On-Line Analytical Mining of Association Rules

by

Hua Zhu
B.S., University of Science and Technology of China, 1995

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

SIMON FRASER UNIVERSITY
December 1998

APPROVAL

Name: Hua Zhu

Degree: Master of Science

Title of thesis: On-Line Analytical Mining of Association Rules

Examining Committee: Dr. James Delgrande
Chair

Dr. Jiawei Han
Senior Supervisor

Dr. Qiang Yang
Supervisor

Dr. William Havens
External Examiner

Date Approved:

Abstract

With wide applications of computers and automated data collection tools, massive amounts of data have been continuously collected and stored in databases, which creates an imminent need and great opportunities for mining interesting knowledge from data. Association rule mining is one kind of data mining techniques which discovers strong association or correlation relationships among data. The discovered rules may help market basket or cross-sales analysis, decision making, and business management.

In this thesis, we propose and develop an interesting association rule mining approach, called on-line analytical mining of association rules, which integrates the recently developed OLAP (on-line analytical processing) technology with some efficient association mining methods. It leads to flexible, multi-dimensional, multi-level association rule mining with high performance. Several algorithms are developed based on this approach for mining various kinds of associations in multi-dimensional databases, including intra-dimensional association, inter-dimensional association, hybrid association, and constraints-based association. These algorithms have been implemented in the DBMiner system. Our study shows that this approach presents great advantages over many existing algorithms in terms of both flexibility and efficiency.

Keywords: OLAP (on-line analytical processing), data mining, association rule mining, data warehouse.

Acknowledgement

I would like to thank my senior supervisor, Dr. Jiawei Han, for inspiring me in data warehousing and date mining, which are very interesting and exciting. My gratitude goes to him for his invaluable support and guidance through my research, development, and the writing of my thesis. I also thank my supervisor, Dr. Qiang Yang, and external examiner, Dr. William S. Havens, for reading my thesis and giving good suggestions.

I would like to take this opportunity to express my thanks toward everyone in the Intelligent Database Laboratory of the School of Computing Science at Simon Fraser University for their invaluable suggestions and help in these two years of study. Thanks also go to all the other friends I have made in Simon Fraser University for making me have an enjoyable peroid of time.

Finally, I love to thank my family, especially my parents ShongZi Zhu and Xiaoqiao Lei, my sister Rong Zhu and my brother Lin Zhu for their continuous love and encouraging. Gratitude also go to my beloved husband, Wenping Luo, for his supporting, understanding and love. Without them, my thesis could not have been written.

Dedication

To my parents.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

With the wide applications of computers and automated data collection tools, massive amounts of data have been continuously collected in databases, which creates great demands for analyzing such data and turning them into useful knowledge [13]. Therefore, Knowledge Discovery and Data Mining (KDD) has become an important field in recent years to address the need for analyzing data contained in very large database [13].

Among discovering many kinds of knowledge in large databases, *association rule mining* has attracted great attention in database research communities in recent years [2, 4, 25, 36, 33, 20, 39, 40, 42, 30, 41, 31, 8]. Association rule mining is a form of data mining to discover interesting relationships among attributes in those data. The discovered rules may help marketing, decision making, and business management. An example of such a rule might be that *98% of customers that purchase tires and auto accessories also get automotive services done*. Since rules are simple, easy to understand, explain, and catch some important relationships among the data in large databases, there is no wonder that mining association rules from large data sets has been a focused topic in recent research into data mining.

Similar to other mining tasks, mining association rules involves several major issues, including efficiency, scalability, usability, and understandability. In real world applications, data mining tasks are applied to data consisting of millions of tuples, if not more. Consequently, our first concern is the efficiency and scalability at mining associations in large databases in order to substantially reduce the computational complexity of the data intensive process. Thus an essential research issue surrounding association rule mining is to find fast and effective association rule mining algorithms.

In this thesis, we propose and develop an interesting method, called *on-line analytical mining of association rules*, which integrates the recently developed OLAP technology with efficient association mining methods. The developed method achieves flexible, multi-dimensional, multi-level association rule mining with high performance. The developed method presents great advantages over many existing algorithms in terms of both flexibility and efficiency.

1.1 On-line analytical processing

Due to the increasing importance of data warehousing, many techniques have been developed to analyze data in large databases efficiently and effectively. Among many available tools, *On-Line Analytical Processing (OLAP)* technique has been proven to be one of the most popular tools for on-line, fast, and effective multidimensional data analysis [10].

OLAP is a technology that uses a *multidimensional* view of aggregated data to provide quick access to strategic information for further analysis. It facilitates queries on large amounts of data much more quickly than traditional relational database techniques [15]. Using OLAP techniques, raw data from large databases is organized into multiple dimensions, and each dimension contains multiple levels of abstraction. This organization provides users with the flexibility to view data from different perspectives.

Data generalization can be achieved through approaches such as *data cube* and *attribute-oriented induction.* Basically, OLAP includes three stages: (1) selecting data from data warehouses, (2) building the data cube, and (3) on-line analysis using the cube. Morever, many data cube computation algorithms exist to materialize data cubes efficiently [10, 43, 1], and many OLAP algorithm exist to manipulate data cubes, such as roll-up, drill-down, slice and dice, pivot, etc. In this thesis, we will take advantage of these widely used techniques and combine them with association rule mining techniques for *OLAP-based association rule mining.*

1.2 Association rule discovery

1.2.1 What is association rule

As we mentioned above, mining association rules is to find interesting association or correlation relationships among a large set of data, i.e., to identify sets of *attribute-values* (predicate or item) that frequently occur together, and then formulate rules that characterize these relationships. A formal definition is given below.

Definition 1.1 An **association rule** is a rule in the form of

$$A_1, A_2, \ldots, A_m \;\rightarrow\; B_1, B_2, \ldots, B_n \tag{1.1}$$

where A_i and B_j are predicates or items. Such rules are usually interpreted as "*When items A_1, A_2, ..., A_m occur, it is often the case that items B_1, B_2, ..., B_n occur as well in the same transaction*". What exactly constitutes an item or a transaction depends on the application.

Here we give some association rules with different concepts of item and transaction.

Example 1.2.1 The following is a rule from transaction database of an AllElectronics branch.

$$Product(X, \text{"}IBM\ home\ computer\text{"}) \;\rightarrow\; Product(X, \text{"}Sony\ b/w\ printer\text{"}) \tag{1.2}$$

where X is a variable representing a transaction.

In this case, the items are the things that the customer buys, and a transaction is the set of all items purchased together. The rule says that the customers who purchase IBM home computers also tend to buy Sony b/w printers at the same time. □

Example 1.2.2 The following rule from a data warehouse with three dimensions: Age, Occupation, and Product.

$$Age(X, \text{“}30-39\text{”}), Occupation(X, \text{“}student\text{”}) \rightarrow Product(X, \text{“}laptop\text{”}) \tag{1.3}$$

The items here are the distinct attribute values from three dimensions, and a transaction is one tuple in the data warehouse. The rule says that the customs who are students between the age of 30 and 39 will most possibly buy laptops.

1.2.2 Area of application

Although association rules mining originated with the problem of market basket analysis, it can also be applied to many other problem domains including business, engineering, medicine, and finance [13]. Moreover, as one of the basis tasks of data mining, association rules have been used to achieve the goal of other data mining tasks, such as modeling of data, prediction of future outcomes, and support of decision.

- Market basket analysis.

  Understanding customers' buying habits and preferences is essential for retailers to make decisions including what to put on sale, how to design coupons, how to place merchandise on shelves in order to maximize the profit, etc. Association rules mining can provide such information. An effective mining application in the retail environment is market basket analysis or shopping basket analysis. It analyzes the attributes of customers' shopping basket from Electronic Point of Sale data and applies the findings to launch effective promotions and advertising. For example, all rules that have “Diet Coke” as consequent may help plan what the store should do to boost the sales of Diet Coke.

- Cross-sale

  In the present strong competitive environment within the service industries, retaining customers and making better use of the customer base have become as important as, if not more important than, attracting new customers. Most companies are involved in providing more than one service or product. Since they know more about their existing customers through data collected about them, targeting those customers with products that they do not already have is seen as a way of reaping quick profits. **cross-sales** is the term given to this problem of attempting to sell a product to existing customers of the company who are not already customers of that particular product. However, customer databases are very large in large organizations and manually sifting through it is difficult. Thus automatic association rules discovery techniques can incorporate domain expertise and discover useful knowledge for the domain expert to help solve cross-sales problems.

- Partial classification

  Many real-life problems require a partial classification of the data, i.e., describe the discovery of models that show characteristics of the data classes, but may not cover all classes and all examples of any given class. Conventional classifiers would be ineffective when there are a very large number of attributes, and most of the values of each attribute are missing. This problem can be solved using association analysis. One of the examples is the data consisted of medical tests given to patients. As usual, there are hundreds of medical tests but only a few of them are given to any single patient. doctors can use rules and models discovered by association analysis to see whether any of the medical test results could be predicted by combinations of other test results. If such tests are found, they can be used to avoid giving patients redundant tests, or complex tests could be replaced with simple tests.

- Financial services

  Association rules mining is used extensively by financial service industries. Security analysts are using this to analyze massive financial data in order to build trading and risk models for developing investment strategies. A number of companies in the financial sector have been testing this technique that have produced positive results. These products may be utilized in currency trading, data tidying, claims processing, automated underwriting, stock selection, credit scoring, identifying fraud patterns, and mortgage screening in the near future.

1.3 Motivation

Although association rule technology has reached a level of maturity where association rule techniques are included in commercial mining products, this area also remains an area of active research. Some serious limitation of current association rules and several related issues motivate continued studies into knowledge based on-line association analysis.

1.3.1 Problem 1 - lack of a general association rules mining algorithm

Association rules can be classified into several categories according to different criteria. For instance,

- Based on the dimensions of data involved in the rule, there are two cases: single-dimensional and multi-dimensional rules.
- Based on the levels of abstractions involved in the rule, there are single-level and multi-level rules.

Although various algorithms are proposed for different cases, there is no such a general one which is applicable for all situations. Especially for multi-dimensional rules mining, all current research is confined into a simple case: multi-dimensional association with no repetitive predicates, such as

$$City(X, \text{``}Vancouver\text{''}), Cost(X, \text{``}0-100\text{''}) \rightarrow Product(X, \text{``}SportBag\text{''}) \quad (1.4)$$

However, it is very natural to ask if we can discover multi-dimensional association rules with repetitive predicates, this is a case which hasn't attracted much attention yet. For instance,

$$City(X, \text{``}Vancouver\text{''}), Product(X, \text{``}Tents\text{''}) \rightarrow Product(X, \text{``}SportBag\text{''}) \quad (1.5)$$

[Our contribution]: In this thesis, we propose a more general algorithm based on the current algorithms. This algorithm is a combination of the Aprior algorithm used for single-dimensional rules mining and the cube-based algorithm used for multi-dimensional rules mining. Experiments show that this algorithm resolves the limitation we discussed above very efficiently without losing any scalability and speed when applied to those previous cases.

1.3.2 Problem 2 - users may be overwhelmed by the number of rules identified

Support and confidence thresholds can be used to limit the number of rules generated, but there is always the danger that rules of genuine interest to the user may be lost if the thresholds are set too low. This is the motivation for allowing the user to specify the desired form and partial contents of the rules returned, i.e., constraint-based mining [23]. More specifically, the users are allowed to provide hypotheses in the form of constraints, or pattern templates. The system attempts to confirm these hypotheses by searching for rules which satisfy the given constraints.

Example 1.3.1 An example of such a constraint is

$$The\ item\ Product(\text{“}Tents\text{”})\ must\ be\ in\ the\ head\ of\ the\ rule \tag{1.6}$$

Then, the following is a rule satisfies the above constraint,

$$Cost(X, \text{“}100-200\text{”}), Location(X, \text{“}Belgium\text{”}) \rightarrow Product(X, \text{“}Tents\text{”}) \tag{1.7}$$

□

Currently, the method for constraint-based mining is to add one more step, rule filtering, after mining. Obviously, it is far from efficient enough, especially when applied to a very large database.

[Our contribution]: In this thesis, we propose a cube-based approach to slove this problem. We try to push more constraints into the procedure of association rules mining, and thus make it more efficient by reducing the rule search space.

1.3.3 Problem 3 - strong association rules are not necessarily interesting

In some cases, association analysis itself is not sufficient to find interesting rules. For example,

Example 1.3.2 Consider the Sales database, it shows that 60% of those transactions are made in Vancouver, 75% include "Sleeping Bag", and 40% involve both "Vancouver"and "Sleeping Bag". Suppose we use a minimum support of 40% and a minimum confidence of 60%. Then the following association rule is discovered.

$$\begin{aligned} &Location(X, \text{"}Vancouver\text{"}) \rightarrow \\ &\quad Product(X, \text{"}SleepingBag\text{"})[support = 40\%, confidence = 66\%] \end{aligned} \tag{1.8}$$

However, this rule is misleading since the overall percentage of transactions including "Sleeping Bag"is 75%, even larger than 66%. In fact, the location of "Vancouver"and the product of "Sleeping Bag"are negatively associated because being involved in the former actually decreases the likelihood of being involved the latter. Without fully understanding this phenomenon, one could make unwise business decisions based on this rule.

[**Our contribution**]: To help filter out such misleading "strong" associations of the form $A \rightarrow B$, it is necessary to perform correlation analysis between A and B. In this thesis, we propose one method to get *strong and interesting* Rules by calculating the interestingness between A and B.

1.4 Thesis organization

The remaining of the thesis is organized as follows. Chapter 2 outlines some existing work related to the thesis. A general description of association mining system is given in Chapter 3. Based on the discussion above, we will address three problems. The first is more general multidimensional association rules mining, second is correlation analysis, last is on constraints-based mining. The approaches and algorithms towards solving these problems and experimental results will be presented in Chapters 4, 5. A brief discussion on system implementation is in Chapter 6. Some conclusions and future work will be in Chapter 7.

Chapter 2

Related Work

The basic concept on association rules is introduced in Chapter 1. In this chapter, we will review current research on OLAP technology and association rules mining. Our overview of related topics focuses on two major themes: (1) efficient *OLAP operations* in large data warehouses, and (2) fast and effective association mining *algorithms*.

2.1 Data warehousing and OLAP technology

2.1.1 Data warehousing and data cube

Data warehouse is a subject-oriented, integrated, time-variant, and non-volatile collection of data for decision support applications [24]. The construction of data warehouse, with data cleaning and data integration, can be viewed as an important preprocessing step for knowledge discovery tasks.

The proposal of the construction of large data warehouse for multi-dimensional analysis is credited to Codd who coined the term OLAP for on-line analytical processing [12]. Portions of data warehouses can be precomputed and materialized for efficient processing and such a materialized multidimensional database is popularly called **data cube** [15]. From the data structure point of view, data cube can be viewed as a large multi-dimensional array which consists of a set of *dimensions* with respect to the analyzed data, and a set of values in each cell called *measures* [10].

From the operational point of view, a data cube is referred to as a relational operator which computes group-by aggregations over all possible subsets of the specified dimensions [15]. It treats each of the n aggregated attributes as an n-dimensional subcube, or cuboids. The aggregation of a particular set of attribute values is a point in this space. The rapid acceptance of this operator has led to a variant of the CUBE being proposed for the SQL standard.

Data embedded in a data cube can be a primitive concept level, i.e., the raw data from the database. If knowledge is extracted from and expressed using this *base cube*, it is often not meaningful and useful enough. Therefore, it is usually more desirable to abstract the data from a low concept level to a reasonable higher one. This important functionality is called **data generalization**.

2.1.2 On-line analytical processing

OLAP (on-line Analytical processing) is a set of operations provided by data warehouses to manipulate data at multiple dimensions and multiple levels of abstraction. The basic OLAP operations include *roll-up* (increasing the level of aggregation) and *drill-down* (decreasing the level of aggregation), *slice-and-dice* (selection and projection), and *pivot* (re-orienting the multidimensional view of data).

An OLAP engine demands fast processing on very large volume of data contained in data warehouse. This requires highly efficient cube computation and query processing techniques. A common and powerful query optimization technique is to materialize some or all of subcubes rather than compute them from raw data each time. Harinarayan, et al. [21] gave a detailed discussion on this technique and proposed a greedy algorithm for selective materialization of certain most expensive views and further studied an indexing technique for this task. Moreover, research on how to speed up the computation of multi-dimensional aggregates has also been done. It consists of two influential techniques: One is represented by Agarwal et al.'s work [1], which extends the sort-based and hash-based grouping methods with several efficient optimizations, like combining common operations across multiple group-bys, caching, etc. The other is represented by Zhao, et al.'s [43] multi-way array-based algorithm for MOLAP systems. Besides these, to facilitate efficient data accessing, most current data warehouse systems support index structures including bitmap indexing and join indexing, or even bitmapped join index [10].

2.1.3 Concept hierarchy

The essential background knowledge applied in data generalization is the concept hierarchy associated with each dimension. A concept hierarchy is a tree or lattice structure that organizes concepts in a dimension into a partial order such that those in levels closer to the root are more general than those closer to the leaf nodes.

Generally, a concept hierarchy can be directly derived from the database schema which is referred to as the *schema-based specification*. Or it can be defined by user or domain experts through knowledge of an attribute called the *instance-based specification*. However, there are some cases that automatic generation of concept hierarchies and dynamic adjustment of some existing hierarchies are desirable. The methods for automatic generation of concept hierarchy for numerical attribute based on data distributions and for dynamic refinement of a given or generated concept hierarchy based on a learning request are introduced in [19].

2.2 Association rules discovery

The concept of association rules was first introduced in [2]. Since then, efficient association mining mechanism in large databases and its extensions to different domains have been the subject of many studies. These studies cover a broad spectrum of topics including: (1) fast algorithms based on the level-wise Apriori framework [4, 25] and its variations, including partitioning [36, 33], and sampling [42]; (2) incremental updating and parallel algorithms [11, 34, 16]; (3) mining of generalized and multi-level rules [39, 20]; (4) mining of quantitative and multi-dimensional rules [40, 14, 30, 27, 23]; (5) mining long patterns and dense data sets [6, 7]; (6) mining correlations and causal structures [8, 38]; (7) mining ratio rules [26]; (8) query-based constraint mining of associations [41, 31]; (9) mining cyclic and calendric association rules [32, 35]; (10)

mining partial periodicities [18]; and (11) rule mining query languages [28]. With the limited space and the limited scope of this study, we are not going to survey all of the aspects of such studies and extensions. Our overview of the field of association rule mining is confined to the scope and the methods that we are going to use or extend in this thesis.

Motivated by very huge amounts of sales data (referred as the basket data) and the interest in instituting information-driven marketing processes, the problem of association rules mining was first introduced in [2]. Since then, a lot of research has been done on various aspects of this topic. A lot of exciting results made this technique go far beyond what it was originally applicable. Now, association mining has been a useful and important tool in data mining.

2.2.1 Market basket analysis

The initial association rules mining was mainly market analysis on sales basket data. A record in such data typically consists of the transaction identification and items bought in the transaction. An interesting and influential algorithm for mining frequent itemsets, called **Apriori**, was proposed by Rakesh Agrawal and Ramakrishnan Srikant [4]. Apriori employs an iterative approach: First, the set of frequent 1-itemsets (L_1) is found. L_1 is used to find L_2, the frequent 2-itemsets, which is used to find L_3, and so on, until no more frequent k-itemsets can be found. During this iteration, an important property, called the Apriori property, is used to reduce the scan times of the database and improve the efficiency of such generation of frequent items. The more details on Apriori algorithm will be discussed further in Chapter 4.

Based on the Apriori algorithm, variations of it have been studied, aimed for further improvements of the performance of the algorithm. For example, a hash-based technique [33] can be used to reduce the size of the candidate k-itemsets; a scan reduction technique can be used to reduce the number of database scan; and a transaction reduction technique can be used to reduce the number of transactions scanned in future iteration. Recently, a strategy based on partitioning the data shows a stronger effect than the other scan reduction methods studied [36] reducing the number of scans required to two.

2.2.2 Multi-level association rules mining

Some researchers [5, 20] found that for many applications, it is difficult to find strong and interesting associations among data items at the primitive levels of abstraction due to the sparsity of data. However, many strong associations discovered at rather high concept levels are common sense knowledge. Therefore, a mining system with the capabilities to mine association rules at multiple levels of abstraction and traverse easily among different abstraction spaces is more desirable.

Based on the concept hierarchy and some existing algorithms for mining single-level association rules, a group of efficient algorithms for mining multilevel association rules are proposed [5, 20]. The method proposed by Han and Fu [20] uses concept hierarchy information encoded transaction table, instead of the original transaction table, in iterative data mining. A top-down progressive deepening technique is developed, which extends the existing single level association rules mining algorithms, i.e., first finds large data items at the top-most level and then deepens the mining progress into their large descendants at lower concept levels.

Moreover, reduced support is recommended to progressively lower down the minimum support threshold when mining touches deeper levels of abstraction. For example, for the location dimension, one may use 10% as the minimum support threshold when mining items at the high level of abstraction, such as "region", then use a smaller threshold, like 5%, for mining one level lower as "country", and even smaller threshold, such as 2% for mining one more level lower as "city". It is found much better than uniform support at different levels to generate sufficient meaningful association at low levels but keep the rules generated at high levels sufficiently interesting.

2.2.3 Multi-dimensional association rules mining

According to the terminology used in multi-dimensional databases, the rules commonly mined from transactional data can be referred as single-dimensional association rules since they contain a single predicate with repetitive occurrence. With recent progress on data warehousing and OLAP technology, rather than in a transactional databases, we want to find associations in a multi-dimensional data warehouses, referred as multi-dimensional association rules.

A data cube model for mining multi-dimensional association rules is proposed by Kamber et al. [23], which combines the cube data structure with OLAP techniques, like multi-dimensional slicing and layered cube search. Efficient algorithms are developed by either using an existing data cube or constructing of a data cube on the fly. The idea is to first find the frequent 1-itemsets in each dimension, and then use frequent (k-1)-itemsets to grow frequent k-itemsets by multi-dimensional slicing on the data cube. The performance analysis shows that this method outperforms the direct extension of table-Apriori algorithm. Furthermore, this method can be easily extended to mine multi-level, multi-dimensional association rules.

2.2.4 Constraint-based association mining

Recently, another interesting and important problem, **constraint-based association mining** attracts much attention [37]. User can provide some additional constraints on the rule pattern to be mined, so that the generated rules are more of interests to the user and then more specific and more useful.

Some effective and influential research on constraint-based mining of transaction database has been conducted by Ng, et al. [31]. They introduce and analyze two properties of constraints that are critical to pruning: *anti-monotonicity* and *succinctness*. An efficient algorithm, called CAP, is proposed and it achieves a significant degree of constraints pruning optimization by pushing the constraints deeply into the mining process [31].

Another type of constraint-based mining is meta-rule guided multi-dimensional mining, where each constraint is expressed by a *meta-rule* [23]. This method extends the cube-based method for multi-dimensional association rules mining to solve this problem. Several algorithms are proposed based on the availability of data cubes. The basic idea is that, given an appropriate n-D cube and a meta-rule containing p predicates, then an n-D cube search is applied on the p-layer summary cells and directly find frequent p-predicates sets, L_p.

2.2.5 Interestingness of association rules

As discussed before, not all the strong association rules discovered are interesting enough to present to users. Objective interestingness measures, based on the statistics behind the data, can be used as one step towards the goal of weeding out uninteresting rules. Some research papers are discussing this problem. Craig et al. use the chi-squared test for measuring the independence from classical statistics.

There are many other topics in association rules mining that are being studied. For example, parallel and distributed mining of association rule with the goal of increasing the mining speed; extension of the applicability of association rules to other data types, like multi-media data, sequential data, etc.; and formulation of mining operations as SQL queries and the integration of association rule mining into database management systems.

2.3 Current projects and the DBMiner system

There are a very large number of projects on association rules under way. Since many association rules applications are commercial in nature, there has been a great deal of interest from commercial software vendors. Several universities and government agencies also have significant projects in this area.

Among large commercial software vendors, the most pronounced research projects have been initiated by IBM, Silicon Graphics, and SAS Institutue. The contributions of the IBM Almaden Research Center have been especially notable in the development of association rule technology [22]. They have investigated many efficient methods for finding associations and for formulating association rules mining queries. There are continuing projects including the Quest Data Mining Project [3]. Silicon Graphics has developed a sophisticated data mining and visualization package called MineSet [9] which is based on the public domain machine learning package MLC++. MineSet

[9] includes the ability to mine association rules, and many other mining capabilities. SAS Institute has been focused on data mining using statistical analysis techniques. Microsoft funds basic research on data mining through its Decision Theory & Adaptive Systems group. Recently, Microsoft put forward OLE DB for OLAP [29], a set of objects and interfaces that extend OLE DB to provide access to multidimensional data sources. This specification is supported by large majority of companies doing data mining applications. In addition to these major corporations, numerous smaller companies have developed products that support association rules.

Among government agencies, the most notable supporter of data mining research is NASA, which has supported research directly and in collaboration with universities. Many universities also have data mining projects including SFU, Stanford Univ., University of Helsinki. Among these projects, DBMiner [17], developed by the Intelligent Database Research Laboratory at SFU, is an intelligent data mining and data warehousing system. It consists of several modules, including cube computation, data viewer, cube viewer, summarization tool, classification tool, association tool, and prediction tool. In which, the association module discovers various association relationships among large datasets.

Chapter 3

General Framework

The problem of association mining can be viewed as a pattern search problem. What we try to find out is whether, with respect to a given requirement, there exist strong associations among the interesting dimensions, and if so, what the patterns are like.

In this chapter, we introduce the general architecture of the association mining system and the framework of association rule mining which is implemented and used successfully in our DBMiner system. We also explain some basic concepts that will be used frequently in further discussion.

3.1 Overview

As we mentioned before, most researches on association mining are based on flat relational table structure, such as [2, 4, 25, 36, 33, 20, 39, 40, 42, 30, 41, 31, 8]. Whereas, our association mining method is based on a data cube structure and it integrates association mining with OLAP techniques. Such a method is referred by us as *on-line analytical association mining*. The following observations motivate us to explore this method.

1. The minimum support threshold that an association rule must satisfy implies that each value to be examined must appear a nontrivial number of times in the corresponding attribute of the initial relation.

This observation will serve as a constraint to prevent us from examining some meaningless candidate attributes or their combinations, such as key or candidate key attributes. For example, to mine association rules in a student relation of a university database, the attributes such as student name and student ID should be removed since a rule referring to such attributes, such as the following rule, can not accumulate nontrivial supports to pass the minimum support threshold.

$$\forall x \in student \quad student\_name(x, \text{``}TomLinden\text{''}) \quad \rightarrow$$
$$home\_phone(x, \text{``294\_3251''})$$

Similarly, data in numerical attributes should be generalized to a certain level of abstraction in order to accumulate sufficient support to pass the minimum support threshold. For example, the values in an attribute gpa in the student relation should be generalized to range values, such as "3.8-4.0", or some high level concepts, such as "excellent", rather than retaining at the primitive level, such as 3.873.

2. Usually, given a large database with a large number of attributes, the user is interested in only a small subset of attributes.

   A data relation may have a large number of attributes, e.g., some data relations may have over 50 or 100 attributes. In most cases, we want to extract interesting relationships among only a small number of attributes, such as five. Thus, a facility with a friendly interface should be provided to specify the set of attributes to be mined and exclude the set of irrelevant attributes from examination.

   As we can see, both automatic and user-controlled removal and generalization of attributes in the original table will reduce the number of dimensions and/or raise the level of abstraction for attributes to be examined in the mining, thus the cubes to be constructed could be relatively small compared the huge amount of raw data.

3. The enriched functions on OLAP and date cube facilitate association mining.

   The data cube constructed contains the information not only from the raw data, but also the summarized data. With the methods for efficient construction,

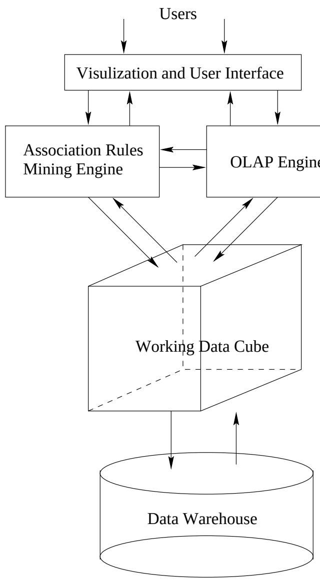


Figure 3.1: The architecture of on-line analytical association mining

generalization, and accessing of data cubes being extensively researched, and with many OLAP systems successfully constructed, the exploration of data cube structures becomes realistic with reasonable efforts. These facilitate efficient and effective mining of associations.

Next, we will lay out the architecture of the association mining system. As shown in Figure 3.1, the general architecture consists of three parts: (1) *data warehouse*, (2) *working data cube and OLAP Engine*, and (3) *Association Mining Engine*, each part will be discussed in the following several sections.

3.2 Data warehouse

Data warehouse is a semantically consistent data store, serving as a physical implementation of a decision support data model and storing the information on which an enterprise needs to make strategic decisions. It includes:

- Integrated historical data

This allows the miner to easily and quickly look across vistas of historical data. Data cleansing and data integration have been performed by dedicated system components. Keys may have to be reconstructed, encoded values reconciled, structures of data standardized, and so on. With such great efforts on data cleansing and data integration, the miner can work on an integrated data warehouse and concentrate on the task of association mining.

- Materialized data cube

  The substantial cost for on-the-fly computation of a data cube calls for precomputation and materialization of cubes. However, materialization of every possible view may require a huge amount of disk space. Therefore, partial materialization of selected cuboids has been suggested as an alternative solution [21]. Since the selectively materialized cuboids are saved in the data warehouse, it may save the amount of mining work for the association miner.

- Metadata

  Metadata serves as a road map to the miner. It is used to describe not the content of data but the context of information. One of the most important kinds of metadata is concept hierarchy. As mentioned before, hierarchy is the basic knowledge of data warehouse and OLAP, and is often embedded in the dimension specification.

Therefore, the data warehouse sets the stage for successful and efficient exploration of the world of data.

3.3 Working data cube and OLAP engine

3.3.1 Working data cube

Given an association mining task involved with dimensions A_1, ..., A_n, we precompute and materialize the task-relevant data from data warehouse into an n-dimensional

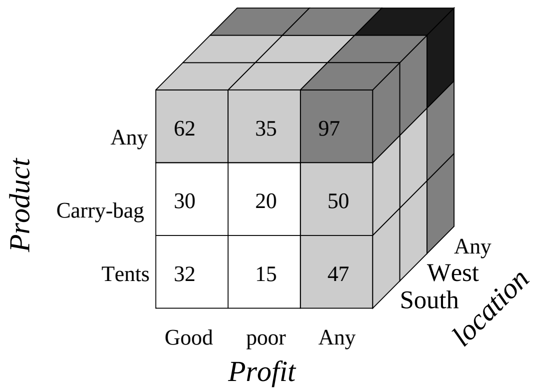


Figure 3.2: A base cube with three dimensions, location, product and profit

data cube. Each dimension of the cube contains $|A_i|+1$ values where $|A_i|$ is the number of distinct values in the dimension A_i. The first $|A_i|$ rows represent the distinct values of A_i. Each cell in these rows stores the "count"value generated from the initial relation (raw data), $r(A_1 = a_1, \ldots, A_n = a_n, count)$. The last row is a special "Any"value, in which each cell stores the aggregation value of the previous rows. Those aggregation values represent one of the essential features of data cube structure. We will explore this more in next two chapters. Generally, we can map the data cube to an $(n+1)$-attribute table, referred as *cube table*, with each attribute representing a dimension and the $(n+1)$-th attribute representing *the count*. Then, each cube cell can be matched into one tuple in the table.

Example 3.3.1 As an example, Figure 3.2 shows a tiny cube which consists of three dimensions: *Location, Product*, and *Profit*. Each dimension has two values. The cube table corresponding to this cube is in Table 3.1. □

Conceptually, a data cube can also be viewed as being partitioned into multiple dimension spaces based on different kinds of cells. The n-D space consists of all data cells without *"Any"* value, i.e., $c(a_1, a_2, \ldots, a_n)$. The $(n-1)$-D space consists of all the cells with a single *"Any"* value, e.g., $c(Any, a_2, \ldots, a_n)$. The 1-D space consists of the cells with $n-1$ *Any"* values, e.g., $c(a_1, Any, \ldots, Any)$. Finally, the 0-D space consists of only one cell with all *"Any"* values, i.e., $c(Any, Any, \ldots, Any)$.

location	product	profit	the count
South	Carry Bags	poor	20
South	Carry Bags	good	30
South	Carry Bags	any	50
South	Tent	poor	15
South	Tent	good	32
South	Tent	any	47
South	any	poor	35
South	any	good	62
South	any	any	97
...	...	...	...
North	any	any	118
any	any	any	215

Table 3.1: The cube table for cube in Figure 3.2

Example 3.3.2 Based on the cube in Example 3.3.1, we get four cube spaces according to the definition,

1. The 3-D space consists of 8 rows in which there is no *"Any"* value. The first, second, forth and fifty row shown in Table 3.1 are in this space;
2. The 2-D space consists of 12 rows in which there is only one *"Any"* value. The third, sixth, seventh and eighth row shown in Table 3.1 are some examples.
3. the 1-D space consists of 6 rows, each containing two *"Any"* values. The ninth and tenth row shown in Table 3.1 are two such cases.
4. Finally, one cell { *"Any"*, *"Any"*, *"Any"*} composes the 0-D space which is shown as the last row in Table 3.1.

3.3.2 OLAP engine

Once the working cube is constructed, the complete information needed for the given task is preserved in the cube cells including the aggregated ones, so that we do not have to refer to the original relation any more. Such a cube serves as a data source and an interface between the data and the mining task. Since data mining works on a very large database, it is essential to have enriched and efficient functions to create and play with the data cube, Thus, OLAP Engine, whose major task is to compute user's OLAP instructions, such as creating data cube, drilling, dicing, pivoting, etc., and return the results to the user interface layer, shows its important role in the entire data mining architecture.

Although the details of OLAP techniques are not our major part in this thesis, there are many papers on efficient computation of data cube and performing OLAP operations, such as [1, 21, 10, 15, 43].

3.4 Association mining engine

Association Mining Engine is the major component of the association mining system. It is also the major part discussed in this thesis. Next, we will introduce the general framework of association mining engine. The detailed algorithm will be discussed in the following several chapters.

First, we introduce some important terms that are frequently used in this thesis.

3.4.1 Basic concepts

First of all, a set of items is referred as an *itemset*, an itemset that contains k items is a *k-itemset*. The left hand side of the rule (the part of A_1, A_2, ..., A_m in the definition 1.1) is known as the *body* of the rule, and the right hand side is the *head* of the rule. The *frequency* of an itemset is the total number of transactions that contain the itemset.

Support and *confidence* are two major measures of rule interestingness. They respectively reflect the usefulness and certainty of a discovered rule. Given a rule $X \rightarrow Y$, the support is the probability for a transaction to contain $X \cup Y$, and the confidence is the probability for a transaction which contains X to contain Y. By the concept of itemset frequency, the computation of *support* and *confidence* can be defined by the following two equations:

$$support(X \rightarrow Y) = frequency(X \cup Y)/totalcount \tag{3.1}$$

$$confidence(X \rightarrow Y) = frequency(X \cup Y)/frequency(X) \tag{3.2}$$

An itemset is *frequent* if the support is no less than a minimum support threshold. And, an association rule is considered strong if it satisfies both a *minimum support threshold* and a *minimum confidence threshold*.

Here, there is a concrete example.

Example 3.4.1 The following rule is generated from the Sales relational database.

$$\begin{aligned} &Product(X, \text{“}CarryBags\text{”}), Location(X, \text{“}North\text{”}) \rightarrow \\ &\quad Profit(X, \text{“}good\text{”}) \;\; [confidence = 80\%, support = 20\%] \end{aligned}$$

This rule means that *Carry Bags* bought in in *North* is likely to have a *good* profit(with a 80% confidence, or certainty) and such customers represent 20% of all customers under study. Now, if the minimum support threshold is 15% and the minimum confidence threshold is 75%, the above rule is interesting. □

In this thesis, several types of association analysis are discussed.

- *Inter-dimensional Association.*

  This is the association among a set of dimensions. For example,

  $$Location(X, \text{“}South\text{”}), Profit(X, \text{“}bad\text{”}) \rightarrow Product(X, \text{“}Tents\text{”})$$

  We can see that all items in this rule are from different dimension. So, we also call it *multi-dimensional association without repetition.*

- *Intra-dimensional Association.*

  This is the association within one dimension. For example,

  $$Product(X, \text{“}CarryBags\text{”}) \rightarrow Product(X, \text{“}Tents\text{”})$$

  Since all items in this rule are from the same dimension, we also call it *single-dimensional association.*

- *Hybrid Association*

  This is the association among a set of dimensions, but some items in the rule are from one dimension. So, this is also called *multi-dimensional association with repetition*, while the previous inter-dimensional association mining referred as multi-dimensional association mining without repetition. For example,

  $$Location(X, \text{“}North\text{”}), Product(X, \text{“}CarryBags\text{”}) \rightarrow Product(X, \text{“}Tents\text{”})$$

- *Constraint-based Association*

  It is preferable to have some user-specified constraints to guide an association rule mining process. Such constraints can be applied to rule items or rule forms. For example, user want to find all rules item *Product(X, “Carry Bags”)* in the rule head, then the following is a desired rule satisfying this constraint,

  $$Profit(X, \text{“}good\text{”}) \rightarrow Location(X, \text{“}South\text{”}), Product(X, \text{“}CarryBags\text{”})$$

3.4.2 Framework of association mining engine

Generally, after generating the working cube, the problem of discovering association rules can be decomposed into two subproblems:

1. Find all frequent itemsets with support above minimum support.

2. Use the frequent itemsets to generate the desired rules.

Correspondingly, based on data cube structure, we use a two-phase framework in our system as following:

- *Phase 1: Generate frequent itemsets satisfying thresholds and specified constraints.*

  - INPUT.

    1. task-relevant working data cube
    2. minimum support threshold: *min_supp*

3. A set of constraints C on rule items

– OUTPUT.

A list of frequent itemsets satisfying constraints is the output. According to the type of association we want to generate, the frequent itemset can be inter-dimensional itemset, intra-dimensional itemset or hybrid itemset.

- *Phase 2: Generate desired association rules.*

  – INPUT.

  1. frequent itemsets output from phase 1
  2. minimum confidence threshold: *min_conf*
  3. a set of constrain C' on the rule forms.

  – OUTPUT.

  Once getting the satisfied frequent itemsets, this phase is proceeded to generate the desired rules. We use the confidence as the significance metric, and whatever further constraints C' to be imposed on the body and head can be specified. Then, all the association rules that satisfy confidence threshold and constraints are output. Furthermore, upon examining the rules, the user has the option to perform an correlation analysis if association analysis itself is not sufficient enough to find interesting rules. Finally, all association rules that satisfy the confidence threshold and C' are output.

3.5 Visualization and graphical user interface

Allowing visualization of discovered association rules in various forms can help users with different backgrounds to identify rules of interest and to guide the system into further discovery. Also, a good user interface, enriched by query constraints, we use a convinent mining tool to serve as the high-level user interface, and implement two forms of visualizations, i.e., table view and graphical view. We will have more discussion on this topic in Chapter 6.

Chapter 4

OLAP-Based Association Mining

As we discussed in the previous chapters, the task for OLAP-based association mining is to explore how to use the data cube structure for mining association rules and study efficient mining methods using such a structure. According to the type of rules to be mined, such a problem can be decomposed into three sub-categories: *inter-dimensional association*, *intra-dimensional association*, and *hybrid association*. These subclasses of mining problems are consequently referred to as *inter-dimensional*, *intra-dimensional*, and *hybrid association mining* respectively.

Basically, there are three steps in OLAP-based association mining, that is, all the algorithms are divided into three parts.

1. Generating the task-relevant working cube with desired dimensions,
2. Generating frequent itemsets, i.e., the itemsets whose support is greater than the given minimum support threshold.
3. Generating association rules from the frequent itemsets.

In this chapter, we will give the algorithms for each subclass of association mining. Multi-level association mining is also discussed.

Before we start the discussion, first, we will introduce the data source *SALES* which will be used in the examples through the whole chapter. SALES is a sales database which includes sales information of a company. Part of the data is shown in Table 4.1. We can see that there are three attributes: *Location*, *Product* and *Profit*, with two of them being categorical and the third one, Profit, numerical. From now on, without further specification, all the examples are using this data source as the raw data.

location	product	profit
Seattle	Alert Devices	6.8
Seattle	Alert Devices	2.7
Seattle	Alert Devices	11.0
...	...	...
Seattle	Alert Devices	4.8
Seattle	Carry-Bags	2.7
Seattle	Carry-Bags	3.0
Seattle	Carry-Bags	2.0
...	...	...
Tokyo	Tents	4.5
Tokyo	Tents	9.9

Table 4.1: The SALES example data with three attributes

4.1 Generating working cube

Given a mining task, we need to first generate a working data cube by extracting relevant date from data warehouse. Let us take an example to show how this working cube is generated.

Example 4.1.1 Suppose we want to perform association mining involving three dimensions: *Location, Product* and *Profit*, then a three-dimensional cube is generated by the OLAP Engine, as shown in Figure 4.1.

First, numerical dimension *Profit* is generalized. Suppose the generalized level contains the following 5 values:

$$poor(1 \sim 3), \quad mediocre(3 \sim 5), \quad average(5 \sim 7),$$

location	product	profit	count
HongKong	Sport Wear	good	30
HongKong	Sport Wear	excellent	10
HongKong	Water Purifier	good	30
Mexico	Alert Devices	poor	20
Mexico	Carry Bags	poor	85
Mexico	Carry Bags	average	25
Mexico	Tents	mediocre	25
Mexico	Tents	excellent	20
Seattle	Carry Bags	good	100
Seattle	Sport Wear	poor	20
Seattle	Sport Wear	good	40
Seattle	Water Purifier	average	25
Tokyo	Carry Bags	mediocre	10
Tokyo	Sport Wear	average	20
Tokyo	Tents	mediocre	20

Table 4.2: A generalized relation from SALES with count attributes

$$good(7 \sim 9), \quad excellent(9 \sim 11)$$

Then we map the real profit values in the raw data to these five values, aggregate it into a new relation with one more attribute *count*. This new relation is shown in Table 4.2.

Based on the new relation, a cube construction algorithm will generate a 3-dimension cube with:

1. dimensions : Location, Product, Profit.

   Each dimension has several values including the distinct attribute values from Table 4.2 plus the special "*Any*" value.

2. the *count* value from Table 4.2 saved in each cube cell. For example, the count in $cell(Location = \text{"}Seattle\text{"}, Product = \text{"}CarryBags\text{"}, Profit = \text{"}good\text{"})$ is 100.

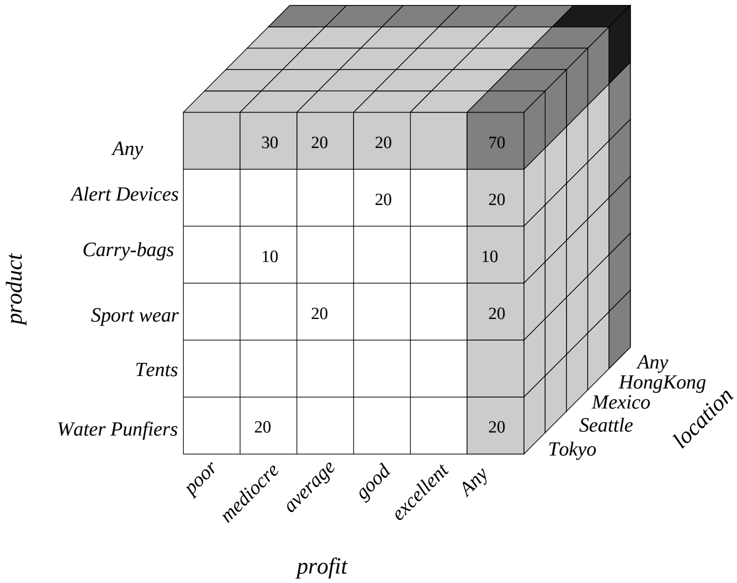


Figure 4.1: The three-dimensional data cube generated from SALES

Furthermore, according to the definition of the cube space, we can get four cube spaces from this purticular cube, depicted in Figure 4.1 with different colors. For example, the space comprised of the white cells is the 3-D space of the cube.

4.2 Generating frequent itemsets

Now we have learned how to generate working data cube for the mining task. In the following sections, we study algorithms for generating frequent itemsets for the purpose of mining association rules. Different categories of associations, i.e., inter-dimensional, intra-dimensional, and hybrid association rules, use different algorithms.

Because of the importance of the Apriori algorithm [4], we will first fully discuss it in the following section.

4.2.1 The Apriori algorithm

The Apriori algorithm, first proposed by Agrawal and Srikant [4], is an interesting and influential algorithm for generating intra-dimensional frequent itemsets based on the transaction table. The typical transaction table consists of two fields: one is *transaction ID*, the other is the *items* associated with the transaction ID. By grouping the transaction ID, a transaction table can be transformed into a set-based table in which the items sharing the same transaction ID are merged into one tuple. An example of such a table is as follows.

Example 4.2.1 Based on the SALES data, we take the *Location* as the transaction ID attribute and product as the item attribute. After grouping, there are totally four transactions, i.e., $totalcount = 4$. We assume that the items within a transaction are stored in the lexicographic order as shown in Table 4.3.

trans ID(location)	product
HongKong	Water Purifiers, Sport Wear
Mexico	Alert Devices, Carry-Bags, Tents
Seattle	Carry-Bags, Water Purifiers, Sport Wear
Tokyo	Alert Devices, Carry-Bags, Water Purifiers, Sport Wear

Table 4.3: The transaction table generated from SALES

Detailed algorithm is depicted in the following paragragh. Step 2 and 3 of Apriori find the frequent 1-itemsets, L_1. And then Step 4, divided into two sub-steps, generates frequent k-itemset. First, candidate k-itemsets C_k are generated by the *gen_candidiate* procedure, which joins the frequent $(k-1)$-itemset L_k and eliminates those having a $(k-1)$-subset that is not frequent. Second, frequent k-itemsets L_k are generated from C_k in procedure *gen_frequent* as described below: once all the candidates have been generated, the database is scanned; for each transaction, a *subset* function is used to find all subsets of the transaction that are candidate, and the count for each of these candidates is accumulated; finally, all those candidates satisfying minimum support comprise L_k.

Algorithm 4.2.1 *(Apriori)* Generating frequent itemsets using an iterative approach.

Input: Transaction databse D and minimum support threshold min_supp.

Output: The set of frequent itemsets L in D.

Method

```
1. k = 1; L = φ;
2. Candidate 1-itemsets C_1 = {all the distinct values in item
   attribute}.
3. Compute frequent 1-itemsets L_1.  L_1 = gen_frequent(1, C_1);
4. Repeat
     k = k + 1;
     Generate Candidate k-itemsets C_k = gen_candidate(k, L_{k-1});
     Compute frequent k-itemsets L_k = gen_frequent(k, C_k);
     L = L ∪ L_k;
   Until L_k = φ;

Function gen_frequent(k, C_k)
% Generate frequent k-itemsets L_k from candidates C_k.
   for each itemset c ∈ C_k do
        c.frequency = 0;
   for each transaction t ∈ D { %scan D for counts
        C_t = subset(C_k, t);% get the subsets of t ∈ C_k
        for each candidate c ∈ C_t
             c.frequency ++;
   }
   return L_k = {c ∈ C_k | c.support ≥ min_supp}


Function gen_candidate(k, L_{k-1})
% Generate candidate k-itemsets C_k from L_{k-1}.
   C_k = φ;
   for each itemset I_1 ∈ L_{k-1}
        for each itemset I_2 ∈ L_{k-1}
```

```
        if (first k-2 items in I_1 and I_2 are same but
              the last item are different) then {
              c = I_1 ⋈ I_2;
              if ∃(k-1)-subset s of c, s ∉ L_{k-1} then
                    delete c;
              else add c to C_k;
        }
return C_k;
```

□

Rationale. The Apriori algorithm employs an iterative approach. First, the frequent 1-itemsets is found, denoted as L_1. L_1 is used to find L_2, the frequent 2-itemsets; which is in turn used to find L_3, and so on, until no more frequent k-itemsets can be found. To improve the efficiency of such level-wise (i.e., level-by-level) generation of frequent itemsets, an important property, called the *Apriori property*, presented below is explored to reduce the search space.

Theorem 4.1 *Apriori property: All non-empty subsets of a frequent itemset must be frequent.*

This property is proposed and proved by Agrawal and Srikant [4]. It is based on the following observation. Since all the itemsets adopt the same minimum support threshold *min_supp*, if an itemset I is not frequent, i.e., $Prob\{I\} < min\_supp$, by adding an item i to the itemset I, such a combined set cannot occur more frequent than I alone, and thus we have $Prob\{I \cup \{i\}\} < min\_supp$.

This property belongs to a special category of the properties called *anti-monotonicity* [31] in the sense that *if a set cannot pass a test, all of its superset will fail the same test as well.* It is called *anti-monotone* because the property is monotonic in the context of failing a test. Later we will see that this property is important in constraint-based optimization. It will be discussed in details in the next chapter.

Based on the Apriori property, the frequent k-itemsets, L_k, can be obtained by joining the frequent $(k-1)$-itemsets, L_{k-1}. Moreover, for a k-itemset l_k to be in L_k, all of its $(k-1)$-subsets must be frequent. If any of such a subset is not in L_{k-1}, l_k can be removed from the candidate list.

This observation facilitates the generation of candidate k-itemset, C_k, from L_{k-1}. C_k is a superset of L_k, that is, its members may or may not be frequent but all the frequent k-itemsets are in C_k. A final check of the support of each candidate in C_k results in the determination of L_k (i.e., all candidates having a support no less than the minimum support threshold are frequent by definition, and therefore, belong to L_k). How to generate a small C_k? Based on the Apriori property, C_k can be generated in two steps. First, a candidate is generated by joining the members in L_{k-1}, where two members are joinable if they share $(k-2)$ common items, i.e.,

$$L_{k-1} \bowtie L_{k-1} = \{A \bowtie B | A, B \subset L_{k-1}, |A \wedge B| = k-2\}. \tag{4.1}$$

Second, remove from C_k any of its members that contains a $(k-1)$-subset that is not frequent. Now we have a basic understanding about how the Apriori algorithm works on the transaction database.

Since the efficient construction, generalization, and accessing of data cubes have been extensively researched with many successfully constructed OLAP systems, the exploration of data cube structures becomes realistic with reasonable efforts. Moreover, the data cube structure facilitates the mining of associations at *multiple levels of abstraction*. So, from next section on, we will examine efficient methods for mining association rules in a large database using a data cube structure. First, let us start from the simplest case: *mining intra-dimensional associations*.

4.2.2 Intra-dimensional association mining

Intra-dimensional association is the association within one dimension, which we call *item dimension*, by grouping another dimension, which we call *transaction dimension*. So, generally, there are two dimensions involved in the intra-dimensional association mining. Thus, a two-dimensional cube is created by the OLAP Engine and served as the working data cube for mining. Here we give an example of such a cube as follows.

Products \ location	Tokyo	Seattle	Mexico	HongKong	Any
Any	70	185	175	70	500
Alert Devices	20		20		40
Carry-bags	10	100	110		220
Sport wear	20	60		40	120
Tents			45		45
Water Punfiers	20	25		30	75

Figure 4.2: A data cube for mining intra-dimensional association rules

Example 4.2.2 Based on the SALES data (Table 4.1), we take the *Location* as the transaction dimension and *Product* as the item dimension. The corresponding two-dimensional cube is then created and shown in Figure 4.2. According to the cube definition, each cell saves the *count* value generated from the original relation. □

Now, we want to do the intra-dimensional association mining based on such a cube. Algorithm 4.2.2 shows the detailed algorithm. The procedure is very similar to Apriori algorithm except that the support for each candidate itemset is computed by scanning part of the data cube, not the transaction table.

Algorithm 4.2.2 Generating intra-dimensional frequent itemsets.

Input:

- A two-dimensional data cube: $Cube[transaction, item]$.
- Minimum support threshold: min_supp.

Output: The frequent intra-dimensional itemsets L.

Method

```
1. k = 1;  L = φ;
2. Candidate 1-itemsets C_1 = {all the distinct values in item
   dimension};
3. Generate frequent 1-itemsets L_1 = gen_frequent(1, C_1);
4. Repeat
       k = k + 1;
       Generate Candidate k-itemsets C_k = gen_candidate(k, L_{k-1});
       Generate frequent k-itemsets L_k = gen_frequent(k, C_k);
       L = L ∪ L_k;
   Until L_k = φ;

Function gen_frequent(k, C_k)
% Generate frequent k-itemsets L_k from candidates C_k.
   L_k = φ;
   for each candidate k-itemset, I = {i_1, i_2, ..., i_k} ∈ C_k {
        frequency = 0;
        for each transaction t {
             AllIn = TRUE;
             for each item i ∈ I {
                  Get the count saved in the cube cell(t, i);
                  if (the count is 0) then {
                  % item i is not in this transaction t
                       AllIn = FALSE;
                       break;
                  }
             }
             if( AllIn = true)
             % all items in this itemset are in transaction t
                  frequency ++;
        }
        support = frequency/totalcount;
```

```
        if (support > min_supp) % I is a frequent itemset
            L_k = L_k ∪ I;
    }
    return L_k;

Function gen_candidate(k, L_{k-1})
% Generate candidate k-itemsets C_k from frequent (k-1)-itemsets
% L_{k-1}.
    This function works the same as Algorithm 4.2.1.
```

Rationale Based on the Apriori property, the algorithm starts with finding frequent 1-itemsets. For each candidate item i, we take the slicing from the cube corresponding to item i. This slicing has n cube cells(n is the number of transactions), with each cell(t, i) for each transaction t. According to the defination of cube, each cell has a *count* value saved in it. This value can be used to decide whether the item i is contained in transaction t: if it is larger than 0, it means that item i is in this transaction t, otherwise, it is not in t. In the sense, we can see that the number of cells in this slicing whose account is larger than 0 is the number of transctions containing the item i, that is, the frequency for i. By computing the support and comparing with min_supp threshold, we can judge whether i is frequent or not.

Next, to discover the frequent k-itemsets L_k, the algorithm uses $L_{k-1} \bowtie L_{k-1}$ to generate the candidate k-itemsets, C_k. This join procedure is exactly the same as Apriori algorithm. Then, for every candidate itemset $I \in C_k$, since the items in I are all in the same dimension but different positions from data cube, we have to search through the corresponding slicing covering those items and all transactions to find how often these items occur together. For each transaction t, we look at those k cells in this slicing where transaction $= t$. If the *count* values saved in all these k cells are larger than 0, it means that these items occur in this transaction together. So, the number of such transactions is exactly the frequency for this itemset I. This frequency computation procedure can be extened to any candidate k-itemset. □

Example 4.2.3 Let us try the example based on the cube in Figure 4.2. We want to mine intra-dimensional association rules taken the *Location* as transaction dimension, *Product* as the item dimension and min_supp = 50%.

- In the first iteration of the algorithm, the total five items(*Products*) comprise the candidate 1-itemsets, C_1. Then, for every product in C_1, e.g., *"Water Purifier"*, we do slicing on the cube and get a plane of four cells for *Product* = *"Water Purifier"*. Comparing all the *count* values in this slicing, {30, 0, 25, 20}, with 0, there are three values greater than 0. So the frequency for "Water Purifier"is 3. The algorithm simply repeats this procedure for all candidates in C_1. The frequency results are shown in Figure 4.3.

- Compared with the minimum support threshold, 50%, the frequent 1-itemsets L_1 can then be determined. It consists of those candidate 1-itemset with frequency at least 2.

- To discover the candidate 2-itemsets C_2, the algorithm uses $L_1 \bowtie L_2$ and generate 6 itemsets comprising C_2.

- For each itemset in C_2, e.g., {*"Carry Bags", "Water Purifier"*}, we do slicing on the cube and get a plane of eight cells corresponding to *Product* = *"Carry Bags"* and *Product* = *"Water Purifier"*. we check the two *count* values in each pair of cells within the same transaction (Location). We find that only two pairs of cells with both values larger than 0, i.e., {{*"Seattle", "Water Purifier"*} {*"Seattle", "Carry Bags"*}} and {{*"Tokyo", "Water Purifier"*}, {*"Tokyo", "Carry Bags"*}}. Therefore, the frequency of this itemset is 2. We repeat this procedure for all other itemset in C_2, the results are also shown in Figure 4.3.

- The frequent 2-itemsets L_2 is then determined, consisting of those candidate 2-itemsets in C_2 with frequency at least 2.

- The generation of the candidate 3-itemsets C_3 is divided into two steps. First let $C_3 = C_2 \bowtie L_2$ = {{"Alert Devices", "Carry Bags", "Sport Wear"}, {"Alert

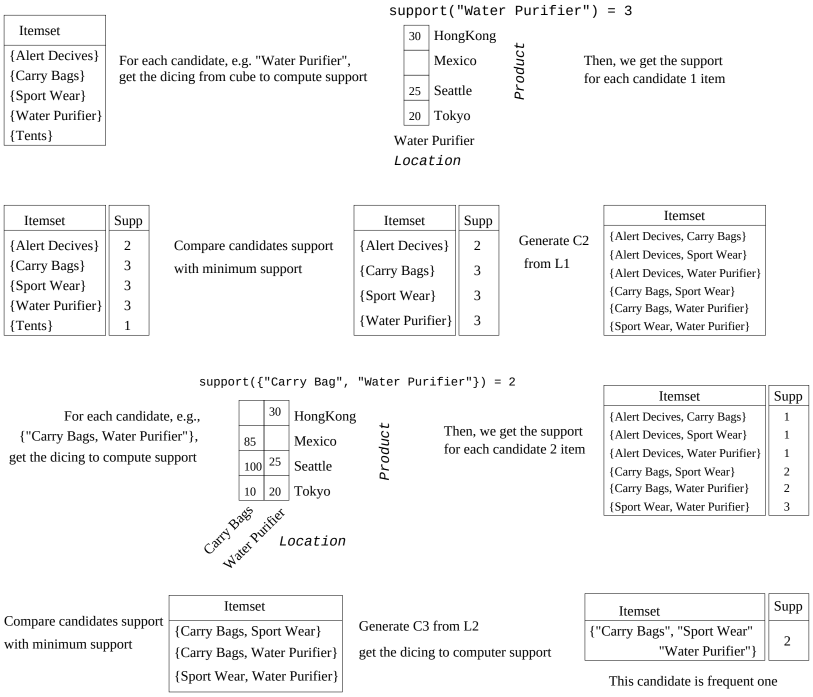


Figure 4.3: Generating frequent itemsets with minimum support = 2.

Devices", "Carry Bags", "Water Purifier"}, {"Carry Bags", "Sport Wear", "Water Purifier"}. Based on the Apriori property that all subsets of a frequent itemset must also be frequent, we can determine that the candidates {"Alert Devices", "Carry Bags", "Sport Wear"} and {"Alert Devices", "Carry Bags", "Water Purifier"} cannot possibly be frequent. We therefore remove them from C_3, thereby saving the effort of unnecessary support computation.

- In order to determine L_3, the cube is sliced and scanned in the same way as generating frequent 2 itemset . The final result for L_3 is shown in Figure 4.3.

- No more frequent itemsets can be found (since here, $C_4 = NULL$), and so the algorithm terminates, with all of the frequent itemsets found.

□

Discussion In this subsection, the problem of finding intra-dimensional association rules with OLAP technologies is discussed. A cube-based algorithm is presented.

Basically, the algorithm works as follows: it generates the candidate itemsets, and for every candidate itemset, it scans the necessary cube cells to compute the support and then decides whether this itemset is frequent or not.

The first part takes advantage of the Apriori property, that is, for every two frequent $(k-1)$-itemset with (k-2) items overlapping, we join them together to get a k-itemset and then decide whether it is a candidate or not by *tracking back* the other (k-1)-subset of this itemset. If any of these subsets is not frequent, then this k-itemset is not candidate. By analyzing the algorithm 4.2.2, the total number of *tracking back* can be computed using Equation 4.2, where n is the size of the longest possible candidate itemset, $|L_{k-1} \bowtie L_{k-1}|$ is the number of possible k-itemset generated by joining the frequent (k-1)-itemsets L_{k-1}, and $(k-2)$ is the number of other (k-1)-subsets since for each k-itemset I joined by two (k-1)-itemsets, I_1, I_2, we only need to check the other (k-1)-subset except I_1, I_2.

$$\sum_{k=2}^{n}(|L_{k-1} \bowtie L_{k-1}| \times (k-2)) \tag{4.2}$$

The second part is mainly for cube scanning. The number of scanning is the total number of cells used in the frequency computation for all candidate itemsets and can be computed using Equation 4.3, where $|C_k|$ is the number of itemsets in the candidate k-itemses C_k, and $|transaction|$ is the number of transactions in the data cube.

$$\sum_{k=1}^{n}(|C_k| \times k \times |transactions|) \tag{4.3}$$

Therefore, according to the above two equations, the time spent in the association mining can roughly be divided into two parts, *tracking back* time and *cube scanning* time. Totally, with a fixed minimum support threshold, there are three variables in the above two equations: $|L_{k-1} \bowtie L_{k-1}|$, $|C_k|$ and n. The major factors affecting their values are the data cube density and the number of distinct values in item dimension. The denser the cube and the bigger the number of distinct values in item dimension, the more time that has to be spent in the mining.

4.2.3 Inter-dimensional association mining

Inter-dimensional association is the association among a set of dimensions. The whole procedure is also based on the Apriori algorithm with a different support computation method. In this case, since the items in a itemset come from different dimensions, by using the available summary layers of data cube, the frequency for each itemset can be directly obtained from one cube cell, making the mining procedure very efficient. The detailed algorithm is shown as follows.

Algorithm 4.2.3 Generating inter-dimensional frequent itemsets among n dimensions with Apriori pruning.

Input:

- An n-dimensional data cube: $CB[d_1, \ldots, d_n]$.
- Minimum support threshold: min_sup.

Output: The set of frequent inter-dimensional itemsets L among the n dimensions.

Method

1. $k = 1$; $L = \phi$;
2. Generate candidate 1-itemset for each dimension
   C_{1,d_i} = {all distinct values in dimension d_i}, $C_1 = \bigcup_{i=1}^{n} C_{1,d_i}$;
3. Generate frequent 1-itemset L_1 = $gen\_frequent(1, C_1)$;
4. Repeat
   $k = k + 1$;
   Generate candidate k-itemsets C_k = $gen\_candidate(k, L_{k-1})$;
   Generate frequent k-itemsets L_k = $gen\_frequent(k, C_k)$;
   $L = L \cup L_k$;
   Until $L_k = \phi$;

Function gen_frequent(k, C_k)
% Generate frequent k-itemset L_k from candidates C_k

```
    L_k = φ;
    for each candidate I = {i_1, i_2, ..., i_k} ∈ C_k do {
        frequency = the count in the cell(i_1, i_2, ..., i_k) in
        the k-D cube space
        support = frequency/totalcount;
        if (support > min_supp) then
        %I is a frequent itemset
            L_k = L_k ∪ {I};
    }


Function gen_candidate(k, L_{k-1})
% Generate candidate k-itemsets C_k from frequent (k-1)-itemsets
% L_{k-1}
    C_k = φ;
    for each item l_1 ∈ L_{k-1} {
        for each item l_2 ∈ L_{k-1} {
            if( l_1 and l_2 has k-2 same items, and the other
                one from different dimensions ) then {
                c = l_1 ⋈ l_2
                if c has infrequent (k-1)-subset then
                    delete c
                else add c to C_k
            }
        }
    }
    return C_k
```

Rationale Based on the Apriori property, the algorithm starts with finding frequent 1-itemsets, then to discover the set of frequent k-itemsets L_k, the algorithm uses $L_{k-1} \bowtie L_{k-1}$ to generate the candidate k-itemsets C_k. For every candidate k-itemset

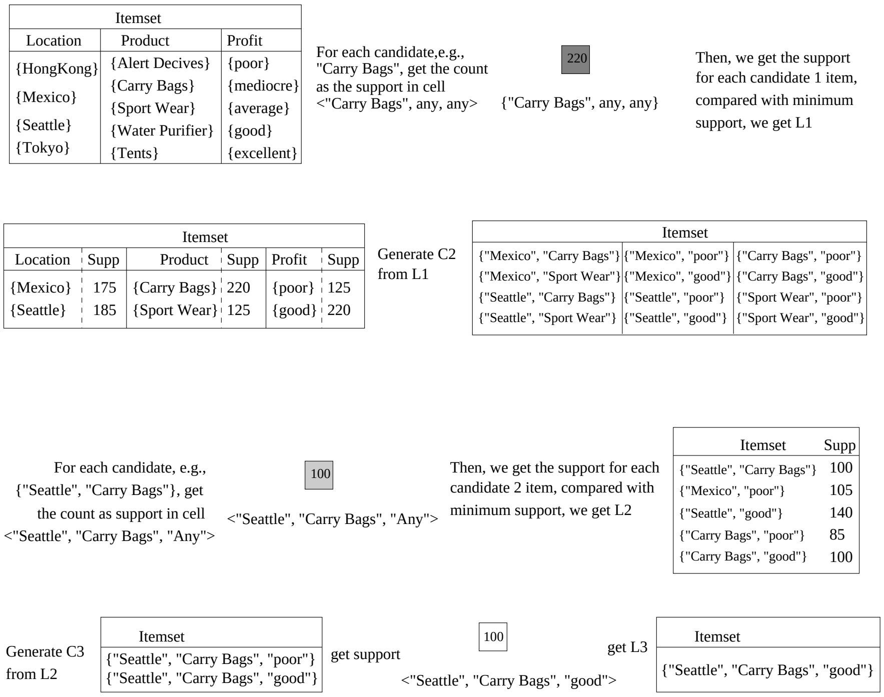


Figure 4.4: Generating frequent itemsets with minimum support, 18%.

$I \in C_k$, we check the corresponding cell in the k-D cube space. According to the definition of cube, the *count* value saved in each cell is the aggregation from the initial data base, thus it is exactly the frequency for the itemset represented by the cell. Then we compare it with the threshold, and collect those whose support is no less than min_sup. □

Example 4.2.4 Let us use an example to show how this algorithm works. This example is based on the cube shown in Figure 4.1 in Section 4.1. Now, we want to do inter-dimensional association mining among three dimensions: *Location, Production and Profit.*

Following Algorithm 4.2.3, the mining is performed as follows.

- First, we get all the distinct values in the three dimensions as the candidate 1-itemsets C_1. Then, for each item in C_1, for example, "Carry Bags"in *Product* dimension, we get the cell (Any, "Carry Bags", Any) in 1-D cube space, the count value saved in this cell is 220, thus 220 is the frequency for itemset {"Carry Bags"}. We iterate for every item in C_1 in this way to compute the frequency.

- According to the minimum support threshold (i.e., *min_sup* = 18%), the frequent 1-itemset L_1 can then be determined. It consists of the candidate 1-itemsets having frequency at least 80. The results are shown in Figure 4.4.

- By the intersection of each two frequent 1-itemsets from different dimensions $L_{1,(A_i)} \bowtie L_{1,(A_j)}$, we generate 12 candidate 2-itemsets comprising C_2 referred to Figure 4.4.

- For each itemset in C_2, e.g., {*"Seattle", "Sport Wear"*}, we get the cell ("Seattle", "Sport Wear", Any) from 2-D cube space, the *count value* saved in this cell, i.e., is 100, it means that the frequency for this itemset is 100 as well. In the same way, we can get the frequency for all 2-itemset in C_2. The resulting counts are also shown in Figure 4.4.

- The frequent 2-itemsets, L_2, is then determined, consisting of those candidate 2-itemsets in C_2 having a frequency of at least 80.

- Now, it is another join-and-prune step to get the candidate 3-itemset. For example, by joining {*"Seattle", "Carry Bags"*} and {*"Seattle", "poor"*}, we get {*"Seattle", "Carry Bags", "poor"*}. Next, we need to track back whether {*"Carry Bags", "poor"*} is a frequent 2-itemset. The fact is that it is, then {*"Seattle", "Carry Bags", "poor"*} is added to C_3.

- We compute the frequency for each itemset in C_3 by searching the 3-D cube space and checking the *count* value saved in the corresponding cell.

□

So far, we have described one cube-based algorithm for inter-dimensional association mining. This method is mainly based on the Apriori property, that is, the candidate k-itemsets C_k are generated by joining frequent $(k-1)$-itemsets L_{k-1} and pruning all those containing subsets which are not in L_{k-1}. This join-and-prune procedure reduces the size of C_k, and this reduces the frequency computation for C_k. But in another way, we need to spend time in joining L_{k-1}, and then for each joined k-itemset, each $(k-1)$-subset should be checked to see whether it is frequent $(k-1)$-itemset. If the data cube is sparse and the size of frequent k-itemset is not big, this join-and-check procedure is worthy compared to the optimization gained by pruning the candidate itemsets. But if the data cube is very dense, frequent itemsets become very huge. It is very possible that we will spend much time in the joining and checking, but cannot prune off many candidate itemsets. Furthermore, implementing Apriori property requires a complex data structure. Usually, a B+ tree or more advanced tree structure is used. In this case, the Apriori algorithm does not show its advantage. In the following, we give another cube-based algorithm to solve this problem. In this method, we do not use the Apriori property for pruning. After we get the frequent 1-itemsets L_1, we generate the candidate k-itemsets by obtaining all possible combination of k items within L_1.

Algorithm 4.2.4 Generating inter-dimensional frequent itemsets among n dimensions without Apriori pruning

Input:

- An n-dimensional data cube: $C[d_1, \ldots, d_n]$.
- Minimum support threshold: min_sup.

Output: The frequent inter-dimensional itemsets L.

Method

1. $k = 1$; $L = \phi$;
2. Generate candidate 1-itemset for each dimension C_{1,d_i} = {all distinct values in dimension d_i}. $C_1 = \bigcup_{i=1}^{n} C_{1,d_i}$
3. Generate frequent 1-itemsets L_1 = $gen\_frequent(1, C_1)$;

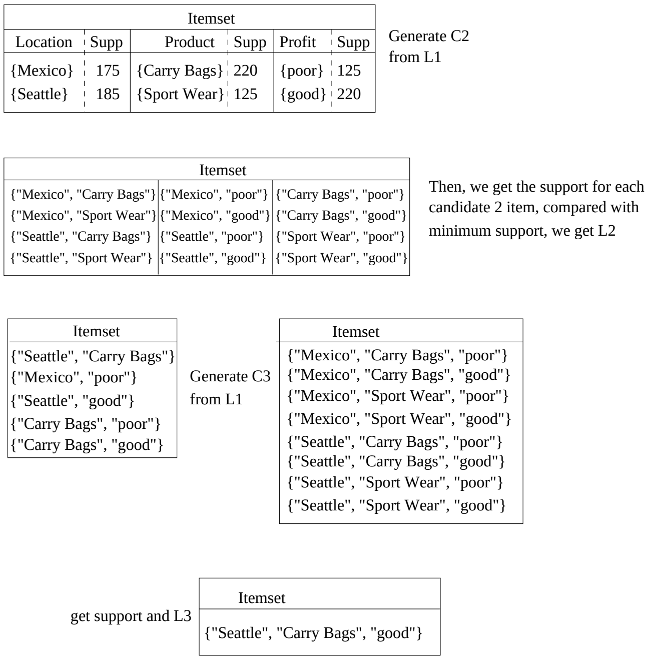


Figure 4.5: Generating frequent inter-itemsets without Apriori property, minimum support is 18%.

4. Repeat

  $k = k + 1$;

  Generate candidate k-itemsets C_k = $gen\_candidate(k, L_1)$;

  Generate frequent k-itemsets L_k = $gen\_frequent(k, C_k)$;

  $L = L \cup L_k$;

  Until $L_k = \phi$;

Function gen_frequent(k, C_k)

% Generate frequent k-itemsets L_k from candidates k-itemsets C_k

  for each candidate $I = \{i_1, i_2, \ldots, i_k\} \in C_k$ do {

    frequency = the count in the corresponding cell$(i_1, i_2, \ldots, i_k)$ in the k-D cube space

    $support = frequency/totalcount$;

    if $(support > min\_supp)$ then

    %I is a frequent itemset

```
        L_k = L_k ∪ {I};
    }


Function gen_candidate(k, L_1)
% Generate candidate k-itemsets C_k from frequent 1-itemsets L_1.
    C_k = φ;
    for ( each k items l_1, l_2, ... l_k ∈ L_1 and
          l_1, l_2, ... l_k are from different dimension ){
          c = l_1 ⋈ l_2 ... ⋈ l_k
          add c to C_k
    }
    return C_k
```

□

Example 4.2.5 Now, let us use the same example as Example 4.2.4 to show how this method works and how these two algorithms differ from each other.

- First, we iterate each dimension to generate candidate 1-itemsets C_1. For each itemset in C_1, we obtain the frequency from the 1-D cube space and compare it with min_supp. Then, the frequent itemsets will be decided. This step is the same as Example 4.2.4. The result is shown in Figure 4.5.

- The generation of frequent 2-itemsets L_2 is also exactly the same as in Example 4.2.4. The candidate 2-itemsets C_2 are formed by joining every $L_{1,(A_i)} \bowtie L_{1,(A_j)}$. The frequency of each itemset in C_2 is from the *count* value saved in the corresponding cell in the 2-D cube space. The results are also shown in Figure 4.5.

- Next, we first get the candidate 3-itemsets. It is the join of every three frequent 1-itemsets in L_1 with different dimensions. For example, we join the item {*"Seattle"*} from *Location* dimension, {*"Sport Wear"*} from *Product* dimension and {*"good"*} from *Profit* dimension to form a candidate 3-itemset {*"Seattle", "Sport Wear", "good"*}. In such a way, compared with 2 in Example 4.2.4, there are totally 8 itemsets are generated for C_3, shown in Figure 4.5.

- We can get the frequency for each itemset in C_3 and then generate frequent 3-itemsets L_3 by scanning the 3-D cube space in the same way in Example 4.2.4, the result is shown in Figure 4.5.

□

Discussion In this subsection, the problem of finding inter-dimensional association rules with OLAP technologies is discussed. Two cube-based algorithms are presented. The first one is based on the Apriori property. It generates candidate k-itemsets C_k by a join-and-prune procedure, that is, joining frequent $(k-1)$-itemsets L_{k-1} and pruning those containing $(k-1)$-subsets which are not frequent. The second method does not use the Apriori property, C_k is generated by joining frequent 1-itemset. Now, we give a comparison between these two algorithms based on the number of *tracking back* in pruning and the number of scanned cube cells in computing the frequency (see the discussion for Algorithm 4.2.2).

The algorithm 4.2.3 completely follows Apriori property, therefore, based on the discussion for algorithm 4.2.2, the total number of *tracking back* is

$$\sum_{k=2}^{n}(|L_{k-1} \bowtie L_{k-1}| \times (k-2)) \tag{4.4}$$

Since for each candidate inter-dimensional itemset, only one cell is checked to obtain the frequency. So, the number of scanned cube cells is

$$\sum_{k=1}^{n}(|C_k|) \tag{4.5}$$

The Algorithm 4.2.4 does not use Apriori property. The candidate k-itemsets are generated from frequent 1-itemsets directly. So, there is no *tracking back* procedure. Same as the first method, one cell is scanned for each candidate itemset, so, the number of scanned cube cell can be computed by the following Equation 4.6, where

n is the number of dimensions, and $|L_j|$ is the number of frequent 1-items in the j_th dimension.

$$\sum_{k=1}^{n}(|C_k|), |C_k| = \binom{n}{k} \prod(|L_{1,j}|),\ j\ is\ from\ the\ chosen\ k\ dimensions \qquad (4.6)$$

From the above discussion, we can see that if the data cube is sparse, only a small number of itemsets can pass over the frequency threshold, especially when the size of itemset grows. In this case, the algorithm with the Apriori property gains more benefit than the other by pruning candidate itemsets and thus reducing the frequency computation for those pruned itemsets. Furthermore, the more number of dimensions and the more itemsets are generated by joining frequent 1-itemset L_1, the more work that needs to be done for the frequency computation by Algorithm 4.2.4. On the other hand, if the cube is very dense, we cannot prune many candidate itemsets based on the Apriori property by tracking back the subsets of each itemset. In this case, the algorithm without Apriori property is better since it does not do any *tracking back*. Next, we will do the experiment analysis by using some real data to show this difference between these two algorithms.

□

Performance Analysis

In this subsection, we will present the results of some experiments conducted to analyze the performance of the two algorithms presented above with respect to various factors such as cube size, density of data, number of dimensions, etc.

The experiments were done on a Pentium Pro 200 with 64MB of memory running Windows NT. The testing data are two real databases, one is with five attributes and 67475 tuples and the other with four attributes and $287,280$ tuples. For each experiment, we pick up some attributes as dimensions from one of the two databases to construct a data cube, then the two algorithms for generating frequent inter-dimensional itemsets are run on this cube.

Experiment 4.1 *Sparse Cube Scale-up*

First, the effects of sparse date cube for these two algorithms are examined. We select 2, 3, 4 dimensions respectively and construct three data cubes. Suppose n is

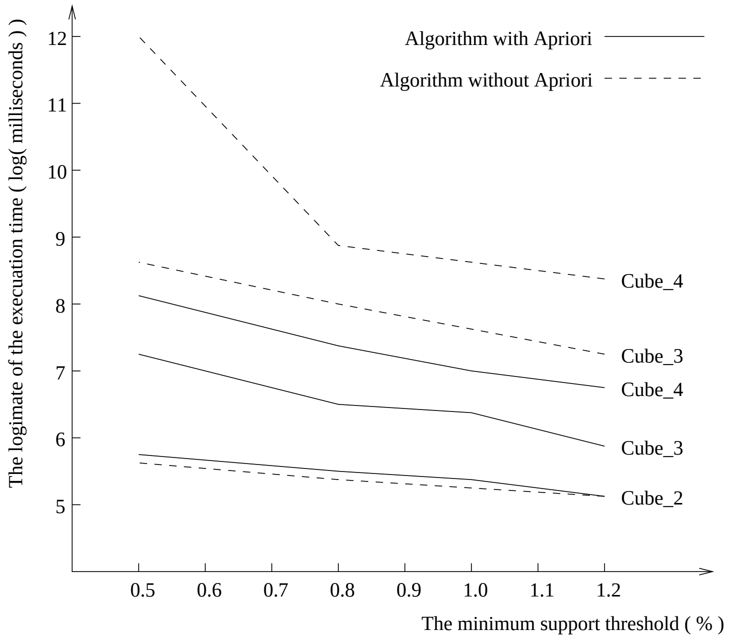


Figure 4.6: Performance comparison with sparse cube.

the number of dimensions in the cube, and d_i is the i-th dimension, $|d_i|$ is the size of dimension d_i, i.e., the number of distinct value in dimension d_i, then the cube size and cube density can be computed following the equations:

$$\begin{aligned} Cube\ Size &= \prod_{n}^{i=1} |d_i| \\ Cube\ Density &= number\ of\ non-emptycells/Cube\ Size \end{aligned} \tag{4.7}$$

According to the above equation, the information of all three cubes is listed in the following table:

Cube	Number of Dimensions	Cube Size	Cube Density
$Cube_2$	2	35636	3.4×10^{-2}
$Cube_3$	3	1,496,712	8.7×10^{-4}
$Cube_4$	4	41,907,936	7×10^{-5}

Table 4.4: All dimensions in the cube for Experiment 4.1

According to the information above, we can see that all the three cubes are sparse. Now, we run the two algorithms on each sparse cube using 4 different minimum support 0.5%, 0.8%, 1.0% and 1.2%. Figure 4.6 shows the resulting execution time.

Analysis As indicated in the graph, the performance of the two-dimensional cube $Cube_2$ is almost the same for these two algorithms with different thresholds. The reason is that for two-dimensional cube, at most frequent 2-itemset needed to be generated, there is no further need to generate candidate 3-itemset, etc. In the two algorithms, the steps from beginning to generating frequent 2-itemsets are the same, i.e., generating candidate 2-itemsets from joining frequent 1- itemset and then performing the frequency computation. So, the two curves of $Cube_2$ are almost overlapped.

For the three-dimensional cube $Cube_3$ and four-dimensional cube $Cube_4$, there are distinct differences between the two algorithms. From the two pairs of curves in the figure, we can see that Algorithm 4.2.3 with the Apriori property is always better than Algorithm 4.2.4 without the Apriori property. This is exactly what we concluded in the previous discussion. When data cube is very sparse, using Apriori property can prune many candidate itemsets, thus reducing the further support computation. While on the other hand, there are lots of combinations within frequent 1-itemset when generating candidate k-itemsets without the Apriori property, so the frequency computation will expode. In the experiment, we gathered all the information about the number of candidate k-itemsets and list the part for cube $Cube_3$ and $min\_supp =$ 0.8% in Table 4.5. We can see that, the size of candidate 3-itemset without Apriori property is almost 50 times the size with Apriori optimization.

Cube	Algorithm	Candidate 1-itemset	Candidate 2-itemsets	Candidate 3-itemsets	Execuation Time(sec)
$Cube_3$	without Apriori	702	2,470	20,832	5.638
$Cube_3$	with Apriori	702	2,470	462	1.362

Table 4.5: The size of candidate itemsets for inter-dimensional association mining algorithms, min_supp = *0.8%*

Another phenomenon can be seen from the curves: the difference between these two algorithms for $Cube_4$ is bigger than the difference for $Cube_3$. For a sparse cube, if we use the Apriori property, then as k becomes larger, the number of frequent $(k-1)$-itemsets will become smaller, reducing the size of candidate k-itemsets. But for the algorithm without the Apriori property, the number of candidate itemsets depends on the combination within frequent 1-itemsets, so, the larger k is, the bigger the number of candidate k-itemsets. As a result, the more dimensions involved in the association mining, the more benifit the algorithm with Apriori property gains.

Also, we can see that the difference between these two algorithms depends on the *min_supp* threshold as well. When the threshold is smaller, the difference will be greater. This can be easily understood since more 1-itemsets will be frequent when the threshold becomes smaller. Then the possible combinations will increase very tremendously if without any Apriori optimization, thus making the number of candidate itemsets very huge. □

Experiment 4.2 *Dense Cube Scale_up*

The above experiment is the performance analysis of these two algorithms on sparse cubes. Next, we will examine the effects of dense date cube on these two algorithms. Similarly, three data cubes are constructed with 2, 3 and 4 dimensions respectively. The following table is the information of all cubes in this experiment:

Cube	Number of Dimensions	Cube Size	Cube Density
$Cube_2$	2	126	0.81
$Cube_3$	3	41,040	0.42
$Cube_4$	4	287,280	0.16

Table 4.6: Cubes information in Experiment 4.2

According to the information above, we can see that all the three cubes are very dense cubes. We run the two algorithms on these dense cubes with different minimum support: 1.5%, 1.2%, 1.0%, 0.8% and 0.5%. Figure 4.7 shows the resulting execution time.

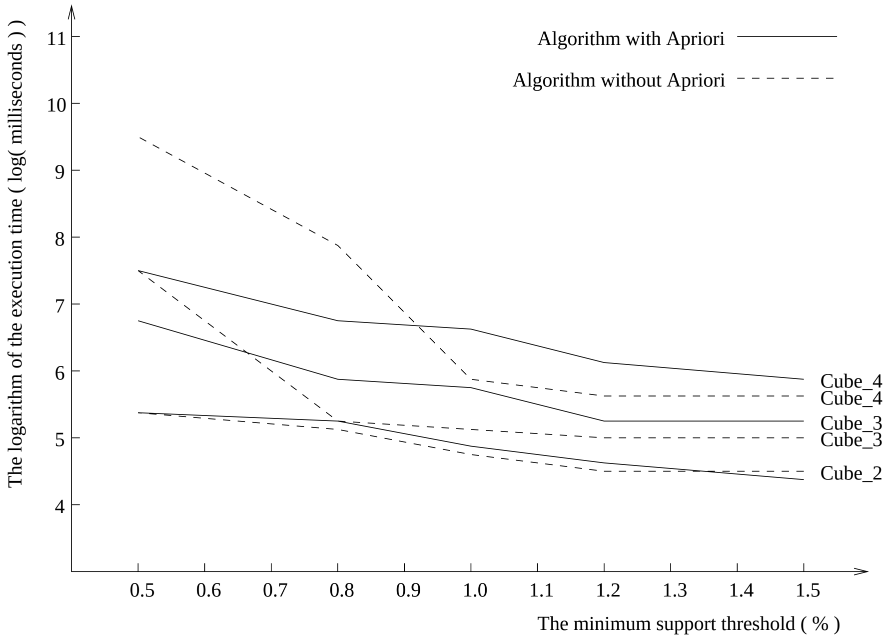


Figure 4.7: Performance comparison with dense cube.

Analysis As indicated in the graph, the performance of the two-dimensional cube $Cube_2$ is also the same with different thresholds for these two algorithms. The reason is discussed in Experiment 4.1: the steps from the beginning to the generating frequent 2-itemsets are the same with generating candidate 2-itemsets by joining frequent 1-itemsets.

For the three-dimensional cube $Cube_3$ and four-dimensional cube $Cube_4$, we notice that when threshold is relatively high, Algorithm 4.2.4 without the Apriori property is better the the one with the Apriori property. we also gathered all the information about the number of candidate k-itemsets and list part of it for cube $Cube_4$ and $min\_supp = 1.0\%$ in the following table:

From this table, we can see that after generating frequent 1-itemset, since the cube is very dense, it is very likely that the subsets of candidate k-itemsets ($k > 2$) generated after this step are all frequent. In this case, using the Apriori property cannot prune many candidate itemsets, thus further support computation can not be saved while we have to spend much time in the *tracking back* procedure.

Cube	Algorithm	Candidate 1-itemset	Candidate 2-itemsets	Candidate 3-itemsets	Candidate 4-itemsets	Execuation Time(sec)
$Cube_4$	without Apriori	595	171	306	0	0.351
$Cube_4$	with Apriori	595	171	249	11	0.711

Table 4.7: The size of candidate itemsets for inter-dimensional association mining algorithms, *min_supp = 1.0%*

On the other hand, when the threshold is relatively small, Algorithm 4.2.3 with the Apriori property has a better performance than the other one. Also, first, let us have a look at with the information about candidate itemsets on $Cube_4$ and *min_supp = 0.5%* in the following table.

Cube	Algorithm	Candidate 1-itemset	Candidate 2-itemsets	Candidate 3-itemsets	Candidate 4-itemsets	Execuation Time(sec)
$Cube_4$	without Apriori	595	2641	16551	29070	13.109
$Cube_4$	with Apriori	595	2641	630	11	1.743

Table 4.8: The size of candidate itemset for inter-dimensional association mining algorithms, *min_supp = 0.5%*

From the table, we can find that when the threshold is smaller, there are more frequent 1-itemsets, and the combination within frequent 1-itemsets, i.e., the size of candidate k-itemset ($k > 2$), is much larger. The number of candidate 3-itemset is 26 times difference between the two algorithms, while the number of candidate 4-itemsets is almost 270 times. In such a case, the *tracking back* time becomes trivial compared with the reduced frequency computation with the Apriori property.

From the curves, we also notice that the difference between these two algorithms for $Cube_4$ is bigger than that for $Cube_3$. This is also discussed in Experiment 4.1.

□

4.2.4 Hybrid association mining

Up to now, we discuss two simple cases of association rules: intra-dimensional associations and inter-dimensional associations. From the view of dimension, the former is also called single-dimensional association mining as we can see that all predicates from the rule are from the same dimension, while the latter is called multi-dimensional association mining. But here, what we discussed only covers part of multi-dimensional association mining, i.e., multi-dimensional without petition since we can see that all the predicates in the rule are not repetitive. Can we do more general multi-dimensional association mining, i.e., the desired rule includes not only multiple dimensions, but also some repetitive predicates from one dimension? The answer is yes. In this subsection, we will have some discussions on this problem and give a simple algorithm.

Actually, if we take the dimension with repetitive predicates as the intra-dimension, and the other dimensions as the inter-dimension, the case of multi-dimension with repetition is the combination of intra-dimension and inter-dimension. In this case, each candidate itemset L can be written as $L = \{L_{inter}, L_{intra}\}$, where L_{inter} are the items from inter-dimensions, whereas L_{intra} are those from intra-dimension. We can easily see that if either L_{inter} or L_{intra} are not a frequent itemset, then $L = \{L_{inter}, L_{intra}\}$ is not a frequent itemset as well. From this observation, we find it natural and easier to separate the whole procedure into two sub-procedures: instead of finding frequent itemsets among all the dimensions, we first find frequent itemsets from intra-dimension, and inter-dimensions respectively, and then merge the two parts together to abtain the frequent hybrid itemsets.

The detailed algorithm is depicted in Algorithm 4.2.5. Part1 (Step 2) generates all frequent intra-itemsets L_{intra}. Part2 is an iterative procedure: first generates frequent k-inter-itemsets $L_{inter,k}$, and then merges $L_{inter,k}$ with L_{intra} to generate the candidate hybrid itemsets, finally all those candidate itemsets satisfying minimum support are added to the set of frequent itemsets, L.

Algorithm 4.2.5 Generating hybrid frequent itemsets among n dimensions in a data cube C.

Input:

- An $n+2$-dimensional data cube: $C[d_1, \ldots, d_n+2]$,
- a transaction dimension d_t and an item dimension d_{item} (intra-dimension)
- other n associated inter-dimensions
- the minimum support threshold: min_sup.

Output: The set of frequent itemsets L.

Method

```
1. L = φ;
2. Generate the frequent intra-itemsets L_intra by Apriori property;
3. if L_intra is empty then exit.
4. L = L ∪ L_intra;
5. k = 1;
   While(1) { % iteration for size of inter-itemset
      Generate the frequent k-inter-itemsets L_inter,k;
      if L_inter,k = φ then exit;
      l = 1; % do merging iterately for size of intra-itemset
      Repeat
           l = l + 1;
           C_l+k = Merge L_intra,l and L_inter,k;
           % generate the candidate (l + k)-hybrid-itemsets C_l+k
           generate frequent itemsets L_l+k from C_l+k;
           L = L ∪ L_l+k;
           prune any itemset in L_intra,l and its extensions
           if it is not in L_l+k;
           prune any itemset in L_inter,k if it does not appear in L_l+k;
      Until L_l+k = φ or L_intra = φ;
      k = k + 1;
   }
```

□

4.3 Generating strong association rules from frequent itemsets

After generating the frequent itemsets, we reach the final step of association mining, i.e., generating strong association rules from the frequent itemsets. This step is quite easy and same for different association mining, including inter-dimensional, intra-dimensional and hybrid association mining.

According to the definitions in Chapter 3, strong association rules are those rules with confidence above the minimum confidence threshold. The confidence is computed using the following equation:

$$confidence(A \rightarrow B) = support(A \cup B)/support(A) \tag{4.8}$$

Based on this equation, a simple and straightforward algorithm for this step is described as follows.

Algorithm 4.3.1 *Generating strong association rules from the given frequent itemsets.*

Input:

1. Frequent itemsets, L.
2. minimum confidence threshold, min_conf.

Output: Strong association rules R.

Method

```
R = φ;
for each frequent itemset I in L {
   for each non-empty subset S of I {
        conf(S → I − S) = support(I)/support(S);
        if conf ≥ min_conf % generate an interesting rule
             rule r = ''s → (I − s)'';
```

$$R = R \cup \{r\};$$

}

}

□

Every rule so generated automatically satisfies the minimum support requirement because the rules are generated from the frequent itemsets. All the frequent itemsets are stored along with their frequency so that they can be accessed quickly.

Example 4.3.1 Since the algorithm 4.3.1 is just applying the same procedure for every frequent itemset, we only need to take an itemset as an example to demonstrate how this algorithm works step by step. Based on Example 4.2.2, such an itemset, I = {"Carry Bags", "Sport Wear", "Water Purifier"}, is generated as a frequent itemset. What are the association rules that can be generated from I? The non-empty subsets of I are $\{``CarryBags", ``SportWear"\}$, {"Carry Bags", "Water Purifier"}, {"Sport Wear", "Water Purifier"}, {"Carry Bags"}, {"Sport Wear"}, and {"Water Purifier"}. The resulting association rules are as shown below, each being listed with its confidence, where $\forall X \in Location$:

$$\begin{aligned}
&``CarryBags" \wedge ``SportWear" \rightarrow ``WaterPurifier" \quad conf = 2/2 = 100\% \\
&``CarryBags" \wedge ``WaterPurifier" \rightarrow ``SportWear" \quad conf = 2/2 = 100\% \\
&``SportWear" \wedge ``WaterPurifier" \rightarrow ``CarryBags" \quad conf = 2/3 = 67\% \\
&``CarryBags" \rightarrow ``SportWear" \wedge ``WaterPurifier" \quad conf = 2/3 = 67\% \\
&``SportWear" \rightarrow ``CarryBags" \wedge ``WaterPurifier" \quad conf = 2/3 = 67\% \\
&``WaterPurifier" \rightarrow ``CarryBags" \wedge ``SportWear" \quad conf = 2/3 = 67\% \quad (4.9)
\end{aligned}$$

If the minimum confidence threshold is, say, 70%, only the first and second rules above are strong since these are the only ones with confidence greater than 70%.

In the same way, the following strong rules can be generated from other frequent itemsets generated in Example 4.2.2:

$$``AlertDevices" \rightarrow ``CarryBags" \quad conf = 2/2 = 100\%$$

$$
\begin{aligned}
\text{“}SportWear\text{”} &\rightarrow \text{“}WaterPurifier\text{”} \quad conf = 3/3 = 100\% \\
\text{“}WaterPurifier\text{”} &\rightarrow \text{“}SportWear\text{”} \quad conf = 3/3 = 100\%
\end{aligned} \tag{4.10}
$$

4.4 Mining multi-level association rules

In the previous sections, algorithms were studied for mining association rules at a single level of abstraction, i.e., each dimension in a data cube is examined at a fixed level of abstraction. However, in many applications, it is difficult to find strong associations among data at the primitive levels of abstraction due to the sparsity of data in multi-dimensional space. On the other hand, many strong associations discovered at rather high concept levels are common sense knowledge. Therefore, it is desirable to mine associations at multiple levels of abstraction.

Based on our discussion in the previous Chapters, many data warehouse systems have implemented efficiently interactive drilling, slicing, dicing and pivoting operations in data cubes and taken them as primitive OLAP operations. This data cube structure and its associated OLAP operations provide a convenient base for interactively mining associations at multiple levels of abstraction. Hence, the basic method of such a process is that first we drill-down or roll-up for any chosen dimension along the concept hierarchy to the desire level, and then find association rules at the new levels of abstraction using one of the mining algorithms. An example of such an OLAP operation is shown in the following example.

Example 4.4.1 Based on the cube in Figure 4.1. Suppose the hierarchy for dimension *Location* is in Figure 4.8. Now, we want to perform inter-dimensional association mining with dimension *Location* at level *region* instead of level *city*.

Given this mining task, we first generate the corresponding working cube, shown in Figure 4.9 by rolling-up the base cube in Figure 4.1 along dimension *Location*. Then next, Algorithm 4.2.3 for inter-dimensional association mining is run on this working cube.

□

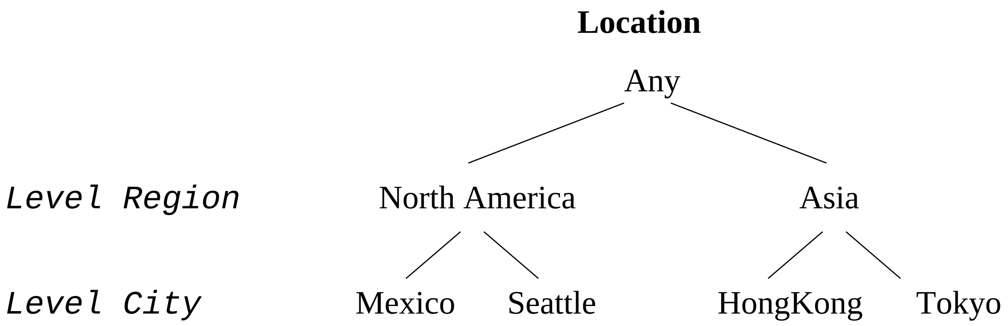


Figure 4.8: The hierarchy for *Location* dimension.

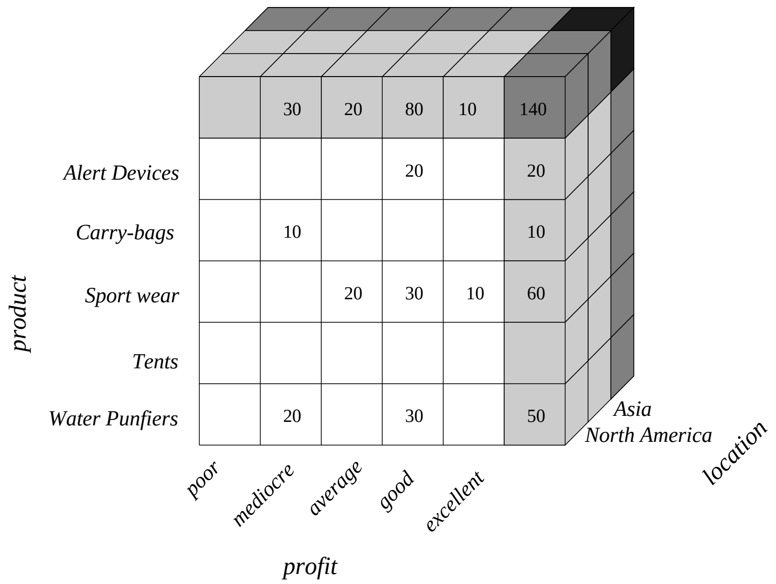


Figure 4.9: The cube generated from rolling up.

Discussion Different strategies of setting minimum support thresholds at different levels of abstraction can be used, depending on whether a threshold is to be changed at different levels:

1. independent min_support across levels, where there is no relationship of min_support among different levels of abstraction;

2. same min_support across levels, where the same min_support is used at different levels of abstraction;

3. reduced min_support across levels, where lower level of abstraction uses smaller min_support in the mining.

Let us compare these approaches. It is simple for users or experts to provide a single uniform minimum support threshold for association rules mining. Also, it is simple to execute a mining algorithm using a fixed minimum support threshold. However, since it is unlikely that the itemsets at a lower level of abstraction will occur as frequent as those at a higher level of abstraction, a fixed minimum support threshold may cause some problems. If it is set too high, it may miss a lot of meaningful associations occurring at low levels; however, if it set too low, it may generate many uninteresting associations occurring at high levels. Therefore, in comparison between the two approaches, we recommend to use a reduced support threshold to progressively lower down the minimum support threshold when mining touches deeper levels of abstraction since one would like to generate sufficient meaningful associations at low levels but keeps the rules to be generated at high levels sufficiently interesting.

4.5 Interestingness of association rules

So far, we have discussed the algorithms from generating frequent itemsets to finally discovering rules. During our experiments with real data, sometimes, more than a thousand rules can be found even from a small size of data. Are all the rules discovered interesting enough to be presented to the user? Not necessarily. Whether a rule is interesting or not can be judged either subjectively or objectively. Ultimately, only the user can judge if a given rule is interesting or not, and this judgement, being subjective, may differ from one user to another. However, objective interestingness criterion, based on the statistics "behind"the data, can be used as one step towards the goal of weeding out uninteresting rules from presentation to the user. How can we tell which rules are really interesting?

Example 4.5.1 Take a rule derived in Example 4.2.2:

$$\text{“}CarryBags\text{”} \;\rightarrow\; \text{“}SportWear\text{”} \tag{4.11}$$

the support of this rule is 50%, confidence is 66.7. We can say that rule 4.11 is a strong association rule based on the support-confidence framework. However, this rule is incomplete and misleading since the overall support of "Carry Bags"is 75%, even greater than 66.7%. In other words, a customer who buys "Carry Bags"is less likely to buy "Sport Wear"than a customer about whom we have no information. The truth here is that there is a negative dependence between buying "Carry Bags"and buying "Sport Wear". □

To help filter out such misleading strong association rules $A \Rightarrow B$, we need to study how the two events A and B are correlated. We want to find the dependence of a given rule in order to give a more precise characterization of the rule.

Definition 4.1 We define the interestingness of events x, y to be

$$I(x, y) = \frac{p(x, y)}{p(x)p(y)} \tag{4.12}$$

where $p(x)$ is the possibility of event x.

The fact that the interestingness of events x and y is less than 1 indicates negativly correlation, since the nominator is the actually likelyhood of both events happen, and the denominator is what the likelyhood would have been in the case when the two activities are independent. In the above example, we can see that the interestingness of "Carry Bags", "Sport Wear"is:

$$\begin{aligned} p(\text{“}CarryBags\text{”}, \text{“}SportWear\text{”})/p(\text{“}CarryBags\text{”}) \times p(\text{“}SportWear\text{”}) \\ = 0.5/0.75 \times 0.75 = 0.89 < 1 \end{aligned} \tag{4.13}$$

It means that buying “Carry Bags”is negative associated with buying “Sport Wear”, so this rule is not interesting enough to be reported.

Combining the computation into association mining and returning those rules having positive dependence will reduce the size of results and make those generated rules more precise. We notice that $p(x)$ is the support of item x in term of association rules. Therefore, we have the following definition:

Definition 4.2 We define the interestingness of a rule $A \Rightarrow B$ to be

$$I(A \Rightarrow B) = \frac{support(A, B)}{support(A)support(B)} \tag{4.14}$$

Next the algorithm to generate *interesting and strong* rules is in Algorithm 4.5.1. Basically, one step(step 4) is added after we generate all strong association rules by the method we discussed in the previous sections. This step is to compute the interestingness of each rule and filter out those rules whose interestingness is less than 1.

Algorithm 4.5.1 *Generate interesting and strong association rules*

Input:

- A mining task and related working data cube C
- the minimum support and confidence threshold: min_sup, min_conf

Output: All interesting and strong association rules R.

Method

1. `Generate the frequent itemsets L using alogorithm 4.2.2, or 4.2.3, etc.`
2. `Generate all strong association rules` R_c `satisfying with` min_conf `threshold using algorithm 4.3.1;`
3. $R = \phi$;

```
4. for each rule r(A → B) ∈ R_c {
       interestingness = I(A ⇒ B) = support(A,B) / (support(A)support(B));
       if interestingness ≥ 1 then % this rule is interesting
           R = R ∪ {r};
   }
```

$\square$

Example 4.5.2 Let us continue example 4.3.1. Now, we want to generate all interesting rules from those generated strong rules. After interestingness computation for each rule based on Equation 4.14. Only the following three rules are interesting enough to be further presented to the users.

$$\text{“}CarryBags\text{”} \wedge \text{“}SportWear\text{”} \rightarrow \text{“}WaterPurifier\text{”} \qquad interestingness = 133\%$$

$$\text{“}CarryBags\text{”} \wedge \text{“}WaterPurifier\text{”} \rightarrow \text{“}SportWear\text{”} \qquad interestingness = 133\%$$

$$\text{“}WaterPurifier\text{”} \rightarrow \text{“}SportWear\text{”} \qquad interestingness = 133\%$$

$\square$

Chapter 5

Constraint-based Association Mining

In the last chapter, the problem of mining association rules has been discussed for several cases. From the view of interface, the process of association mining can be summarized as three steps. First, a user specifies the part of the database to be mined. Next, the user specifies minimum thresholds such as support and confidence. Then the system executes one of the mining algorithms. At the end of a highly intensive data processing, a large number of associations are returned, some of which are hopefully what the user is looking for.

But in practice, often times, users may be only interested in a subset of associations, for instance, those containing one user-defined item, like $\{Location = \text{“}Paris\text{”}\}$. It is desirable for an association mining system to have the ability to let users specify various constraints during the mining. In this chapter, we will focus on this topic, referred as *constraint-based association mining* [31]. The main task is that given a set of constraints C, we seek to find algorithms that are both sound and complete. Algorithm is sound means that it finds only the frequent itemsets that satisfy the given constraints, while it is complete means that all frequent itemsets satisfying given constraints are found.

5.1 Apriori+

A simple way that we can put forward right away is to apply one of the algorithms discussed in the last chapter for generating all rules satisfying the min_sup and $min\_confidence$ thresholds, and then we add one more step to select the rules generated. That is, we take those rules as candidates and check each of them against the constraints. Since this is a simple extension to the Apriori algorithm [4], we call this algorithm *Apriori+*. The details are described in Algorithm 5.1.1

Algorithm 5.1.1 (Apriori+) A simple Apriori-like algorithm for constraint-based association mining

Input:

- A task_relevant data cube
- thresholds: min_sup, min_conf
- a set of constraints C on rules

Output: The association rules R satisfying thresholds and constraints.

Method:

1. `Generate all of the frequent itemsets` L `satisfying` min_sup `threshold. According to rule type to be mined, one of the algorithms, 4.2.2, 4.2.3, 4.2.4 or 4.2.5, is selected to perform at this step.`
2. `Generate all association rules` R_c `satisfying the` min_conf `threshold by applying Algorithm 4.3.1 on` L.
3. `Generate the desired rules` R `satisfying constraints` C.
   R `= rules_pruning(` R_c, C `).`

`Function rules_pruning(`R_c, C`)`

`% Return those rules in` R_c `satisfying constraints` C

```
for each rule r ∈ R_c do {
     if r does not satisfy C{
          Prune r from R_c
     }
}
return R_c
```

□

Example 5.1.1 We continue examining example 4.2.4. Suppose for a local store in "Seattle", they are only interested in the rules related with $\{Location = \text{"Seattle"}\}$ and the confidence threshold is 70%. The mining procedure is basically three steps. First, we generate the frequent itemsets in the exact same method as used in example 4.2.4, the result is in Figure 4.4. Next, by performing Algorithm 4.2.3, five strong association rules are generated as follows:

$$\begin{aligned} Profit(X, \text{"poor"}) &\rightarrow Location(X, \text{"Mexico"}) \\ Location(X, \text{"Seattle"}) &\rightarrow Profit(X, \text{"good"}) \\ Location(X, \text{"Seattle"}), Product(X, \text{"CarryBags"}) &\rightarrow Profit(X, \text{"good"}) \\ Location(X, \text{"Seattle"}), Profit(X, \text{"good"}) &\rightarrow Product(X, \text{"CarryBags"}) \\ Profit(X, \text{"good"}), Product(X, \text{"CarryBags"}) &\rightarrow Location(X, \text{"Seattle"}) \end{aligned}$$

Finally, for each association rule, we check whether it has the item $\{Location = \text{"Seattle"}\}$. We find only the last four rules are what we are interested in.

Apriori+ is simple, but it can be inefficient. For example, suppose we change the constraint to $\{Location = \text{"HongKong"}\}$ in the above example. We notice that after getting the frequent 1-itemsets from the *Location* dimension, as shown in Figure 4.4, there is no item satisfying the constraint. Obviously, all the rules derived further will violate the constraint. Therefore, from the beginning, we can conclude there is no association rule that can be generated. But using the Apriori+ algorithm, we do not do the constraint checking until we generate all the rules satisfying the thresholds. This obviously involves a lot of unnecessary processing.

The key technical challenge is that how to guarantee a level of performance commensurate with the selectivity of constraints. From the common sense, to test all the candidate itemsets for constraint satisfaction before generating association rules is the most efficient way. But this approach is not even guaranteed to be sound. Consider the intra-dimensional mining and the constraint that item $\{Product = \text{"}Carry\ Bags\text{"}\}$ *must be in rule*, if we apply this constraint in getting frequent 1-itemsets, we can only get one frequent item, i.e., "Carry Bag". So, we will miss all the other frequent 1-itemsets and rules. We can easily see that this way is not right. In this chapter, we will perform some analysis on the properties of constraints and try to push them as deeply as possible inside the frequent itemsets computation.

In this chapter, two kinds of constraints are discussed, i.e., *anti-monotone* constraint and *pre-prunable constraint* .

5.2 Anti-monotone constraints

The first property of constraints that we analyze is *anti-monotonicity*. The simple description of it is that if a set S violates the constraint, any superset of S violates the constraint as well. The formal definition is as follows:

Definition 5.1 A constraint c is *anti-monotone* iff for any set S,

$$S\ does\ not\ satisfy\ C \Rightarrow \forall S' \supseteq S,\ S'\ does\ not\ satisfy\ C. \tag{5.1}$$

Here, there are some examples.

Example 5.2.1 Constraint $\sum(Price) \leq 100$ on intra-dimensional itemset is anti-monotone.

Proof Given an intra-dimensional itemset I, if I does not satisfy the constraint, it means that $\sum price(a_i), a_i \in I > 100$. For any superset of I, I', obviously, $\sum price(a_j), a_j \in I' > \sum price(a_i), a_i \in I$. Thus, $\sum price(a_j), a_j \in I' > 100$.

□

Similarly, we can present and proof a lot of such anti-monotone constraints. Next let us have a look at the following anti-monotone constraint:

Example 5.2.2 For all kinds of associations, constraint "$support(itemset) \leq min\_sup$" is an anti-monotone constraint.

We can easily see that this constraint is the minimum support constraint (also called frequency constraint). That is, if an itemset violates the minimum support threshold, so does any of its supersets. From the discussion in last chapter, we can see that this property enables the Apriori algorithm to prune away a significant number of candidate itemsets that require support computation. Motivated by the fact that the minimum support constraint is anti-monotone, we propose a question that whether we can do pruning on other anti-monotone constraints similar to Arprior algorithm. The answer is yes. First, let us give the following lemma.

$$
\begin{aligned}
&S,\ where\ |S| = k,\ is\ frequent \Rightarrow \\
&\quad \forall S' \in S,\ where\ |S'| = k-1, S'\ is\ frequent. \qquad (5.2)\\
&S,\ where\ |S| = k,\ satisfies\ C_{am} \Rightarrow \\
&\quad \forall\ S' \in S,\ where\ |S'| = k-1,\ S'\ satisfies\ C_{am}. \qquad (5.3)
\end{aligned}
$$

From the above lemma, we can conclude that, in the similar way, other anti-monotone constraints can be incorporated into the mining algorithm with the same efficiency as the frequency constraint in the Apriori Algorithm. More specifically, if L_{k-1} consists of all the frequent $k-1$-itemsets that satisfy C_{am}, then the candidate k-itemsets, C_k, can be generated in exactly the same way as is done in the Apriori algorithm. Furthermore, C_k can be further pruned by checking whether each itemset in C_k satisfies every constraint in C_{am} other than the frequency constraint. Any element that violates any constraint in C_{am} is not needed for support computation.

Next we lay out the new algorithm for association mining with anti-monotone constraints. The whole algorithm is divided into two parts. The first part is to generate frequent itemsets which satisfy constraints as well, and the second part is to generate association rules satisfying min_conf threshold. Basically there are three sub-steps in generating frequent k-itemsets L_k. First, candidate k-itemsets C_k are generated, either from L_{k-1} according to Apriori property, or from the data directly (for C_1). The procedure for generating C_k from L_{k-1} is the same as the method we discussed in the last chapter, therefore, we do not discuss the details here. After generating C_k, next we prune it by checking constraints using procedure *do_pruning*, which simply checks each itemset to see whether it complies with the constraints or not. Finally, the frequent k-itemsets L_k is generated by support computation. This sub-step is also the same as what we did for the algorithms in the last chapter.

Algorithm 5.2.1 *Improved mining method with anti-monotone constraints.*

Input:

- A task-relevant data cube
- Thresholds: min_sup and min_conf
- a set of anti-monotone constraints C_{am}

Output: The association rules R satisfying thresholds and constraints C_{am}.

Method:

```
1. k = 1; L = φ;
2. Collect the candidate 1-itemsets C_1;
3. Prune C_1 based on constraints C_am, do_pruning(C_1, C_am);
4. Generate the frequent 1-itemsets L_1 = gen_frequent(1, C_1);
5. Repeat
       k = k + 1;
       Generate candidate k-itemsets C_k from L_{k-1};
       Prune C_k based on constraints C_am, do_pruning(C_k, C_am);
       Generate frequent k-itemset L_k = gen_frequent(k, C_k);
       L = L ∪ L_k;
   Until L_k is empty;
6. Generating association rules R on the frequent itemsets L;
```

```
Function do_pruning(C_k, C_am)
% Prune away those itemsets in C_k which do not satisfy
%constraints C_am
   for each itemset I ∈ C_k do {
        if I does not satisfy C_am{
             Prune I from C_k
        }
   }

Function gen_frequent(k, C_k)
% generate frequent k-itemset L_k from candidate C_k
   L_k = φ;
   for each itemset I ∈ C {
        compute the support for I
        if support ≥ min_supp then
             L_k = L_k ∪ {I};
   }
   return L_k;
```

Rationale Based on the algorithms we discussed in the last chapter, Algorithm 5.2.1 basically adds steps to perform constraints checking. As we can see, instead of checking the constraints after we generate all the frequent itemsets, which is used in Algorithm *Apriori+*, Algorithm 5.2.1 conducts the constraints checking on candidate itemsets before the support computation. The anti-monotone property guarantees that any k-itemsets satisfying the constraint must be derived from the pruned candidate $(k-1)$-itemsets. Thus, it can prune away a significant number of candidate sets that require support counting. □

Here is an example to show how the improvement is gained using the anti-monotone property.

product	price
Alert Devices	20
Carry Bags	65
Sport Wear	80
Tents	50
Water Purifier	15

Table 5.1: The price for each product

Example 5.2.3 We continue with the example 4.2.2. Suppose we have the *price* information for each product in Table 5.1.

Now, we want to generate association rules with minimum support threshold as 2, minimum confidence threshold as 70% and satisfying the following constraint: *the sum of the price of all the items in an itemset should be no more than 70*.

From Example 5.2.1, we know that this constraint, i.e., $\sum price \leq 70$, is anti-monotone. Then, we do association mining based on Algorithm 5.2.1 as follows:

1. First, each item (*product*) is a member of the set of candidate 1-itemsets, C_1.

2. Then, from Table 5.1, we get the price for each product, check it against the constraint, and then drop "Sport Wear"from C_1 because its price is 80, greater than 70. The results of candidate before and after constraints checking are shown in Figure 5.1.

3. Next, we form C_2 using L_1 which consists of three 2-itemsets: { *"Alert Devices", "Carry Bags"*}, { *"Alert Devices", "Water Purifier"*} and { *"Carry Bags", "Water Purifier"*}. After computing the total price for each itemset $I \in C_2$, we drop the set { *"Alert Devices", "Carry Bags"*} because its total price is 75, greater than 70. After minimum support checking, finally, we get L_2. Since there is only one set in L_2, the procedure of generating frequent itemsets finishes.

4. Now, we generate the association rules from frequent itemsets in exactly the same way as we discussed in the last chapter. Details are not shown here.

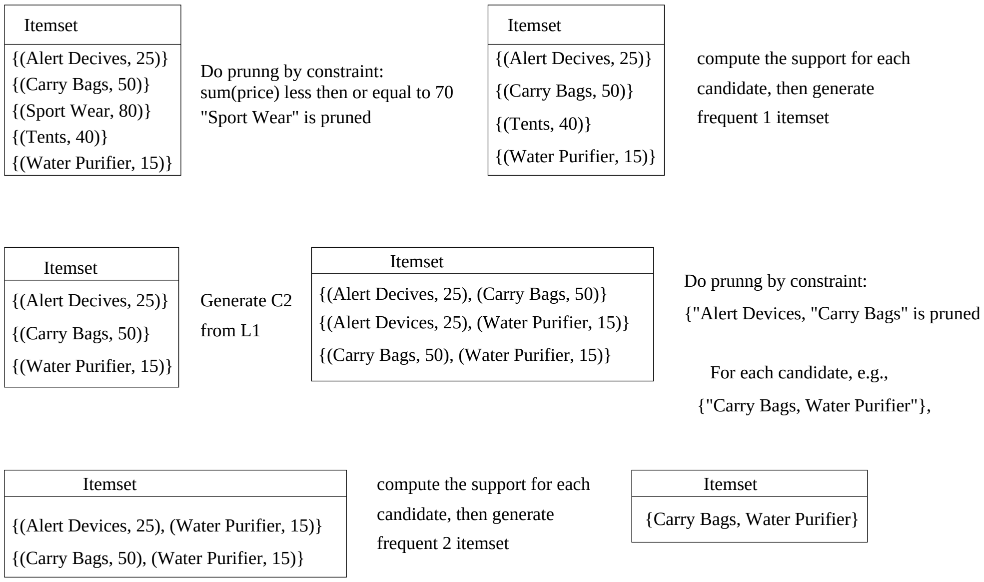


Figure 5.1: The frequent 1-itemset in Example 5.2.3, the number in each bracket is the price associated with the product

□

So far, we address the problem that how to use anti-monotone property in constraint-based association mining. The question left is that how to recognize those constraints belonging to the anti-monotone class before we can do the pruning optimization. In the following, we analyze some categories of constraints and summarize them in Table 5.2, where S is an attribute of itemset or the itemset itself, v is a single constant from dimensions of S, and V is a set of such constants. The following are some simple proofs for Table 5.2.

Proof

1. $S = v$ is anti-monotone. Given an itemset I, if I does not satisfy the constraint, it means that $\exists i \in I$, $S(i) \neq v$. For any superset of I, I', obviously, it is true that $i \in I'$. Because $S(i) \neq v$, I' does not satisfy the constraint as well.

   In the same way, we can prove that $S \geq v$ and $S \leq v$ are anti-monotone.

2. $S \subseteq V$ is anti-monotone. Given an itemset I, if I does not satisfy the constraint, it means that $\exists i \in V$, $S(i) \notin V$. For any superset of I, I', obviously, it is true

Constraint	anti-monotone
$S = v$	yes
$S \geq v$	yes
$S \leq v$	yes
$S \subseteq V$	yes
$S \supseteq V$	no
$count(S) \leq v$	yes
$count(S) \geq v$	no
$count(S) = v$	no
$min(S) \leq v$	no
$min(S) \geq v$	yes
$min(S) = v$	no
$max(S) \leq v$	yes
$max(S) \geq v$	no
$max(S) = v$	no
$sum(S) \leq v$	yes
$sum(S) \geq v$	no
$sum(S) = v$	no
minimum support constraint	yes

Table 5.2: Anti-monotone Constraint Categories

that $i \in I'$. Because $S(i) \notin V$, I' does not satisfy the constraint as well.

Nevertheless, $S \supseteq V$ is not anti-monotone. For a given itemset I, if I does not satisfy the constraint, in another word, it means that $\exists v_1, \ldots v_n \in V$, $\forall i \in I$, $S(i) \neq v_1, \ldots v_n$. But suppose $S(i_1) = v_1$, $\ldots S(i_n) = v_n$, then a superset of I, $I' = I \cup \{i_1\} \ldots \cup \{i_n\}$, obviously satisfies the constraint.

3. $count(S) \leq v$ is anti-monotone. Given an itemset I, if I does not satisfy the constraint, it means that $count(I) > v$. For any superset of I, I', obviously, it is true that $count(I') > count(I) > v$. So I' does not satisfy the constraint as well.

   Nevertheless, $count(S) \geq v$ is not anti-monotone. For a given itemset I, if I does not satisfy the constraint, in another word, it means that $count(I) < v$. If we get a superset I, I', by adding enough number of items into I to make it true that $count(I') \geq v$. Then, I' satisfies the constraint. In the same way, we can prove That $count(S) = v$ is not anti-monotone.

4. Using the same prove procedure as the above, we can easily demonstrate the left three groups of constraints. Here, we do not give more details in the sake of space.

5. The final constraint is the min_support threshold constraint. Based on the Apriori property we discussed before, we already know that this constraint is anti-monotone.

5.3 Pre-prunable constraints

From the algorithm above, we can see that the optimization achievable using anti-monotone property is restricted to iterative pruning. That is, at each iteration or level of the Apriori algorithm, we can prune the candidates which require support counting according to the satisfaction with the anti-monotone constraints. On the other hand, it also means that at each iteration, we still need to generate and test these candidates. Let us discuss an example first.

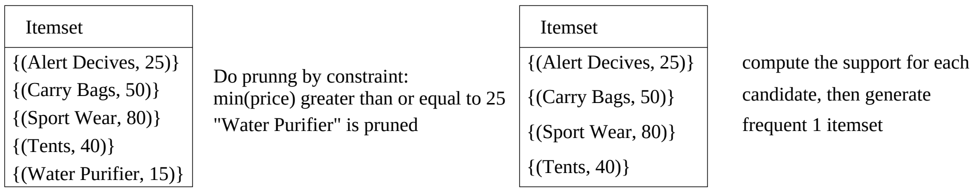


Figure 5.2: The frequent 1-itemset in Example 5.3.1

Example 5.3.1 We still take example 5.2.3. But we change the constraint to be *the minimum price of all items in a itemset should be no less than 25.*

According to Table 5.2, we know that this constraint, i.e., $min(price) \geq 25$, is anti-monotone. So, we can do optimization similar to Example 5.2.3. That is, every time we get a candidate itemset I, we check every item $\in I$, if $\exists i \in I$ that if $price(i) < 25$, then we prune it. $\square$

If we look into this example, we can see that after the constraints checking for candidate 1-itemsets C_1, which is shown in Figure 5.2, any frequent k-itemsets I' derived from C_1, are absolutely satisfying the constraints. So, any further checking is not necessary. This observation poses a question: whether there are some classes of constraints for which constraints checking can be done before any iteration takes place, i.e., for such constraints, we can generate all itemsets that satisfy the given constraints by avoiding the generate-and-test paradigm completely. The answer is yes. First, we formalize a notion of another constraint property: self-closed property.

Definition 5.2 (Self-Closed) *A constraint C is* self-closed *iff for a given k-itemset I_k, the following equation is satisfied.*

$$\forall (k-1) - itemset\ I_{k-1} \subset I,\ I_{k-1}\ satisfies\ C \Rightarrow I_k\ satisfies\ C.$$

Let us give an example of self-closed constraint.

Example 5.3.2 The constraint in Example 5.3.1, $min(price) \geq 25$, is a self-closed constraint. This is reasoned as follows. Given a k-itemset I, if $\forall (k-1) - itemset\ I_{k-1} \subseteq I$, the minimum price of all items in $I_{k-1} \geq 25$, then we can conclude that the minimum price of all items in $I \geq 25$ as well. Otherwise, if it is not true, it means that $\exists i \in I$,

$price(i) < 25$. Taking off one item i_1 from I, $i_1 \neq i$, to form a subset I' of I. Then minimum price of $I' < 25$ because $i \in I'$ and $price(i) < 25$. In another word, I' does not satisfy the constraint. This contradicts the premise.

$\square$

We should notice that although the above self-closed constraint, $min(price) \geq 25$, is also anti-monotone according to the Table 5.2, not all self-closed constraints are anti-monotone. Let us give an example.

Example 5.3.3 The constraint $min(price) \leq 25$ is self-closed, but it is not anti-monotone.

Proof From Table 5.2, we know that the constraint $min(price) \leq 25$ is not anti-monotone. So, we only need to prove that it is self-closed instead. This is very straightword. Actually, given a k-itemset I, if $\forall (k-1)_i temset\ I_{k-1} \subseteq I$, the minimum price of items in $I_{k-1} \leq 25$, based on the fact that the minimum price of items in I is less than any $I' \subseteq I$, we can conclude that the minimum price of $I \leq 25$ as well.

$\square$

From the definition of self-closed property, there is a conclusion that after pruning once by a self-closed constraint before any further computation, the satisfaction of the constraint alone is not affected in any way by the result of the iterative candidate itemsets generation and support computation. That is, after we do the pruning on the candidate 1-itemsets by a self-closed constraint before any support computation, any itemset derived from this pruned 1-itemsets will absolutely satisfy the constraint. Now the question is how to take full advantage of self-closed property. Since the approach of directly applying self-closed property can not gurantee to be sound as we discussed in the beginning of this chapter, we explore another way by combining self-closed property with anti-monotone property. First, let us define

Definition 5.3 (Pre-Prunable) *A constraint c is pre-prunable iff c is both anti-monotone and self-closed.*

Then, we can easily have the fact that: if a constraint c is pre-prunable, then for any k-itemset I,

$$I\ satisfies\ c \Longleftrightarrow \forall (k-1) - itemset\ I^{'} \subset I,\ I^{'}\ satisfies\ c \tag{5.4}$$

Actually, the left direction of the above equation is anti-monotone property, while the right direction is self-closed property. From the optimization point of view, this fact means that those constraints are pre-prunable, after we prune the candidate 1-itemsets C_1 before any support computation, if and only if those itemsets derived from C_1 will satisfy the constraints. It means we do not need to do any further pruning, or checking, on those pre-prunable constraints.

The following is the algorithm we lay out for association mining with pre-prunable constraints. First, candidate 1-itemsets C_1 are generated by selecting all the values from data directly. Then, we prune those itemsets from C_1 which do not satisfy the pre-prunable constraints and generate frequent 1-itemsets L_1 by support computation. Next, we do the iteration of generating candidate k-itemsets C_k from frequent (k-1)-itemsets L_{k-1} by Apriori property, and generating the frequent k-itemsets L_k from C_k by support computation. This procedure is the same as the method we discussed in the last chapter. We do not discuss the details here. Finally, association rules are generated from the frequent itemsets.

Algorithm 5.3.1 *Improved association algorithm with pre-prunable constraints.*

Input.

- A working data cube
- min_sup and min_conf thresholds
- a set of pre-prunable constraints C_{pp}

Output. The set of association rules R satisfying thresholds and constraints C_{pp}.

Method

1. $k = 1$; $L = \phi$;

2. Collect the candidate 1-itemsets, C_1;

3. Prune C_1 based on C_{pp}, *do_pruning*(C_1, C_{pp});

4. Generate frequent 1-itemsets L_1 = *gen_frequent*(1, C_1);

5. Repeat

    $k = k + 1$;

    Generate candidate k-itemsets C_k from L_{k-1};

    Generate frequent k-itemsets L_k = *gen_frequent*(k, C_k);

    $L = L \cup L_k$;

  Until L_k is empty;

6. Generate association rules R on the frequent itemsets L;

Function do_pruning(C_1, C_{pp})
% Prune away those 1-itemsets in C_1 which do not satisfy
% constraints C_{pp}
  for each itemset $I \in C_1$ do {
      if I does not satisfy C_{pp}{
          Prune I from C_1
      }
  }

Function gen_frequent(k, C_k)
% generate frequent k-itemset L_k from candidate k-itemsets C_k
  $L_k = \phi$;
  for each itemset $I \in C_k$ {
      compute the support for I
      if $support \geq min\_supp$ then
          $L_k = L_k \cup \{I\}$;
  }

Rationale. The basic structure of this algorithm is very similar to Algorithm 5.2.1 for anti-monotone constraints optimization. The difference between them is in the constraints checking procedure. Compared with the iterative pruning for anti-monotone

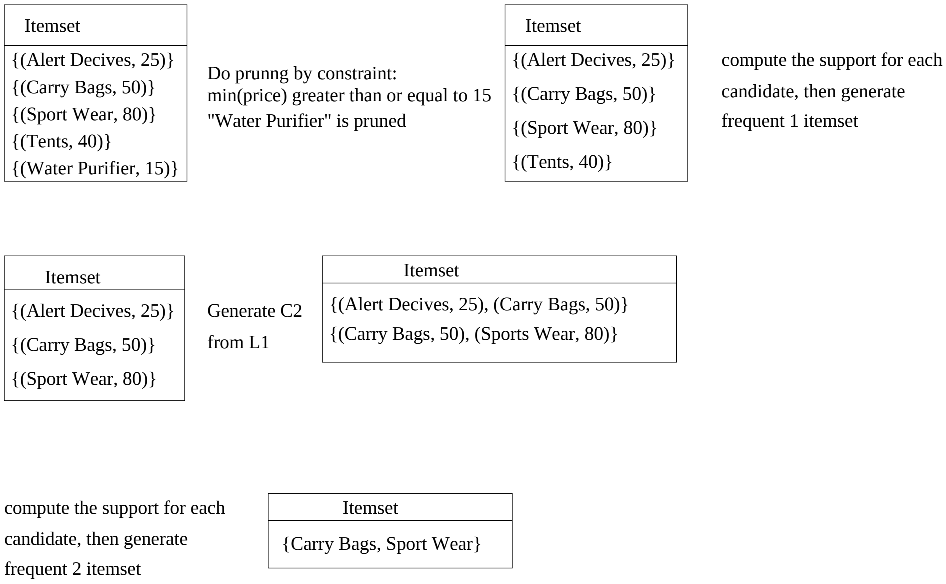


Figure 5.3: The frequent itemsets in Example 5.3.4

constraitns, this algorithm only performs pruning on candidate 1-itemset, and there is no any further constraints checking after this. This is because that, according to the pre-prunable property, any k-itemsets derived from the pruned candidate 1-itemsets will definitely satisfy these pre-prunable constraints as well. On the other hand, all possible k-itemsets satisfying the constraints have to be derived from the pruned candidate 1-itemsets.

□

Example 5.3.4 Let us finish Example 5.3.2, i.e., we want to generate the association rules under a condition that all items in each rule satisfy constraint $min(price) \geq 25$. We know that this constraint is proven to be anti-monotone and self-closed. Therefore, it is pre-prunable. Then based on Algorithm 5.3.1. the association mining is as follows with the results shown in Figure 5.3:

1. First, each item (*product*) is a member of the candidate 1-itemsets, C_1.
2. Then, each item price is checked against the constraint. The result is that "Water Purifier"is dropped from C_1 because its price is less than 25.

Constraint	pre-prunable
$S = v$	yes
$S \geq v$	yes
$S \leq v$	yes
$S \subseteq V$	yes
$min(S) \geq v$	yes
$max(S) \leq v$	yes

Table 5.3: Pre-prune Constraints

3. Frequent 1-itemsets L_1 are generated by support computation.

4. Next, the candidate 2-itemsets C_2 are generated from L_1.

5. Then we compute the support for each itemset in C_2 and get L_2 with only two itemsets as shown in Figure 5.3.

6. No candidate 3-itemset can be generated from L_2.

7. All association rules are generated from frequent itemsets.

□

In the last section, we analyze some categories of anti-monotone constraints and summarize them in Table 5.2 . Next, we analyze the same constraints on pre-prunable property. The result is in Table 5.3.

Proof Just like the anti-monotone constraints, we give some simple proof for this table.

1. From Table 5.2, $S = v$ is anti-monotone. So, we only need to prove it is self-closure. That is, given an k-itemset I, if any $k-1$-itemset I' satisfies this constraint, so does I. Otherwise, if I does not satisfy the constraint, it means that $\exists i \in I$, $S(i) \neq v$. Taking off one item i_1 from I, $i_1 \neq i$, to form a subset I' of I. We say that I' does not satisfy the constraint because $i \in I'$ and $S(i) \neq v$. This contradicts the premise. So, constraint $S = v$ is pre-prunable.

   In the same way, we can prove that $S \geq v$ and $S \leq v$ are anti-monotone.

2. Also from Table 5.2, $S \subseteq V$ is anti-monotone. Next, we prove that this constraint is self-closure as well. In another word, we need to demonstrate that given a k-itemset I, if any $(k-1)$-subset of I satisfies the constraint, then I satisfies the constraint as well. Otherwise, if I does not satisfy the constraint, it means that $\exists i \in I$, $i \notin V$. Taking off one item i_1 from I, $i_1 \neq i$, to form a subset I' of I. We say that I' does not satisfy the constraint because $i \in I'$ and $i \notin V$. This contradicts the premise. Therefore constraint $S \subseteq V$ is pre-prunable.

The other entries in Table 5.2 are not in Table 5.3 simply because they are not self-closure, and not pre-prunable as well. The prove can refer to Example 5.3.3. We will not give more details.

5.4 Performance analysis

In this section, we will present the results of two experiments conducted to analyze the performance of the above algorithms with respect to anti-monotone and pre-prunable optimization.

The experiments were performed on a Pentium Pro 200 with 64MB of memory running Windows NT. The same databases are used as in Experiment 4.1 and 4.2. Since these two optimization techniques are affecting only the step of generating frequent itemsets, the running time referred here are the execution time within this step.

Experiment 5.1 *Anti-monotone Optimization Experiment*

First, the optimization gained by applying anti-monotone constraints optimization is examined. The experiment is run based on an intra-dimensional data cube with two dimensions: *Location* and *Production.* We also have another associated table about the *price* information for each product. The association mining task is as follows:

Taking production as item dimension and location as transaction dimension, we want to mine intra-dimensional association rules with minimum support threshold 10% and constraint that the total price of items in a rule $\leq$ *min_sum.*

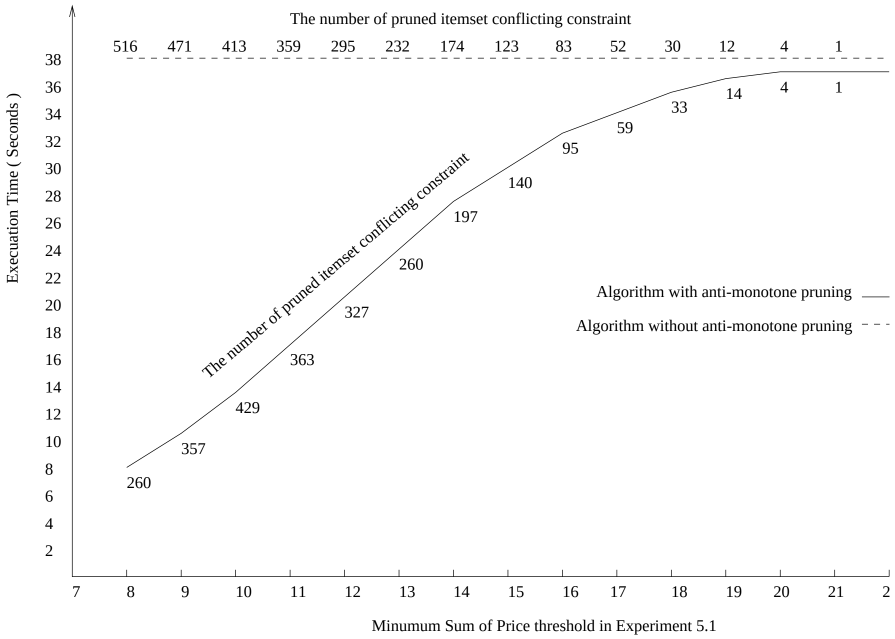


Figure 5.4: The performance analysis with different threshold on Algorithm 5.2.1 and Apriori+.

According to Table 5.2, we know that the constraint in this mining task is anti-monotone. Two algorithms, Algorithm 5.2.1 with anti-monotone optimization, the other one based on Algorithm Apriori+ without anti-monotone optimization, are executed on this same task. Fourteen different values of *min_sum* are applied: $\{8, 9, 10, \ldots 21\}$. Intuition tells us that the smaller of the threshold *min_sum* and the more candidates can be pruned when performing anti-monotone optimization, so the better Algorithm 5.2.1 should demenstrate. Figure 5.4 shows the performance results. The numbers above each curve represent the numbers of itemset pruned in each algorithm with different thresholds.

Analysis We can see that the curve complies with our expectation quite well. Generally Algorithm 5.2.1 outperforms the Apriori+. When *min_sun* becomes larger, the difference between these two algorithms becomes less. When $min\_sum = 22$, no itemset can be pruned in the whole procedure, in which the two curves approach very close. When $min\_sum = 8$, a large part of itemsets can be pruned before frequency computation, Algorithm 5.2.1 is much better than the other one.

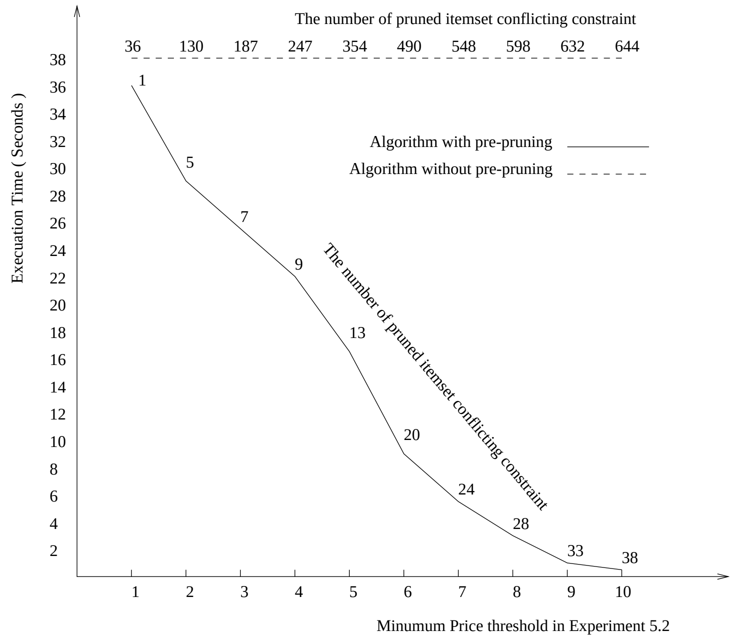


Figure 5.5: The performance analysis with different threshold on Algorithm 5.3.1 and Apriori+.

Experiment 5.2 *Pre-prunable Optimization Experiment*

Next, we conducted another experiment on the optimization gained by pre-prunable constraints optimization. The experiment is run based on the same cube as in Experiment 5.1. The association mining task in this experiment is:

Taking Production as item dimension, Location as transaction dimension, we want to mine intra-dimensional association rules with minimum support threshold 10% and constraints that the minimum price of items in a rule $\geq$ *min_price.*

According to Table 5.3, we know that the constraint in this mining task is pre-prunable. Two algorithms, Algorithm 5.3.1 with pre-prunable optimization and Apriori+, are executed on this same task. Ten different values of *min_price* are applied: $\{1, 2, \ldots 10\}$. We can easily conclude that the bigger the *min_price* and the more candidates can be pruned when performing pre-prunable optimization, the better Algorithm 5.3.1. Figure 5.5 shows the performance results.

Analysis We can see that Algorithm 5.3.1 is really better than the Apriori+.

When the *min_price* threshold is small enough that almost all item prices are higher than the threshold, these two algorithms show almost the same performance becuase for Algorithm 5.3.1 with pre-pruned optimization, the size of candidate 1-itemset is not reduced after pruning against the pre-prunable constraints, then the further computation is not saved.

Chapter 6

System Implementation

In the previous several chapters, we mainly focused on the association rules mining techniques, discussing several problems in this topic and described our algorithms. From system implementation point of view, there are lots of issues and interesting topics as well, such as how to design a user friendly interface, how to mine mutli-level association rules, how to specify various constraints, how to represent the mining result in a visualization way etc.

In this chapter, based on our implementation, we will introduce the user interface and the visualization of association rules mining.

6.1 Interface

Our association mining system is an on-line analytical mining system, which integrate both on-line analytical processing and association mining algorithms. This integration demenstrates a very powerful functionality, flexibility, usability and performance.

- Mining Various Association Rules

  In many cases, users want to generate various rules within the same mining system. We design a GUI(shown in Figure 6.1) to conveniently guide users to do various association mining. In the figure, the dimensions involved in the mining task are listed in the left side, for example here, we are interested in the association rules along two dimensions:*Location and Product.* There are two parts in the right side. The upper part can be used to specify the type of rules to be mined, and the lower part is to specify the transaction(group-by) dimension used in intra-dimensional and hybrid association mining. If we only want to do inter-dimensional association rules mining, we just click *Next* button to go ahead. Otherwise, we can use this page to perform intra-dimensional or hybrid association mining, as shown in the following two examples.

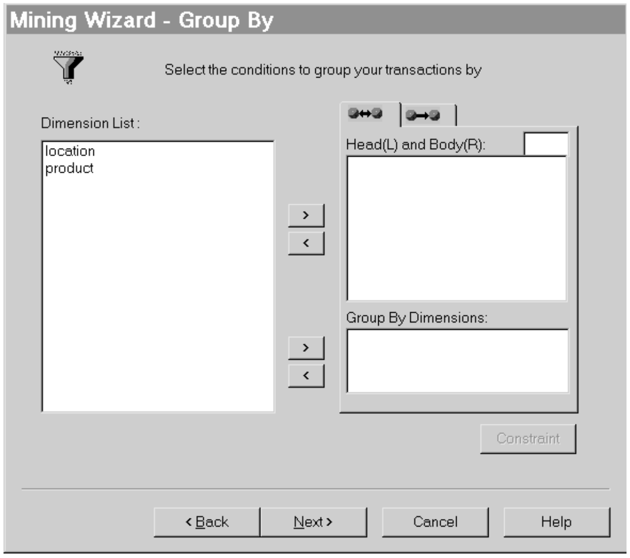


Figure 6.1: GUI page for various association rules mining

Example 6.1.1 *Mining Intra-dimensional Association rules*

Now, we want to generate intra-dimensional association rules within *Product* dimension, taken *Location* as the transaction dimension. What we do is to pick up *Location* dimension into the *group-by* list-box in the GUI, put *Product* dimension into the upper list box and check in the associated check-box. The check mark means that this dimension is the item dimension in intra-dimensional association mining and is repetitive in the resulting rules.

Example 6.1.2 *Mining Hybrid Association rules*

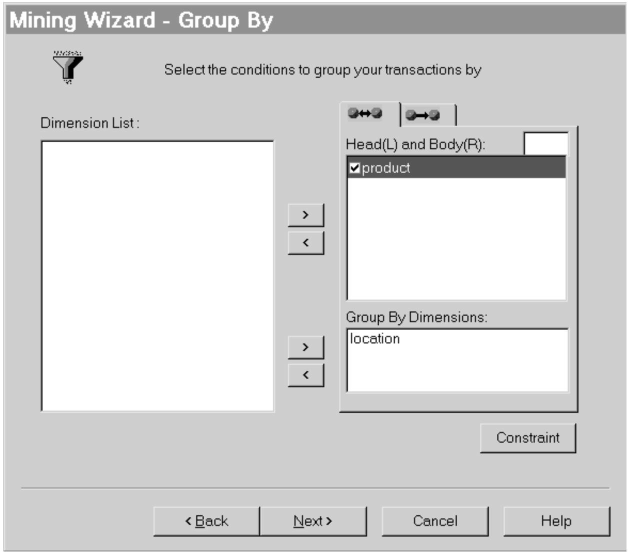


Figure 6.2: GUI page for intra-dimensional association rules mining

Suppose, we want to generate hybrid association rules within three dimensions: *Location, Product and Quantity*, taken *Product* dimension as the *item dimension* and *Location* as the *transaction dimension*. Then, originally, there should be three dimensions in the left side list-box in the GUI. Then, we pick up *Location* dimension into *group-by* list-box, put the *Product* and *Quantity* dimensions into the upper list-box and check the check-box associated with *Product* dimension.

- Mining multi-level association rules

  In many applications, since it is difficult to find strong associations among data at the primitive abstraction levels due to the sparsity of data in multi-dimensional space, and also on the other hand, many strong associations discovered at rather high concept levels are common sense knowledge. So, users may want to mine associations at multiple levels of abstraction. Our system provides a convenient way to do association mining in multiple levels. As shown in Figure 6.5, at the top of the main window, there is a toolbar with two combo-boxes. With this tool, user can select the desired dimension and specify desired level for the dimension. Here is an example.

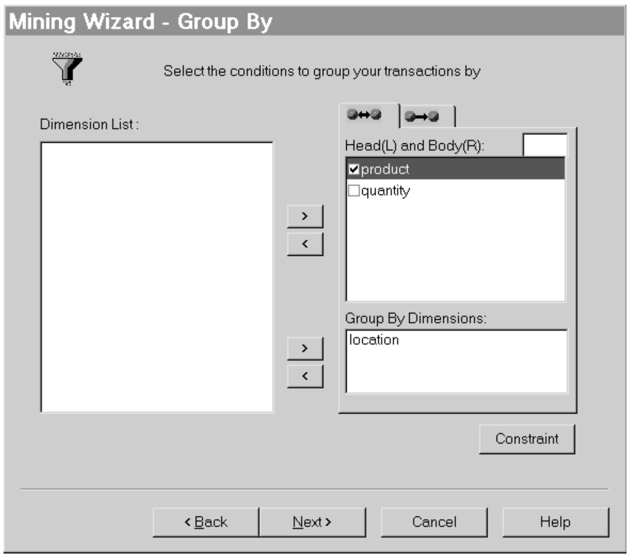
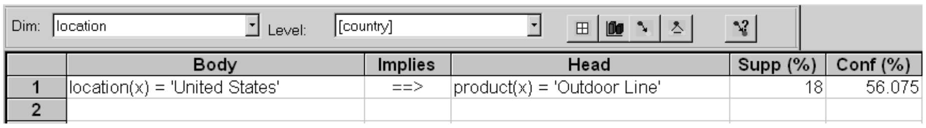


Figure 6.3: GUI page for hybrid association rules mining

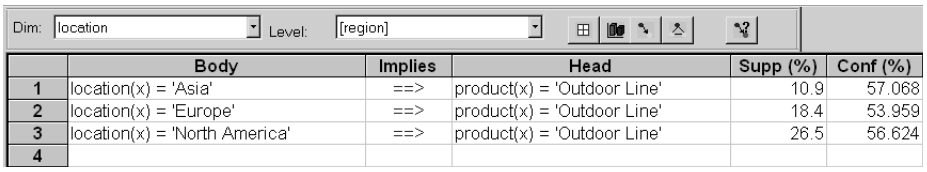
Example 6.1.3 *Mining multi-level association rules*

Suppose we are now performing inter-dimensional association mining among two dimensions: *Location and Product*. First, we choose *Location* at the *country* level, only one rule is output as shown in Figure 6.4. If we want to look into the higher level of *Location*, by selecting the *region* level as the current level of *Location* dimension, we get three rules as the output, as shown in Figure 6.5.



	Body	Implies	Head	Supp (%)	Conf (%)
1	location(x) = 'United States'	==>	product(x) = 'Outdoor Line'	18	56.075
2					

Figure 6.4: Mining in the *country* level of *Location* dimension



	Body	Implies	Head	Supp (%)	Conf (%)
1	location(x) = 'Asia'	==>	product(x) = 'Outdoor Line'	10.9	57.068
2	location(x) = 'Europe'	==>	product(x) = 'Outdoor Line'	18.4	53.959
3	location(x) = 'North America'	==>	product(x) = 'Outdoor Line'	26.5	56.624
4					

Figure 6.5: Mining in the *region* level of *Location* dimension

- Constraint-based association mining

  On-line analytical association mining requires fast response upon data mining requests whereas most mining requests are constraint-based. This requires mining not only be performed with a limited scope of data, confined by queries and/or constraints, but also adopt efficient, constraint-based data mining algorithms. We use a constraint pushing technology in our system to optimize this procedure, so that the whole system is very fast and efficient. Since the lack of time, we only implemented a part of constraints discussed in Chapter 5, but based on the careful design of data structure, it is easy to hook up other constraints into the system. In the interface part, we provide two GUI pages for specifying the constraints, such as the dimensions and the number of items required to be in the rule, the values limited for a dimension and if desirable, we can detail the constraints into rule body and head. Here, we will give some examples.

  Example 6.1.4 *Constraints on dimensions*

  Suppose we want to do inter-dimensional association mining among four dimensions:*Location, Product, Quantity and Cost.* And we are only interested in those rules related to the *Location* dimension. Using the GUI page shown in Figure 6.1, we put the *Location* into the upper list-box in the right side which means that *Location* must be shown in the rule. We can also limit the size of rule at the same time. For example, we want to get all rules with less than or equal to three items. Then, we put the string "<= 3"into the edit box in the upper-left side in the GUI, shown in Figure 6.6

  If what we are interested are only those rules related to the *Location = "United States"*, then we can specify this constraint in another page obtained by clicking the *"Constraint"* button in the current GUI page, as shown in Figure 6.7.

  Example 6.1.5 *Constrains on rule BODY and HEAD*

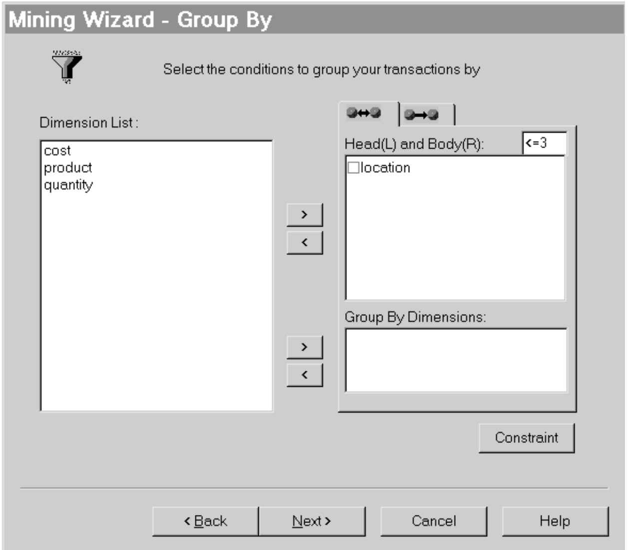


Figure 6.6: Interface for specifying constraints on dimensions

Continuing the above example, now, we want *Location* dimension only appeared in the head of the rule with the value *"United States"*, and we want *Quantity* dimension shown in the body of the rule with the value *"0.00   100.00"*. Also, we limit the size of the rule body should be less than or equal to 2. Then using the two GUI pages, we can perform constraints specification as shown in Figure 6.8 and Figure 6.9.

6.2 Association rules visualization

Visualization tools allow users to work in an interactive environment with ease in understanding the rules. In our system, we implemented three kinds of visualization tools to view the mining results: tabular view, ball graphical view and bar graphical view.

In the tabular view of association rules, all strong rules are represented in a tabular table (*rule table*) with each tuple corresponding to a rule. There are five columns in the rule table: body of the rule, the implication symbol, head of the rule, rule support and confidence. All the rules can be displayed in different order, such as order by head,

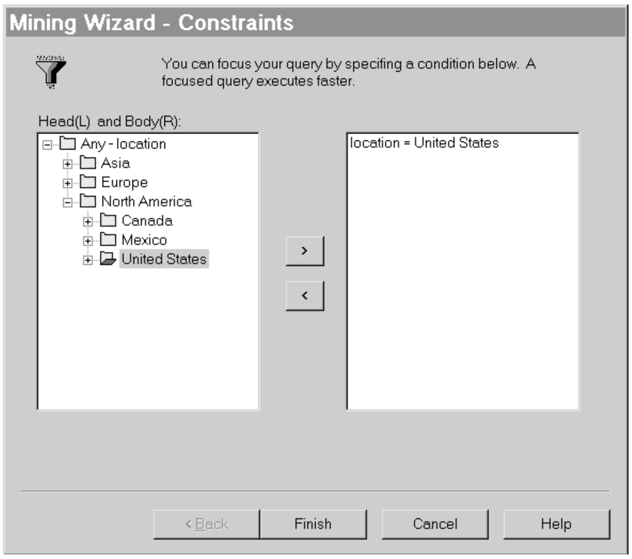


Figure 6.7: Interface for specifying constraints on dimension values

body, support or confidence. This helps users have a clearer view of the rules and locates a paticular rule more easily. The tabular view is very suitable for representing large number of rules with varied length, and easily understood. The Figure 6.10 gives an example of this kind of visualization.

Although the tabular view is good and scalable, sometimes users may have to view the rules from different aspects. For example, now we want to have a comprehensive view of the relationship between rules and items. In this case, the tabular view is not very convenient, since an item is repetitively appeared in the rule table as long as it is contained by a rule. Based on this requirement, our system provides two kinds of rules visualization tools. One is ball graphical display and the other is bar graphical display.

As shown in the Figure 6.11, there are two components in the ball visualization, the ball and the line connecting two balls with an arrow. Basically, the ball represents frequent 1-item and each line represents one rule. The arrow of line points to the head of the rule. Furthermore, the width of the arrow corresponds to the support of the rule, whereas the width of the arrow head corresponds to the confidence. Correspondingly, in the bar visualization tool, as shown in Figure 6.12, each rule is represented as a

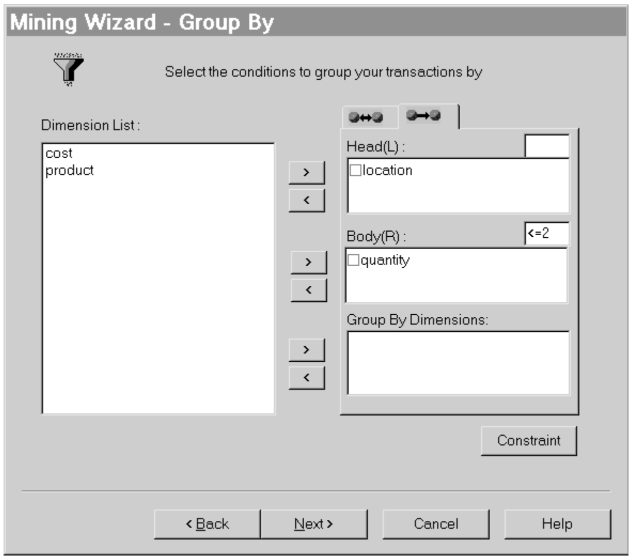


Figure 6.8: Interface for specifying constraints on dimension BODY and HEAD

bar at the conjunction of its head and body item. The rule support is height of the bar, and the rule confidence is represented by the color of the bar.

From the figures, we can see that the graphical visualization gives a more clear and vivid view of the rules and items. But limited by the graphical software which is used to implement the visualization, this method can only be used for small size of rules.

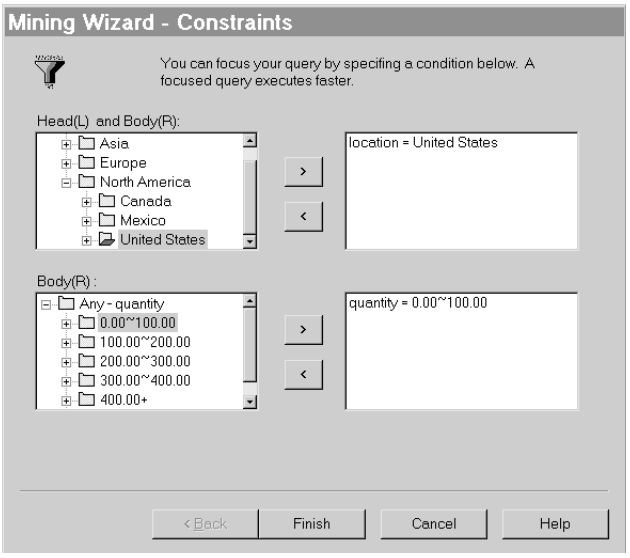


Figure 6.9: Interface for specifying constraints on dimension BODY and HEAD values

	Body	Implies	Head	Supp (%)	Conf (%)
1	category(x) = 'Camping Equipment'	==>	quantity(x) = '0.00~100.00'	22.9	99.565
2	cost(x) = '0.00~500.00'	==>	quantity(x) = '0.00~100.00'	25.9	93.165
3	cost(x) = '0.00~500.00'	==>	revenue(x) = '0.00~1000.00'	26.4	94.964
4	region(x) = 'United States'	==>	quantity(x) = '0.00~100.00'	26	80.997
5	revenue(x) = '0.00~1000.00'	==>	quantity(x) = '0.00~100.00'	55.5	80.202
6					
7					
8	cost(x) = '0.00~500.00'	==>	revenue(x) = '0.00~1000.00' AND quantity(x) = '0.00~100.00'	24.5	88.129
9	quantity(x) = '0.00~100.00' AND cost(x) = '0.00~500.00'	==>	revenue(x) = '0.00~1000.00'	24.5	94.595
10	revenue(x) = '0.00~1000.00' AND cost(x) = '0.00~500.00'	==>	quantity(x) = '0.00~100.00'	24.5	92.803

Figure 6.10: Tabular visulization of association rules

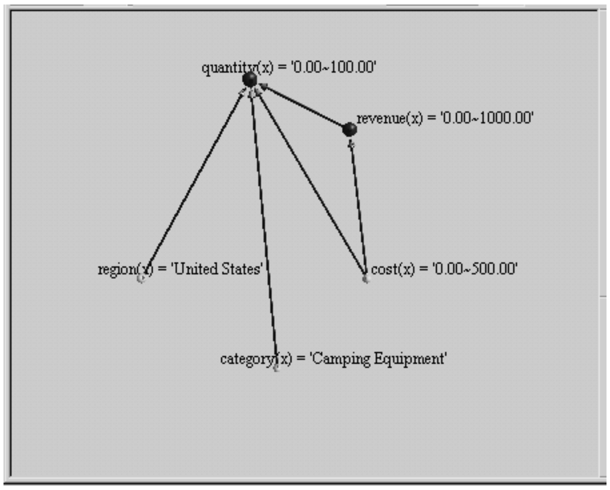


Figure 6.11: Ball graphical visulization of association rules

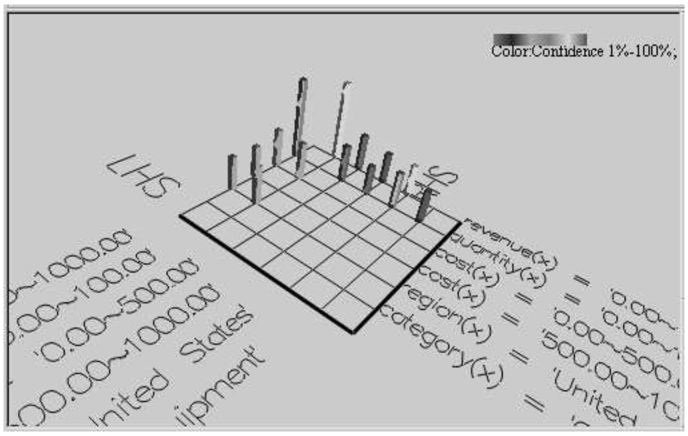


Figure 6.12: Bar graphical visulization of association rules

Chapter 7

Conclusion and Future Work

In this thesis, we performed a comprehensive study on association rules mining, proposed several algorithms with detailed discussion and experimental results. In this chapter, we will conclude with a summary of this study and propose some future research directions in this field.

7.1 Summary of research

The goal of this thesis is to propose the solution to the problem of generating association rules in large data warehouse. The research reported in this thesis proposes and develops an interesting approach, called on-line analytical mining of association rules, which integrates the recently developed OLAP (on-line analytical processing) technology with efficient association mining algorithms. It covers a few interesting studies on the issues including discovering different categories of association rules, measuring rule interestingness, and optimizing mining with different constraints.

- First, an architecture for association mining was proposed in Chapter 3. Different from other architectures based on a flat relational table structure, this is an OLAP-based architecture which integrates the data warehouse, OLAP technologies and association mining algorithms together. Data warehouse serves as a data store with integrated historical data, materialized data cube, meta data, etc. OLAP technologies act as the interface between data and mining task. It is the essential part in efficiently dealing with large data sets by providing data cube structure, enriched and efficient functions to play with data cube. And finally, association mining algorithms is to generate the desired rules complying with uers's requirement.

- Three sub-problems of mining association rules are covered in Chapter 4, that is, mining intra-dimensional, inter-dimensional, and hybrid rules. A layer-by-layer method is adopted. We first generate frequent 1-itemset, then 2-itemset, ..., until no more frequent itemsets can be found. An important property, the Apriori property, is used to reduce the size of candidate itemsets. Detailed algorithms are provided for each category with some discussions on the efficiency issue. Furthermore, for inter-dimensional association mining, another algorithm without the Apriori property is also presented. The discussion and performance analysis show that in the case of sparse data set and large number of dimensions, the algorithm with the Apriori property is more efficient, while in the case of dense data set, the latter is better.

- In this thesis, we designed a method based on the OLAP technologies to perform multi-level association mining in different abstract levels on the concept hierarchy. Several approaches of setting the minimum support threshold are compared. Based on the discussion, we recommend to use a reduced support threshold method in which the threshold is progressively lowered down when mining touches deeper levels of abstraction. It can help user generate sufficient meaningful associations at low levels but keep the rules to be generated at high levels sufficiently interesting.

- Some study on the rule interestingness measure are conducted at the end of Chapter 4. A simple and efficient method based on the chi-square property is proposed.

- In Chapter 5, the constraint-based association rules mining was discussed. We analyzed two important constraint properties: anti-monotone and pre-pruning.

> Anti-monotone constraints are used to prune candidate itemsets in each iteration, while pre-pruning constraints are used to prune candidate 1-itemset and any further k-itemset derived from the pruned 1-itemset automatically satisfies the constraints as well. The performance showed that the improved algorithms integrating these properties were much more efficient.

We should notice that all algorithms in this thesis rely on the manipulation of data cube that has been an active research topic for recent several years. The advantages of using such data cube structure is listed in Chapter 3. In general, multi-dimensional data warehouse and OLAP technologies facilitate the selection of dimensions and abstraction levels conveniently and efficiently. This makes "the ability to mine anywhere"realistic. The implementation of the DBMiner system shows that this OLAP-based approach presents great advantages over many existing algorithms in terms of both flexibility and efficiency.

7.2 Discussion and future work

During our previous discussion on the on-line analytical association mining, we always assume that a fully materialized cube can be built for the mining task. In real application, there are many cases that such a cube can not be stored on disk when the number of dimensions is large and the cube size is too big. In this case, only some subcubes are materialized. Then, a data mining method can take advantage of these materialized subcubes to speed up the mining process. In this situation, more researches are needed to be done including how to build the cube index, how to get the support for candidate itemset from the materialized subcubs, and how to generate those frequent itemsets which are not materialized, etc.

In this thesis, for the interestingness measure of association rules, we proposed a method based on the simple chi-square property. From recent study, some other methods have been explored with more predictive power to generated association rules. Further work can be done in this part on how to integrate those methods with OLAP technologies to get better results.

Also, for the interface part, we feel that our implementation is limited, although from the algorithm of view, the power of our association rules module is beyond the current interface. On the other hand, different visualization tools can help user understand the rules in various respects. Thus, a better interface and integration with more visualization tools is more and more important in the future.

Another issue of concern is the constraint-based association mining. In this thesis, we only discussed some simple categories of constraints, that is, anti-monotone and pre-pruning. In reality, there are many other constraints which do not fall into these two categories and are more complex. How to perform optimization with these constraints is a big research issue in the future. Moreover, Extended work on this can be defining a unifying template for association mining and a general language to design templates for the extraction of arbitrary association types.

Bibliography

[1] S. Agarwal, R. Agrawal, P. M. Deshpande, A. Gupta, J. F. Naughton, R. Ramakrishnan, and S. Sarawagi. On the computation of multidimensional aggregates. In *Proc. 1996 Int. Conf. Very Large Data Bases*, pages 506–521, Bombay, India, Sept. 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. Conf. Data Mining and Knowledge Discovery (KDD'96)*, pages 244–249, 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] R. J. Bayardo. Efficiently mining long patterns from databases. In *Proc. 1998 ACM-SIGMOD Int. Conf. Management of Data*, pages 85–93, Seattle, Washington, June 1998.

[7] R. J. Bayardo, R. Agrawal, and D. Gunopulos. Constraint-based rule mining on large, dense data sets. In *Proc. 1999 Int. Conf. Data Engineering (ICDE'99)*, Sydney, Australia, April 1999.

[8] S. Brin, R. Motwani, and C. Silverstein. Beyond market basket: Generalizing association rules to correlations. In *Proc. 1997 ACM-SIGMOD Int. Conf. Management of Data*, pages 265–276, Tucson, Arizona, May 1997.

[9] C. Brunk, J. Kelly, and R. Kohavi. MineSet: An integrated system for data mining. In *Proc. 3rd Int. Conf. Knowledge Discovery and Data Mining (KDD'97)*, pages 135–138, Newport Beach, California, August 1997.

[10] S. Chaudhuri and U. Dayal. An overview of data warehousing and OLAP technology. *ACM SIGMOD Record*, 26:65–74, 1997.

[11] D.W. Cheung, J. Han, V. Ng, and C.Y. Wong. Maintenance of discovered association rules in large databases: An incremental updating technique. In *Proc. 1996 Int. Conf. Data Engineering*, pages 106–114, New Orleans, Louisiana, Feb. 1996.

[12] E. F Codd, S. B. Codd, and C. T. Salley. Beyond decision support. *Computer World*, 27, July 1993.

[13] U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy (eds.). *Advances in Knowledge Discovery and Data Mining.* AAAI/MIT Press, 1996.

[14] T. Fukuda, Y. Morimoto, S. Morishita, and T. Tokuyama. Data mining using two-dimensional optimized association rules: Scheme, algorithms, and visualization. In *Proc. 1996 ACM-SIGMOD Int. Conf. Management of Data*, pages 13–23, Montreal, Canada, June 1996.

[15] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, and H. Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab and sub-totals. *Data Mining and Knowledge Discovery*, 1:29–54, 1997.

[16] E.-H. Han, G. Karypis, and V. Kumar. Scalable parallel data mining for association rules. In *Proc. 1997 ACM-SIGMOD Int. Conf. Management of Data*, pages 277–288, Tucson, Arizona, May 1997.

[17] J. Han, J. Chiang, S. Chee, J. Chen, Q. Chen, S. Cheng, W. Gong, M. Kamber, G. Liu, K. Koperski, Y. Lu, N. Stefanovic, L. Winstone, B. Xia, O. R. Zaïane, S. Zhang, and H. Zhu. DBMiner: A system for data mining in relational databases and data warehouses. In *Proc. CASCON'97: Meeting of Minds*, pages 249–260, Toronto, Canada, November 1997.

[18] J. Han, G. Dong, and Y. Yin. Efficient mining of partial periodic patterns in time series database. In *Proc. 1999 Int. Conf. Data Engineering (ICDE'99)*, Sydney, Australia, April 1999.

[19] J. Han and Y. Fu. Dynamic generation and refinement of concept hierarchies for knowledge discovery in databases. In *Proc. AAAI'94 Workshop Knowledge Discovery in Databases (KDD'94)*, pages 157–168, Seattle, WA, July 1994.

[20] 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.

[21] V. Harinarayan, A. Rajaraman, and J. D. Ullman. Implementing data cubes efficiently. In *Proc. 1996 ACM-SIGMOD Int. Conf. Management of Data*, pages 205–216, Montreal, Canada, June 1996.

[22] IBM. *IBM Intelligent Miner User's Guide, Version 1, Release 1.* IBM, 1996.

[23] M. Kamber, J. Han, and J. Y. Chiang. Metarule-guided mining of multi-dimensional association rules using data cubes. In *Proc. 3rd Int. Conf. Knowledge Discovery and Data Mining (KDD'97)*, pages 207–210, Newport Beach, California, August 1997.

[24] R. Kimball. *The Data Warehouse Toolkit.* John Wiley & Sons, New York, 1996.

[25] 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. Conf. Information and Knowledge Management*, pages 401–408, Gaithersburg, Maryland, Nov. 1994.

[26] F. Korn, A. Labrinidis, Y. Kotidis, and C. Faloutsos. Ratio rules: A new paradigm for fast, quantifiable data mining. In *Proc. 1998 Int. Conf. Very Large Data Bases*, pages 582–593, New York, NY, August 1998.

[27] B. Lent, A. Swami, and J. Widom. Clustering association rules. In *Proc. 1997 Int. Conf. Data Engineering (ICDE'97)*, pages 220–231, Birmingham, England, April 1997.

[28] R. Meo, G. Psaila, and S. Ceri. A new SQL-like operator for mining association rules. In *Proc. 1996 Int. Conf. Very Large Data Bases*, pages 122–133, Bombay, India, Sept. 1996.

[29] Microsoft. OLEDB for OLAP programmer's reference version 1.0. In *http://www.microsoft.com/data/oledb/olap*, 1998.

[30] R.J. Miller and Y. Yang. Association rules over interval data. In *Proc. 1997 ACM-SIGMOD Int. Conf. Management of Data*, pages 452–461, Tucson, Arizona, May 1997.

[31] R. Ng, L. V. S. Lakshmanan, J. Han, and A. Pang. Exploratory mining and pruning optimizations of constrained associations rules. In *Proc. 1998 ACM-SIGMOD Int. Conf. Management of Data*, pages 13–24, Seattle, Washington, June 1998.

[32] B. Özden, S. Ramaswamy, and A. Silberschatz. Cyclic association rules. In *Proc. 1998 Int. Conf. Data Engineering (ICDE'98)*, pages 412–421, Orlando, FL, Feb. 1998.

[33] 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.

[34] J.S. Park, M.S. Chen, and P.S. Yu. Efficient parallel mining for association rules. In *Proc. 4th Int. Conf. Information and Knowledge Management*, pages 31–36, Baltimore, Maryland, Nov. 1995.

[35] S. Ramaswamy, S. Mahajan, and A. Silberschatz. On the discovery of interesting patterns in association rules. In *Proc. 1998 Int. Conf. Very Large Data Bases*, pages 368–379, New York, NY, August 1998.

[36] A. Savasere, E. Omiecinski, and S. Navathe. An efficient algorithm for mining association rules in large databases. In *Proc. 1995 Int. Conf. Very Large Data Bases*, pages 432–443, Zurich, Switzerland, Sept. 1995.

[37] W. Shen, B. Mitbander, K. Ong, and C. Zaniolo. Using metaqueries to integrate inductive learning and deductive database technology. In *Proc. AAAI'94 Workshop Knowledge Discovery in Databases (KDD'94)*, pages 335–346, Seattle, WA, July 1994.

[38] C. Silverstein, S. Brin, R. Motwani, and J. Ullman. Scalable techniques for mining causal structures. In *Proc. 1998 Int. Conf. Very Large Data Bases*, pages 594–605, New York, NY, August 1998.

[39] 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.

[40] R. Srikant and R. Agrawal. Mining quantitative association rules in large relational tables. In *Proc. 1996 ACM-SIGMOD Int. Conf. Management of Data*, pages 1–12, Montreal, Canada, June 1996.

[41] R. Srikant, Q. Vu, and R. Agrawal. Mining association rules with item constraints. In *Proc. 3rd Int. Conf. Knowledge Discovery and Data Mining (KDD'97)*, pages 67–73, Newport Beach, California, August 1997.

[42] H. Toivonen. Sampling large databases for association rules. In *Proc. 1996 Int. Conf. Very Large Data Bases*, pages 134–145, Bombay, India, Sept. 1996.

[43] Y. Zhao, P. M. Deshpande, and J. F. Naughton. An array-based algorithm for simultaneous multidimensional aggregates. In *Proc. 1997 ACM-SIGMOD Int. Conf. Management of Data*, pages 159–170, Tucson, Arizona, May 1997.