8). Finally, the rules in the form of meta-rules are generated from the large p-predicate-sets at each conceptual level based on the specified confidence threshold at this level.

This technique is illustrated in the following example.

Example 6.3.1 We examine how to derive the multiple-level strong association rules for query (q_1) of Example 6.2.1.

1. The initial data relation $\mathcal{R}_0$ (a fragment shown in Table 6.1) is derived by performing selection to collect the students who were born in Canada and then projection on the set of relevant attributes: *major, gpa, status, birth_place,* and *address.*

major	gpa	status	birth_place	address
CS	3.85	Senior	Vancouver, B.C., Canada	123 Curtis, Burnaby, B.C., Canada
...	...	...	...	...

Table 6.1: A fragment of *student* relation in relevance to the data mining task.

2. Large 1-predicate-set tables at multiple conceptual levels, (as shown in Table 6.2), i.e., $\mathcal{L}[1,1]$, $\mathcal{L}[2,1]$, ..., $\mathcal{L}[max\_1,1]$, are derived by scanning the initial data relation $\mathcal{R}_0$ once.

$\mathcal{L}[1,1]$

major	count
Science	4,850
...	...
gpa	count
Excellent	2,173
...	...
status	count
Underg.	20,204
...	...

$\mathcal{L}[2,1]$

major	count
Appl._Sci.	1,364
...	...
gpa	count
3.8_4.0	1,731
...	...
status	count
Senior	4,204
...	...

$\mathcal{L}[3,1]$

major	count
CS	675
...	...
gpa	count
3.8_3.9	1,043
...	...
status	count
Senior	4,204
...	...

Table 6.2: A fragment of large 1-predicate tables at different conceptual levels.

3. Large 2-predicate-sets at multiple conceptual levels (as shown in Table 6.3), i.e., $\mathcal{L}[1,2]$, $\mathcal{L}[2,2]$, ..., $\mathcal{L}[max\_1,2]$, are derived by first generating the candidate

large 2-predicate-sets and then scanning $\mathcal{R}_0$ to compute the large 2-predicate-sets.

$\mathcal{L}[1,2]$

major	gpa	count
Science	Excellent	819
...	...	...
major	status	count
Science	Underg.	6,914
...	...	...

$\mathcal{L}[2,2]$

major	gpa	count
Appl._Sci.	3.8_4.0	327
...	...	...
major	status	count
Appl._Sci.	Senior	2,149
...	...	...

$\mathcal{L}[3,2]$

major	gpa	count
CS	3.8_3.9	174
...	...	...
major	status	count
CS	Senior	891
...	...	...

Table 6.3: A fragment of large 2-predicate tables at different conceptual levels.

4. This process continues until the large p-predicate-sets at multiple conceptual levels, i.e., $\mathcal{L}[1,p]$, $\mathcal{L}[2,p]$, ..., $\mathcal{L}[max\_l,p]$, where p is the total number of predicates in the meta-rule, are derived. The tables so derived for the large 3-predicate sets are presented in Table 6.4.

5. The rules in the form of meta-rules are generated in Table 6.5 from the large 3-predicate-sets at multiple conceptual levels, based on the specified confidence threshold at each level. □

The above example leads to the following algorithm for mining meta-rule guided single-variable strong ML-association rules using large predicate growing technique.

Algorithm 6.3.1 (large predicate-growing) *Meta-rule guided mining of single-variable strong ML-association rules using large predicate growing technique.*

$\mathcal{L}[1,3]$

major	gpa	status	count
Science	Excellent	Underg.	526
...	...	...	...

$\mathcal{L}[2,3]$

major	gpa	status	count
Appl._Sci.	3.8_4.0	Senior	274
...	...	...	...

$\mathcal{L}[3,3]$

major	gpa	status	count
CS	3.8_3.9	Senior	180
...	...	...	...

Table 6.4: A fragment of large 3-predicate tables at different conceptual levels.

Input: (1) $\mathcal{DB}$, a relational database, (2) $\mathcal{H}$, a conceptual hierarchy, (3) $minsup[l]$, the minimum support threshold, and $minconf[l]$, the minimum confidence threshold, for each conceptual level l, and (4) $meta\_R$, the meta-rule in the form of (6.3.8).

Output: Multiple-level strong association rules in the form of (6.3.8) discovered in relational database $\mathcal{DB}$.

Method: A top-down, progressively deepening process which collects large predicate sets at different conceptual levels as follows.

1. The initial data relation $\mathcal{R}_0$ is derived by executing an SQL query specified by the data mining query.

2. Large 1-predicate-set tables at each conceptual level, i.e., $\mathcal{L}[1,1]$, $\mathcal{L}[2,1]$, ..., $\mathcal{L}[max\_l,1]$, are derived by scanning the initial data relation $\mathcal{R}_0$ once. Note that a predicate $p_i(t,c_i)$ is large at level l (and thus being included in $\mathcal{L}[l,1]$) if (1) its support is lower than $minsup[l]$, and its corresponding concept c_i' at a higher-level $l-1$ is large.

Rule	Support	Confidence
...	...	...
major(s, "Science") $\wedge$ *birth_place(s, "B.C.")* $\rightarrow$ *address(s, "Burnaby")*	25%	55%
...	...	...
major(s, "CS") $\wedge$ *gpa(s, "3.8_3.9")* $\rightarrow$ *status(s, "Senior")*	5%	25.6%
...	...	...

Table 6.5: Rules generated from the large 3-predicate tables at different conceptual levels.

3. Derive the large k-predicate-set tables at each conceptual level and for each k from 2 to p, i.e., derive $\mathcal{L}[l,k]$, for $l = 1,\ldots,max\_l$, and $k = 2,\ldots,p$, where p is the total number of predicates in the meta-rule.

   Note that a set of k predicates is large at level l if (1) each of its k subsets of $(k-1)$ predicates is large at level l, and (2) the support of the k predicates at level l is no less than $minsup[l]$.

4. For each conceptual level l, generate the rules in the form of meta-rules from the large p-predicate set tables $\mathcal{L}[l,p]$ if the confidence of the rule is no less than $minconf[l]$, the specified confidence threshold at this level. □

6.3.2 A Direct p-predicate Testing Technique

The previous algorithm is a natural extension of the method developed in the study of mining multiple-level association rules [47]. A major difference of the requirements in meta-rule guided mining from that in the mining of general multiple-level association rules is that p, the number of large predicates in the rules to be generated, is predefined by the given meta-rule. This heuristic can be used in the development of the variations of the rule mining algorithms.

Here we consider one variation of the mining technique: a *direct p-predicate generation and testing* technique. At the third step of Algorithm 6.3.1, instead of deriving

large 2-predicate-sets at each conceptual level, and then large 3-predicates, etc., p-predicate sets are generated directly from the large 1-predicate sets and tested against the support threshold at each level. This technique is illustrated in the following similar example, followed by the algorithm for mining meta-rule guided single-variable strong ML-association rules using the p-predicate testing technique.

Example 6.3.2 We examine the derivation of the multiple-level strong association rules for query (q_1) of Example 6.2.1.

1. The same as Step 1 and Step 2 of Example 6.3.1.
2. Large p-predicate-sets at multiple conceptual levels, i.e., $\mathcal{L}[1,p]$, $\mathcal{L}[2,p]$, ..., $\mathcal{L}[max\_l,p]$, are derived based on the large 1-predicate sets derived at previous step. This skips the generation of the large 2-predicate tables of Example 6.3.1 and generates only the large 3-predicate tables as in Table 6.4.
3. The rules in the form of meta-rules are generated from the large p-predicate-sets at each conceptual level based on the specified confidence threshold at this level. This generates the same rule table as in Table 6.5. □

Algorithm 6.3.2 (Direct p-predicate testing) *Meta-rule guided mining of single-variable strong ML-association rules using the direct p-predicate derivation technique.*

Input: The same as Algorithm 6.3.1.

Output: The same as Algorithm 6.3.1.

Method: A top-down, progressively deepening process which collects large p predicate sets at multiple conceptual levels as follows.

1. The same as Step 1 and 2 of Algorithm 6.3.1.
2. Derive the large p-predicate-set tables at each conceptual level from level 1 to max_l, i.e., derive $\mathcal{L}[l,p]$, for $l = 1,\ldots,max\_l$, where p is the total number of predicates in the meta-rule.

Note that a set of p predicates is large at level l if (1) each of its component 1-predicates is large at level l, and (2) the support of the p predicates at level l is no less than $minsup[l]$.

3. For each conceptual level l, generate the rules in the form of meta-rules from the large p-predicate set tables $\mathcal{L}[l, p]$ if the confidence of the rule is no less than $minconf[l]$, the specified confidence threshold at this level. □

6.3.3 A Performance Comparison of the Two Algorithms

We implemented the large-predicate growing and the direct p-predicate testing algorithms on a SUN SparcStation5 with 32MB main memory. A synthetic database is used to test the algorithms. The database has five attributes each of which has 100 values at the primitive level. The values are organized into a conceptual hierarchy with four levels. The numbers of higher level (nonprimitive) nodes in the hierarchy are 1, 5 and 20 at level 1, 2, 3 respectively. Since there is only one node at the level 1, it is treated as a virtual level and does not join the computation. The meta-rule we used has the form: $A(t, x) \wedge B(t, y) \rightarrow C(t, z)$. The minimal confidences are 50% at all levels.

First, we test the scale-up properties of the two algorithms. They are tested on the database with the number of tuples from 10,000 to 100,000. The minimal supports are (4%, 1%, 0.2%) at levels 2, 3 and 4. The performance data are shown in Figure 6.1. As we can see, both algorithms scale up well. Algorithm 6.3.1 has better scale-up behavior since the overhead of computing $\mathrm{L}[l, k]$ for small k weights less and less as the database size grows.

We then compared the performance of the algorithms under different minimal supports. Figure 6.2 shows the execution times of both algorithms with different minimal supports. The database size is fixed at 10,000 tuples. The minimal supports used are: T1(6%, 1%, 0.5%), T2(4%, 1%, 0.1%), T3(4%, 0.5%, 0.1%), T4(2%, 0.5%, 0.1%), and T5(2%, 0.5%, 0.05%). When the minimal supports decrease, the execution times increase since the filter becomes weaker. We find that Algorithm 6.3.1 is sensitive to the minimal supports since it uses them to cut out small patterns at each iteration.

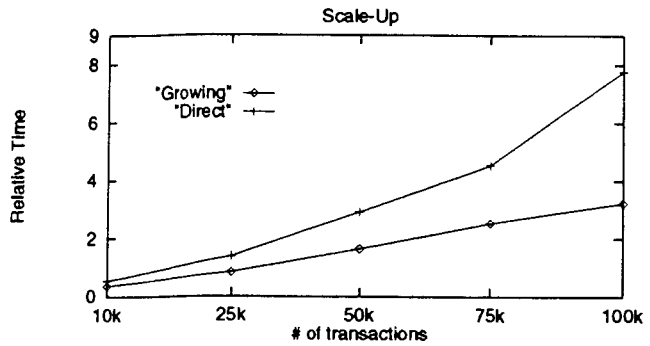

Figure 6.1: Scale up of the algorithms.

On the other hand, Algorithm 6.3.2 is not so sensitive to the change. Algorithm 6.3.1 outperforms Algorithm 6.3.2 when the minimal supports are large (so the filter is strong) while Algorithm 6.3.2 outperforms Algorithm 6.3.1 when the filter is not very strong. Generally, we feel Algorithm 6.3.1 should be tried for most reasonable support thresholds. Algorithm 6.3.2 is a good candidate when lots of details are interested, i.e., when the support thresholds are small.

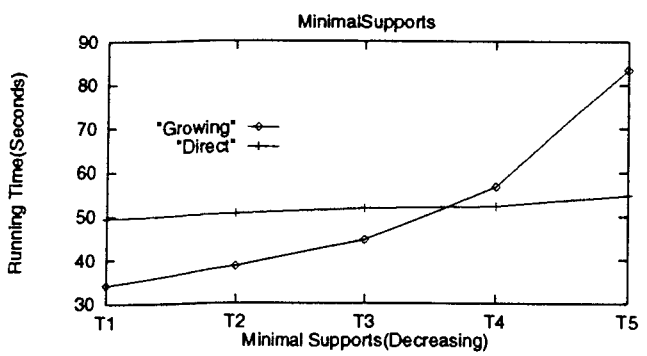

Figure 6.2: Relative performance with respect to minimal support.

6.4 Meta-Rule Guided Mining of Multiple-Variable Rules

Now we examine the meta-rule guided mining of multiple-variable rules. Since a multiple-variable association rule presents relationships among several relations, a join of these relations should be performed in the data collection step based on the join relationship explicitly expressed in the meta-rules.

Taking query (q_2) in Example 6.2.1 as an example, we analyze the data mining process as follows.

Example 6.4.1 The meta-rule presented in query (q_2) of Example 6.2.1 contains three predicates: $major(s, x)$, $P(c, y)$, and $Q(s, c, z)$. The first predicate is from the attribute *major* of the relation *student*, the second is a property in relevance to the relation *course* because it contains one variable from *course*, and the third is a property in relevance to the relation *grading* since it contains two variables, each from *student* and *course*, respectively.

The data mining process is to discover the relationships in relevance to three relations: *student*, *course*, and *grading*. It is necessary to perform a join of the three relations. Since only one predicate $major(s, x)$ is from the relation *student*, only the attribute *major* in the relation *student* is retained in the joined relation. Therefore, the joined relation should have the following schema.

s_c_g (sno, major, cno, title, dept, instructor, semester, grade)

The possible instantiations of the two candidate predicates P and Q should be: $P \in \{title, dept\}$, and $Q \in \{instructor, semester, grade\}$. Moreover, since *title* is unique in the relation *course*, which is similar to the behavior of the key *cno*, the predicate P in the meta-rule can only be instantiated to *dept*. Therefore, the data mining process is essentially to find multiple-level association rules in relevance to the following three properties: (1) $major(s, x)$, (2) $dept(c, y)$, and (3) one of the following three predicates: $instructor(s, c, z)$, $semester(s, c, z)$, $grade(s, c, z)$.

Except for the restriction on the instantiation of predicate variables, the data mining methods are like that of mining single-variable association rules. □

6.5 Discussion

This section discusses some closely-related issues on meta-rule guided mining of multiple-level association rules, including meta-rule-guided mining of mixed-level rules and variations of constraints on the forms of meta-rules.

6.5.1 Meta-Rule-Guided Mining of Mixed-Level Rules

In the method developed in Section 6.3, it is assumed that the concepts of the predicates in the discovered rules are lined up among different predicates according to the levels of their conceptual hierarchies. For example, major "Science" is lined up with gpa "Excellent" and birth_place "B.C.", whereas major "CS" is lined up with gpa "3.8_3.9" and birth_place "Vancouver", etc. However, it may not be the case in practical applications. It could be desirable to line up major "CS" with gpa "Excellent" and birth_place "British Columbia", etc. That is, it is often necessary to link concepts among different predicates at multiple levels of hierarchies for effective knowledge mining.

Interestingly, the method studied in the last two sections needs only minor modifications in order to accommodate this flexible data mining requirement. For example, Algorithm 6.3.1 can be modified as the following for mining rules across multiple conceptual levels. At the third step, the candidate large 2-predicate-sets will enclose the pairs of two large 1-predicate-sets at any conceptual levels instead of pairing only those at the same conceptual levels.

6.5.2 Variations of Constraints on the Forms of Meta-Rules

In our previous discussion, there has been another constraint on the possible forms of meta-rules: there are no repetitive predicate variables in the meta-rule, and all the predicates in an instantiated rule will be different.

Although this restriction may cover a large number of applications, there are applications which would like to study the association relationships involving the same predicates. For example, one may like to find the general association relationships

among the courses taken by the same student. Such a query could be presented and examined in the following example.

Example 6.5.1 In the university database of Example 6.2.1, one may like to find the association relationships among the courses taken by the same student. The query can be presented as follows.

(q_4) : discover rules in the form of
$P(s:S, c_1:C, x_1) \wedge P(s, c_2:C, x_2) \rightarrow P(s, c_3:C, x_3)$
from student S, grading G, course C

The system may find some meaningful rules like the following.

$$grade(s, \text{“}CMPT100\text{”}, \text{“}Excellent\text{”}) \wedge grade(s, \text{“}MATH100\text{”}, \text{“}Excellent\text{”}) \rightarrow grade(s, \text{“}CMPT300\text{”}, \text{“}A\text{”}) \quad (82\%)$$

Note in this case the data mining process can be viewed as a similar process of mining association rules in transaction databases [2]. This is because the relational table can be compressed into a table consisting of two fields: (1) a set of distinct students, each corresponding to a transaction identifier in a transaction database, and (2) a set of corresponding grading records associated with each student, each corresponding a set of data items processed by that transaction. Thus the transaction-based data mining algorithms developed in previous studies [4, 47] can be applied in the efficient processing of association relationships. However, the previously developed transaction-based association rule mining algorithms still need to be modified to accommodate more complicated queries.

6.6 Summary

We have studied the meta-rule guided mining of multiple-level association rules in large relational databases. Meta-rule guided mining of multiple-level association rules provides syntactic constraints on the desired rule forms to be discovered, which leads

to the constrained and progressive mining of refined knowledge from data and thus has interesting applications for knowledge discovery in large databases.

A top-down progressive deepening data mining technique is developed for rule-guided mining of multiple-level association rules, which extends the multiple-level association rule mining algorithms for rule-guided mining of association rules. Two algorithms, the large-predicate growing and the direct p-predicate testing, have been proposed and tested against synthetic databases, and their performance study shows that different algorithms may have the best performance for different distributions of data.

Related issues, including methods for mining flexible multiple-level association rules and relaxations of constrains on the forms of mete-rules are also discussed in this chapter. Our study shows that meta-rule guided mining of multiple-level association rules from databases has wide applications and efficient algorithms can be developed for discovery of interesting and strong such rules in large databases.

Chapter 7

Cooperative Query Answering Using Multiple Layered Databases

The data mining techniques we discussed in previous chapters can be applied to many areas, such as scheme evolution and integration [26], cooperative query answering [49], knowledge discovery on the Internet [52], etc.

In this chapter, we discuss the applications of data mining techniques in intelligent query answering.

7.1 Introduction

Cooperative (or intelligent) query answering refers to a mechanism which answers information system queries cooperatively and intelligently by analyzing the intent of a query and providing some generalized, neighborhood, or associated answers [20, 24, 39, 8]. Many interesting techniques [58, 25, 19, 38, 63] have been developed for cooperative query answering, by integration of the methods developed in several related fields, such as semantic data modeling, deductive databases, knowledge discovery in databases, etc.

In this chapter, we propose a new technique: *the construction and application of a multiple layered database*, and explore its potential and effectiveness in cooperative query answering. A multiple layered database (MLDB) is a database composed of

several layers of information, with the lowest layer corresponding to the primitive information stored in a conventional database, and with higher layers storing more general information extracted from lower layers.

We have the following motivations to promote the idea of *multiple layered databases.*

First, with the wide availability of database systems and rapid progress of information technologies, a database may store a huge set of data objects with complex structures. A large set of data objects may be organized in classes and class-subclass hierarchies and may contain complex structured or unstructured subobjects, texts, images, and spatial or multimedia data. Moreover, the data may be distributed to different sites and be stored in heterogeneous *multi*-databases. Queries on such kind of databases could be costly to process. A multiple layered database system may preprocess and generalize some primitive data, resolve certain semantic ambiguities of heterogeneous data, and store the preprocessed data at a more general conceptual layer, which may facilitate high-level querying and reduce the cost of query processing [92, 109].

Secondly, a database user may not be familiar with a database schema, a query language, or specific data constraints. It is likely that such a user may pose queries which are not exactly what (s)he wants to know. Such kind of queries are better treated as information probes and answered by providing general or associated information with data distribution statistics, which may help users to better understand the data and form more accurate queries [19, 8, 24]. In a multiple layered database system, probe queries can be mapped to a relatively higher conceptual layer and be processed in such a layer. Such answers may provide associative and summary information and assist users to refine their queries.

Thirdly, a multiple layered database may provide a global view of the current contents in a database with summary statistics. It is a natural resource to assist users to browse database contents, pose progressively refined queries, and perform knowledge discovery in databases. Some users may even be satisfied with the examination of the general or abstract data with associated statistical information in a high layer instead of examining the concrete data in every detailed level.

Finally, schema-directed semantic query optimization can be performed in a multiple layered database. A higher layer database, storing more general and abstract information, could be much smaller than its corresponding lower layer one. Thus, it is faster and less costly to retrieve data in a higher layer database. Moreover, since a multiple layered database provides statistical information of database contents in the higher layers, it may provide guided assistance for query processing and query optimization of its lower level counterparts.

In this chapter, we propose a model for a multiple layered database and study how to construct a multiple layered database and how to perform cooperative query answering using MLDBs.

The following sections are organized as follows. In Section 7.2, the concept of multiple layered database is introduced. The techniques for construction of a multiple layered database are studied in Section 7.3. Cooperative query answering using MLDBs is investigated in Section 7.5. The chapter is summarized in Section 7.6.

7.2 A Multiple Layered Database

To facilitate our discussion, we assume that the database to be studied is constructed based on an extended-relational data model with the capabilities to store and handle different kinds of complex data, such as structured or unstructured data, hypertext, spatial or multimedia data, etc. It is straightforward to extend our study to other data models, such as object-oriented, deductive, etc., and to other kinds of databases, such as distributed and heterogeneous databases.

Definition 7.2.1 A multiple layered database (MLDB) consists of 4 major components: ⟨S, H, C, D⟩, defined as follows.

1. S: a database schema, which contains the meta-information about the layered database structures;

2. H: a set of conceptual hierarchies;

3. C: a set of integrity constraints; and

4. D: a set of database relations, which consists of all the relations (primitive or generalized) in the multiple layered database. □

The first component, a database schema, outlines the overall database structure of an MLDB. It stores general information such as types, ranges, and data statistics about the relations at different layers, their relationships, and their associated attributes. More specifically, it describes which higher-layer relation is generalized from which lower-layer relation(s) and how the generalization is performed. Therefore, it presents a route map for schema browsing and database content browsing and for assistance of cooperative query answering and query optimization.

The second component, a set of conceptual hierarchies, is used to generalize lower layer relations to high layer ones and map queries to appropriate conceptual layers for processing, as discussed in Chapter 3.

The third component, a set of integrity constraints, consists of a set of integrity constraints to ensure the consistency of an MLDB.

The fourth component, a set of database relations, stores data relations, in which some of them are primitive (i.e., layer-0) relations, whereas others are higher layer ones, obtained by generalization.

Example 7.2.1 Suppose a real-estate database contains the following four data relations.

1. *house(house_id, address, construction_date, constructor(...), owner(name, ...), living_room(length, width), bed_room_1(...), ..., surrounding_map, house_layout, house_picture, house_video, listing_price).*

2. *customer (name, social_insurance_#, birth_date, education, income, work_address, home_address, spouse, children (...), phone, ...).*

3. *sales (house, buyer, agent, contract_date, sell_price, mortgage (...), ..., notes).*

4. *agent(...).*

These relations are layer-0 relations in the MLDB. Suppose the database contains the conceptual hierarchies for *geographic locations, occupations, income ranges*, etc. An MLDB can be constructed as follows.

First, the relation *house* can be generalized to a higher layered relation *house'*. The generalization can be performed, for example, as follows: (1) transform the *house construction date* to *years_old*, e.g., from "*Sept. 10, 1980*" to 16; (2) preserve the *owner's name* but remove other information associated with the *owner*; (3) compute the *total floor area* of all the rooms and the *number of rooms* but remove the detailed specification for each room; and (4) remove some attributes: *surrounding_map, house_layout, house_video*, etc. The generalized relation *house'* can be considered as the layer-1 information of the house, whose schema is presented as follows.

house'(house_id, address, years_old, owner_name, floor_area, #_of_rooms, ..., house_picture, listing_price).

Secondly, further generalization on *house'* can be performed to produce an even higher layered relation *house''*. For example, generalization may be performed as follows: (1) remove the attributes *house_id*, *owner*, *house_picture*, etc.; (2) generalize the *address* to *areas*, such as *North_Burnaby, East_Vancouver*, etc.; (3) generalize *years_old* to *year_range*, etc.; (4) transform *#_of_rooms* and other associate information into *category*, such as *5-bedroom house, 3-bedroom town-house*, etc.; and (5) merge identical tuples in the relation and store the total *count* of such merged tuples. The generalized relation *house''* could be as follows.

house''(area, year_range, floor_area_range, category, ..., price_range, count).

Similarly, *customer* can be generalized to *customer'*, *customer''*, etc., which forms multiple layers of a *customer* relation. Multiple layers can also be formed in a similar way for the relations, *sales* and *agent*.

A higher layered relation can also be formed by joining two or more primitive or generalized relations. For example, *customer_sales'* can be produced by generalization on the join of *customer'* and *sales'* as long as it follows the regulation(s) for the construction of MLDBs (to be presented in the next section). Similarly, one may join several relations at different layers to form new higher-layered relations, such as *house_sales_customer'*, etc.

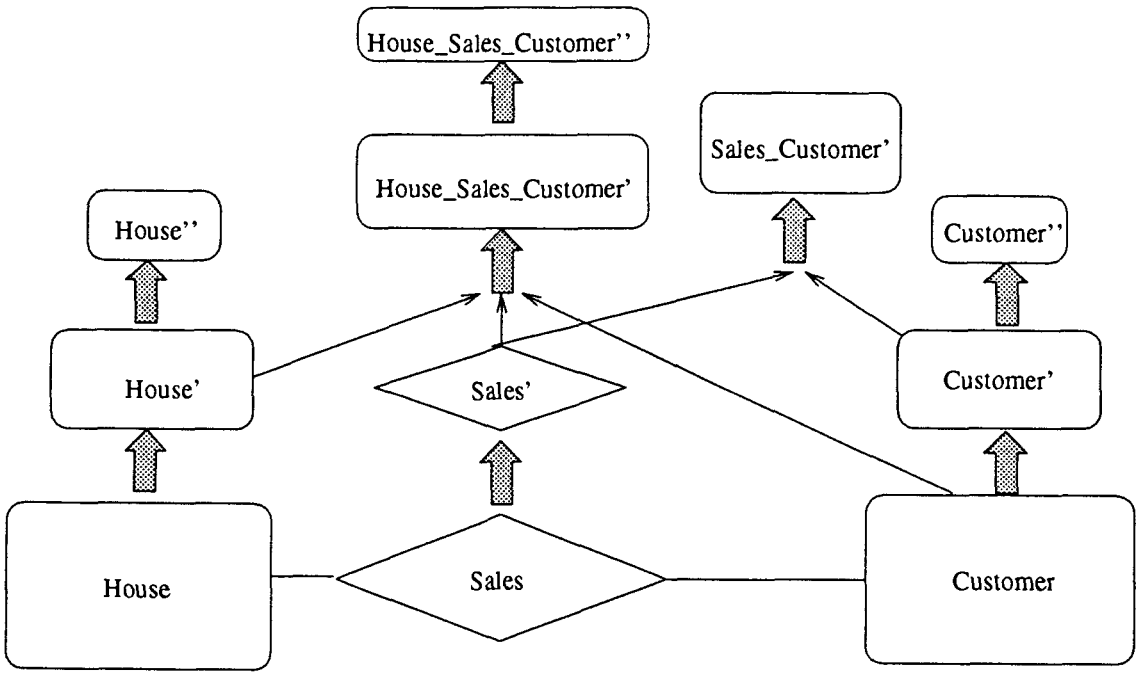

Figure 7.1: The route map of a real-estate DB.

A possible overall MLDB structure, i.e., the schema of an MLDB, is presented in Fig. 7.1.

Queries can be answered efficiently and intelligently using the MLDB. For example, a user may ask the information about the houses with the price range between \$250k and \$300k. The query can be answered intelligently by first using *house''*, which may return "*none in West Vancouver, 10% in East Vancouver, 15% in South Burnaby, etc.*". Such an answer may help the user form more accurate queries to search for houses in specific regions. □

7.3 Generalization of Different Kinds of Data

An MLDB is constructed by generalization of the layer-0 (original) database. Since a database may contain different kinds of complex data, it is important to examine the method for generalization of each kind of data, including unstructured and structured values, spatial and multimedia data, etc [70, 28].

7.3.1 Generalization of Unstructured Data

Single valued, numerical and nonnumerical data are the most popularly encountered attribute values in databases. The generalization on simple values can be performed using conceptual hierarchy climbing, i.e., replacing the lower data or concepts with corresponding high level data or concepts. The generalization may rely on the available hierarchies, specified by domain experts or users or implicitly stored in the database, as we discussed in Section 3.1.3. In addition, conceptual hierarchies may be dynamically adjusted based on the data distribution in order to best meet the current request, as we discussed in Section 3.2. Moreover, conceptual hierarchies can be automatically generated for numerical data as shown in Section 3.3.

7.3.2 Generalization of Structured Data

Complex structure-valued data, such as set-valued and list-valued data and data with nested structures, can be generalized in several ways in order to be interesting.

A set-valued attribute may be of homogeneous or heterogeneous types. Typically, a set-valued data can be generalized in two ways: (1) generalization of each value in a set into its corresponding higher level concepts, or (2) derivation of the general behavior of a set, such as the number of elements in the set, the types or value ranges in the set, the weighted average for numerical data, etc. Moreover, the generalization can be performed by applying different generalization operators to explore alternative generalization paths. In this case, the result of generalization is a heterogeneous set.

For example, the *hobby* of a person is a set-valued attribute which contains a set of values, such as {*tennis, hockey, chess, violin, nintendo*}, which can be generalized into a set of high level concepts, such as {*sports, music, video_games*}, or into 5 (the number of hobbies in the set), or both, etc. Moreover, a *count* can be associated with a generalized value to indicate how many elements are generalized to the corresponding generalized value, such as {*sports(3), music(1), video_games(1)*}, where *sports(3)* indicates *three kinds of sports*, etc.

A list-valued or a sequence-valued attribute can be generalized in a way similar to the set-valued attribute except that the order of the elements in the sequence should

be observed in the generalization.

Set- and list-valued attributes are simple structure-valued attributes. In general, a structure-valued attribute may contain sets, tuples, lists, trees, records, etc. and their combinations. Furthermore, one structure can be nested in another structure at any level. Similar to the generalization of set- and list-valued attributes, a general structure-valued attribute can be generalized in several ways, such as (1) generalize each attribute in the structure whereas maintain the shape of the structure, (2) flatten the structure and generalize on the flattened structure, (3) remove the low-level structures or summarize the low-level structures by high-level concepts or aggregation, and (4) return the type or an overview of the structure.

7.3.3 Aggregation and Approximation as a Means of Generalization

Besides conceptual hierarchy ascension and structured data summarization, aggregation and approximation [101, 95] should be considered as an important means of generalization, which is especially useful for generalization of attributes with large sets of values, complex structures, spatial or multimedia data, etc.

Take spatial data as an example. It is desirable to generalize detailed geographic points into clustered regions, such as business, residential, industry, or agricultural areas, according to the land usage. Such generalization often requires the merge of a set of geographic areas by spatial operations, such as spatial union, or spatial clustering algorithms. Approximation is an important technique in such generalization. In spatial merge, it is necessary not only to merge the regions of similar types within the same general class but also to ignore some scattered regions with different types if they are unimportant to the study. For example, different pieces of land for different purposes of agricultural usage, such as vegetables, grain, fruits, etc. can be merged into one large piece of land by spatial merge. However, such an agricultural land may contain highways, houses, small stores, etc. If the majority land is used for agriculture, the scattered spots for other purposes can be ignored, and the whole region can be

claimed as an agricultural area by approximation. The spatial operators, such as *spatial_union*, *spatial_overlapping*, *spatial_intersection*, etc., which merge scattered small regions into large, clustered regions can be considered as generalization operators in spatial aggregation and approximation.

7.3.4 Generalization on Multimedia Data

A multimedia database may contain complex text, graphics, images, maps, voice, music, and other forms of audio/video information. Such multimedia data are typically stored as sequences of bytes with variable lengths, and segments of data are linked together for easy reference. Generalization on multimedia data can be performed by recognition and extraction of the essential features and/or general patterns of such data.

There are many ways to extract the essential features or general patterns from segments of multimedia data. For an image, the size and color of the contained objects or the major regions in the image can be extracted by aggregation and/or approximation. For a segment of music, its melody can be summarized based on the approximate patterns that repeatedly occur in the segment and its style can be summarized based on its tone, tempo, major musical instruments played, etc. For an article, its abstract or general organization such as the table of contents, the subject and index terms frequently occurring in the article, etc. may serve as generalization results. In general, it is a challenging task to generalize multimedia data to extract the interesting knowledge implicitly stored in the data [29]. Further research should be devoted to this issue.

7.4 Construction of MLDB

7.4.1 Frequently Referenced Attributes and Frequently Used Patterns

With attribute generalization techniques available, the next important question is how to selectively perform appropriate generalizations to form useful layers of databases. In principle, there could be a large number of combinations of possible generalizations by selecting different sets of attributes to generalize and selecting the levels for the attributes to reach in the generalization. However, in practice, a few layers containing most frequently referenced attributes and patterns will be sufficient to handle most practically important cases.

Frequently used attributes and patterns should be determined before generation of new layers of an MLDB by the analysis of the statistics of query history or by receiving instructions from users or experts. If users are often interested in one set of attributes but rarely asking things related to another set, it is wise to remove those rarely used attributes in a higher layer. Similar guidelines apply when generalizing attributes to a more general conceptual level. For example, users may like the oldness of a house to be expressed by the *ranges* (of the construction years) such as {*below_5, 6-15, 16-30, over_30*} instead of the exact *construction date*, etc.

A new layer could be formed by performing generalization on one relation or on a join of several relations based on the selected, frequently used attributes and patterns. Generalization is performed by removing a set of less-interested attributes, substituting the concepts in one or a set of attributes by their corresponding higher level concepts, performing aggregation or approximation on certain attributes, etc. [13]

Since most joins of several relations are performed on their key and/or foreign key attributes, whereas generalization may remove or generalize the key or foreign key attributes of a data relation, it is important to distinguish the following two classes of generalizations.

1. key-preserving generalization, in which all the key or foreign key values are preserved.

2. **key-altering generalization**, in which some key or foreign key values are generalized, and thus altered. The generalized keys should be marked explicitly since they cannot be used as join keys at generating subsequent layers.

It is crucial to identify altered keys since if the altered keys were used to perform joins of different relations, it may generate incorrect information. This is observed in the following example.

Example 7.4.1 Suppose one would like to find the relationships between the ages of the houses sold and the household income level of the house buyers. Let the relations *house'*, *sales_customer'* contain the following tuples.

$house'(945\_Austin, \ldots, 35(years\_old), \ldots)$.

$house'(58\_Austin, \ldots, 4(years\_old), \ldots)$.

$sales\_customer'(945\_Austin, mark\_lee, 30\_40k(income), \ldots)$.

$sales\_customer'(58\_Austin, tim\_akl, 60\_70k(income), \ldots)$.

Their further generalization may result in the relations *house''*, *sales_customer''* containing the following tuples.

$house''(North\_Burnaby, \ldots, over\_30(years\_old), \ldots)$.

$house''(North\_Burnaby, \ldots, below\_5(years\_old), \ldots)$.

$sales\_customer''(North\_Burnaby, 30\_40k(income), \ldots)$.

$sales\_customer''(North\_Burnaby, 60\_70k(income), \ldots)$.

If the join is performed between *house'* and *sales_customer'*, it will still produce the correct information as below.

$house\_customer'(945\_Austin, 35, mark\_lee, 30\_40k, \ldots)$.

$house\_customer'(58\_Austin, 4, tim\_akl, 60\_70k, \ldots)$.

Further generalization can still be performed on such a joined relation.

However, if the join is performed on the altered keys between *house''* and *sales_customer''*, it will generate 4 tuples, which is incorrect.

$house\_customer''(North\_Burnaby, over\_30, 30\_40k, \ldots)$.

$house\_customer''(North\_Burnaby, over\_30, 60\_70k, \ldots)$.

$house\_customer''(North\_Burnaby, below\_5, 30\_40k, \ldots)$.

$house\_customer''(North\_Burnaby, below\_5, 60\_70k, \ldots)$.

Obviously, joins on the generalized attributes may produce more tuples than joins on the original ones since different values in the attribute may have been generalized to identical ones at a high layer. □

This restriction leads to the following regulation.

Regulation 7.4.1 (Join in MLDB) A join in an MLDB cannot be performed on the generalized attributes.

Based on this regulation, if the join in an MLDB is performed on the generalized attributes, it is called an information-loss join (since the information could be lost by such a join). Otherwise, it is called an information-preserving join.

7.4.2 An MLDB Construction Algorithm

Based on the previous discussion, the construction of an MLDB can be summarized into the following algorithm.

Algorithm 7.4.1 Construction of an MLDB.

Input: A relational database, a set of conceptual hierarchies, and a set of frequently referenced attributes and frequently used query patterns.

Output: A multiple layered database.

Method. An MLDB is constructed in the following steps.

1. Determine the multiple layers of the database based on the frequently referenced attributes and frequently used query patterns.
2. Starting with the most specific layer, generalize the relation step-by-step (using the given conceptual hierarchies) to form multiple layered relations (according to the layers determined in Step 1).
3. Merge identical tuples in each generalized relation and update the *count* of the generalized tuple.

4. Construct a new schema by recording all the primitive and generalized relations, their relationships and the generalization paths. □

Rationale of Algorithm 7.4.1.

Step 1 indicates that the layers of an MLDB should be determined based on the frequently referenced attributes and frequently used query patterns. This is reasonable since to ensure the elegance and efficiency of an MLDB, only a small number of layers should be constructed, which should provide maximum benefits to the frequently accessed query patterns. Obviously, the frequently referenced attributes should be preserved in higher layers, and the frequently referenced conceptual levels should be considered as the candidate conceptual levels in the construction of higher layers. Steps 3 and 4 are performed in a method similar to attribute-oriented induction [45, 13]. Step 5 constructs a new schema which records a route map and the generalization paths for database browsing and cooperative query answering, which is discussed in detail below. □

7.4.3 Schema: A Route Map and a Set of Generalization Paths

Since an MLDB schema provides a route map, i.e., a general structure of the MLDB for query answering and database browsing, it is important to construct a concise and information-rich schema. In addition to the schema information stored in a conventional relational database system, an MLDB schema should store two more important pieces of information:

1. A route map, which outlines the relationships among the relations at different layers of the database. For example, it shows which higher layered relation is generalized from one or a set of lower layered relations.

2. A set of generalization paths, each of which shows *how* a higher layered relation is generalized from one or a set of lower layered relations.

Similar to many extended relational databases, a *route map* can be represented by an extended E-R (entity-relationship) diagram [106], in which the entities and relationships at layer-0 (the original database) can be represented in a conventional E-R diagram [67]; whereas generalization is represented by a double-line arrow pointed from the generalizing entity (or relationship) to the generalized entity (or relationship). For example, *house'* is a higher layered entity generalized from a lower layer entity *house*, as shown in Fig. 7.1. Similarly, *sales_customer'* is a higher layered relationship, obtained by generalizing the join of *sales'* and *customer'*. It is represented as a generalization from a relationship obtained by joining one entity and one relationship in the route map (Fig. 7.1). Since an extended E-R database can be easily mapped into an extended relational one [67], our discussion assumes such mappings and still adopts the terminologies from an extended relational model.

A *generalization path* is created for each high layer relation to represent how the relation is obtained in the generalization. Such a high layer relation is possibly obtained by removing a set of infrequently used attributes, preserving some attributes and/or generalizing the remaining set of attributes. Since attribute removing and preserving can be obviously observed from a relational schema, the generalization path need only register how a set of attributes are generalized. A generalization path consists of a set of entries, each of which contains three components: ⟨*old_attr(s), new_attr(s), rules*⟩, which tells how one or a set of old attributes is generalized into a set of new (generalized) attributes by applying some generalization rule(s), such as generalizing to which conceptual levels of a conceptual hierarchy, applying which aggregation operations, etc. If an existing hierarchy is adjusted or a new hierarchy is created in the formation of a new layer, such a hierarchy should also be registered in H, the hierarchy component of an MLDB.

7.4.4 Maintenance of MLDBs

Since an MLDB is resulted from extracting extra-layers from an existing database by generalization, an MLDB will take more disk space than its corresponding single layered database. However, since a higher layer database is usually much smaller

than the original database, query processing is expected to be more efficient if done in a higher database layer. The rapid progress of computer hardware technology has reduced the cost of disk space dramatically in the last decade. Therefore, it could be more beneficial to trade disk space with intelligent and fast query answering.

- In response to the updates to the original relations, the corresponding higher layers should be updated accordingly to keep the MLDB consistent. Incremental update algorithms should be developed to minimize the cost of update propagation. Here we examine how to propagate incremental database updates at insertion, deletion and update of tuples in an original relation.

  When a new tuple t is inserted into a relation R, t should be generalized to t' according to the route map and be inserted into its corresponding higher layer. Such an insertion will be propagated to higher layers accordingly. However, if the generalized tuple t' is equivalent to an existing tuple in this layer, it needs only to increment the count of the existing tuple, and further propagations to higher layers will be confined to count increment as well. The deletion of a tuple from a data relation can be performed similarly.

  When a tuple in a relation is updated, one can check whether the change may affect any of its high layers. If not, do nothing. Otherwise, the algorithm will be similar to the deletion of an old tuple followed by the insertion of a new one.

  Although an MLDB consists of multiple layers, database updates should always be performed at the primitive database (i.e., layer-0) and the updates are then propagated to their corresponding higher layers. This is because a higher layer represents more general information, and it is impossible to transform a more general value to a more specific one, such as from *age* to *birth-date* (but it is possible in the reverse direction by applying appropriate generalization rules).

- To response to the changes in the frequently referenced attributes or the frequently used patterns, the affected layers should be updated accordingly and the changes should be recorded in the schema of the MLDB.

When an attribute becomes a frequently referenced attribute, it is added into higher layer relations and the data of the attribute are generalized into these higher layer databases. When an attribute is no longer a frequently referenced attribute, it may be removed from the higher layer databases. However, a "grace" period can be waited before the deletion, in case the attribute became a frequently referenced attribute again in the near future. The changes in frequently used patterns can be assimilated in a similar way, by inserting or deleting corresponding relations. All the changes should be recorded in the schema of the MLDB.

An incremental update approach can be taken because only the affected layers or relations need to be changed. For example, if the attribute *constructor_warranty* becomes frequently referenced, it can be added into relations *house'* and *house''*, but not others.

- The changes in the schema of primitive layer relations can be assimilated easily, by updating the corresponding higher layer relations and the schema of the MLDB.

  When a new attribute is added into a relation, it is recorded in the schema of the MLDB. The higher layers are not affected unless the attribute is a frequently referenced attribute. The deletion of an attribute from a primitive layer relation, however, will cause the attribute (if there is any) to be deleted from the higher layers.

Note that the changes can be reported by the database administrators or detected automatically by periodical sampling.

7.5 Query Answering in an MLDB

A query consists of user-provided information (*query constants*) and inquired information, where the former (query constants) could be the concepts matching different layers; whereas the latter may be mapped to different layers of an MLDB as well.

Moreover, one may expect that the query be answered *directly* by strictly following the request, or *intelligently* by providing some generalized, neighborhood, or associated answers.

We first examine the mechanisms for *direct* answering of queries in an MLDB and then extend the results to *cooperative* query answering.

7.5.1 Direct Query Answering in an MLDB

Direct query answering refers to answering queries by strictly following query specifications without providing (extra) associative information in the answers. Rigorously speaking, if all the provided and inquired information of a query are at the primitive conceptual level, a query can be answered directly by searching the primitive layer without exploring higher layers. However, a cooperative system should provide users with flexibility of expressing query constants and inquiries at a relatively high conceptual level. Such kind of "high-level" queries can be answered directly in an MLDB.

At the first glance, it seems to be easy to process such high-level queries by simply matching the constants and inquires in the query to a corresponding layer and then directly processing the query in this layer. However, there could be dozens of attributes in a relation and each attribute may have several conceptual levels. It is impossible and often undesirable to construct all the possible generalized relations whose different attributes are at different conceptual levels. In practice, only a small number of all the possible layers will be stored in an MLDB based on the analysis of the frequently referenced query patterns. This implies that transformations often need to be performed on some query constants to map those constants to a conceptual level corresponding to that of an existing layered database.

In principle, a high-level query constant is defined in a conceptual hierarchy, based on which the high-level constant can be mapped to primitive level concepts. For example, "*greater Vancouver area*" can be mapped to all of its composite regions, and "*big house*" can be mapped to "$total\_floor\_area > 3,000(sq.\ ft.)$", etc. Thus, a query can always be transformed into a primitive level query and be processed in

a layer-0 database. However, to increase processing efficiency and present high-level (and more meaningful) answers, our goal is to process a query in the highest possible layer, *consistent* with all of the query constants and inquiries.

Definition 7.5.1 A database layer L is consistent on an attribute A_i with a query q if the constants of attribute A_i in query q can absorb (i.e., level-wise higher than) the concept(s) (level) of the attribute in the layer.

For example, if the query constant in query q for the attribute "*house_area*" is "*big*", whereas the conceptual level for "*house_area*" in layer L is the same as "*big*", or lower, such as "3,000-4,999", "over_5,000", etc., then layer L is consistent with query q on the attribute "*house_area*".

Definition 7.5.2 The watermark of a (nonjoin) attribute A_i for query q is the topmost database layer which is consistent with the conceptual level of query constants/inquiries of attribute A_i in query q.

Lemma 7.5.1 *All the layers lower than the watermark of an attribute A_i for query q must be consistent with the values of attribute A_i in query q.*

We first examine the case that a query references only one generalized relation and all the high level query constants are nonnumerical values.

Proposition 7.5.1 *If a query q references only one generalized relation and all the high level query constants are nominal (nonnumerical) values, the highest possible layer consistent with the query should be the lowest watermark of all the participant attributes of q in the route map of the MLDB.*

Rationale. Suppose layer L is the lowest watermark of all the participant attributes of q in the route map of the MLDB. Since a layer lower than the watermark of attribute A_i must be consistent with the corresponding query constant/inquiry on attribute A_i, L must be consistent with all the constants and inquiries of all the participant attributes of query q. Furthermore, since a watermark for an attribute is the highest possible database layer for such an attribute, the layer so derived must be the highest possible layer which is consistent with all the participating attributes in the query. □

We then examine the case of queries involving join(s) of two or more relations. If such a join or its lower layer is already stored in the MLDB by an information-preserving join, the judgement should be the same as the case for single relations. However, if no such a join has been performed and stored as a new layer in the MLDB, the watermark of such a join attribute must be the highest database layer in which generalization has not been performed on this attribute (i.e., on which the information-preserving join can be performed). This is because a join cannot be performed on the generalized attributes according to Regulation 1.

Definition 7.5.3 The watermark of a join attribute A_i for query q is the topmost database layer which is consistent with the conceptual level of query constants/inquiries of attribute A_i in query q and in which the information-preserving join can be performed on A_i.

Thus, we have the following proposition.

Proposition 7.5.2 *If a query q involves a join of two or more relations, and all the high level query constants are nominal (nonnumerical) constants, the highest possible layer consistent with the query should be the lowest watermark of all the participant attributes (including the join attributes) of q in the route map of the MLDB.*

Example 7.5.1 Suppose the query on the real-estate MLDB is to describe the relationship between *house* and *sales* with the following given information: *located in North-Vancouver, 3-bedroom house, and sold in the summer of 1993.* Moreover, suppose the route map of an MLDB corresponding to this query is shown in Fig. 7.2.

The query involves a join of *sales* and *house* and the provided query constants are all at the levels high enough to match those in *house''* and *sales''*. However, joins cannot be performed at these two high layer relations since the join attributes of *house''* and *sales''* have been generalized (with their join keys altered). The watermarks of the join attributes, *house.location* and *sales.house_loc*, are one layer lower than their topmost layers.

If there exists a relation such as *house_sales* in the MLDB, which represents the join between the two relations and/or their further generalizations, the query can

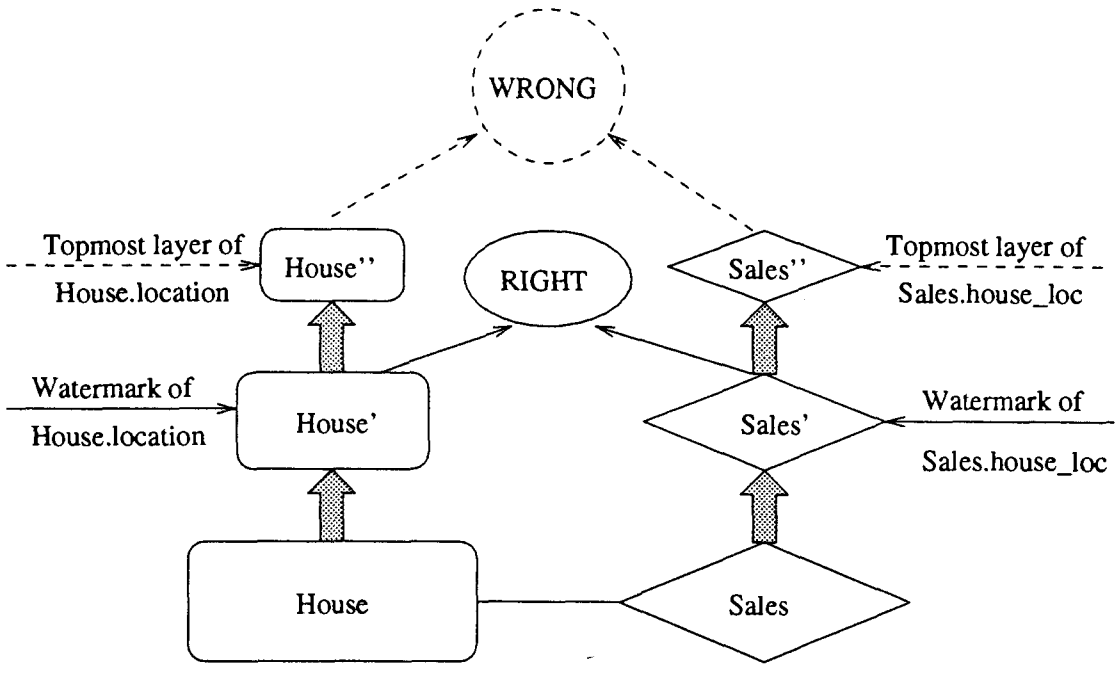

Figure 7.2: Perform joins in an MLDB.

be processed within such a layer. Otherwise (as shown in Fig. 7.2), a join must be performed on the highest joinable layers (which should be *house'* and *sales'*, as shown in Fig. 7.2). Further generalization can then be performed on this joined relation to form appropriate answers. □

Finally, we examine the determination of the highest possible database layers if the query contains numeric attributes. If the value in a numeric attribute in the query is expressed as a generalized constant, such as "*expensive*", or the specified range in the query has an exact match with some (generalized) range in a conceptual hierarchy, such as "*$300-400k*", the numeric value can be treated the same as a nonnumeric concept. Otherwise, we have two choices: (1) set the watermark of the attribute to the highest layer in which such numeric attributes has not been generalized, or (2) relax the requirement of the preciseness of the query answering. In later case, the appropriate layer is first determined by nonnumeric attributes. A coverage test is then performed to see whether the generalized range is entirely covered by the range provided in the query. For those entirely covered (generalized) ranges, the

precision of the answer remains the same. However, for those partially covered, the answer provided should be associated with certain probability for uncovered numerical ranges (e.g., by assuming that the data are relatively uniformly distributed within the generalized range), or be associated with a necessary explanation to clarify that the answer occupies only a portion of the entire generalized range.

Example 7.5.2 Suppose the query on the real-estate database is to describe the big houses in North-Vancouver with the price ranged from \$280k to \$350k. Since the query is to *describe* houses (not to find exact houses), the inquired portion can be considered at a high layer, matching any layers located by its query constants. To find the layer of its query constants, we have "*house_size* = *big*", "*address* = *North-Vancouver*", and *price range* = \$280k-\$350k. The watermarks of the first two are at the layer *house*", whereas the third one is a range value. Suppose in the layer *house*", the generalized tuples may have the ranges like \$250k-\$300k, \$300k-\$350k, etc., which do not have the exact match of the range \$280k-\$350k. Still, the query can be processed at this layer, with the information within the range \$300k-\$350k returned without additional explanation, but with the information within the range \$250k-\$300k returned, associated with an explanation that the returned information is for the range of \$250k-\$300k instead of \$280k-\$300k to avoid misunderstanding. □

7.5.2 Cooperative Query Answering in an MLDB

Since an MLDB stores general database information in higher layers, many techniques investigated in previous research on cooperative query answering in (single layered) databases [58, 25, 19, 38, 20] can be extended to cooperative query answering in MLDBs, easily, effectively and efficiently.

The following reasoning may convince us that an MLDB can greatly facilitate cooperative query answering.

1. Many cooperative query answering techniques need certain kinds of generalization [20, 39]; whereas different kinds of frequently used generalizations are performed and stored in the higher layers of an MLDB.

2. Many cooperative query answering techniques need to compare the "neighborhood" information [25, 20]; whereas the generalized neighborhood tuples are usually stored in the same higher layer relations, ready for comparison and investigation.

3. Many cooperative query answering techniques need to summarize the answer-related information, associated with data statistics or with certain aggregations [19, 112]. Interestingly, a higher-layered relation not only presents the generalized tuples but also the *counts* of the identical tuples or other computed aggregation values (such as sum, average, etc.). Such high-level information with counts conveys important information for data summarization and statistical data investigation.

4. Since the layer selection in the construction of an MLDB is based on the study of the frequently referenced attributes and frequently used query patterns, the MLDB itself embodies rich information about the history of the most regular query patterns and also implies the potential intent of the database users. It forms a rich source for query intent analysis and plays the role of confining the cooperative answers to frequently referenced patterns automatically.

5. An MLDB constructs a set of layers step-by-step, from most specific data to more general information. It facilitates progressive query refinement, from general information browsing to specific data retrieval. Such a process represents a top-down information searching process, which matches human's reasoning and learning process naturally, thus provides a cooperative process for step-by-step information exploration [109, 113].

Clearly, with these advantages, MLDB may become a valuable tool in cooperative query answering.

Since the cooperative query answering has been studied relatively thoroughly in previous research, instead of "reinventing" the technologies of cooperative query answering, we briefly present some examples to illustrate the use of MLDBs in the implementation of cooperative query answering mechanisms.

Example 7.5.3 A query like "*what kind of houses can be bought with $300k in the Vancouver area?*" can be answered using an MLDB efficiently and effectively. Here we examine several ways to answer this simple query using the MLDB constructed in Example 1.

1. Relaxation of query conditions using conceptual hierarchies and/or high layer relations:

   Instead of answering the query using "*house_price = $300k*", the condition can be relaxed to *about $300k*, that is, the price range covering $300k in a high layer relation, such as *house''*, can be used for query answering. This kind of relaxation can be done by mapping query constants up or down using conceptual hierarchies, and once the query is mapped to a level which fits a corresponding database layer, it can be processed within the layer.

2. Generalized answers with summarized statistics:

   Instead of printing thousands of houses within this price range, it searches through the top layer *house* relation, such as *house''*, and print the generalized answer, such as "20% 20-30 years-old, medium-sized, 3_bedrooms house in East Vancouver, ...". With the availability of MLDBs, such kind of generalized answers can be obtained directly from a high layered DB by summarization of the answers (such as giving percentage, general view, etc.) at a high layer.

3. Comparison with the neighborhood answers:

   Furthermore, the printed general answer can be compared with its neighborhood answers using the same top-level relation, such as "10% 3-bedroom 20-30 years-old houses in the Central Vancouver priced between $250k to $350K, while 30% such houses priced between $350 to $500k, ...". Notice that such comparison information can be presented as concise tables using an existing high layer relation.

4. Query answering with associative information:

It is often desirable to provide some "extra" information associated with a set of answers in cooperative query answering. Query answering with associative information can be easily achieved using high layer data relations. For example, the query can be answered by printing houses with different price ranges (such as $230-280k, $330-380k, etc.) as *row extension*, or printing houses in neighboring cities, printing other interesting features as *column extension*, or printing sales information related to such houses as *table extension*. These can be performed using high layer relations.

5. Progressively query refinement or progressive information focusing:

   The query can be answered by progressively stepping down the layers to find more detailed information. The top layer is often examined first, with general data and global views presented. Such a presentation often gives a user better idea of what should be searched further with additional constraints. For example, a user may focus the search to the East Vancouver area after s(he) finds a high percentage of the houses within this price range since it is likely to find a satisfiable house within this area. Such a further inquiry may lead the search to lower layer relations and may also promote users to pose more restricted constraints or refine the original ones. In this case, the route map associated with the MLDB will act as a tour guide to locate related lower layer relation(s).

7.6 Summary

We discuss the applications of data mining techniques in cooperative query answering in this chapter. A multiple layered database (MLDB) model has been proposed and examined. An MLDB can be constructed using data mining techniques. Data generalization and layer construction methods have been developed to guarantee new layers can be constructed efficiently, effectively and consistent with the primitive information stored in the database. Direct and cooperative query answering in such a MLDB are also examined. Our study shows that data mining techniques can be applied to facilitate cooperative query answering.

Chapter 8

Conclusions and Future Research

Our study on the discovery of multiple-level rules is concluded in this chapter. My major thesis work is summarized in Section 8.2. The conclusions drawn from our study are presented in Section 8.2. Finally, in Section 8.3, some future research problems are discussed.

8.1 Summary of My Thesis Work

In this thesis, we proposed the mining of multiple-level rules from large databases. A set of algorithms for the manipulation of conceptual hierarchies was proposed, as well as a set of algorithms for mining multiple-level rules, including characteristic rules, discriminant rules, and association rules. The algorithms were implemented in our data mining system, DBMiner. The experiments showed the algorithms performed well on large databases. The major contributions of my thesis work are summarized as follows.

1. The idea of mining multiple-level rules from large databases has been introduced. The mining of multiple-level rules was proposed and studied for several kinds of rules: characteristic, discriminant, and association rules. As discussed in Section 1.2, multiple-level rules can provide richer information than single-level rules, and may represent the hierarchical nature of the knowledge discovery

process. The mining of multiple-level rules extends previous KDD studies on the discovery of single-level rules.

2. The use of conceptual hierarchies in data mining has been examined. A set of algorithms for conceptual hierarchy manipulation has been developed. Two algorithms for conceptual hierarchy adjustment were proposed, one for adjustment using an attribute threshold and the other for adjustment without the use of an attribute threshold. In addition, an algorithm for the generation of conceptual hierarchies for numerical attributes was also proposed. All algorithms have been implemented and tested. The experiments on real databases showed very satisfactory results.

3. The mining of multiple-level characteristic, discriminant, and association rules has been investigated.

   An interactive progression method has been proposed for mining multiple-level characteristic and discriminant rules. An algorithm for the mining of multiple-level characteristic rules was presented as well as an algorithm for the mining of multiple-level discriminant rules. Both algorithms were implemented in the DBMiner system, and demonstrated the desired performance.

   A progressive deepening method for mining multiple-level association rules has been proposed. Several variants of the method, using different optimization techniques, were implemented and tested, and their performances were compared and analyzed. The experiments show that the method finds multiple-level association rules efficiently and effectively.

   A meta-rule guided approach for mining multiple-level association rules is proposed. Two algorithms, the large-predicate growing and the direct p-predicate testing, have been proposed and tested. The experiments show that meta-rule guided mining of multiple-level association rules is effective for discovery of interesting and strong such rules in large databases.

4. The application of data mining techniques in cooperative query answering has been studied. A multiple layered database (MLDB) model was proposed and

examined in Chapter 7. An MLDB can be constructed using data mining techniques. Direct and cooperative query answering in such an MLDB was studied.

5. A data mining prototype system, DBMiner, has been developed (several functional modules were developed by other researchers) which can find several kinds of knowledge from database.

8.2 Conclusions

Our study demonstrates that mining multiple level knowledge is both practical and desirable.

- The scope of data mining has been broadened by the study on the mining of multiple-level rules. The mining of multiple-level rules can provide more information for the users and enhance the flexibility and power of data mining systems. Therefore, the discovery of multiple-level rules represents an interesting research direction in data mining.

- The use of conceptual hierarchies facilitates the mining of multiple-level rules. Moreover, conceptual hierarchies can be adjusted dynamically to meet the need of the current data mining task. For numerical attributes, conceptual hierarchies can be generated automatically based on the current data distribution.

- Interesting multiple-level characteristic, discriminant, and association rules can be discovered efficiently from large databases.

  - The interactive progression algorithm finds mining multiple-level characteristic and discriminant rules flexibly and efficiently.
  - The progressive deepening method finds multiple-level association rules effectively and efficiently.
  - Meta-rule guided mining is a powerful tool for mining multiple-level association rules. Efficient algorithms have been developed for the meta-rule guided mining of multiple-level association rules, as we have demonstrated.

- Data mining techniques are very useful for cooperative query answering. Multiple layered databases can be built using data mining techniques in order to facilitate cooperative query answering.

- As a data mining prototype system, DBMiner successfully integrates data mining and database techniques, in order to find multiple-level knowledge from large databases. The DBMiner experience is a valuable example for the development of future data mining systems.

8.3 Future Research

Some interesting future research problems are presented as follows.

8.3.1 Discovery of Other Kinds of Multiple-Level Knowledge

The idea of mining multiple-level knowledge can be applied to discover other kinds of multiple-level rules or patterns, such as multiple-level sequential patterns, multiple-level deviation patterns, etc.

- Mining multiple-level sequential patterns.
  A sequential pattern is a series of items bought together by customers in a transactional database. For example, a sequential pattern, "TV followed by VCR followed by video camera", reveals that people buy TV, then VCR, and then video camera. An Apriori-like algorithm was proposed by Agrawal and Srikant [5] for the mining of frequently occurring sequential patterns.

  As for association rules, taxonomical or hierarchical organizations exist for many items. For example, a "26-inch Sony TV" is a "Sony TV" which is in turn a " TV". A multiple-level sequential pattern is a series of items, at primitive or nonprimitive level, occurring in transactions. For example, the search of multiple-level sequential patterns may find "Sony TV followed by Sony VCR

followed by Sony Video Camera", which gives more specific information than the above sequential pattern.

- Mining multiple-level deviation patterns.
  A deviation pattern describes the deviation of a variable from its expected value. For example, the following deviation pattern maybe discovered from a stock price database, "the price of company ABC's stock rose 30% more than the average in the last month".

  Conceptual hierarchies may exist for many attributes, for example, the companies can be grouped into several major sectors: service, finance, manufacture, etc. The mining of multiple-level deviation patterns may report the low-level deviation patterns as in the above example, or high-level deviation patterns, such as "the prices of the hi-tech stocks dropped 20% more than the average in the last quarter".

8.3.2 Meta-Rule Guided Mining of Other Kinds of Rules

Meta-rule guided mining is a powerful tool for specifying the interesting rules. It is interesting to investigate the meta-rule guided mining of other kinds of multiple-level rules, for example, prediction rules, sequential rules, etc.

- Meta-Rule guided mining of multiple-level prediction rules.
  A method for mining multiple-level prediction rules was proposed by Wang [110]. A multiple-level prediction rule can be used to predict an unknown attribute of an object based on its other attributes, which may be at non-primitive levels as well as the primitive level. For example, high-level prediction rules, such as "if a car is a subcompact Japanese car, its annual repair cost may range from five hundred to one thousand dollars, with 40% probability", may be discovered, together with low-level prediction rules, such as "if a car is a five-year-old Toyota, its annual repair cost may range from two hundred to five hundred dollars, with 60% probability". A meta-rule can specify the expected form of the prediction rules. For example, a meta-rule, $make(c : car, x) \wedge Q(c, y) \rightarrow repair\_cost(c, z)$,

can guide us to find the above rules, which predict a car's potential repair cost based on its make and any other attribute.

- Meta-rule guided mining of sequential rules.
  A sequential rule describes the associations of the sequential patterns in a transactional database. Sequential rules can be derived from sequential patterns. For example, a sequential rule, "50% of customers who buy TV also buy VCR within a year", can be derived from a sequential pattern, "TV followed by VCR".

  A meta-rule can specify the kind of sequential rules we are interested in. For example, a meta-rule, $P(c : customer, t_1) \rightarrow Q(c, t_2) \wedge t_2 \in [t_1, t_1 + 365]$, can help us to find the interesting rules which involve the sequential items bought by customers within a year. This may lead to the discovery of the above rule.

8.3.3 Automatic Generation of Conceptual Hierarchies for Nonnumerical Attributes

We studied the automatic generation of conceptual hierarchies for numerical attributes. It is natural to investigate the automatic generation of conceptual hierarchies for nonnumerical attributes. There are a number of studies on automatic hierarchical clustering, such as Cluster/2 [76], COBWEB [34, 35], and many others. However, they usually depend on a set of other attributes to define the search space, in which the best hierarchical clustering for the targeted attribute, based on some given criteria, is searched.

Some of the previous algorithms can be adapted to generate conceptual hierarchies for nonnumerical attributes. For example, a relevance analysis can determine the attributes that are related to the targeted attribute, and the objects can be clustered based on these relevant attributes. The conceptual hierarchy of the targeted attribute can then be extracted from the clusters. For example, the attribute *province* in Example 4.1.4 can be clustered based on the attributes of the *geo_location* of each province, and geographically nearby provinces will be organized into a region.

A more difficult problem is to find hierarchy for an attribute, with little or no

reference to other attributes, i.e., where the search space is not defined by other attributes.

8.3.4 Data Mining on the Internet (WWW)

The World Wide Web (WWW or Web) is rapidly growing and becoming increasingly popular. A great deal of information is available now on the Web, which provides huge amount of data for data mining.

Many resource discovery tools have been developed which can search for documents containing specific keywords [22, 23]. For example, a user can find documents about "data mining" through one of the search engines. However, not much research has been done on knowledge discovery on the Internet or Web. For example, it is difficult, if not impossible, to find all research groups that have published more than five papers on data mining since 1994.

A multiple layered, structured approach was proposed by Han et al. [52], in which multiple layered databases were constructed by the generalization of raw, primitive level information. For example, the description of an image, instead of the image itself, can be recorded in a high layer database.

However, some problems remain unsolved. For example, the extraction of basic information from complex data, such as multimedia data and structured data, requires sophisticated tools. Other problems include the heterogeneity of the data, the autonomy of the data, the maintenance of these multiple layered databases, and so on.

Bibliography

[1] R. Agrawal, A. Gupta, and S. Sarawagi. Modeling multidimensional databases. In *IBM Research Report*, 1996.

[2] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *Proc. 1993 ACM-SIGMOD Int. Conf. Management of Data*, pages 207–216, Washington, D.C., May 1993.

[3] R. Agrawal, M. Mehta, J. Shafer, R. Srikant, A. Arning, and T. Bollinger. The Quest data mining system. In *Proc. 1996 Int'l Conf. on Data Mining and Knowledge Discovery (KDD'96)*, Portland, Oregon, August 1996.

[4] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. 1994 Int. Conf. Very Large Data Bases*, pages 487–499, Santiago, Chile, September 1994.

[5] R. Agrawal and R. Srikant. Mining sequential patterns. In *Proc. 1995 Int. Conf. Data Engineering*, pages 3–14, Taipei, Taiwan, March 1995.

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

[7] A. Borgida and R. J. Brachman. Loading data into description reasoners. In *Proc. 1993 ACM-SIGMOD Int. Conf. Management of Data*, pages 217–226, Washington, D.C., May 1993.

[8] P. Bosc and O. Pivert. Some approaches for relational databases flexible querying. *Journal of Intelligent Information Systems*, 1:323–354, 1992.

[9] R. Brachman, P. Selfridge, L. Terveen, B. Altman, F. Halper, T. Kirk, A. Lazar, D. McGuinness, L. Resnick, and A. Borgida. Integrated support for data archaeology. *Inter. J. Intelligent and Cooperative Information Systems*, 2:159–185, June, 1993.

[10] R. J. Brachman and T. Anand. The process of knowledge discovery in databases : a first sketch. In *Proc. AAAI'94 Workshop on Knowledge Discovery in Databases (KDD'94)*, pages 1–12, Seattle, WA, July 1994.

[11] W. Buntine. Graphical models for discovering knowledge. In U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 59–82. AAAI/MIT Press, 1996.

[12] W. L. Buntine. Theory refinement on bayesian networks. In *Proc. 7th Conference on Uncertainty in Artificial Intelligence*, pages 52–60, San Francisco, CA, 1991.

[13] Y. Cai, N. Cercone, and J. Han. Attribute-oriented induction in relational databases. In G. Piatetsky-Shapiro and W. J. Frawley, editors, *Knowledge Discovery in Databases*, pages 213–228. AAAI/MIT Press, 1991.

[14] P. Cheeseman, J. Kelly, M. Self, J. Stutz, W. Taylor, and D. Freeman. Autoclass: a bayesian classification system. In *Proc. Fifth Int. Conf. on Machine Learning*, pages 54–64, San Mateo, California, 1988.

[15] P. Cheeseman and J. Stutz. Bayesian classification (AutoClass): Theory and results. In U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 153–180. AAAI/MIT Press, 1996.

[16] P. Cheeseman and J. Stutz. Bayesian classification (AUTOCLASS): Theory and results. In U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*. AAAI/MIT Press, 1996.

[17] J. Cheng, U. M. Fayyad, K. B. Irani, and Z. Qian. Improved decision trees: a generalized version of id3. In *Proc. Fifth Int. Conf. on Machine Learning*, pages 100–107, San Mateo, California, 1988.

[18] D. K. Y. Chiu, A. K. C. Wong, and B. Cheung. Information discovery through hierarchical maximum entropy discretization and synthesis. In G. Piatetsky-Shapiro and W. J. Frawley, editors, *Knowledge Discovery in Databases*, pages 125–141. AAAI/MIT Press, 1991.

[19] W. W. Chu and Q. Chen. Neighborhood and associative query answering. *Journal of Intelligent Information Systems*, 1:355–382, 1992.

[20] W. W. Chu, Q. Chen, and R. Lee. Cooperative query answering via type abstraction hierarchy. In S.M Dee, editor, *Cooperating Knowledge Based System*, pages 271–292. Now York: Elsevier, 1990.

[21] E. F. Codd, S. B. Codd, and C. T. Salley. *Providing OLAP (On-Line Analytical Processing) to User-Analysis: An IT Mandate.* http://www.arborsoft.com/papers/coddTOC.html, E. F. Codd Associates, 1993.

[22] Infoseek Corporation. *Infoseek.* http://guide.infoseek.com/, 1995-96.

[23] Yohoo! Corporation. *Yahoo.* http://www.yahoo.com/, 1994-96.

[24] F. Cuppens and R. Demolombe. Cooperative answering: a methodology to provide intelligent access to databases. In *Proc. 2nd Int. Conf. Expert Database Systems*, pages 621–642, 1989.

[25] F. Cuppens and R. Demolombe. Extending answers to neighbor entities in a cooperative answering context. *Decision Support Systems*, 7:1–11, 1991.

[26] S. Dao and B. Perry. Applying a data miner to heterogeneous schema integration. In *Proc. First Int. Conf. on Knowledge Discovery and Data Mining*, pages 63–68, Montreal, Canada, Aug. 1995.

[27] G. Dunn and B. Everitt. *An Intorduction to Mathematical Taxonomy.* Cambridge Press, 1982.

[28] T. Duong and J. Hiller. Modelling the real world by multi-world data model. In *Proc. 1st Int. Conf. Cooperative Information Systems*, pages 279–290, 1993.

[29] Jr. F. S. Hill, S. Walker, and F. Gao. Interactive image query system using progressive transmission. *Computer Graphics*, 17, July 1983.

[30] A. Fall. Reasoning with taxonomies. In *Ph.D. Dissertation, Simon Fraser University*, Burnaby, B.C., Canada, 1996.

[31] A. Fall. Sparse logical terms. *Appl. Math. Lett.*, 8:11–15, 1996.

[32] U. M. Fayyad, S. G. Djorgovski, and N. Weir. Automating the analysis and cataloging of sky surveys. In U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 471–493. AAAI/MIT Press, 1996.

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

[34] D. Fisher. Improving inference through conceptual clustering. In *Proc. 1987 AAAI Conf.*, pages 461–465, Seattle, Washington, July 1987.

[35] D. Fisher, M. Pazzani, and P. Langley. *Concept Formation: Knowledge and Experience in Unsupervised Learning.* Morgan Kaufmann, 1991.

[36] W. J. Frawley, G. Piatetsky-Shapiro, and C. J. Matheus. Knowledge discovery in databases: An overview. In G. Piatetsky-Shapiro and W. J. Frawley, editors, *Knowledge Discovery in Databases*, pages 1–27. AAAI/MIT Press, 1991.

[37] Y. Fu and J. Han. Meta-rule-guided mining of association rules in relational databases. In *Proc. 1st Int'l Workshop on Integration of Knowledge Discovery with Deductive and Object-Oriented Databases (KDOOD'95)*, pages 39–46, Singapore, Dec. 1995.

[38] T. Gaasterland. Restricting query relaxation through user constraints. In *Proc. 1st Int. Conf. Cooperative Information Systems*, pages 359–366, 1993.

[39] T. Gaasterland, P. Godgrey, and J. Minker. Relaxation as a platform for cooperative answering. *Journal of Intelligent Information Systems*, 1:293–321, 1992.

[40] B. R. Gains. Exception dags as knowledge structures. In *Proc. AAAI'94 Workshop on Knowledge Discovery in Databases (KDD'94)*, pages 13–24, Seattle, WA, July 1994.

[41] M. Genesereth and N. Nilsson. *Logical Foundations of Artificial Intelligence.* Morgan Kaufmann, 1987.

[42] DBMiner group. *DBMiner User's Manual.* Database Systems Lab, Simon Fraser University, 1996.

[43] A. Gupta, V. Harinarayan, and D. Quass. Aggregate-query processing in data warehousing environment. In *Proc. 21st Int. Conf. Very Large Data Bases*, pages 358–369, Zurich, Switzerland, Sept. 1995.

[44] J. Han, Y. Cai, and N. Cercone. Concept-based data classification in relational databases. In *1991 AAAI Workshop on Knowledge Discovery in Databases*, pages 77–94, Anaheim, CA, July 1991.

[45] J. Han, Y. Cai, and N. Cercone. Knowledge discovery in databases: An attribute-oriented approach. In *Proc. 18th Int. Conf. Very Large Data Bases*, pages 547–559, Vancouver, Canada, August 1992.

[46] J. Han, Y. Cai, and N. Cercone. Data-driven discovery of quantitative rules in relational databases. *IEEE Trans. Knowledge and Data Engineering*, 5:29–40, 1993.

[47] J. Han and Y. Fu. Discovery of multiple-level association rules from large databases. In *Proc. 1995 Int. Conf. Very Large Data Bases*, pages 420–431, Zurich, Switzerland, Sept. 1995.

[48] J. Han and Y. Fu. Exploration of the power of attribute-oriented induction in data mining. In U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 399–421. AAAI/MIT Press, 1996.

[49] J. Han, Y. Fu, and R. Ng. Cooperative query answering using multiple-layered databases. In *Proc. 2nd Int. Conf. Cooperative Information Systems*, pages 47–58, Toronto, Canada, May 1994.

[50] J. Han, Y. Fu, W. Wang, J. Chiang, W. Gong, K. Koperski, D. Li, Y. Lu, A. Rajan, N. Stefanovic, B. Xia, and O. R. Zaiane. DBMiner: A system for mining knowledge in large relational databases. In *Proc. 1996 Int'l Conf. on Data Mining and Knowledge Discovery (KDD'96)*, Portland, Oregon, August 1996.

[51] J. Han, Y. Fu, W. Wang, K. Koperski, and O. R. Zaiane. DMQL: A data mining query language for relational databases. In *Proc. 1996 SIGMOD'96 Workshop on Research Issues on Data Mining and Knowledge Discovery (DMKD'96)*, Montreal, Canada, June 1996.

[52] J. Han, O. R. Zaïane, and Y. Fu. Resource and knowledge discovery in global information systems: A scalable multiple layered database approach. In *Proc. of a Forum on Research and Technology Advances in Digital Libraries (ADL'95)*, McLean, Virginia, May 1995.

[53] V. Harinarayan, A. Rajaraman, and J. D. Ullman. Implementing data cubes efficiently. In *Proc. 1996 ACM-SIGMOD Int. Conf. Management of Data*, Montreal, Canada, June 1996.

[54] D. Heckerman. Bayesian networks for knowledge discovery. In U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 273–306. AAAI/MIT Press, 1996.

[55] D. Heckerman, D. Geiger, and D. M. Chickering. Learning bayesian networks: the combination of knowledge and statistical data. In *Proc. AAAI'94 Workshop on Knowledge Discovery in Databases (KDD'94)*, pages 85–96, Seattle, WA, July 1994.

[56] J. Hong and C. Mao. Incremental discovery of rules and structure by hierarchical and parallel clustering. In G. Piatetsky-Shapiro and W. J. Frawley, editors, *Knowledge Discovery in Databases*, pages 177–193. AAAI/MIT Press, 1991.

[57] X. Hu and N. Cercone. Rough set similarity-based learning from databases. In *Proc. First Int. Conf. on Knowledge Discovery and Data Mining*, pages 162–167, Montreal, Canada, Aug. 1995.

[58] T. Imielinski. Intelligent query answering in rule based systems. *J. Logic Programming*, 4:229–257, 1987.

[59] T. Imielinski and A. Virmani. DataMine – interactive rule discovery system. In *Proc. 1995 ACM-SIGMOD Int. Conf. Management of Data*, page 472, San Jose, CA, May 1995.

[60] MicroStrategy
Inc. *The Case for Relational OLAP.* http://www.strategy.com/dwf/wp_b_a1.htm, 1995.

[61] G. H. John. Robust decision trees: Removing outliers from databases. In *Proc. First Int. Conf. on Knowledge Discovery and Data Mining*, pages 174–179, Montreal, Canada, Aug. 1995.

[62] M. Kamber and R. Shinghal. Evaluating the interestingness of characteristic rules. In *Proc. 1996 Int'l Conf. on Data Mining and Knowledge Discovery (KDD'96)*, Portland, Oregon, August 1996.

[63] K. A. Kaufman, R. S. Michalski, and L. Kerschberg. Mining for knowledge in databases: Goals and general description of the INLEN system. In G. Piatetsky-Shapiro and W. J. Frawley, editors, *Knowledge Discovery in Databases*, pages 449–462. AAAI/MIT Press, 1991.

[64] D. Keim, H. Kriegel, and T. Seidl. Supporting data mining of large databases by visual feedback queries. In *Proc. 10th of Int. Conf. on Data Engineering*, pages 302–313, Houston, TX, Feb. 1994.

[65] M. Klemettinen, H. Mannila, P. Ronkainen, H. Toivonen, and A. I. Verkamo. Finding interesting rules from large sets of discovered association rules. In *Proc. 3rd Int'l Conf. on Information and Knowledge Management*, pages 401–408, Gaithersburg, Maryland, Nov. 1994.

[66] W. Klösgen. Explora: a multipattern and multistrategy discovery assistant. In U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 249–271. AAAI/MIT Press, 1996.

[67] H. F. Korth and A. Silberschatz. *Database System Concepts, 2ed.* McGraw-Hill, 1991.

[68] H. Leslie, R. Jain, D. Birdsall, and H. Yaghmai. Efficient search of multidimensional b-tree. In *Proc. 21st Int. Conf. Very Large Data Bases*, pages 710–719, Zurich, Switzerland, Sept. 1995.

[69] H. Lu, R. Setiono, and H. Liu. Neurorule: A connectionist approach to data mining. In *Proc. 21st Int. Conf. Very Large Data Bases*, pages 478–489, Zurich, Switzerland, Sept. 1995.

[70] M. Manago and Y. Kodratoff. Induction of decision trees from complex structured data. In G. Piatetsky-Shapiro and W. J. Frawley, editors, *Knowledge Discovery in Databases*, pages 289–306. AAAI/MIT Press, 1991.

[71] H. Mannila and K-J. Raiha. Dependency inference. In *Proc. 1987 Int. Conf. Very Large Data Bases*, pages 155–158, Brighton, England, Sept. 1987.

[72] C. Matheus, P. K. Chan, and G. Piatetsky-Shapiro. Systems for knowledge discovery in databases. *IEEE Trans. Knowledge and Data Engineering*, 5:903–913, 1993.

[73] C. J. Matheus and G. Piatesky-Shapiro. An application of kefir to the analysis of healthcare information. In *Proc. AAAI'94 Workshop on Knowledge Discovery in Databases (KDD'94)*, pages 441–452, Seattle, WA, July 1994.

[74] C.J. Matheus, G. Piatetsky-Shapiro, and D. McNeil. Selecting and reporting what is interesting: The KEFIR application to healthcare data. In U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 495–516. AAAI/MIT Press, 1996.

[75] R. S. Michalski. A theory and methodology of inductive learning. In Michalski et al., editor, *Machine Learning: An Artificial Intelligence Approach, Vol. 1*, pages 83–134. Morgan Kaufmann, 1983.

[76] R. S. Michalski and R. Stepp. Automated construction of classifications: Conceptual clustering versus numerical taxonomy. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 5:396–410, 1983.

[77] R. S. Michalski and G. Tecuci. *Machine Learning, A Multistrategy Approach, Vol. 4*. Morgan Kaufmann, 1994.

[78] R. Missaoui and R. Godin. An incremental concept formation approach for learning from databases. In V.S. Alagar, L.V.S. Lakshmanan, and F. Sadri, editors, *Formal Methods in Databases and Software Engineering*, pages 39–53. Springer-Verlag, 1993.

[79] T. M. Mitchell. Version spaces: A candidate elimination approach to rule learning. In *Proc. 5th Int. Joint Conf. Artificial Intelligence*, pages 305–310, Cambridge, MA, 1977.

[80] T. M. Mitchell. An analysis of generalization as a search problem. In *Proc. 6th Int. Joint Conf. Artificial Intelligence*, pages 577–582, Tokyo, Japan, 1979.

[81] R. Ng. Spatial data mining: Discovering knowledge of clusters from maps. In *Proc. 1996 SIGMOD'96 Workshop on Research Issues on Data Mining and Knowledge Discovery (DMKD'96)*, Montreal, Canada, June 1996.

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

[83] Z. Pawlak. Rough sets. *Inter. J. of Computer and Information Sciences*, 11:341–356, 1995.

[84] G. Piatesky-Shapiro and C. J. Matheus. The interestingness of deviations. In *Proc. AAAI'94 Workshop on Knowledge Discovery in Databases (KDD'94)*, pages 25–36, Seattle, WA, July 1994.

[85] G. Piatetsky-Shapiro. Discovery, analysis, and presentation of strong rules. In G. Piatetsky-Shapiro and W. J. Frawley, editors, *Knowledge Discovery in Databases*, pages 229–238. AAAI/MIT Press, 1991.

[86] G. Piatetsky-Shapiro, U. Fayyad, and P. Smith. From data mining to knowledge discovery: An overview. In U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 1–35. AAAI/MIT Press, 1996.

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

[88] J. R. Quinlan. Discovering rules by induction from large collections of examples. In D. Michie, editor, *Expert Systems in the Micro Electronic Age.* Edinburgh, England, 1979.

[89] J. R. Quinlan. Learning efficient classification procedures and their application to chess end-games. In Michalski et al., editor, *Machine Learning: An Artificial Intelligence Approach, Vol. 1*, pages 463–482. Morgan Kaufmann, 1983.

[90] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

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

[92] R.L. Read, D.S. Fussell, and A. Silberschatz. A multi-resolution relational data model. In *Proc. 18th Int. Conf. Very Large Data Bases*, pages 139–150, Vancouver, Canada, Aug. 1992.

[93] K. A. Ross, D. Srivastava, and S. Sudarshan. Materialized view maintenance and integrity constraint checking: Trading space for time. In *Proc. 1996 ACM-SIGMOD Int. Conf. Management of Data*, pages 447–458, Montreal, Canada, May 1996.

[94] W. Shen, K. Ong, B. Mitbander, and C. Zaniolo. Metaqueries for data mining. In U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 375–398. AAAI/MIT Press, 1996.

[95] C. Shum and R. Muntz. An information-theoretic study on aggregate responses. In *Proc. 14th Int. Conf. Very Large Data Bases*, Los Angeles, USA, August, 1988.

[96] A. Silberschatz, M. Stonebraker, and J. D. Ullman. Database systems: Achievements and opportunities. *Comm. ACM*, 34:94–109, 1991.

[97] A. Silberschatz, M. Stonebraker, and J. D. Ullman. Database research: Achievements and opportunities into the 21st century. *SIGMOD Record*, 25:52–63, March 1996.

[98] A. Silberschatz and A. Tuzhilin. On subjective measure of interestingness in knowledge discovery. In *Proc. 1st Int. Conf. on Knowledge Discovery and Data Mining (KDD'95)*, pages 275–281, Montreal, Canada, Aug. 1995.

[99] E. Simoudis, B. Livezey, and R. Kerber. Using Recon for data cleaning. In *Proc. First Int. Conf. on Knowledge Discovery and Data Mining*, pages 258–262, Montreal, Canada, Aug. 1995.

[100] E. Simoudis, B. Livezey, and R. Kerber. Integrating inductive and deductive reasoning. In U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 353–376. AAAI/MIT Press, 1996.

[101] J.M. Smith and D.C.P. Smith. Database abstractions: Aggregation and generalization. *ACM Trans. Database Syst.*, 2:105–133, June 1977.

[102] P. Smyth and R.M. Goodman. Rule induction using information theory. In G. Piatetsky-Shapiro and W. J. Frawley, editors, *Knowledge Discovery in Databases*, pages 159–176. AAAI/MIT Press, 1991.

[103] P. Spirtes, C. Glymour, and R. Scheines. *Causation, Prediction and Search.* Springer-Verlag, 1993.

[104] R. Srikant and R. Agrawal. Mining generalized association rules. In *Proc. 1995 Int. Conf. Very Large Data Bases*, pages 407–419, Zurich, Switzerland, Sept. 1995.

[105] R. Srikant and R. Agrawal. Mining quantitative association rules in large relational tables. In *Proc. 1996 ACM-SIGMOD Int. Conf. Management of Data*, Montreal, Canada, June 1996.

[106] T. J. Teorey, D. Yang, and J. P. Fry. A logical design methodology for relational databases using the extended entity-relationship model. *ACM Comput. Surv.*, 18:197–222, 1986.

[107] P. E. Utgoff. An incremental id3. In *Proc. Fifth Int. Conf. on Machine Learning*, pages 107–120, San Mateo, California, 1988.

[108] R. Uthurusamy, U. M. Fayyad, and S. Spnggler. Learning useful rules from inconclusive data. In G. Piatetsky-Shapiro and W. J. Frawley, editors, *Knowledge Discovery in Databases*, pages 141–158. AAAI/MIT Press, 1991.

[109] S.V. Vrbsky and J. W. S. Liu. An object-oriented query processor that returns monotonically improving answers. In *Proc. 7th IEEE Conf. on Data Engineering*, pages 472–481, Kobe, Japan, April 1991.

[110] W. Wang. *Predictive Modeling for Knowledge Discovery in Databases.* Master's Thesis, Simon Fraser University, 1996.

[111] R. Wille. Conceptual lattices and conceptual knowledge systems. *Computers Math. Applic.*, 23:493–515, 1992.

[112] C. Wittemann and H. Kunst. Intelligent assistance in flexible decisions. In *Proc. 1st Int. Conf. Cooperative Information Systems*, pages 377–381, 1993.

[113] M.F. Wolf. Successful integration of databases, knowledge-based systems, and human judgement. In *Proc. 1st Int. Conf. Cooperative Information Systems*, pages 154–162, 1993.

[114] Y. Zhuge, H. Garcia-Molina, J. Hammer, and J. Widom. View maintenance in a warehousing environment. In *Proc. 1995 ACM-SIGMOD Int. Conf. Management of Data*, pages 316–327, San Jose, CA, May 1995.

[115] W. Ziarko. The discovery, analysis, and representation of data dependancies in databases. In G. Piatetsky-Shapiro and W. J. Frawley, editors, *Knowledge Discovery in Databases*, pages 195–209. AAAI/MIT Press, 1991.

[116] J. Zytkow and J. Baker. Interactive mining of regularities in databases. In G. Piatetsky-Shapiro and W. J. Frawley, editors, *Knowledge Discovery in Databases*, pages 31–54. AAAI/MIT Press, 1991.