

# Contents

<b>5 Mining Frequent Patterns, Associations, and Correlations</b>	<b>3</b>
5.1 Basic Concepts and a Road Map	3
5.1.1 Market Basket Analysis: A Motivating Example	4
5.1.2 Frequent Itemsets, Closed Itemsets, and Association Rules	5
5.1.3 Frequent Pattern Mining: A Road Map	7
5.2 Efficient and Scalable Frequent Itemset Mining Methods	8
5.2.1 The Apriori Algorithm: Finding Frequent Itemsets Using Candidate Generation	9
5.2.2 Generating Association Rules from Frequent Itemsets	11
5.2.3 Improving the Efficiency of Apriori	12
5.2.4 Mining Frequent Itemsets without Candidate Generation	15
5.2.5 Mining Frequent Itemsets Using Vertical Data Format	17
5.2.6 Mining Closed Frequent Itemsets	19
5.3 Mining Various Kinds of Association Rules	20
5.3.1 Mining Multilevel Association Rules	20
5.3.2 Mining Multidimensional Association Rules from Relational Databases and Data Warehouses	23
5.4 From Association Mining to Correlation Analysis	27
5.4.1 Strong Rules Are Not Necessarily Interesting: An Example	27
5.4.2 From Association Analysis to Correlation Analysis	28
5.5 Constraint-Based Association Mining	32
5.5.1 Metarule-Guided Mining of Association Rules	32
5.5.2 Constraint Pushing: Mining Guided by Rule Constraints	33
5.6 Summary	36
5.7 Exercises	38
5.8 Bibliographic Notes	43



# List of Figures

5.1	Market basket analysis. . . . .	4
5.2	Generation of candidate itemsets and frequent itemsets, where the minimum support count is 2. . . . .	11
5.3	Generation and pruning of candidate 3-itemsets, $C_3$ , from $L_2$ using the Apriori property. . . . .	12
5.4	The Apriori algorithm for discovering frequent itemsets for mining Boolean association rules. . . . .	13
5.5	Hash table, $H_2$ , for candidate 2-itemsets: This hash table was generated by scanning the transactions of Table 5.1 while determining $L_1$ from $C_1$ . If the minimum support count is, say, 3, then the itemsets in buckets 0, 1, 3, and 4 cannot be frequent and so they should not be included in $C_2$ . . . . .	14
5.6	Mining by partitioning the data. . . . .	14
5.7	An FP-tree registers compressed, frequent pattern information. . . . .	16
5.8	The conditional FP-tree associated with the conditional node I3. . . . .	16
5.9	The FP-growth algorithm for discovering frequent itemsets without candidate generation. . . . .	17
5.10	A concept hierarchy for <i>AllElectronics</i> computer items. . . . .	21
5.11	Multilevel mining with uniform support. . . . .	22
5.12	Multilevel mining with reduced support. . . . .	23
5.13	Lattice of cuboids, making up a 3-D data cube. Each cuboid represents a different group-by. The base cuboid contains the three predicates <i>age</i> , <i>income</i> , and <i>buys</i> . [TO EDITOR For consistency with rest of book, please kindly italicize all instances of <i>age</i> , <i>income</i> , and <i>buys</i> .] . . . . .	25
5.14	A 2-D grid for tuples representing customers who purchase high-definition TVs. . . . .	27



# List of Tables

5.1	Transactional data for an <i>AllElectronics</i> branch. . . . .	10
5.2	Mining the FP-tree by creating conditional (sub)-pattern bases. . . . .	16
5.3	The vertical data format of the transaction data set $D$ of Table 5.1. . . . .	18
5.4	The 2-itemsets in vertical data format. . . . .	18
5.5	The 3-itemsets in vertical data format. . . . .	18
5.6	Task-relevant data, $D$ . . . . .	21
5.7	A $2 \times 2$ contingency table summarizing the transactions with respect to game and video purchases. . . . .	29
5.8	The above contingency table, now shown with the expected values. . . . .	29
5.9	A $2 \times 2$ contingency table for two items. . . . .	30
5.10	Comparison of four correlation measures using contingency tables for different data sets. . . . .	30
5.11	Comparison of the four correlation measures for game-and-video data sets. . . . .	31
5.12	Characterization of commonly used SQL-based constraints. . . . .	36
5.13	Generalized relation for Exercise 5.9. . . . .	40



## Chapter 5

# Mining Frequent Patterns, Associations, and Correlations

**Frequent patterns** are patterns (such as itemsets, subsequences, or substructures) that appear in a data set frequently. For example, a set of items, such as milk and bread, that appear frequently together in a transaction data set is a *frequent itemset*. A subsequence, such as buying first a PC, then a digital camera, and then a memory card, if it occurs frequently in a shopping history database, is a (*frequent*) *sequential pattern*. A *substructure* can refer to different structural forms, such as subgraphs, subtrees, or sublattices, which may be combined with itemsets or subsequences. If a substructure occurs frequently, it is called a (*frequent*) *structured pattern*. Finding such frequent patterns plays an essential role in mining associations, correlations, and many other interesting relationships among data. Moreover, it helps in data classification, clustering, and other data mining tasks as well. Thus, frequent pattern mining has become an important data mining task and a focused theme in data mining research.

In this chapter, we introduce the concepts of frequent patterns, associations and correlations, and study how they can be mined efficiently. The topic of frequent pattern mining is indeed rich. This chapter is dedicated to methods of *frequent itemset mining*. We delve into the following questions: How can we find frequent itemsets from large amounts of data, where the data are either transactional or relational? How can we mine association rules in multilevel and multidimensional space? Which association rules are the most interesting? How can we help or guide the mining procedure to discover interesting associations or correlations? How can we take advantage of user preferences or constraints to speed up the mining process? The techniques learned in this chapter may also be extended for more advanced forms of frequent pattern mining, such as from sequential and structured data sets, as we will study in later chapters.

### 5.1 Basic Concepts and a Road Map

Frequent pattern mining searches for recurring relationships in a given data set. This section introduces the basic concepts of frequent pattern mining for the discovery of interesting associations and correlations between itemsets in transactional and relational databases. We begin in Section 5.1.1 by presenting an example of market basket analysis, the earliest form of frequent pattern mining for association rules. The basic concepts of mining frequent patterns and associations are given in Section 5.1.2. Section 5.1.3 presents a road map to the different kinds of frequent patterns, association rules, and correlation rules that can be mined.

### 5.1.1 Market Basket Analysis: A Motivating Example

Frequent itemset mining leads to the discovery of associations and correlations among items in large transactional or relational data sets. With massive amounts of data continuously being collected and stored, many industries are becoming interested in mining such patterns from their databases. The discovery of interesting correlation relationships among huge amounts of business transaction records can help in many business decision making processes, such as catalog design, cross-marketing, and customer shopping behavior analysis.

A typical example of frequent itemset mining is **market basket analysis**. This process analyzes customer buying habits by finding associations between the different items that customers place in their “shopping baskets” (Figure 5.1). The discovery of such associations can help retailers develop marketing strategies by gaining insight into which items are frequently purchased together by customers. For instance, if customers are buying milk, how likely are they to also buy bread (and what kind of bread) on the same trip to the supermarket? Such information can lead to increased sales by helping retailers do selective marketing and plan their shelf space.

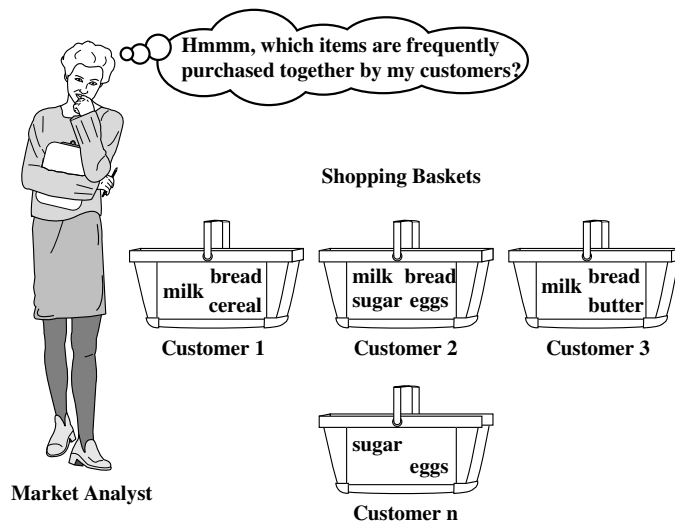


Figure 5.1: Market basket analysis.

Let’s look at an example of how market basket analysis can be useful.

**Example 5.1 Market basket analysis.** Suppose, as manager of an *AllElectronics* branch, you would like to learn more about the buying habits of your customers. Specifically, you wonder, “Which groups or sets of items are customers likely to purchase on a given trip to the store?” To answer your question, market basket analysis may be performed on the retail data of customer transactions at your store. You can then make use of the results to plan marketing or advertising strategies, or in the design of a new catalog. For instance, market basket analysis may help you design different store layouts. In one strategy, items that are frequently purchased together can be placed in close proximity in order to further encourage the sale of such items together. If customers who purchase computers also tend to buy antivirus software at the same time, then placing the hardware display close to the software display may help to increase the sales of both of these items. In an alternative strategy, placing hardware and software at opposite ends of the store may entice customers who purchase such items to pick up other items along the way. For instance, after deciding on an expensive computer, a customer may observe security systems for sale while heading towards the software display to purchase antivirus software and may decide to purchase a home security system as well. Market basket analysis can also help retailers to plan which items to put on sale at reduced prices. If customers tend to purchase computers and printers together, then having a sale on printers may encourage the sale of printers *as well as* computers. ■

If we think of the universe as the set of items available at the store, then each item has a Boolean variable

representing the presence or absence of that item. Each basket can then be represented by a Boolean vector of values assigned to these variables. The Boolean vectors can be analyzed for buying patterns that reflect items that are frequently *associated* or purchased together. These patterns can be represented in the form of **association rules**. For example, the information that customers who purchase computers also tend to buy antivirus software at the same time is represented in Association Rule (5.1) below:

$$\text{computer} \Rightarrow \text{antivirus\_software} \quad [\text{support} = 2\%, \text{confidence} = 60\%] \quad (5.1)$$

Rule **support** and **confidence** are two measures of rule interestingness. They respectively reflect the usefulness and certainty of discovered rules. A support of 2% for Association Rule (5.1) means that 2% of all the transactions under analysis show that computer and antivirus software are purchased together. A confidence of 60% means that 60% of the customers who purchased a computer also bought the software. Typically, association rules are considered interesting if they satisfy both a **minimum support threshold** and a **minimum confidence threshold**. Such thresholds can be set by users or domain experts. Additional analysis can be performed to uncover interesting statistical correlations between associated items.

### 5.1.2 Frequent Itemsets, Closed Itemsets, and Association Rules

Let  $\mathcal{I} = \{I_1, I_2, \dots, I_m\}$  be a set of items. Let  $D$ , the task-relevant data, be a set of database transactions where each transaction  $T$  is a set of items such that  $T \subseteq \mathcal{I}$ . Each transaction is associated with an identifier, called TID. Let  $A$  be a set of items. A transaction  $T$  is said to contain  $A$  if and only if  $A \subseteq T$ . An association rule is an implication of the form  $A \Rightarrow B$ , where  $A \subset \mathcal{I}$ ,  $B \subset \mathcal{I}$ , and  $A \cap B = \phi$ . The rule  $A \Rightarrow B$  holds in the transaction set  $D$  with **support**  $s$ , where  $s$  is the percentage of transactions in  $D$  that contain  $A \cup B$  (i.e., the *union* of sets  $A$  and  $B$ , or say, both  $A$  and  $B$ ). This is taken to be the probability,  $P(A \cup B)$ .<sup>1</sup> The rule  $A \Rightarrow B$  has **confidence**  $c$  in the transaction set  $D$ , where  $c$  is the percentage of transactions in  $D$  containing  $A$  that also contain  $B$ . This is taken to be the conditional probability,  $P(B|A)$ . That is,

$$\text{support}(A \Rightarrow B) = P(A \cup B) \quad (5.2)$$

$$\text{confidence}(A \Rightarrow B) = P(B|A). \quad (5.3)$$

Rules that satisfy both a minimum support threshold (*min\_sup*) and a minimum confidence threshold (*min\_conf*) are called **strong**. By convention, we write support and confidence values so as to occur between 0% and 100%, rather than 0 to 1.0.

A set of items is referred to as an **itemset**.<sup>2</sup> An itemset that contains  $k$  items is a  **$k$ -itemset**. The set  $\{\text{computer}, \text{antivirus\_software}\}$  is a 2-itemset. The **occurrence frequency of an itemset** is the number of transactions that contain the itemset. This is also known, simply, as the **frequency**, **support count**, or **count** of the itemset. Note that the itemset support defined in Equation (5.2) is sometimes referred to as *relative support*, whereas the occurrence frequency is called the **absolute support**. If the relative support of an itemset  $I$  satisfies a pre-specified **minimum support threshold** (i.e., the absolute support of  $I$  satisfies the corresponding **minimum support count threshold**), then  $I$  is a **frequent** itemset.<sup>3</sup> The set of frequent  $k$ -itemsets is commonly denoted by  $L_k$ .<sup>4</sup>

From Equation (5.3), we have

$$\text{confidence}(A \Rightarrow B) = P(B|A) = \frac{\text{support}(A \cup B)}{\text{support}(A)} = \frac{\text{support\_count}(A \cup B)}{\text{support\_count}(A)}. \quad (5.4)$$

<sup>1</sup>Notice that the notation  $P(A \cup B)$  indicates the probability that a transaction contains the *union* of set  $A$  and set  $B$ , i.e., it contains every item in  $A$  and in  $B$ . This should not be confused with  $P(A \text{ or } B)$ , which indicates the probability that a transaction contains either  $A$  or  $B$ .

<sup>2</sup>In the data mining research literature, “itemset” is more commonly used than “item set.”

<sup>3</sup>In early work, itemsets satisfying minimum support were referred to as **large**. This term, however, is somewhat confusing as it has connotations to the number of items in an itemset rather than the frequency of occurrence of the set. Hence, we use the more recent term **frequent**.

<sup>4</sup>Although the term **frequent** is preferred over **large**, for historical reasons frequent  $k$ -itemsets are still denoted as  $L_k$ .

Equation (5.4) shows that the confidence of rule  $A \Rightarrow B$  can be easily derived from the support counts of  $A$  and  $A \cup B$ . That is, once the support counts of  $A$ ,  $B$  and  $A \cup B$  are found, it is straightforward to derive the corresponding association rules  $A \Rightarrow B$  and  $B \Rightarrow A$  and check whether they are strong. Thus the problem of mining association rules can be reduced to that of mining frequent itemsets.

In general, association rule mining can be viewed as a two-step process:

1. **Find all frequent itemsets:** By definition, each of these itemsets will occur at least as frequently as a pre-determined minimum support count,  $min\_sup$ .
2. **Generate strong association rules from the frequent itemsets:** By definition, these rules must satisfy minimum support and minimum confidence.

Additional interestingness measures can be applied for the discovery of correlation relationships between associated items, as will be discussed in Section 5.4. Since the second step is much less costly than the first, the overall performance of mining association rules is determined by the first step.

A major challenge in mining frequent itemsets from a large data set is the fact that such mining often generates a huge number of itemsets satisfying the minimum support ( $min\_sup$ ) threshold, especially when  $min\_sup$  is set low. This is because if an itemset is frequent, each of its subsets is frequent as well. A long itemset will contain a combinatorial number of shorter, frequent sub-itemsets. For example, a frequent itemset of length 100, such as  $\{a_1, a_2, \dots, a_{100}\}$ , contains  $\binom{100}{1} = 100$  frequent 1-itemsets:  $a_1, a_2, \dots, a_{100}$ ,  $\binom{100}{2}$  frequent 2-itemsets:  $(a_1, a_2), (a_1, a_3) \dots, (a_{99}, a_{100})$ , and so on. The total number of frequent itemsets that it contains is thus,

$$\binom{100}{1} + \binom{100}{2} + \dots + \binom{100}{100} = 2^{100} - 1 \approx 1.27 \times 10^{30}. \quad (5.5)$$

This is too huge a number of itemsets for any computer to compute or store. To overcome this difficulty, we introduce the concepts of *closed frequent itemset* and *maximal frequent itemset*.

An itemset  $X$  is **closed** in a data set  $S$  if there exists no proper super-itemset<sup>5</sup>  $Y$  such that  $Y$  has the same support count as  $X$  in  $S$ . An itemset  $X$  is a **closed frequent itemset** in set  $S$  if  $X$  is both closed and frequent in set  $S$ . An itemset  $X$  is a **maximal frequent itemset** (or **max-itemset**) in set  $S$  if  $X$  is frequent, and there exists no super-itemset  $Y$  such that  $X \subset Y$  and  $Y$  is frequent in  $S$ .

Let  $\mathcal{C}$  be the set of closed frequent itemsets for a data set  $S$  satisfying a minimum support threshold,  $min\_sup$ . Let  $\mathcal{M}$  be the set of maximal frequent itemsets for  $S$  satisfying  $min\_sup$ . Suppose that we have the support count of each itemset in  $\mathcal{C}$  and  $\mathcal{M}$ . Notice that  $\mathcal{C}$  and its count information can be used to derive the whole set of frequent itemsets. Thus we say that  $\mathcal{C}$  contains complete information regarding its corresponding frequent itemsets. On the other hand,  $\mathcal{M}$ , registers only the support of the maximal itemsets. It usually does not contain the complete support information regarding its corresponding frequent itemsets. We illustrate these concepts with the following example.

**Example 5.2 Closed and maximal frequent itemsets.** Suppose that a transaction database has only two transactions:  $\{\langle a_1, a_2, \dots, a_{100} \rangle; \langle a_1, a_2, \dots, a_{50} \rangle\}$ . Let the minimum support count threshold be  $min\_sup = 1$ . We find two closed frequent itemsets and their support counts, that is  $\mathcal{C} = \{\{a_1, a_2, \dots, a_{100}\} : 1; \{a_1, a_2, \dots, a_{50}\} : 2\}$ . There is one maximal frequent itemset:  $\mathcal{M} = \{\{a_1, a_2, \dots, a_{100}\} : 1\}$ . (We cannot include  $\{a_1, a_2, \dots, a_{50}\}$  as a maximal frequent itemset since it has a frequent super-set,  $\{a_1, a_2, \dots, a_{100}\}$ .) Compare this to the above, where we determined that there are  $2^{100} - 1$  frequent itemsets, which is too huge a set to be enumerated anywhere!

The set of closed frequent itemsets contains complete information regarding the frequent itemsets. For example, from  $\mathcal{C}$ , we can derive, say, (1)  $\{a_2, a_{45} : 2\}$  since  $\{a_2, a_{45}\}$  is a sub-itemset of the itemset  $\{a_1, a_2, \dots, a_{50} : 2\}$ ; and

<sup>5</sup> $Y$  is a proper super-itemset of  $X$  if  $X$  is a proper sub-itemset of  $Y$ , that is, if  $X \subset Y$ . In other words, every item of  $X$  is contained in  $Y$  but there is at least one item of  $Y$  that is not in  $X$ .

(2)  $\{a_8, a_{55} : 1\}$  since  $\{a_8, a_{55}\}$  is not a sub-itemset of the previous itemset but of the itemset  $\{a_1, a_2, \dots, a_{100} : 1\}$ . However, from the maximal frequent itemset, we can only assert that both itemsets ( $\{a_2, a_{45}\}$  and  $\{a_8, a_{55}\}$ ) are frequent, but we cannot assert their actual support counts. ■

### 5.1.3 Frequent Pattern Mining: A Road Map

Market basket analysis is just one form of frequent pattern mining. In fact, there are many kinds of frequent patterns, association rules and correlation relationships. Frequent pattern mining can be classified in various ways, based on the following criteria:

- **Based on the *completeness of patterns to be mined*:** As we discussed in the previous subsection, we can mine the **complete set of frequent itemsets**, the **closed frequent itemsets**, and the **maximal frequent itemsets**, given a minimum support threshold. We can also mine **constrained frequent itemsets** (i.e., those that satisfy a set of user-defined constraints), **approximate frequent itemsets** (i.e., those that derive only approximate support counts for the mined frequent itemsets), **near-match frequent itemsets** (i.e., those that tally the support count of the near or almost matching itemsets), **top- $k$  frequent itemsets** (i.e., the  $k$  most frequent itemsets for a user-specified value,  $k$ ), and so on.

Different applications may have different requirements regarding the completeness of the patterns to be mined, which in turn, can lead to different evaluation and optimization methods. In this chapter, our study of mining methods focuses on mining the *complete set of frequent itemsets*, *closed frequent itemsets*, and *constrained frequent itemsets*. We leave the mining of frequent itemsets under other completeness requirements as an exercise.

- **Based on the *levels of abstraction involved in the rule set*:** Some methods for association rule mining can find rules at differing levels of abstraction. For example, suppose that a set of association rules mined includes the following rules where  $X$  is a variable representing a customer:

$$\text{buys}(X, \text{"computer"}) \Rightarrow \text{buys}(X, \text{"HP\_printer"}) \quad (5.6)$$

$$\text{buys}(X, \text{"laptop\_computer"}) \Rightarrow \text{buys}(X, \text{"HP\_printer"}) \quad (5.7)$$

In Rules (5.6) and (5.7), the items bought are referenced at different levels of abstraction (e.g., “*computer*” is a higher-level abstraction of “*laptop computer*”.) We refer to the rule set mined as consisting of **multilevel association rules**. If, instead, the rules within a given set do not reference items or attributes at different levels of abstraction, then the set contains **single-level association rules**.

- **Based on the *number of data dimensions involved in the rule*:** If the items or attributes in an association rule reference only one dimension, then it is a **single-dimensional association rule**. Note that Rule (5.1), for example, could be rewritten as Rule (5.8):

$$\text{buys}(X, \text{"computer"}) \Rightarrow \text{buys}(X, \text{"antivirus\_software"}) \quad (5.8)$$

Rules (5.6), (5.7), and (5.8) are single-dimensional association rules since they each refer to only one dimension, *buys*.<sup>6</sup>

If a rule references two or more dimensions, such as the dimensions *age*, *income*, and *buys*, then it is a **multidimensional association rule**. The following rule is an example of a multidimensional rule.

$$\text{age}(X, \text{"30...39"}) \wedge \text{income}(X, \text{"42K...48K"}) \Rightarrow \text{buys}(X, \text{"high resolution TV"}) \quad (5.9)$$

<sup>6</sup>Following the terminology used in multidimensional databases, we refer to each distinct predicate in a rule as a *dimension*.

- **Based on the *types of values* handled in the rule:** If a rule involves associations between the presence or absence of items, it is a **Boolean association rule**. For example, Rules (5.1), (5.6), and (5.7) are Boolean association rules obtained from market basket analysis.

If a rule describes associations between quantitative items or attributes, then it is a **quantitative association rule**. In these rules, quantitative values for items or attributes are partitioned into intervals. Rule (5.9) above is also considered a quantitative association rule. Note that the quantitative attributes, *age* and *income*, have been discretized.

- **Based on the *kinds of rules* to be mined:** Frequent pattern analysis can generate various kinds of rules and other interesting relationships. **Association rules** are the most popular kind of rules generated from frequent patterns. Typically, such mining can generate a large number of rules, many of which are redundant or do not indicate a correlation relationship among itemsets. Thus, the discovered associations can be further analyzed to uncover statistical correlations, leading to **correlation rules**.

We can also mine **strong gradient relationships** among itemsets, where a gradient is the ratio of the measure of an item when compared with that of its parent (a generalized itemset), its child (a specialized itemset) or its sibling (a comparable itemset). One such example could be: The average sales from *Sony\_Digital\_Camera* increases over 16% when sold together with *Sony\_Laptop\_Computer*: both are siblings of the parent itemset, *Sony*.

- **Based on the *kinds of patterns* to be mined:** There are many kinds of frequent patterns that can be mined from different kinds of data sets. For this chapter, our focus is on **frequent itemset mining**, that is, the mining of frequent itemsets (sets of items) from transactional or relational data sets. However, other kinds of frequent patterns can be found from other kinds of data sets. **Sequential pattern mining** searches for frequent *subsequences* in a *sequence data set*, where a sequence records an ordering of events. For example, with sequential pattern mining, we can study the order in which items are frequently purchased. For instance, customers may tend to first buy a PC, followed by a digital camera, and then a memory card. **Structured pattern mining** searches for frequent *substructures* in a *structured data set*. Notice that *structure* is a general concept that covers many different kinds of structural forms, such as graphs, lattices, trees, sequences, sets, single items, or combinations of such structures. Single items are the simplest form of structure. Each element of an itemset may contain a subsequence, a subtree, etc., and such containment relationships can be defined recursively. Therefore, structured pattern mining can be considered as the most general form of frequent pattern mining.

In the next section, we will study efficient methods for mining the basic (i.e., single-level, single-dimensional, Boolean) frequent itemsets from transactional databases, and show how to generate association rules from such itemsets. The extension of this scope of mining to multi-level, multi-dimensional, and quantitative rules is discussed in Section 5.3. The mining of strong-correlation relationships is studied in Section 5.4. Constraint-based mining is studied in Section 5.5. We address the more advanced topic of mining sequence and structured patterns in later chapters. Nevertheless, most of the methods studied here can be easily extended for the mining of more complex kinds of patterns.

## 5.2 Efficient and Scalable Frequent Itemset Mining Methods

In this section, you will learn methods for mining the simplest form of frequent patterns—*single-dimensional, single-level, Boolean frequent itemsets*, such as those discussed for market basket analysis in Section 5.1.1. We begin by presenting **Apriori**, a basic algorithm for finding frequent itemsets (Section 5.2.1). In Section 5.2.2, we look at how to generate strong association rules from frequent itemsets. Section 5.2.3 describes several variations to the Apriori algorithm for improved efficiency and scalability. Section 5.2.4 presents methods for mining frequent itemsets that, unlike Apriori, do not involve the generation of “candidate” frequent itemsets. Section 5.2.5 presents methods for mining frequent itemsets that take advantage of vertical data format. Methods for mining closed frequent itemsets are discussed in Section 5.2.6.

### 5.2.1 The Apriori Algorithm: Finding Frequent Itemsets Using Candidate Generation

**Apriori** is a seminal algorithm proposed by R. Agrawal and R. Srikant in 1994 for mining frequent itemsets for Boolean association rules. The name of the algorithm is based on the fact that the algorithm uses *prior knowledge* of frequent itemset properties, as we shall see below. Apriori employs an iterative approach known as a *level-wise* search, where  $k$ -itemsets are used to explore  $(k + 1)$ -itemsets. First, the set of frequent 1-itemsets is found by scanning the database to accumulate the count for each item, and collecting those items that satisfy minimum support. The resulting set is denoted  $L_1$ . Next,  $L_1$  is used to find  $L_2$ , the set of frequent 2-itemsets, which is used to find  $L_3$ , and so on, until no more frequent  $k$ -itemsets can be found. The finding of each  $L_k$  requires one full scan of the database.

To improve the efficiency of the level-wise generation of frequent itemsets, an important property called the **Apriori property**, presented below, is used to reduce the search space. We will first describe this property, and then show an example illustrating its use.

[TO EDITOR Please Note: There were some errors/misunderstandings regarding the typesetting of this property in edition 1! We would like it to be one line, on its own (e.g., treated as a paragraph), not joined with the paragraph following it. We would like the words ‘Apriori property’ to be in boldface and the sentence following it to be italicized. Thank you!]

**Apriori property:** *All nonempty subsets of a frequent itemset must also be frequent.*

The Apriori property is based on the following observation. By definition, if an itemset  $I$  does not satisfy the minimum support threshold,  $min\_sup$ , then  $I$  is not frequent, that is,  $P(I) < min\_sup$ . If an item  $A$  is added to the itemset  $I$ , then the resulting itemset (i.e.,  $I \cup A$ ) cannot occur more frequently than  $I$ . Therefore,  $I \cup A$  is not frequent either, that is,  $P(I \cup A) < min\_sup$ .

This property belongs to a special category of properties called **antimonotone** in the sense that *if a set cannot pass a test, all of its supersets will fail the same test as well*. It is called *antimonotone* because the property is monotonic in the context of failing a test.<sup>7</sup>

“How is the Apriori property used in the algorithm?” To understand this, let us look at how  $L_{k-1}$  is used to find  $L_k$  for  $k \geq 2$ . A two-step process is followed, consisting of **join** and **prune** actions.

1. **The join step:** To find  $L_k$ , a set of **candidate**  $k$ -itemsets is generated by joining  $L_{k-1}$  with itself. This set of candidates is denoted  $C_k$ . Let  $l_1$  and  $l_2$  be itemsets in  $L_{k-1}$ . The notation  $l_i[j]$  refers to the  $j$ th item in  $l_i$  (e.g.,  $l_1[k - 2]$  refers to the second to the last item in  $l_1$ ). By convention, Apriori assumes that items within a transaction or itemset are sorted in lexicographic order. For the  $(k - 1)$ -itemset,  $l_i$ , this means that the items are sorted such that  $l_i[1] < l_i[2] < \dots < l_i[k - 1]$ . The join,  $L_{k-1} \bowtie L_{k-1}$ , is performed, where members of  $L_{k-1}$  are joinable if their first  $(k - 2)$  items are in common. That is, members  $l_1$  and  $l_2$  of  $L_{k-1}$  are joined if  $(l_1[1] = l_2[1]) \wedge (l_1[2] = l_2[2]) \wedge \dots \wedge (l_1[k - 2] = l_2[k - 2]) \wedge (l_1[k - 1] < l_2[k - 1])$ . The condition  $l_1[k - 1] < l_2[k - 1]$  simply ensures that no duplicates are generated. The resulting itemset formed by joining  $l_1$  and  $l_2$  is  $l_1[1]l_1[2] \dots l_1[k - 2]l_1[k - 1]l_2[k - 1]$ .
2. **The prune step:**  $C_k$  is a superset of  $L_k$ , that is, its members may or may not be frequent, but all of the frequent  $k$ -itemsets are included in  $C_k$ . A scan of the database to determine the count of each candidate in  $C_k$  would result in the determination of  $L_k$  (i.e., all candidates having a count no less than the minimum support count are frequent by definition, and therefore belong to  $L_k$ ).  $C_k$ , however, can be huge, and so this could involve heavy computation. To reduce the size of  $C_k$ , the Apriori property is used as follows. Any  $(k - 1)$ -itemset that is not frequent cannot be a subset of a frequent  $k$ -itemset. Hence, if any  $(k - 1)$ -subset of a candidate  $k$ -itemset is not in  $L_{k-1}$ , then the candidate cannot be frequent either and so can be removed from  $C_k$ . This **subset testing** can be done quickly by maintaining a hash tree of all frequent itemsets.

---

<sup>7</sup>The Apriori property has many applications. It can also be used to prune search during data cube computation (Chapter 4).

Table 5.1: Transactional data for an *AllElectronics* branch.

<i>TID</i>	<i>List of item_IDs</i>
T100	I1, I2, I5
T200	I2, I4
T300	I2, I3
T400	I1, I2, I4
T500	I1, I3
T600	I2, I3
T700	I1, I3
T800	I1, I2, I3, I5
T900	I1, I2, I3

**Example 5.3 Apriori.** Let's look at a concrete example, based on the *AllElectronics* transaction database,  $D$ , of Table 5.1. There are nine transactions in this database, that is,  $|D| = 9$ . We use Figure 5.2 to illustrate the Apriori algorithm for finding frequent itemsets in  $D$ .

1. In the first iteration of the algorithm, each item is a member of the set of candidate 1-itemsets,  $C_1$ . The algorithm simply scans all of the transactions in order to count the number of occurrences of each item.
2. Suppose that the minimum support count required is 2 (i.e.,  $min\_sup\_count = 2$ , or  $min\_sup = 2/9 = 22\%$ ). The set of frequent 1-itemsets,  $L_1$ , can then be determined. It consists of the candidate 1-itemsets satisfying minimum support. In our example, all of the candidates in  $C_1$  satisfy minimum support.
3. To discover the set of frequent 2-itemsets,  $L_2$ , the algorithm uses the join  $L_1 \bowtie L_1$  to generate a candidate set of 2-itemsets,  $C_2$ .<sup>8</sup>  $C_2$  consists of  $\binom{|L_1|}{2}$  2-itemsets. Note that no candidates are removed from  $C_2$  during the prune step since each subset of the candidates is also frequent.
4. Next, the transactions in  $D$  are scanned and the support count of each candidate itemset in  $C_2$  is accumulated, as shown in the middle table of the second row in Figure 5.2.
5. The set of frequent 2-itemsets,  $L_2$ , is then determined, consisting of those candidate 2-itemsets in  $C_2$  having minimum support.
6. The generation of the set of candidate 3-itemsets,  $C_3$ , is detailed in Figure 5.3. From the join step, we first get  $C_3 = L_2 \bowtie L_2 = \{\{I1,I2,I3\}, \{I1,I2,I5\}, \{I1,I3,I5\}, \{I2,I3,I4\}, \{I2,I3,I5\}, \{I2,I4,I5\}\}$ . Based on the Apriori property that all subsets of a frequent itemset must also be frequent, we can determine that the four latter candidates cannot possibly be frequent. We therefore remove them from  $C_3$ , thereby saving the effort of unnecessarily obtaining their counts during the subsequent scan of  $D$  to determine  $L_3$ . Note that when given a candidate  $k$ -itemset, we only need to check if its  $(k - 1)$ -subsets are frequent since the Apriori algorithm uses a level-wise search strategy. The resulting pruned version of  $C_3$  is shown in the first table of the bottom row of Figure 5.2.
7. The transactions in  $D$  are scanned in order to determine  $L_3$ , consisting of those candidate 3-itemsets in  $C_3$  having minimum support (Figure 5.2).
8. The algorithm uses  $L_3 \bowtie L_3$  to generate a candidate set of 4-itemsets,  $C_4$ . Although the join results in  $\{\{I1,I2,I3,I5\}\}$ , this itemset is pruned since its subset  $\{\{I2,I3,I5\}\}$  is not frequent. Thus,  $C_4 = \phi$ , and the algorithm terminates, having found all of the frequent itemsets. ■

Figure 5.4 shows pseudocode for the Apriori algorithm and its related procedures. Step 1 of Apriori finds the frequent 1-itemsets,  $L_1$ . In steps 2–10,  $L_{k-1}$  is used to generate candidates  $C_k$  in order to find  $L_k$  for  $k \geq 2$ . The `apriori_gen` procedure generates the candidates and then uses the Apriori property to eliminate those having a subset

<sup>8</sup> $L_1 \bowtie L_1$  is equivalent to  $L_1 \times L_1$  since the definition of  $L_k \bowtie L_k$  requires the two joining itemsets to share  $k - 1 = 0$  items.

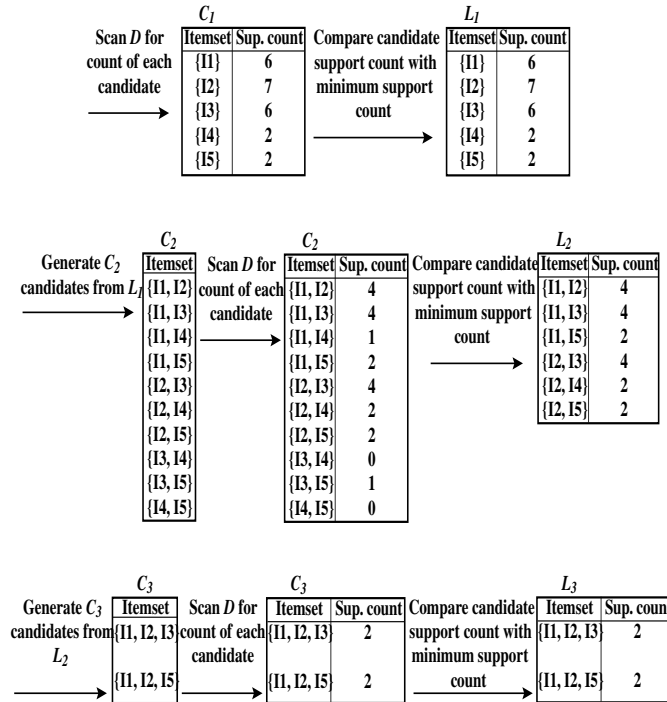


Figure 5.2: Generation of candidate itemsets and frequent itemsets, where the minimum support count is 2.

that is not frequent (step 3). This procedure is described below. Once all the candidates have been generated, the database is scanned (step 4). For each transaction, a **subset** function is used to find all subsets of the transaction that are candidates (step 5), and the count for each of these candidates is accumulated (steps 6 and 7). Finally, all those candidates satisfying minimum support form the set of frequent itemsets,  $L$ . A procedure can then be called to generate association rules from the frequent itemsets. Such a procedure is described in Section 5.2.2.

The `apriori_gen` procedure performs two kinds of actions, namely, **join** and **prune**, as described above. In the join component,  $L_{k-1}$  is joined with  $L_{k-1}$  to generate potential candidates (steps 1–4). The prune component (steps 5–7) employs the Apriori property to remove candidates that have a subset that is not frequent. The test for infrequent subsets is shown in procedure `has_infrequent_subset`.

### 5.2.2 Generating Association Rules from Frequent Itemsets

Once the frequent itemsets from transactions in a database  $D$  have been found, it is straightforward to generate strong association rules from them (where *strong* association rules satisfy both minimum support and minimum confidence). This can be done using Equation (5.4) for confidence, which we show again here for completeness:

$$\text{confidence}(A \Rightarrow B) = P(B|A) = \frac{\text{support\_count}(A \cup B)}{\text{support\_count}(A)},$$

The conditional probability is expressed in terms of itemset support count, where  $\text{support\_count}(A \cup B)$  is the number of transactions containing the itemsets  $A \cup B$ , and  $\text{support\_count}(A)$  is the number of transactions containing the itemset  $A$ . Based on this equation, association rules can be generated as follows:

- For each frequent itemset  $l$ , generate all nonempty subsets of  $l$ .
- For every nonempty subset  $s$  of  $l$ , output the rule “ $s \Rightarrow (l - s)$ ” if  $\frac{\text{support\_count}(l)}{\text{support\_count}(s)} \geq \text{min\_conf}$ , where  $\text{min\_conf}$  is the minimum confidence threshold.

- (a) Join:  $C_3 = L_2 \bowtie L_2 = \{\{I1,I2\}, \{I1,I3\}, \{I1,I5\}, \{I2,I3\}, \{I2,I4\}, \{I2,I5\}\} \bowtie \{\{I1,I2\}, \{I1,I3\}, \{I1,I5\}, \{I2,I3\}, \{I2,I4\}, \{I2,I5\}\} = \{\{I1,I2,I3\}, \{I1,I2,I5\}, \{I1,I3,I5\}, \{I2,I3,I4\}, \{I2,I3,I5\}, \{I2,I4,I5\}\}$ .
- (b) Prune using the Apriori property: All nonempty subsets of a frequent itemset must also be frequent. Do any of the candidates have a subset that is not frequent?
- The 2-item subsets of  $\{I1,I2,I3\}$  are  $\{I1,I2\}$ ,  $\{I1,I3\}$ , and  $\{I2,I3\}$ . All 2-item subsets of  $\{I1,I2,I3\}$  are members of  $L_2$ . Therefore, keep  $\{I1,I2,I3\}$  in  $C_3$ .
  - The 2-item subsets of  $\{I1,I2,I5\}$  are  $\{I1,I2\}$ ,  $\{I1,I5\}$ , and  $\{I2,I5\}$ . All 2-item subsets of  $\{I1,I2,I5\}$  are members of  $L_2$ . Therefore, keep  $\{I1,I2,I5\}$  in  $C_3$ .
  - The 2-item subsets of  $\{I1,I3,I5\}$  are  $\{I1,I3\}$ ,  $\{I1,I5\}$ , and  $\{I3,I5\}$ .  $\{I3,I5\}$  is not a member of  $L_2$ , and so it is not frequent. Therefore, remove  $\{I1,I3,I5\}$  from  $C_3$ .
  - The 2-item subsets of  $\{I2,I3,I4\}$  are  $\{I2,I3\}$ ,  $\{I2,I4\}$ , and  $\{I3,I4\}$ .  $\{I3,I4\}$  is not a member of  $L_2$ , and so it is not frequent. Therefore, remove  $\{I2,I3,I4\}$  from  $C_3$ .
  - The 2-item subsets of  $\{I2,I3,I5\}$  are  $\{I2,I3\}$ ,  $\{I2,I5\}$ , and  $\{I3,I5\}$ .  $\{I3,I5\}$  is not a member of  $L_2$ , and so it is not frequent. Therefore, remove  $\{I2,I3,I5\}$  from  $C_3$ .
  - The 2-item subsets of  $\{I2,I4,I5\}$  are  $\{I2,I4\}$ ,  $\{I2,I5\}$ , and  $\{I4,I5\}$ .  $\{I4,I5\}$  is not a member of  $L_2$ , and so it is not frequent. Therefore, remove  $\{I2,I4,I5\}$  from  $C_3$ .
- (c) Therefore,  $C_3 = \{\{I1,I2,I3\}, \{I1,I2,I5\}\}$  after pruning.

Figure 5.3: Generation and pruning of candidate 3-itemsets,  $C_3$ , from  $L_2$  using the Apriori property.

Since the rules are generated from frequent itemsets, each one automatically satisfies minimum support. Frequent itemsets can be stored ahead of time in hash tables along with their counts so that they can be accessed quickly.

**Example 5.4 Generating associations rules.** Let's try an example based on the transactional data for *All-Electronics* shown in Table 5.1. Suppose the data contain the frequent itemset  $l = \{I1,I2,I5\}$ . What are the association rules that can be generated from  $l$ ? The nonempty subsets of  $l$  are  $\{I1,I2\}$ ,  $\{I1,I5\}$ ,  $\{I2,I5\}$ ,  $\{I1\}$ ,  $\{I2\}$ , and  $\{I5\}$ . The resulting association rules are as shown below, each listed with its confidence:

$I1 \wedge I2 \Rightarrow I5,$	<i>confidence</i> = $2/4 = 50\%$
$I1 \wedge I5 \Rightarrow I2,$	<i>confidence</i> = $2/2 = 100\%$
$I2 \wedge I5 \Rightarrow I1,$	<i>confidence</i> = $2/2 = 100\%$
$I1 \Rightarrow I2 \wedge I5,$	<i>confidence</i> = $2/6 = 33\%$
$I2 \Rightarrow I1 \wedge I5,$	<i>confidence</i> = $2/7 = 29\%$
$I5 \Rightarrow I1 \wedge I2,$	<i>confidence</i> = $2/2 = 100\%$

If the minimum confidence threshold is, say, 70%, then only the second, third, and last rules above are output, since these are the only ones generated that are strong. Note that, unlike conventional classification rules, association rules can contain more than one conjunct in the right hand side of the rule. ■

### 5.2.3 Improving the Efficiency of Apriori

“How can we further improve the efficiency of Apriori-based mining?” Many variations of the Apriori algorithm have been proposed that focus on improving the efficiency of the original algorithm. Several of these variations are summarized below.

**Hash-based technique** (hashing itemsets into corresponding buckets): A hash-based technique can be used to reduce the size of the candidate  $k$ -itemsets,  $C_k$ , for  $k > 1$ . For example, when scanning each transaction in the database to generate the frequent 1-itemsets,  $L_1$ , from the candidate 1-itemsets in  $C_1$ , we can generate all of

**Algorithm: Apriori.** Find frequent itemsets using an iterative level-wise approach based on candidate generation.

**Input:**

- $D$ , a database of transactions;
- $min\_sup$ , the minimum support count threshold.

**Output:**  $L$ , frequent itemsets in  $D$ .

**Method:**

```

(1)   $L_1 = \text{find\_frequent\_1-itemsets}(D)$ ;
(2)  for ( $k = 2; L_{k-1} \neq \phi; k++$ ) {
(3)     $C_k = \text{apriori\_gen}(L_{k-1})$ ;
(4)    for each transaction  $t \in D$  { // scan  $D$  for counts
(5)       $C_t = \text{subset}(C_k, t)$ ; // get the subsets of  $t$  that are candidates
(6)      for each candidate  $c \in C_t$ 
(7)         $c.\text{count}++$ ;
(8)    }
(9)     $L_k = \{c \in C_k | c.\text{count} \geq min\_sup\}$ 
(10) }
(11) return  $L = \cup_k L_k$ ;

```

procedure  $\text{apriori\_gen}(L_{k-1}$ :frequent  $(k-1)$ -itemsets)

```

(1)  for each itemset  $l_1 \in L_{k-1}$ 
(2)    for each itemset  $l_2 \in L_{k-1}$ 
(3)      if ( $l_1[1] = l_2[1] \wedge l_1[2] = l_2[2] \wedge \dots \wedge l_1[k-2] = l_2[k-2] \wedge l_1[k-1] < l_2[k-1]$ ) then {
(4)         $c = l_1 \bowtie l_2$ ; // join step: generate candidates
(5)        if  $\text{has\_infrequent\_subset}(c, L_{k-1})$  then
(6)          delete  $c$ ; // prune step: remove unfruitful candidate
(7)        else add  $c$  to  $C_k$ ;
(8)      }
(9)  return  $C_k$ ;

```

procedure  $\text{has\_infrequent\_subset}(c$ : candidate  $k$ -itemset;

```

 $L_{k-1}$ : frequent  $(k-1)$ -itemsets); // use prior knowledge
(1)  for each  $(k-1)$ -subset  $s$  of  $c$ 
(2)    if  $s \notin L_{k-1}$  then
(3)      return TRUE;
(4)  return FALSE;

```

Figure 5.4: The Apriori algorithm for discovering frequent itemsets for mining Boolean association rules.

the 2-itemsets for each transaction, hash (i.e., map) them into the different *buckets* of a *hash table* structure, and increase the corresponding bucket counts (Figure 5.5). A 2-itemset whose corresponding bucket count in the hash table is below the support threshold cannot be frequent and thus should be removed from the candidate set. Such a hash-based technique may substantially reduce the number of the candidate  $k$ -itemsets examined (especially when  $k = 2$ ).

**Transaction reduction** (reducing the number of transactions scanned in future iterations): A transaction that does not contain any frequent  $k$ -itemsets cannot contain any frequent  $(k+1)$ -itemsets. Therefore, such a transaction can be marked or removed from further consideration since subsequent scans of the database for  $j$ -itemsets, where  $j > k$ , will not require it.

**Partitioning** (partitioning the data to find candidate itemsets): A partitioning technique can be used that requires just two database scans to mine the frequent itemsets (Figure 5.6). It consists of two phases. In Phase I, the algorithm subdivides the transactions of  $D$  into  $n$  nonoverlapping partitions. If the minimum support threshold for transactions in  $D$  is  $min\_sup$ , then the minimum support count for a partition is  $min\_sup \times \text{the number of transactions in that partition}$ . For each partition, all frequent itemsets within the partition are found. These are referred to as **local frequent itemsets**. The procedure employs a special data structure that, for each itemset, records the TIDs of the transactions containing the items in the itemset. This allows it to find all of the local frequent  $k$ -itemsets, for  $k = 1, 2, \dots$ , in just one scan of the database.

$H_2$

Create hash table  $H_2$  using hash function  $h(x, y) = ((\text{order of } x) \cdot 10 + (\text{order of } y)) \bmod 7$

bucket address	0	1	2	3	4	5	6
bucket count	2	2	4	2	2	4	4
bucket contents	{(11, 14)} {(13, 15)}	{(11, 15)} {(11, 15)}	{(12, 13)} {(12, 13)} {(12, 13)}	{(12, 14)} {(12, 14)}	{(12, 15)} {(12, 15)}	{(11, 12)} {(11, 12)}	{(11, 13)} {(11, 13)}

Figure 5.5: Hash table,  $H_2$ , for candidate 2-itemsets: This hash table was generated by scanning the transactions of Table 5.1 while determining  $L_1$  from  $C_1$ . If the minimum support count is, say, 3, then the itemsets in buckets 0, 1, 3, and 4 cannot be frequent and so they should not be included in  $C_2$ .

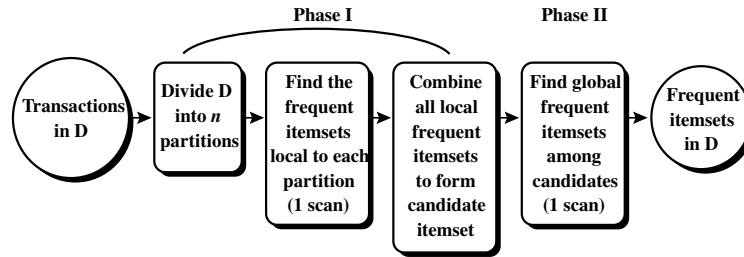


Figure 5.6: Mining by partitioning the data.

A local frequent itemset may or may not be frequent with respect to the entire database,  $D$ . Any itemset that is potentially frequent with respect to  $D$  must occur as a frequent itemset in at least one of the partitions. Therefore, all local frequent itemsets are candidate itemsets with respect to  $D$ . The collection of frequent itemsets from all partitions forms the **global candidate itemsets** with respect to  $D$ . In Phase II, a second scan of  $D$  is conducted in which the actual support of each candidate is assessed in order to determine the global frequent itemsets. Partition size and the number of partitions are set so that each partition can fit into main memory and therefore be read only once in each phase.

**Sampling** (mining on a subset of the given data): The basic idea of the sampling approach is to pick a random sample  $S$  of the given data  $D$ , and then search for frequent itemsets in  $S$  instead of  $D$ . In this way, we trade off some degree of accuracy against efficiency. The sample size of  $S$  is such that the search for frequent itemsets in  $S$  can be done in main memory, and so only one scan of the transactions in  $S$  is required overall. Because we are searching for frequent itemsets in  $S$  rather than in  $D$ , it is possible that we will miss some of the global frequent itemsets. To lessen this possibility, we use a lower support threshold than minimum support to find the frequent itemsets local to  $S$  (denoted  $L^S$ ). The rest of the database is then used to compute the actual frequencies of each itemset in  $L^S$ . A mechanism is used to determine whether all of the global frequent itemsets are included in  $L^S$ . If  $L^S$  actually contains all of the frequent itemsets in  $D$ , then only one scan of  $D$  is required. Otherwise, a second pass can be done in order to find the frequent itemsets that were missed in the first pass. The sampling approach is especially beneficial when efficiency is of utmost importance, such as in computationally intensive applications that must be run on a very frequent basis.

**Dynamic itemset counting** (adding candidate itemsets at different points during a scan): A dynamic itemset counting technique was proposed in which the database is partitioned into blocks marked by start points. In this variation, new candidate itemsets can be added at any start point, unlike in Apriori, which determines new candidate itemsets only immediately prior to each complete database scan. The technique is dynamic in that it estimates the support of all of the itemsets that have been counted so far, adding new candidate itemsets if all of their subsets are estimated to be frequent. The resulting algorithm requires fewer database scans than Apriori.

Other variations involving the mining of multilevel and multidimensional association rules are discussed in the rest of this chapter. The mining of associations related to spatial data, time-series data, and multimedia data are discussed in Chapter 9.

### 5.2.4 Mining Frequent Itemsets without Candidate Generation

As we have seen, in many cases the Apriori candidate generate-and-test method significantly reduces the size of candidate sets, leading to good performance gain. However, it can suffer from two nontrivial costs.

- *It may need to generate a huge number of candidate sets.* For example, if there are  $10^4$  frequent 1-itemsets, the Apriori algorithm will need to generate more than  $10^7$  candidate 2-itemsets. Moreover, to discover a frequent pattern of size 100, such as  $\{a_1, \dots, a_{100}\}$ , it has to generate at least  $2^{100} - 1 \approx 10^{30}$  candidates in total.
- *It may need to repeatedly scan the database and check a large set of candidates by pattern matching.* It is costly to go over each transaction in the database to determine the support of the candidate itemsets.

“Can we design a method that mines the complete set of frequent itemsets without candidate generation?” An interesting method in this attempt is called **frequent-pattern growth**, or simply **FP-growth**, which adopts a *divide-and-conquer* strategy as follows. First, it compresses the database representing frequent items into a **frequent-pattern tree**, or **FP-tree**, which retains the itemset association information. It then divides the compressed database into a set of *conditional databases* (a special kind of projected database), each associated with one frequent item or “pattern fragment”, and mines each such database separately. You’ll see how it works with the following example.

**Example 5.5 FP-growth (finding frequent itemsets without candidate generation).** We reexamine the mining of transaction database,  $D$ , of Table 5.1 in Example 5.3 using the frequent-pattern growth approach.

The first scan of the database is the same as Apriori, which derives the set of frequent items (1-itemsets) and their support counts (frequencies). Let the minimum support count be 2. The set of frequent items is sorted in the order of descending support count. This resulting set or *list* is denoted  $L$ . Thus, we have  $L = \{\{I2: 7\}, \{I1: 6\}, \{I3: 6\}, \{I4: 2\}, \{I5: 2\}\}$ .

An FP-tree is then constructed as follows. First, create the root of the tree, labeled with “null”. Scan database  $D$  a second time. The items in each transaction are processed in  $L$  order (i.e., sorted according to descending support count) and a branch is created for each transaction. For example, the scan of the first transaction, “T100: I1, I2, I5”, which contains three items (I2, I1, I5 in  $L$  order), leads to the construction of the first branch of the tree with three nodes,  $\langle I2: 1 \rangle$ ,  $\langle I1: 1 \rangle$ , and  $\langle I5: 1 \rangle$ , where I2 is linked as a child of the root, I1 is linked to I2, and I5 is linked to I1. The second transaction, T200, contains the items I2 and I4 in  $L$  order, which would result in a branch where I2 is linked to the root and I4 is linked to I2. However, this branch would share a common **prefix**, I2, with the existing path for T100. Therefore, we instead increment the count of the I2 node by 1, and create a new node,  $\langle I4: 1 \rangle$ , which is linked as a child of  $\langle I2: 2 \rangle$ . In general, when considering the branch to be added for a transaction, the count of each node along a common prefix is incremented by 1, and nodes for the items following the prefix are created and linked accordingly.

To facilitate tree traversal, an item header table is built so that each item points to its occurrences in the tree via a chain of **node-links**. The tree obtained after scanning all of the transactions is shown in Figure 5.7 with the associated node-links. In this way, the problem of mining frequent patterns in databases is transformed to that of mining the FP-tree.

The FP-tree is mined as follows. Start from each frequent length-1 pattern (as an initial **suffix pattern**), construct its **conditional pattern base** (a “subdatabase”, which consists of the set of *prefix paths* in the FP-tree co-occurring with the suffix pattern), then construct its (*conditional*) FP-tree, and perform mining recursively on such a tree. The pattern growth is achieved by the concatenation of the suffix pattern with the frequent patterns generated from a conditional FP-tree.

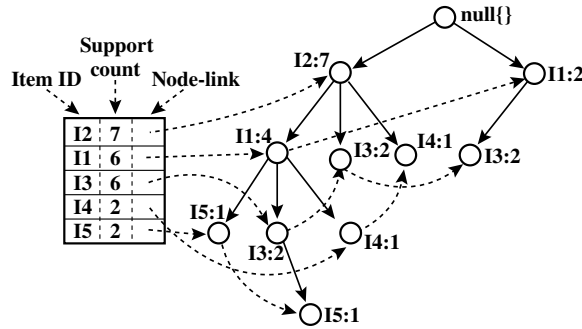


Figure 5.7: An FP-tree registers compressed, frequent pattern information.

Mining of the FP-tree is summarized in Table 5.2 and detailed as follows. We first consider I5 which is the last item in  $L$ , rather than the first. The reason for starting at the end of the list will become apparent as we explain the FP-tree mining process. I5 occurs in two branches of the FP-tree of Figure 5.7. (The occurrences of I5 can easily be found by following its chain of node-links.) The paths formed by these branches are  $\langle I2, I1, I5: 1 \rangle$  and  $\langle I2, I1, I3, I5: 1 \rangle$ . Therefore, considering I5 as a suffix, its corresponding two prefix paths are  $\langle I2, I1: 1 \rangle$  and  $\langle I2, I1, I3: 1 \rangle$ , which form its conditional pattern base. Its conditional FP-tree contains only a single path,  $\langle I2: 2, I1: 2 \rangle$ ; I3 is not included because its support count of 1 is less than the minimum support count. The single path generates all the combinations of frequent patterns:  $\{I2, I5: 2\}$ ,  $\{I1, I5: 2\}$ ,  $\{I2, I1, I5: 2\}$ .

Table 5.2: Mining the FP-tree by creating conditional (sub)-pattern bases.

item	conditional pattern base	conditional FP-tree	frequent patterns generated
I5	$\{\langle I2, I1: 1 \rangle, \langle I2, I1, I3: 1 \rangle\}$	$\langle I2: 2, I1: 2 \rangle$	$\{I2, I5: 2\}, \{I1, I5: 2\}, \{I2, I1, I5: 2\}$
I4	$\{\langle I2, I1: 1 \rangle, \langle I2: 1 \rangle\}$	$\langle I2: 2 \rangle$	$\{I2, I4: 2\}$
I3	$\{\langle I2, I1: 2 \rangle, \langle I2: 2 \rangle, \langle I1: 2 \rangle\}$	$\langle I2: 4, I1: 2 \rangle, \langle I1: 2 \rangle$	$\{I2, I3: 4\}, \{I1, I3: 4\}, \{I2, I1, I3: 2\}$
I1	$\{\langle I2: 4 \rangle\}$	$\langle I2: 4 \rangle$	$\{I2, I1: 4\}$

For I4, its two prefix paths form the conditional pattern base,  $\{\langle I2, I1: 1 \rangle, \langle I2: 1 \rangle\}$ , which generates a single-node conditional FP-tree,  $\langle I2: 2 \rangle$ , and derives one frequent pattern,  $\{I2, I1: 4\}$ . Notice that although I5 follows I4 in the first branch, there is no need to include I5 in the analysis here since any frequent pattern involving I5 is analyzed in the examination of I5.

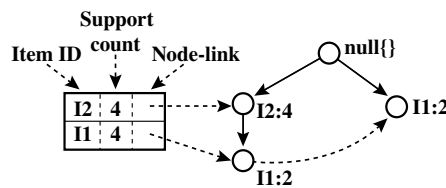


Figure 5.8: The conditional FP-tree associated with the conditional node I3.

Similar to the above analysis, I3's conditional pattern base is  $\{\langle I2, I1: 2 \rangle, \langle I2: 2 \rangle, \langle I1: 2 \rangle\}$ . Its conditional FP-tree has two branches,  $\langle I2: 4, I1: 2 \rangle$  and  $\langle I1: 2 \rangle$ , as shown in Figure 5.8, which generates the set of patterns,  $\{\langle I2, I3: 4 \rangle, \langle I1, I3: 4 \rangle, \langle I2, I1, I3: 2 \rangle\}$ . Finally, I1's conditional pattern base is  $\{\langle I2: 4 \rangle\}$ , whose FP-tree contains only one node,  $\langle I2: 4 \rangle$ , which generates one frequent pattern,  $\{I2, I1: 4\}$ . This mining process is summarized in Figure 5.9. ■

The FP-growth method transforms the problem of finding long frequent patterns to searching for shorter ones recursively and then concatenating the suffix. It uses the least frequent items as a suffix, offering good selectivity.

**Algorithm: FP\_growth.** Mine frequent itemsets using an FP-tree by pattern fragment growth.

**Input:**

- $D$ , a transaction database;
- $min\_sup$ , the minimum support count threshold.

**Output:** The complete set of frequent patterns.

**Method:**

1. The FP-tree is constructed in the following steps.
  - (a) Scan the transaction database  $D$  once. Collect  $F$ , the set of frequent items, and their support counts. Sort  $F$  in support count descending order as  $L$ , the *list* of frequent items.
  - (b) Create the root of an FP-tree, and label it as “null”. For each transaction  $Trans$  in  $D$  do the following. Select and sort the frequent items in  $Trans$  according to the order of  $L$ . Let the sorted frequent item list in  $Trans$  be  $[p|P]$ , where  $p$  is the first element and  $P$  is the remaining list. Call `insert_tree([p|P], T)`, which is performed as follows. If  $T$  has a child  $N$  such that  $N.item-name = p.item-name$ , then increment  $N$ 's count by 1; else create a new node  $N$ , and let its count be 1, its parent link be linked to  $T$ , and its node-link to the nodes with the same *item-name* via the node-link structure. If  $P$  is nonempty, call `insert_tree(P, N)` recursively.
2. The FP-tree is mined by calling `FP_growth(FP_tree, null)`, which is implemented as follows.

```

procedure FP_growth(Tree,  $\alpha$ )
(1)  if Tree contains a single path  $P$  then
(2)    for each combination (denoted as  $\beta$ ) of the nodes in the path  $P$ 
(3)      generate pattern  $\beta \cup \alpha$  with support_count = minimum support count of nodes in  $\beta$ ;
(4)  else for each  $a_i$  in the header of Tree {
(5)    generate pattern  $\beta = a_i \cup \alpha$  with support_count =  $a_i.support\_count$ ;
(6)    construct  $\beta$ 's conditional pattern base and then  $\beta$ 's conditional FP-tree  $Tree_\beta$ ;
(7)    if  $Tree_\beta \neq \emptyset$  then
(8)      call FP_growth(Tree $_\beta$ ,  $\beta$ ); }

```

Figure 5.9: The FP-growth algorithm for discovering frequent itemsets without candidate generation.

The method substantially reduces the search costs.

When the database is large, it is sometimes unrealistic to construct a main memory-based FP-tree. An interesting alternative is to first partition the database into a set of projected databases, and then construct an FP-tree and mine it in each projected database. Such a process can be recursively applied to any projected database if its FP-tree still cannot fit in main memory.

A study on the performance of the FP-growth method shows that it is efficient and scalable for mining both long and short frequent patterns, and is about an order of magnitude faster than the Apriori algorithm. It is also faster than a Tree-Projection algorithm, which recursively projects a database into a tree of projected databases.

### 5.2.5 Mining Frequent Itemsets Using Vertical Data Format

Both the Apriori and FP-growth methods mine frequent patterns from a set of transactions in *TID-itemset* format (that is,  $\{TID : itemset\}$ ), where  $TID$  is a transaction-id and *itemset* is the set of items bought in transaction  $TID$ . This data format is known as **horizontal data format**. Alternatively, data can also be presented in *item-TID\_set* format (that is,  $\{item : TID\_set\}$ ), where *item* is an item name, and  $TID\_set$  is the set of transaction identifiers containing the item. This format is known as **vertical data format**.

In this section, we look at how frequent itemsets can also be mined efficiently using vertical data format, which is the essence of the Eclat algorithm developed by Zaki [Zak00].

**Example 5.6 Mining frequent itemsets using vertical data format.** Consider the horizontal data format of the transaction database,  $D$ , of Table 5.1 in Example 5.3. This can be transformed into the vertical data format shown in Table 5.3 by scanning the data set once.

Table 5.3: The vertical data format of the transaction data set  $D$  of Table 5.1.

<i>itemset</i>	<i>TID_set</i>
I1	{T100, T400, T500, T700, T800, T900}
I2	{T100, T200, T300, T400, T600, T800, T900}
I3	{T300, T500, T600, T700, T800, T900}
I4	{T200, T400}
I5	{T100, T800}

Mining can be performed on this data set by intersecting the *TID\_sets* of every pair of frequent single-items. The minimum support count is 2. Since every single item is frequent in Table 5.3, there are 10 intersections performed in total, which lead to 8 nonempty 2-itemsets as shown in Table 5.4. Notice that since the itemsets {I1, I4} and {I3, I5} each contain only one transaction, they do not belong to the set of frequent 2-itemsets.

Table 5.4: The 2-itemsets in vertical data format.

<i>itemset</i>	<i>TID_set</i>
{I1,I2}	{T100, T400, T800, T900}
{I1,I3}	{T500, T700, T800, T900}
{I1,I4}	{T400}
{I1,I5}	{T100, T800}
{I2,I3}	{T300, T600, T800, T900}
{I2,I4}	{T200, T400}
{I2,I5}	{T100, T800}
{I3,I5}	{T800}

Table 5.5: The 3-itemsets in vertical data format.

<i>itemset</i>	<i>TID_set</i>
{I1,I2,I3}	{T800, T900}
{I1,I2,I5}	{T100, T800}

Based on the Apriori property, a given 3-itemset is a candidate 3-itemset only if every one of its 2-itemset subsets is frequent. The candidate generation process here will generate only two 3-itemsets: {I1, I2, I3} and {I1, I2, I5}. By intersecting the *TID\_sets* of any two corresponding 2-itemsets of these candidate 3-itemsets, it derives Table 5.5, where there are only two frequent 3-itemsets: {I1, I2, I3: 2} and {I1, I2, I5: 2}. ■

Example 5.6 illustrates the process of mining frequent itemsets by exploring the vertical data format. First, we transform the horizontally formatted data to the vertical format by scanning the data set once. The support count of an itemset is simply the length of the *TID\_set* of the itemset. Starting with  $k = 1$ , the frequent  $k$ -itemsets can be used to construct the candidate  $(k + 1)$ -itemsets based on the Apriori property. The computation is done by intersection of the *TID\_sets* of the frequent  $k$ -itemsets to compute the *TID\_sets* of the corresponding  $(k + 1)$ -itemsets. This process repeats, with  $k$  incremented by 1 each time, until no frequent itemsets or no candidate itemsets can be found.

Besides taking advantage of the Apriori property in the generation of candidate  $(k + 1)$ -itemset from frequent  $k$ -itemsets, another merit of this method is that there is no need to scan the database to find the support of  $(k + 1)$  itemsets (for  $k \geq 1$ ). This is because the *TID\_set* of each  $k$ -itemset carries the complete information required for counting such support. However, the *TID\_sets* can be quite long, taking substantial memory space as well as computation time for intersecting the long sets.

To further reduce the cost of registering long *TID\_sets* as well as the subsequent costs of intersections, we can use a technique called *diffset*, which keeps track of only the differences of the *TID\_sets* of a  $(k + 1)$ -itemset and a corresponding  $k$ -itemset. For instance, in Example 5.6 we have {I1} = {T100, T400, T500, T700, T800,

T900}, and  $\{I1, I2\} = \{T100, T400, T800, T900\}$ . The *diffset* between the two, that is,  $\text{diffset}(\{I1, I2\}, \{I1\}) = \{T500, T700\}$ . Thus, rather than recording the four TIDs that make up the intersection of  $\{I1\}$  and  $\{I2\}$ , we can instead use *diffset* to record just two TIDs indicating the difference between  $\{I1\}$  and  $\{I1, I2\}$ . Experiments show that in certain situations, such as when the data set contains many dense and long patterns, this technique can substantially reduce the total cost of vertical format mining of frequent itemsets.

### 5.2.6 Mining Closed Frequent Itemsets

In Section 5.1.2 we saw how frequent itemset mining may generate a huge number of frequent itemsets, especially when the *min\_sup* threshold is set low or when there exist long patterns in the data set. Example 5.2 showed that closed frequent itemsets<sup>9</sup> can substantially reduce the number of patterns generated in frequent itemset mining while preserving the complete information regarding the set of frequent itemsets. That is, from the set of closed frequent itemsets, we can easily derive the set of frequent itemsets and their support. Thus in practice, it is more desirable to mine the set of closed frequent itemsets rather than the set of all frequent itemsets in most cases.

“How can we mine closed frequent itemsets?” A naïve approach would be to first mine the complete set of frequent itemsets and then remove every frequent itemset that is a proper subset of, and carries the same support as, an existing frequent itemset. However, this is quite costly. As shown in Example 5.2, this method would have to first derive  $2^{100} - 1$  frequent itemsets in order to obtain a length-100 frequent itemset, all before it could begin to eliminate redundant itemsets. This is prohibitively expensive. In fact, there exist only a very small number of closed frequent itemsets in the data set of Example 5.2.

A recommended methodology is to search for closed frequent itemsets directly during the mining process. This requires us to prune the search space as soon as we can identify the case of closed itemsets during mining. Pruning strategies include the following:

**Item merging:** *If every transaction containing a frequent itemset  $X$  also contains an itemset  $Y$  but not any proper superset of  $Y$ , then  $X \cup Y$  forms a frequent closed itemset and there is no need to search for any itemset containing  $X$  but no  $Y$ .*

For example, in Table 5.2 of Example 5.5, the projected conditional database for prefix itemset  $\{I5:2\}$  is  $\{\{I2, I1\}, \{I2, I1, I3\}\}$  from which we can see that each of its transactions contains itemset  $\{I2, I1\}$  but no proper superset of  $\{I2, I1\}$ . Itemset  $\{I2, I1\}$  can be merged with  $\{I5\}$  to form the closed itemset,  $\{I5, I2, I1: 2\}$ , and we do not need to mine for closed itemsets that contain  $I5$  but not  $\{I2, I1\}$ .

**Sub-itemset pruning:** *If a frequent itemset  $X$  is a proper subset of an already found frequent closed itemset  $Y$  and  $\text{support\_count}(X) = \text{support\_count}(Y)$ , then  $X$  and all of  $X$ 's descendants in the set enumeration tree cannot be frequent closed itemsets and thus can be pruned.*

Similar to Example 5.2, suppose a transaction database has only two transactions:  $\{\langle a_1, a_2, \dots, a_{100} \rangle, \langle a_1, a_2, \dots, a_{50} \rangle\}$ , and the minimum support count is  $\text{min\_sup} = 2$ . The projection on the first item,  $a_1$ , derives the frequent itemset,  $\{a_1, a_2, \dots, a_{50} : 2\}$ , based on the *itemset merging* optimization. Since  $\text{support}(\{a_2\}) = \text{support}(\{a_1, a_2, \dots, a_{50}\}) = 2$ , and  $\{a_2\}$  is a proper subset of  $\{a_1, a_2, \dots, a_{50}\}$ , there is no need to examine  $a_2$  and its projected database. Similar pruning can be done for  $a_3, \dots, a_{50}$  as well. Thus the mining of closed frequent itemsets in this data set terminates after mining  $a_1$ 's projected database.

**Item skipping:** *In the depth-first mining of closed itemsets, at each level, there will be a prefix itemset  $X$  associated with a header table and a projected database. If a local frequent item  $p$  has the same support in several header tables at different levels, we can safely prune  $p$  from the header tables at higher levels.*

Consider, for example, the transaction database above having only two transactions:  $\{\langle a_1, a_2, \dots, a_{100} \rangle, \langle a_1, a_2, \dots, a_{50} \rangle\}$ , where  $\text{min\_sup} = 2$ . Since  $a_2$  in  $a_1$ 's projected database has the same support as  $a_2$  in the global header table,  $a_2$  can be pruned from the global header table. Similar pruning can be done for  $a_3, \dots, a_{50}$ . There is no need to mine anything more after mining  $a_1$ 's projected database.

<sup>9</sup>Remember that  $X$  is a *closed frequent itemset* in a data set  $S$  if there exists no proper super-itemset  $Y$  such that  $Y$  has the same support count as  $X$  in  $S$ , and  $X$  satisfies minimum support.

Besides pruning the search space in the closed itemset mining process, another important optimization is to perform efficient checking of a newly derived frequent itemset to see whether it is closed since the mining process itself cannot ensure that every generated frequent itemset is closed.

When a new frequent itemset is derived, it is necessary to perform two kinds of closure checking: (1) *superset checking*, which checks if this new frequent itemset is a superset of some already found closed itemsets with the same support, and (2) *subset checking*, which checks if the newly found itemset is a subset of an already found closed itemset with the same support.

If we adopt the *item merging* pruning method under a divide-and-conquer framework, then the superset checking is actually built-in and there is no need to explicitly perform superset checking. This is because if a frequent itemset  $X \cup Y$  is found later than itemset  $X$ , and carries the same support as  $X$ , it must be in  $X$ 's projected database and must have been generated during itemset merging.

To assist in subset checking, a compressed **pattern-tree** can be constructed to maintain the set of closed itemsets mined so far. The pattern-tree is similar in structure to the FP-tree except that all of the closed itemsets found are stored explicitly in the corresponding tree branches. For efficient subset checking, we can use the following property: *If the current itemset  $S_c$  can be subsumed by another already found closed itemset  $S_a$ , then (1)  $S_c$  and  $S_a$  have the same support, (2) the length of  $S_c$  is smaller than that of  $S_a$ , and (3) all of the items in  $S_c$  are contained in  $S_a$ .* Based on this property, a **two-level hash index structure** can be built for fast accessing of the pattern-tree: The first level uses the identifier of the last item in  $S_c$  as a hash key (since this identifier must be within the branch of  $S_c$ ), and the second level uses the support of  $S_c$  as a hash key (since  $S_c$  and  $S_a$  have the same support). This will substantially speed up the subset checking process.

The above discussion illustrates methods for efficient mining of closed frequent itemsets. *“Can we extend these methods for efficient mining of maximal frequent itemsets?”* Since maximal frequent itemsets share many similarities with closed frequent itemsets, many of the optimization techniques developed here can be extended to mining maximal frequent itemsets. However, we leave this as an exercise for interested readers.

## 5.3 Mining Various Kinds of Association Rules

We have studied efficient methods for mining frequent itemsets and association rules. In this section, we consider additional application requirements by extending our scope to include mining multilevel association rules, multidimensional association rules, and quantitative association rules in transactional and/or relational databases and data warehouses. *Multilevel association rules* involve concepts at different levels of abstraction. *Multidimensional association rules* involve more than one dimension or predicate (e.g., rules relating what a customer *buys* as well as the customer's *age*.) *Quantitative association rules* involve numeric attributes that have an implicit ordering among values (e.g., *age*).

### 5.3.1 Mining Multilevel Association Rules

For many applications, it is difficult to find strong associations among data items at low or primitive levels of abstraction due to the sparsity of data at those levels. Strong associations discovered at high levels of abstraction may represent common sense knowledge. Moreover, what may represent common sense to one user may seem novel to another. Therefore, data mining systems should provide capabilities for mining association rules at multiple levels of abstraction, with sufficient flexibility for easy traversal among different abstraction spaces.

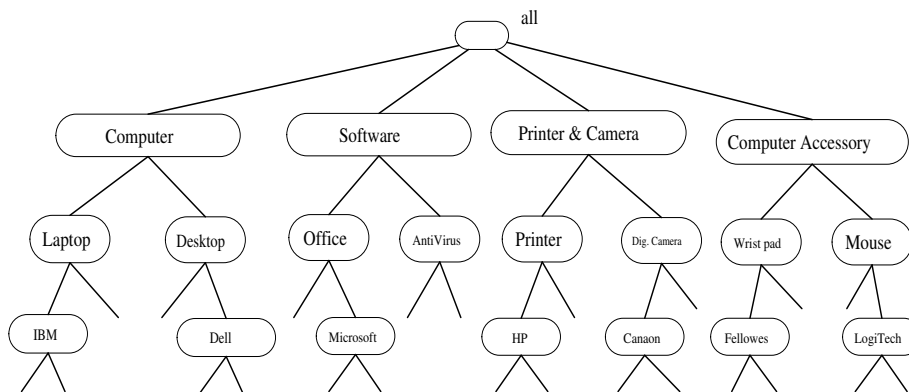
Let's examine the following example.

**Example 5.7 Mining multilevel association rules.** Suppose we are given the task-relevant set of transactional data in Table 5.6 for sales in an *AllElectronics* store, showing the items purchased for each transaction. The concept hierarchy for the items is shown in Figure 5.10. A concept hierarchy defines a sequence of mappings from a set of low-level concepts to higher-level, more general concepts. Data can be generalized by replacing low-level concepts within the data by their higher-level concepts, or *ancestors*, from a concept hierarchy. The concept hierarchy of

Figure 5.10 has five levels, respectively referred to as levels 0 to 4, starting with level 0 at the root node for all (the most general abstraction level). Here, level 1 includes *computer*, *software*, *printer&camera*, and *computer accessory*, level 2 includes *laptop computer*, *desktop computer*, *office software*, *antivirus software*, ..., and level 3 includes *IBM desktop computer*, ..., *Microsoft office software*, and so on. Level 4 represents the most specific abstraction level of this hierarchy. Concept hierarchies for categorical attributes are often implicit within the database schema, in which case they may be automatically generated using methods such as those described in Chapter 2. For our example, the concept hierarchy of Figure 5.10 was generated from data on product specifications. Concept hierarchies for numerical attributes can be generated using discretization techniques, many of which were introduced in Chapter 2. Alternatively, concept hierarchies may be specified by users familiar with the data, such as store managers in the case of our example.

Table 5.6: Task-relevant data,  $D$ .

$TID$	$items\ purchased$
T100	IBM-ThinkPad-T40/2373, HP-Photosmart-7660
T200	Microsoft-Office-Professional-2003, Microsoft-Plus!-Digital-Media
T300	Logitech-MX700-Cordless-Mouse, Fellowes-Wrist-Rest
T400	Dell-Dimension-XPS, Canon-PowerShot-S400
T500	IBM-ThinkPad-R40/P4M, Symantec-Norton-Antivirus-2003
...	...

Figure 5.10: A concept hierarchy for *AllElectronics* computer items.

The items in Table 5.6 are at the lowest level of the concept hierarchy of Figure 5.10. It is difficult to find interesting purchase patterns at such raw or primitive-level data. For instance, if “*IBM-ThinkPad-R40/P4M*” or “*Symantec-Norton-Antivirus-2003*” each occurs in a very small fraction of the transactions, then it can be difficult to find strong associations involving these specific items. Few people may buy these items together, making it unlikely that the itemset will satisfy minimum support. However, we would expect that it is easier to find strong associations between generalized abstractions of these items, such as between “*IBM laptop computer*” and “*antivirus software*”.

Association rules generated from mining data at multiple levels of abstraction are called **multiple-level** or **multilevel association rules**. Multilevel association rules can be mined efficiently using concept hierarchies under a support-confidence framework. In general, a top-down strategy is employed, where counts are accumulated for the calculation of frequent itemsets at each concept level, starting at the concept level 1 and working downwards in the hierarchy, towards the more specific concept levels, until no more frequent itemsets can be found. For each level, any algorithm for discovering frequent itemsets may be used, such as Apriori or its variations. A number of

variations to this approach are described below, where each variation involves “playing” with the support threshold in a slightly different way. The variations are illustrated in Figures 5.11 and 5.12, where nodes indicate an item or itemset that has been examined, and nodes with thick borders indicate that an examined item or itemset is frequent.

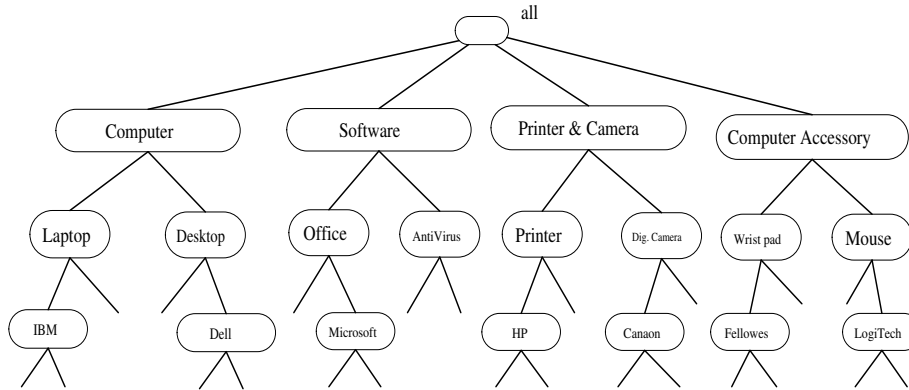


Figure 5.11: Multilevel mining with uniform support.

- **Using uniform minimum support for all levels** (referred to as **uniform support**): The same minimum support threshold is used when mining at each level of abstraction. For example, in Figure 5.11, a minimum support threshold of 5% is used throughout (e.g., for mining from “*computer*” down to “*laptop computer*”). Both “*computer*” and “*laptop computer*” are found to be frequent, while “*desktop computer*” is not.

When a uniform minimum support threshold is used, the search procedure is simplified. The method is also simple in that users are required to specify only one minimum support threshold. An Apriori-like optimization technique can be adopted, based on the knowledge that an ancestor is a superset of its descendants: The search avoids examining itemsets containing any item whose ancestors do not have minimum support.

The uniform support approach, however, has some difficulties. It is unlikely that items at lower levels of abstraction will occur as frequently as those at higher levels of abstraction. If the minimum support threshold is set too high, it could miss some meaningful associations occurring at low abstraction levels. If the threshold is set too low, it may generate many uninteresting associations occurring at high abstraction levels. This provides the motivation for the following approach.

- **Using reduced minimum support at lower levels** (referred to as **reduced support**): Each level of abstraction has its own minimum support threshold. The deeper the level of abstraction, the smaller the corresponding threshold is. For example, in Figure 5.12, the minimum support thresholds for levels 1 and 2 are 5% and 3%, respectively. In this way, “*computer*”, “*laptop computer*”, and “*desktop computer*” are all considered frequent.
- **Using item or group-based minimum support** (referred to as **group-based support**): Since users or experts often have insight as to which groups are more important than others, it is sometimes more desirable to set up user-specific, item or group-based minimal support thresholds when mining multilevel rules. For example, a user could set up the minimum support thresholds based on product price, or on items of interest, such as by setting particularly low support thresholds for *laptop computers* and *flash drives* in order to pay particular attention to the association patterns containing items in these categories.

Notice that the Apriori property may not always hold uniformly across all the items when mining under reduced support and group-based support. However, efficient methods can be developed based on the extension of the property. The details are left as an exercise for interested readers.

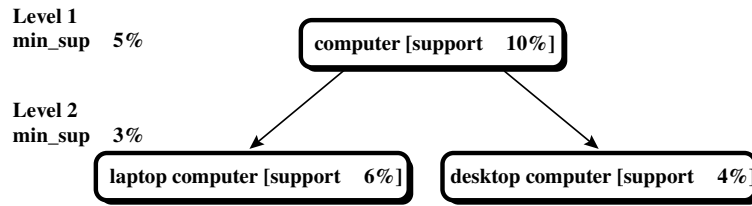


Figure 5.12: Multilevel mining with reduced support.

A serious side-effect of mining multilevel association rules is its generation of many redundant rules across multiple levels of abstraction due to the “ancestor” relationships among items. For example, consider the following rules where “*laptop computer*” is an ancestor of “*IBM laptop computer*” based on the concept hierarchy of Figure 5.10, and where  $X$  is a variable representing customers who purchased items in *AllElectronics* transactions.

$$\text{buys}(X, \text{“laptop computer”}) \Rightarrow \text{buys}(X, \text{“HP printer”}) \quad [\text{support} = 8\%, \text{confidence} = 70\%] \quad (5.10)$$

$$\text{buys}(X, \text{“IBM laptop computer”}) \Rightarrow \text{buys}(X, \text{“HP printer”}) \quad [\text{support} = 2\%, \text{confidence} = 72\%] \quad (5.11)$$

“If Rules (5.10) and (5.11) are both mined, then how useful is the latter rule?” you may wonder. “Does it really provide any novel information?” If the latter, less general rule does not provide new information, it should be removed. Let’s have a look at how this may be determined. A rule  $R1$  is an **ancestor** of a rule  $R2$ , if  $R1$  can be obtained by replacing the items in  $R2$  by their ancestors in a concept hierarchy. For example, Rule (5.10) is an ancestor of Rule (5.11) since “*laptop computer*” is an ancestor of “*IBM laptop computer*”. Based on this definition, a rule can be considered redundant if its support and confidence are close to their “expected” values, based on an ancestor of the rule. As an illustration, suppose that Rule (5.10) has a 70% confidence and 8% support, and that about one quarter of all “*laptop computer*” sales are for “*IBM laptop computers*”. We may expect Rule (5.11) to have a confidence of around 70% (since all data samples of “*IBM laptop computer*” are also samples of “*laptop computer*”) and a support of around 2% (i.e.,  $8\% \times \frac{1}{4}$ ). If this is indeed the case, then Rule (5.11) is not interesting since it does not offer any additional information and is less general than Rule (5.10).

### 5.3.2 Mining Multidimensional Association Rules from Relational Databases and Data Warehouses

So far in this chapter, we have studied association rules that imply a single predicate, that is, the predicate *buys*. For instance, in mining our *AllElectronics* database, we may discover the Boolean association rule

$$\text{buys}(X, \text{“digital camera”}) \Rightarrow \text{buys}(X, \text{“HP printer”}) \quad (5.12)$$

Following the terminology used in multidimensional databases, we refer to each distinct predicate in a rule as a dimension. Hence, we can refer to Rule (5.12) as a **single-dimensional** or **intradimensional association rule** since it contains a single distinct predicate (e.g., *buys*) with multiple occurrences (i.e., the predicate occurs more than once within the rule). As we have seen in the previous sections of this chapter, such rules are commonly mined from transactional data.

Suppose, however, that rather than using a transactional database, sales and related information are stored in a relational database or data warehouse. Such data stores are multidimensional, by definition. For instance,

in addition to keeping track of the items purchased in sales transactions, a relational database may record other attributes associated with the items, such as the quantity purchased or the price, or the branch location of the sale. Additional relational information regarding the customers who purchased the items, such as customer age, occupation, credit rating, income, and address, may also be stored. Considering each database attribute or warehouse dimension as a predicate, we can therefore mine association rules containing *multiple* predicates, such as

$$age(X, "20\dots29") \wedge occupation(X, "student") \Rightarrow buys(X, "laptop") \quad (5.13)$$

Association rules that involve two or more dimensions or predicates can be referred to as **multidimensional association rules**. Rule (5.13) contains three predicates (*age*, *occupation*, and *buys*), each of which occurs *only once* in the rule. Hence, we say that it has **no repeated predicates**. Multidimensional association rules with no repeated predicates are called **interdimensional association rules**. We can also mine multidimensional association rules with repeated predicates, which contain multiple occurrences of some predicates. These rules are called **hybrid-dimensional association rules**. An example of such a rule is the following, where the predicate *buys* is repeated:

$$age(X, "20\dots29") \wedge buys(X, "laptop") \Rightarrow buys(X, "HP printer") \quad (5.14)$$

Note that database attributes can be categorical or quantitative. **Categorical** attributes have a finite number of possible values, with no ordering among the values (e.g., *occupation*, *brand*, *color*). Categorical attributes are also called **nominal** attributes, since their values are “names of things.” **Quantitative** attributes are numeric and have an implicit ordering among values (e.g., *age*, *income*, *price*). Techniques for mining multidimensional association rules can be categorized into two basic approaches regarding the treatment of quantitative attributes.

In the first approach, *quantitative attributes are discretized using predefined concept hierarchies*. This discretization occurs prior to mining. For instance, a concept hierarchy for *income* may be used to replace the original numeric values of this attribute by interval labels, such as “0...20K”, “21K...30K”, “31K...40K”, and so on. Here, discretization is *static* and predetermined. Chapter 2 on data preprocessing gave several techniques for discretizing numeric attributes. The discretized numeric attributes, with their interval labels, can then be treated as categorical attributes (where each interval is considered a category). We refer to this as **mining multidimensional association rules using static discretization of quantitative attributes**.

In the second approach, *quantitative attributes are discretized or clustered into “bins” based on the distribution of the data*. These bins may be further combined during the mining process. The discretization process is *dynamic* and established so as to satisfy some mining criteria, such as maximizing the confidence of the rules mined. Because this strategy treats the numeric attribute values as quantities rather than as predefined ranges or categories, association rules mined from this approach are also referred to as **(dynamic) quantitative association rules**.

Let’s study each of these approaches for mining multidimensional association rules. For simplicity, we confine our discussion to interdimensional association rules. Note that rather than searching for frequent itemsets (as is done for single-dimensional association rule mining), in multidimensional association rule mining we search for frequent *predicate sets*. A ***k*-predicate set** is a set containing *k* conjunctive predicates. For instance, the set of predicates {*age*, *occupation*, *buys*} from Rule (5.13) is a 3-predicate set. Similar to the notation used for itemsets, we use the notation  $L_k$  to refer to the set of frequent *k*-predicate sets.

## 1. Mining Multidimensional Association Rules Using Static Discretization of Quantitative Attributes

Quantitative attributes, in this case, are discretized prior to mining using predefined concept hierarchies or data discretization techniques, where numeric values are replaced by interval labels. Categorical attributes may also

be generalized to higher conceptual levels if desired. If the resulting task-relevant data are stored in a relational table, then any of the frequent itemset mining algorithms we have discussed can easily be modified so as to find all frequent predicate sets rather than frequent itemsets. In particular, instead of searching on only one attribute like *buys*, we need to search through all of the relevant attributes, treating each attribute-value pair as an itemset.

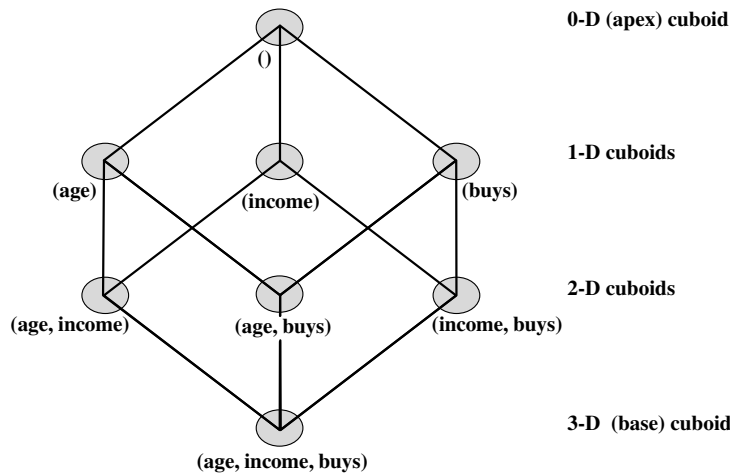


Figure 5.13: Lattice of cuboids, making up a 3-D data cube. Each cuboid represents a different group-by. The base cuboid contains the three predicates *age*, *income*, and *buys*. [TO EDITOR For consistency with rest of book, please kindly italicize all instances of *age*, *income*, and *buys*.]

Alternatively, the transformed multidimensional data may be used to construct a *data cube*. Data cubes are well suited for the mining of multidimensional association rules: They store aggregates (such as counts), in multidimensional space, which is essential for computing the support and confidence of multidimensional association rules. An overview of data cube technology was presented in Chapter 3. Detailed algorithms for data cube computation were given in Chapter 4. Figure 5.13 shows the lattice of cuboids defining a data cube for the dimensions *age*, *income*, and *buys*. The cells of an  $n$ -dimensional cuboid can be used to store the support counts of the corresponding  $n$ -predicate sets. The base cuboid aggregates the task-relevant data by *age*, *income*, and *buys*; the 2-D cuboid,  $(age, income)$ , aggregates by *age* and *income*; the 0-D (apex) cuboid contains the total number of transactions in the task-relevant data, and so on.

Due to the ever-increasing use of data warehouse and OLAP technology, it is possible that a data cube containing the dimensions that are of interest to the user may already exist, fully materialized. If this is the case, we can simply fetch the corresponding aggregate values and return the rules needed using a rule generation algorithm (Section 5.2.2). Notice that even in this case, the Apriori property can still be used to prune the search space. If a given  $k$ -predicate set has support *sup* which does not satisfy minimum support, then further exploration of this set should be terminated. This is because any more specialized version of the  $k$ -itemset will have support no greater than *sup* and, therefore, will not satisfy minimum support either. In cases where no relevant data cube exists for the mining task, we must create one on the fly. This becomes an iceberg cube computation problem, where the minimum support threshold is taken as the iceberg condition (Chapter 4).

## 2. Mining Quantitative Association Rules

Quantitative association rules are multidimensional association rules in which the numeric attributes are *dynamically* discretized during the mining process so as to satisfy some mining criteria, such as maximizing the confidence or compactness of the rules mined. In this section, we will focus specifically on how to mine quantitative association rules having two quantitative attributes on the left-hand side of the rule, and one categorical attribute on the right-hand side of the rule. That is,

$$A_{quan1} \wedge A_{quan2} \Rightarrow A_{cat}$$

where  $A_{quan1}$  and  $A_{quan2}$  are tests on quantitative attribute intervals (where the intervals are dynamically determined), and  $A_{cat}$  tests a categorical attribute from the task-relevant data. Such rules have been referred to as **two-dimensional quantitative association rules**, since they contain two quantitative dimensions. For instance, suppose you are curious about the association relationship between pairs of quantitative attributes, like customer age and income, and the type of television (such as *high definition TV*, i.e., *HDTV*) that customers like to buy. An example of such a 2-D quantitative association rule is

$$age(X, "30...39") \wedge income(X, "42K...48K") \Rightarrow buys(X, "HDTV") \quad (5.15)$$

“How can we find such rules?” Let’s look at an approach used in a system called **ARCS** (Association Rule Clustering System), which borrows ideas from image processing. Essentially, this approach maps pairs of quantitative attributes onto a 2-D grid for tuples satisfying a given categorical attribute condition. The grid is then searched for clusters of points, from which the association rules are generated. The following steps are involved in ARCS:

**Binning:** Quantitative attributes can have a very wide range of values defining their domain. Just think about how big a 2-D grid would be if we plotted *age* and *income* as axes, where each possible value of *age* was assigned a unique position on one axis, and similarly, each possible value of *income* was assigned a unique position on the other axis! To keep grids down to a manageable size, we instead partition the ranges of quantitative attributes into intervals. These intervals are dynamic in that they may later be further combined during the mining process. The partitioning process is referred to as **binning**, that is, where the intervals are considered “bins.” Three common binning strategies are

- **equal-width binning**, where the interval size of each bin is the same,
- **equal-frequency binning**, where each bin has approximately the same number of tuples assigned to it, and
- **clustering-based binning**, where clustering is performed on the quantitative attribute to group *neighboring points* (judged based on various distance measures) into the same bin.

ARCS uses equal-width binning, where the bin size for each quantitative attribute is input by the user. A 2-D array for each possible bin combination involving both quantitative attributes is created. Each array cell holds the corresponding count distribution for each possible class of the categorical attribute of the rule right-hand side. By creating this data structure, the task-relevant data need only be scanned once. The same 2-D array can be used to generate rules for any value of the categorical attribute, based on the same two quantitative attributes. Binning is also discussed in Chapter 2.

**Finding frequent predicate sets:** Once the 2-D array containing the count distribution for each category is set up, this can be scanned in order to find the frequent predicate sets (those satisfying minimum support) that also satisfy minimum confidence. Strong association rules can then be generated from these predicate sets, using a rule generation algorithm like that described in Section 5.2.2.

**Clustering the association rules:** The strong association rules obtained in the previous step are then mapped to a 2-D grid. Figure 5.14 shows a 2-D grid for 2-D quantitative association rules predicting the condition  $buys(X, "HDTV")$  on the rule right-hand side, given the quantitative attributes *age* and *income*. The four Xs correspond to the rules

$$age(X, 34) \wedge income(X, "31K...40K") \Rightarrow buys(X, "HDTV") \quad (5.16)$$

$$age(X, 35) \wedge income(X, "31K...40K") \Rightarrow buys(X, "HDTV") \quad (5.17)$$

$$age(X, 34) \wedge income(X, "41K...50K") \Rightarrow buys(X, "HDTV") \quad (5.18)$$

$$age(X, 35) \wedge income(X, "41K...50K") \Rightarrow buys(X, "HDTV") \quad (5.19)$$

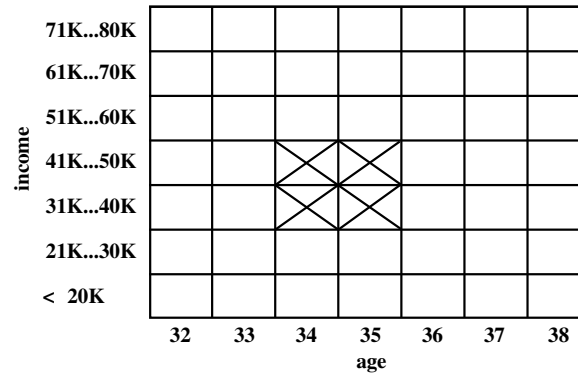


Figure 5.14: A 2-D grid for tuples representing customers who purchase high-definition TVs.

“Can we find a simpler rule to replace the above four rules?” Notice that these rules are quite “close” to one another, forming a rule cluster on the grid. Indeed, the four rules can be combined or “clustered” together to form the following simpler rule, which subsumes and replaces the above four rules:

$$age(X, "34..35") \wedge income(X, "31K..50K") \Rightarrow buys(X, "HDTV") \quad (5.20)$$

ARCS employs a clustering algorithm for this purpose. The algorithm scans the grid, searching for rectangular clusters of rules. In this way, bins of the quantitative attributes occurring within a rule cluster may be further combined, and hence, further dynamic discretization of the quantitative attributes occurs.

The grid-based technique described here assumes that the initial association rules can be clustered into rectangular regions. Prior to performing the clustering, smoothing techniques can be used to help remove noise and outliers from the data. Rectangular clusters may oversimplify the data. Alternative approaches have been proposed, based on other shapes of regions that tend to better fit the data, yet require greater computation effort.

A non-grid-based technique has been proposed to find quantitative association rules that are more general, where any number of quantitative and categorical attributes can appear on either side of the rules. In this technique, quantitative attributes are dynamically partitioned using equal-frequency binning, and the partitions are combined based on a measure of *partial completeness*, which quantifies the information lost due to partitioning. For references on these alternatives to ARCS, see the bibliographic notes.

## 5.4 From Association Mining to Correlation Analysis

Most association rule mining algorithms employ a support-confidence framework. Often, many interesting rules can be found using low support thresholds. Although minimum support and confidence thresholds *help* weed out or exclude the exploration of a good number of uninteresting rules, many rules so generated are still not interesting to the users. Unfortunately, this is especially true *when mining at low support thresholds or mining for long patterns*. This has been one of the major bottlenecks for successful application of association rule mining.

In this section, we first look at how even strong association rules can be uninteresting and misleading. We then discuss how the support-confidence framework can be supplemented with additional interestingness measures based on statistical significance and correlation analysis.

### 5.4.1 Strong Rules Are Not Necessarily Interesting: An Example

Whether or not a rule is interesting can be assessed either subjectively or objectively. Ultimately, only the user can judge if a given rule is interesting, and this judgment, being subjective, may differ from one user to another.

However, objective interestingness measures, 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 strong association rules are really interesting?” Let’s examine the following example.

**Example 5.8 A misleading “strong” association rule.** Suppose we are interested in analyzing transactions at *AllElectronics* with respect to the purchase of computer games and videos. Let *game* refer to the transactions containing computer games, and *video* refer to those containing videos. Of the 10,000 transactions analyzed, the data show that 6000 of the customer transactions included computer games, while 7500 included videos, and 4000 included both computer games and videos. Suppose that a data mining program for discovering association rules is run on the data, using a minimum support of, say, 30% and a minimum confidence of 60%. The following association rule is discovered:

$$\text{buys}(X, \text{“computer games”}) \Rightarrow \text{buys}(X, \text{“videos”}) \quad [\text{support} = 40\%, \text{confidence} = 66\%] \quad (5.21)$$

Rule (5.21) is a strong association rule and would therefore be reported, since its support value of  $\frac{4000}{10,000} = 40\%$  and confidence value of  $\frac{4000}{6000} = 66\%$  satisfy the minimum support and minimum confidence thresholds, respectively. However, Rule (5.21) is misleading since the probability of purchasing videos is 75%, which is even larger than 66%. In fact, computer games and videos are negatively associated because the purchase of one of these items actually decreases the likelihood of purchasing the other. Without fully understanding this phenomenon, we could easily make unwise business decisions based on Rule (5.21). ■

The above example also illustrates that the confidence of a rule  $A \Rightarrow B$  can be deceiving in that it is only an *estimate* of the conditional probability of itemset  $B$  given itemset  $A$ . It does not measure the real strength (or lack of strength) of the correlation and implication between  $A$  and  $B$ . Hence, alternatives to the support-confidence framework can be useful in mining interesting data relationships.

### 5.4.2 From Association Analysis to Correlation Analysis

As we have seen above, the support and confidence measures are insufficient at filtering out uninteresting association rules. To tackle this weakness, a correlation measure can be used to augment the support-confidence framework for association rules. This leads to *correlation rules* of the form

$$A \Rightarrow B \quad [\text{support}, \text{confidence}, \text{correlation}] \quad (5.22)$$

That is, a correlation rule is measured not only by its support and confidence but also by the correlation between itemsets  $A$  and  $B$ . There are many different correlation measures to choose from. In this section, we study various correlation measures to determine which would be good for mining large data sets.

**Lift** is a simple correlation measure that is given as follows. The occurrence of itemset  $A$  is **independent** of the occurrence of itemset  $B$  if  $P(A \cup B) = P(A)P(B)$ ; otherwise itemsets  $A$  and  $B$  are **dependent** and **correlated** as events. This definition can easily be extended to more than two itemsets. The **lift** between the occurrence of  $A$  and  $B$  can be measured by computing

$$\text{lift}(A, B) = \frac{P(A \cup B)}{P(A)P(B)}. \quad (5.23)$$

If the resulting value of Equation (5.23) is less than 1, then the occurrence of  $A$  is *negatively correlated* with the occurrence of  $B$ . If the resulting value is greater than 1, then  $A$  and  $B$  are *positively correlated*, meaning that the occurrence of one implies the occurrence of the other. If the resulting value is equal to 1, then  $A$  and  $B$  are *independent* and there is no correlation between them.

Equation (5.23) is equivalent to  $P(B|A)/P(B)$ , or  $\text{conf}(A \Rightarrow B)/\text{sup}(B)$ , which is also referred as the *lift* of the association (or correlation) rule  $A \Rightarrow B$ . In other words, it assesses the degree of which the occurrence of one

“lifts” the occurrence of the other. For example, if  $A$  corresponds to the sale of computer games and  $B$  corresponds to the sale of videos, then given the current market conditions, the sale of games is said to increase or “lift” the likelihood of the sale of videos by a factor of the value returned by Equation (5.23).

Let’s go back to the computer game and video data of Example 5.8.

Table 5.7: A  $2 \times 2$  contingency table summarizing the transactions with respect to game and video purchases.

	<i>game</i>	$\overline{game}$	$\Sigma_{row}$
<i>video</i>	4,000	3,500	7,500
$\overline{video}$	2,000	500	2,500
$\Sigma_{col}$	6,000	4,000	10,000

**Example 5.9 Correlation analysis using lift.** To help filter out misleading “strong” associations of the form  $A \Rightarrow B$  from the data of Example 5.8, we need to study how the two itemsets,  $A$  and  $B$ , are correlated. Let  $\overline{game}$  refer to the transactions of Example 5.8 that do not contain computer games, and  $\overline{video}$  refer to those that do not contain videos. The transactions can be summarized in a *contingency table*, as shown in Table 5.7. From the table, we can see that the probability of purchasing a computer game is  $P(\{game\}) = 0.60$ , the probability of purchasing a video is  $P(\{video\}) = 0.75$ , and the probability of purchasing both is  $P(\{game, video\}) = 0.40$ . By Equation (5.23), the lift of Rule (5.21) is  $P(\{game, video\}) / (P(\{game\}) \times P(\{video\})) = 0.40 / (0.60 \times 0.75) = 0.89$ . Since this value is less than 1, there is a negative correlation between the occurrence of  $\{game\}$  and  $\{video\}$ . The numerator is the likelihood of a customer purchasing both, while the denominator is what the likelihood would have been if the two purchases were completely independent. Such a negative correlation cannot be identified by a support-confidence framework. ■

The second correlation measure that we study is the  $\chi^2$  measure, which was introduced in Chapter 2 (Equation 2.9). To compute the  $\chi^2$  value, we take the squared difference between the observed and expected value for a slot ( $A$  and  $B$  pair) in the contingency table, divided by the expected value. This amount is summed for all slots of the contingency table. Let’s perform a  $\chi^2$  analysis of the above example.

Table 5.8: The above contingency table, now shown with the expected values.

	<i>game</i>	$\overline{game}$	$\Sigma_{row}$
<i>video</i>	4,000 (4,500)	3,500 (3,000)	7,500
$\overline{video}$	2,000 (1,500)	500 (1,000)	2,500
$\Sigma_{col}$	6,000	4,000	10,000

**Example 5.10 Correlation analysis using  $\chi^2$ .** To compute the correlation using  $\chi^2$  analysis, we need the observed value and expected value (displayed in parenthesis) for each slot of the contingency table, as shown in Table 5.8. From the table, we can compute the  $\chi^2$  value as follows:

$$\chi^2 = \sum \frac{(\text{observed} - \text{expected})^2}{\text{expected}} = \frac{(4000 - 4500)^2}{4500} + \frac{(3500 - 3000)^2}{3000} + \frac{(2000 - 1500)^2}{1500} + \frac{(500 - 1000)^2}{1000} = 555.6.$$

Since the  $\chi^2$  value is greater than one, and the observed value of the slot ( $game, video$ ) = 4000 which is less than the expected value 4500, *buying game* and *buying video* are *negatively correlated*, which is consistent with the conclusion derived from the analysis of the *lift* measure in Example 5.9. ■

Let’s examine two other correlation measures, *all-confidence* and *cosine*, as defined below.

Given an itemset  $X = \{i_1, i_2, \dots, i_k\}$ , the **all\_confidence** of  $X$  is defined as,

$$all\_conf(X) = \frac{sup(X)}{max\_item\_sup(X)} = \frac{sup(X)}{max\{sup(i_j) | \forall i_j \in X\}} \quad (5.24)$$

where  $max\{sup(i_j) | \forall i_j \in X\}$  is the maximum (single) item support of all the items in  $X$ , and hence is called the **max\_item\_sup** of the itemset  $X$ . The *all\_confidence* of  $X$  is the minimal confidence among the set of rules  $i_j \rightarrow X - i_j$ , where  $i_j \in X$ .

Given two itemsets  $A$  and  $B$ , the **cosine** measure of  $A$  and  $B$  is defined as,

$$cosine(A, B) = \frac{P(A \cup B)}{\sqrt{P(A) \times P(B)}} = \frac{sup(A \cup B)}{\sqrt{sup(A) \times sup(B)}} \quad (5.25)$$

The *cosine* measure can be viewed as a harmonized *lift* measure: the two formulae are similar except that for cosine, the *square root* is taken on the product of the probabilities of  $A$  and  $B$ . This is an important difference, however, because by taking the square root, the cosine value is only influenced by the supports of  $A$ ,  $B$  and  $A \cup B$ , and not by the total number of transactions.

“Are these two measures better than lift and  $\chi^2$  in assessing the correlation relationship?” To answer this question, we first look at some other “typical” data sets before returning to our running example.

**Example 5.11 Comparison of four correlation measures on typical data sets.** The correlation relationships between the purchases of two items, *milk* and *coffee*, can be examined by summarizing their purchase history in the form of Table 5.9, a  $2 \times 2$  contingency table, where an entry such as *mc* represents the number of transactions containing both milk and coffee. For the derivation of *all\_confidence*, we let itemset  $X = \{m, c\}$  so that  $sup(X) = mc$  in Equation (5.24).

Table 5.9: A  $2 \times 2$  contingency table for two items.

	<i>milk</i>	$\overline{milk}$	$\Sigma_{row}$
<i>coffee</i>	<i>mc</i>	$\overline{mc}$	<i>c</i>
$\overline{coffee}$	$\overline{m\bar{c}}$	$\overline{\overline{m\bar{c}}}$	$\bar{c}$
$\Sigma_{col}$	<i>m</i>	$\overline{m}$	$\Sigma$

Table 5.10: Comparison of four correlation measures using contingency tables for different data sets.

Data Set	<i>mc</i>	$\overline{mc}$	$\overline{m\bar{c}}$	$\overline{\overline{m\bar{c}}}$	<i>all_conf.</i>	<i>cosine</i>	<i>lift</i>	$\chi^2$
$A_1$	1,000	100	100	100,000	0.91	0.91	83.64	83,452.6
$A_2$	1,000	100	100	10,000	0.91	0.91	9.26	9,055.7
$A_3$	1,000	100	100	1,000	0.91	0.91	1.82	1,472.7
$A_4$	1,000	100	100	0	0.91	0.91	0.99	9.9
$B_1$	1,000	1,000	1,000	1,000	0.50	0.50	1.00	0.0
$C_1$	100	1,000	1,000	100,000	0.09	0.09	8.44	670.0
$C_2$	1,000	100	10,000	100,000	0.09	0.29	9.18	8,172.8
$C_3$	1	1	100	10,000	0.01	0.07	50.0	48.5

Table 5.10 shows a set of transactional data sets with their corresponding contingency tables and values for each of the four correlation measures. From the table, we see that  $m$  and  $c$  are positively correlated in  $A_1$  through  $A_4$ , independent in  $B_1$ , and negatively correlated in  $C_1$  through  $C_3$ . All four measures are good indicators for the independent case,  $B_1$ . *Lift* and  $\chi^2$  are poor indicators of the other relationships, whereas *all\_confidence* and *cosine* are good indicators. Another interesting fact is that between *all\_confidence* and *cosine*, *cosine* is the better

indicator when  $\overline{mc}$  and  $m\overline{c}$  are not balanced. This is because *cosine* considers the supports of both  $A$  and  $B$  whereas *all\_confidence* considers only the maximal support. Such a difference can be seen by comparing  $C_1$  and  $C_2$ .  $C_1$  should be more negatively correlated for  $m$  and  $c$  than  $C_2$  since  $mc$  is the smallest among the three counts,  $mc$ ,  $\overline{mc}$  and  $m\overline{c}$ , in  $C_1$ . However, this can only be seen by checking the *cosine* measure since the *all\_confidence* values are identical in  $C_1$  and  $C_2$ .

“Why are lift and  $\chi^2$  so poor at distinguishing correlation relationships in the above transactional data sets?” To answer this, we have to consider the *null-transactions*. A **null-transaction** is a transaction that does not contain any of the itemsets being examined. In our example,  $\overline{mc}$  represents the number of null-transactions. Lift and  $\chi^2$  have difficulty distinguishing correlation relationships because they are both strongly influenced by  $\overline{mc}$ . Typically, the number of null-transactions can outweigh the number of individual purchases, since many people may buy neither milk nor coffee. On the other hand, *all\_confidence* and *cosine* values are good indicators of correlation because their definitions remove the influence of  $\overline{mc}$ , i.e., they are not influenced by the number of null-transactions. ■

A measure is **null-invariant** if its value is free from the influence of null-transactions. Null-invariance is an important property for measuring correlations in large transaction databases. Among the four above measures, *all\_confidence* and *cosine* are null-invariant measures.

“Are *all\_confidence* and *cosine* the best at assessing correlation in all cases?” Let’s examine the game-and-video examples again.

**Example 5.12 Comparison of four correlation measures on game-and-video data.** We revisit Examples 5.8 to 5.10. Let  $D_1$  be the original game ( $g$ ) and video ( $v$ ) data set from Table 5.7. We add two more data sets,  $D_0$  and  $D_2$ , where  $D_0$  has zero null-transactions, and  $D_2$  has 10,000 null-transactions (instead of only 500 as in  $D_1$ ). The values of all four correlation measures are shown in Table 5.11.

Table 5.11: Comparison of the four correlation measures for game-and-video data sets.

Data Set	$gv$	$\overline{gv}$	$g\overline{v}$	$\overline{g\overline{v}}$	<i>all_conf.</i>	<i>cosine</i>	<i>lift</i>	$\chi^2$
$D_0$	4,000	3,500	2,000	0	0.53	0.60	0.84	1,477.8
$D_1$	4,000	3,500	2,000	500	0.53	0.60	0.89	555.6
$D_2$	4,000	3,500	2,000	10,000	0.53	0.60	1.73	2,913.0

In Table 5.11,  $gv$ ,  $\overline{gv}$ , and  $g\overline{v}$  remain the same in  $D_0$ ,  $D_1$  and  $D_2$ . However, lift and  $\chi^2$  change from rather negative to rather positive correlations, whereas *all\_confidence* and *cosine* have the nice null-invariant property, and their values remain the same in all cases. Unfortunately, we cannot precisely assert that a set of items are positively or negatively correlated when the value of *all\_confidence* or *cosine* is around 0.5. Strictly based on whether the value is greater than 0.5, we will claim that  $g$  and  $v$  are positively correlated in  $D_1$ , however, it has been shown that they are negatively correlated by the lift and  $\chi^2$  analysis. Therefore, a good strategy is to perform the *all\_confidence* or *cosine* analysis first and when the result shows that they are *weakly* positively/negatively correlated, other analyses can be performed to assist in obtaining a more complete picture. ■

Besides null-invariance, another nice feature of the *all\_confidence* measure is that it has the Apriori-like *downward closure* property. That is, if a pattern is *all-confident* (i.e., passing a minimal *all\_confidence* threshold), so is every one of its subpatterns. In other words, if a pattern is not all-confident, further growth (or specialization) of this pattern will never satisfy the minimal *all\_confidence* threshold. This is obvious since according to Equation (5.24), adding any item into an itemset  $X$  will never increase  $sup(X)$ , never decrease  $max\_item\_sup(X)$ , and thus never increase *all\_conf(X)*. This property makes Apriori-like pruning possible: we can prune any patterns that cannot satisfy the minimal *all\_confidence* threshold during the growth of all-confident patterns in mining.

In summary, the use of only support and confidence measures to mine associations results in the generation of a large number of rules, most of which are uninteresting to the user. Instead, we can augment the support-confidence

framework with a correlation measure, resulting in the mining of *correlation rules*. The added measure substantially reduces the number of rules generated, and leads to the discovery of more meaningful rules. However, there seems to be no single correlation measure that works well for all cases. Besides those introduced in this section, many other interestingness measures have been studied in the literature. Unfortunately, most such measures do not have the null-invariance property. Since large data sets typically have many null-transactions, it is important to consider the null-invariance property when selecting appropriate interestingness measures in the correlation analysis. Our analysis shows that both *all\_confidence* and *cosine* are good correlation measures for large applications, although it is wise to augment them with additional tests, such as *lift*, when the test result is not quite conclusive.

## 5.5 Constraint-Based Association Mining

A data mining process may uncover thousands of rules from a given set of data, most of which end up being unrelated or uninteresting to the users. Often, users have a good sense of which “direction” of mining may lead to interesting patterns and the “form” of the patterns or rules they would like to find. Thus, a good heuristic is to have the users specify such intuition or expectations as *constraints* to confine the search space. This strategy is known as **constraint-based mining**. The constraints can include the following:

- **Knowledge type constraints:** These specify the type of knowledge to be mined, such as association or correlation.
- **Data constraints:** These specify the set of task-relevant data.
- **Dimension/level constraints:** These specify the desired dimensions (or attributes) of the data, or levels of the concept hierarchies, to be used in mining.
- **Interestingness constraints:** These specify thresholds on statistical measures of rule interestingness, such as support, confidence, and correlation.
- **Rule constraints:** These specify the form of rules to be mined. Such constraints may be expressed as metarules (rule templates), as the maximum or minimum number of predicates that can occur in the rule antecedent or consequent, or as relationships among attributes, attribute values, and/or aggregates.

The above constraints can be specified using a high-level declarative data mining query language and user interface.

The first four of the above types of constraints have already been addressed in earlier parts of this book and chapter. In this section, we discuss the use of *rule constraints* to focus the mining task. This form of constraint-based mining allows users to describe the rules that they would like to uncover, thereby making the data mining process more *effective*. In addition, a sophisticated mining query optimizer can be used to exploit the constraints specified by the user, thereby making the mining process more *efficient*. Constraint-based mining encourages interactive exploratory mining and analysis. In Section 5.5.1, you will study metarule-guided mining, where syntactic rule constraints are specified in the form of rule templates. Section 5.5.2 discusses the use of additional rule constraints, specifying set/subset relationships, constant initiation of variables, and aggregate functions. For ease of discussion, we assume that the user is searching for association rules. The procedures presented can easily be extended to the mining of correlation rules by adding a correlation measure of interestingness to the support-confidence framework, as described in the previous section.

### 5.5.1 Metarule-Guided Mining of Association Rules

“*How are metarules useful?*” Metarules allow users to specify the syntactic form of rules that they are interested in mining. The rule forms can be used as constraints to help improve the efficiency of the mining process. Metarules may be based on the analyst’s experience, expectations, or intuition regarding the data, or automatically generated based on the database schema.

**Example 5.13 Metarule-guided mining.** Suppose that as a market analyst for *AllElectronics*, you have access to the data describing customers (such as customer age, address, and credit rating) as well as the list of customer transactions. You are interested in finding associations between customer traits and the items that customers buy. However, rather than finding *all* of the association rules reflecting these relationships, you are particularly interested only in determining which pairs of customer traits promote the sale of office software. A metarule can be used to specify this information describing the form of rules you are interested in finding. An example of such a metarule is

$$P_1(X, Y) \wedge P_2(X, W) \Rightarrow \text{buys}(X, \text{"office software"}) \quad (5.26)$$

where  $P_1$  and  $P_2$  are **predicate variables** that are instantiated to attributes from the given database during the mining process,  $X$  is a variable representing a customer, and  $Y$  and  $W$  take on values of the attributes assigned to  $P_1$  and  $P_2$ , respectively. Typically, a user will specify a list of attributes to be considered for instantiation with  $P_1$  and  $P_2$ . Otherwise, a default set may be used.

In general, a metarule forms a hypothesis regarding the relationships that the user is interested in probing or confirming. The data mining system can then search for rules that match the given metarule. For instance, Rule (5.27) matches or **complies with** Metarule (5.26).

$$\text{age}(X, \text{"30...39"}) \wedge \text{income}(X, \text{"41K...60K"}) \Rightarrow \text{buys}(X, \text{"office software"}) \quad (5.27)$$

■

*“How can metarules be used to guide the mining process?”* Let’s examine this problem closely. Suppose that we wish to mine interdimensional association rules, such as in the example above. A metarule is a rule template of the form

$$P_1 \wedge P_2 \wedge \dots \wedge P_l \Rightarrow Q_1 \wedge Q_2 \wedge \dots \wedge Q_r \quad (5.28)$$

where  $P_i$  ( $i = 1, \dots, l$ ) and  $Q_j$  ( $j = 1, \dots, r$ ) are either instantiated predicates or predicate variables. Let the number of predicates in the metarule be  $p = l + r$ . In order to find interdimensional association rules satisfying the template,

- We need to find all frequent  $p$ -predicate sets,  $L_p$ .
- We must also have the support or count of the  $l$ -predicate subsets of  $L_p$  in order to compute the confidence of rules derived from  $L_p$ .

This is a typical case of mining multidimensional association rules, which was discussed in Section 5.3.2. By extending such methods using techniques described in the following section, we can derive efficient methods for meta-rule guided mining.

### 5.5.2 Constraint Pushing: Mining Guided by Rule Constraints

Rule constraints specify expected set/subset relationships of the variables in the mined rules, constant initiation of variables, and aggregate functions. Users typically employ their knowledge of the application or data to specify rule constraints for the mining task. These rule constraints may be used together with, or as an alternative to, metarule-guided mining. In this section, we examine rule constraints as to how they can be used to make the mining process more efficient. Let’s study an example where rule constraints are used to mine hybrid-dimensional association rules.

**Example 5.14 A closer look at mining guided by rule constraints.** Suppose that *AllElectronics* has a sales multidimensional database with the following interrelated relations:

- $sales(customer\_name, item\_name, TID)$
- $lives\_in(customer\_name, region, city)$
- $item(item\_name, group, price)$
- $transaction(TID, day, month, year)$

where  $lives\_in$ ,  $item$ , and  $transaction$  are three dimension tables, linked to the fact table  $sales$  via three keys,  $customer\_name$ ,  $item\_name$ , and  $TID$  ( $transaction\_id$ ), respectively.

Our association mining query is to “Find the sales of what cheap items (where the sum of the prices is less than \$100) that may promote the sales of what expensive items (where the minimum price is \$500) of the same group for Chicago customers in 2004”. This can be expressed in the DMQL data mining query language as follows, where each line of the query has been enumerated to aid in our discussion.

- (1) mine associations as
- (2)  $lives\_in(C, -, “Chicago”) \wedge sales^+(C, ?\{I\}, \{S\}) \Rightarrow sales^+(C, ?\{J\}, \{T\})$
- (3) from sales
- (4) where  $S.year = 2004$  and  $T.year = 2004$  and  $I.group = J.group$
- (5) group by  $C, I.group$
- (6) having  $sum(I.price) < 100$  and  $min(J.price) \geq 500$
- (7) with support threshold = 1%
- (8) with confidence threshold = 50%

Before we discuss the rule constraints, let us have a closer look at the above query. Line 1 is a knowledge type constraint, where association patterns are to be discovered. Line 2 specifies a metarule. This is an abbreviated form for the following metarule for hybrid-dimensional association rules (multidimensional association rules where the repeated predicate here is  $sales$ ):

$$\begin{aligned}
& lives\_in(C, -, “Chicago”) \\
& \wedge sales(C, ?I_1, S_1) \wedge \dots \wedge sales(C, ?I_k, S_k) \wedge I = \{I_1, \dots, I_k\} \wedge S = \{S_1, \dots, S_k\} \\
& \Rightarrow sales(C, ?J_1, T_1) \wedge \dots \wedge sales(C, ?J_m, T_m) \wedge J = \{J_1, \dots, J_m\} \wedge T = \{T_1, \dots, T_m\}
\end{aligned}$$

which means that one or more  $sales$  records in the form of “ $sales(C, ?I_1, S_1) \wedge \dots \wedge sales(C, ?I_k, S_k)$ ” will reside at the rule antecedent (left-hand side), and the question mark “?” means that only  $item\_name$ ,  $I_1, \dots, I_k$  need be printed out. “ $I = \{I_1, \dots, I_k\}$ ” means that all the  $I$ s at the antecedent are taken from a set  $I$ , obtained from the SQL-like where clause of line 4. Similar notational conventions are used at the consequent (right-hand side).

The metarule may allow the generation of association rules like the following:

$$\begin{aligned}
& lives\_in(C, -, “Chicago”) \wedge sales(C, “Census\_CD”, -) \wedge \\
& sales(C, “MS/Office”, -) \Rightarrow sales(C, “MS/SQLServer”, -), \quad [1.5\%, 68\%]
\end{aligned} \tag{5.29}$$

which means that if a customer in Chicago bought “Census\_CD” and “MS/Office”, it is likely (with a probability of 68%) that the customer also bought “MS/SQLServer”, and 1.5% of all of the customers bought all three.

Data constraints are specified in the “ $lives\_in(-, -, “Chicago”)$ ” portion of the metarule (i.e., all the customers who live in Chicago), and in line 3, which specifies that only the fact table,  $sales$ , need be explicitly referenced. In such a multidimensional database, variable reference is simplified. For example, “ $S.year = 2004$ ” is equivalent to the SQL statement “from  $sales$   $S$ ,  $transaction$   $T$  where  $S.TID = T.TID$  and  $T.year = 2004$ ”. All three dimensions ( $lives\_in$ ,  $item$ , and  $transaction$ ) are used. Level constraints are as follows: for  $lives\_in$ , we consider just  $customer\_name$  since  $region$  is not referenced and  $city = “Chicago”$  is only used in the selection; for  $item$ , we consider the levels  $item\_name$  and  $group$  since they are used in the query; and for  $transaction$ , we are only concerned with  $TID$  since  $day$  and  $month$  are not referenced and  $year$  is used only in the selection.

Rule constraints include most portions of the *where* (line 4) and *having* (line 6) clauses, such as “*S.year = 2004*”, “*T.year = 2004*”, “*I.group = J.group*”, “*sum(I.price) ≤ 100*”, and “*min(J.price) ≥ 500*”. Finally, lines 7 and 8 specify two interestingness constraints (i.e., thresholds), namely, a minimum support of 1% and a minimum confidence of 50%. ■

Dimension/level constraints and interestingness constraints can be applied after mining, to filter out discovered rules, although it is generally more efficient and less expensive to use them *during* mining, to help prune the search space. Dimension/level constraints were discussed in Section 5.3, and interestingness constraints have been discussed throughout this chapter. Let’s focus now on rule constraints.

“How can we use rule constraints to prune the search space? More specifically, what kind of rule constraints can be ‘pushed’ deep into the mining process and still ensure the completeness of the answer returned for a mining query?”

Rule constraints can be classified into the following five categories with respect to frequent itemset mining: (1) *antimonotonic*, (2) *monotonic*, (3) *succinct*, (4) *convertible*, and (5) *inconvertible*. For each category, we will use an example to show its characteristics and explain how such kinds of constraints can be used in the mining process.

The first category of constraints is **antimonotonic**. Consider the rule constraint “*sum(I.price) ≤ 100*” of Example 5.14. Suppose we are using the Apriori framework, which at each iteration  $k$ , explores itemsets of size  $k$ . If the price summation of the items in an itemset is no less than 100, this itemset can be pruned from the search space, since adding more items into the set will only make it more expensive and thus will never satisfy the constraint. In other words, if an itemset does not satisfy this rule constraint, none of its supersets can satisfy the constraint. If a rule constraint obeys this property, it is **antimonotonic**. Pruning by antimonotonic constraints can be applied at each iteration of Apriori-style algorithms to help improve the efficiency of the overall mining process while guaranteeing completeness of the data mining task.

The Apriori property, which states that all nonempty subsets of a frequent itemset must also be frequent, is antimonotonic. If a given itemset does not satisfy minimum support, none of its supersets can. This property is used at each iteration of the Apriori algorithm to reduce the number of candidate itemsets examined, thereby reducing the search space for association rules.

Other examples of antimonotonic constraints include “*min(J.price) ≥ 500*”, “*count(I) ≤ 10*”, and so on. Any itemset that violates either of these constraints can be discarded since adding more items to such itemsets can never satisfy the constraints. Note that a constraint such as “*avg(I.price) ≤ 100*” is not antimonotonic. For a given itemset that does not satisfy this constraint, a superset created by adding some (cheap) items may result in satisfying the constraint. Hence, pushing this constraint inside the mining process will not guarantee completeness of the data mining task. A list of SQL-primitives-based constraints is given in the first column of Table 5.12. The antimonotonicity of the constraints is indicated in the second column of the table. To simplify our discussion, only existence operators (e.g., =, ∈, but not ≠, ∉) and comparison (or containment) operators with equality (e.g., ≤, ⊆) are given.

The second category of constraints is **monotonic**. If the rule constraint in Example 5.14 were “*sum(I.price) ≥ 100*”, the constraint-based processing method would be quite different. If an itemset  $I$  satisfies the constraint, that is, the sum of the prices in the set is no less than 100, further addition of more items to  $I$  will increase cost and will always satisfy the constraint. Therefore, further testing of this constraint on itemset  $I$  becomes redundant. In other words, if an itemset satisfies this rule constraint, so do all of its supersets. If a rule constraint obeys this property, it is **monotonic**. Similar rule monotonic constraints include “*min(I.price) ≤ 10*,” “*count(I) ≥ 10*”, and so on. The monotonicity of the list of SQL-primitives-based constraints is indicated in the third column of Table 5.12.

The third category is **succinct constraints**. For this category of constraints, we can *enumerate all and only those sets that are guaranteed to satisfy the constraint*. That is, if a rule constraint is **succinct**, we can directly generate precisely the sets that satisfy it, even before support counting begins. This avoids the substantial overhead of the generate-and-test paradigm. In other words, such constraints are *precounting prunable*. For example, the constraint “*min(J.price) ≥ 500*” in Example 5.14 is succinct. This is because we can explicitly and precisely

Table 5.12: Characterization of commonly used SQL-based constraints.

Constraint	Antimonotonic	Monotonic	Succinct
$v \in S$	no	yes	yes
$S \supseteq V$	no	yes	yes
$S \subseteq V$	yes	no	yes
$\min(S) \leq v$	no	yes	yes
$\min(S) \geq v$	yes	no	yes
$\max(S) \leq v$	yes	no	yes
$\max(S) \geq v$	no	yes	yes
$\text{count}(S) \leq v$	yes	no	weakly
$\text{count}(S) \geq v$	no	yes	weakly
$\text{sum}(S) \leq v$ ( $\forall a \in S, a \geq 0$ )	yes	no	no
$\text{sum}(S) \geq v$ ( $\forall a \in S, a \geq 0$ )	no	yes	no
$\text{range}(S) \leq v$	yes	no	no
$\text{range}(S) \geq v$	no	yes	no
$\text{avg}(S) \theta v, \theta \in \{\leq, \geq\}$	convertible	convertible	no
$\text{support}(S) \geq \xi$	yes	no	no
$\text{support}(S) \leq \xi$	no	yes	no
$\text{all\_confidence}(S) \geq \xi$	yes	no	no
$\text{all\_confidence}(S) \leq \xi$	no	yes	no

generate all the sets of items satisfying the constraint. Specifically, such a set must contain at least one item whose price is no less than \$500. It is of the form  $S_1 \cup S_2$ , where  $S_1 \neq \emptyset$  is a subset of the set of all those items with prices no less than \$500, and  $S_2$ , possibly empty, is a subset of the set of all those items with prices no greater than \$500. Because there is a precise “formula” for generating all of the sets satisfying a succinct constraint, there is no need to iteratively check the rule constraint during the mining process. The succinctness of the list of SQL-primitives-based constraints is indicated in the fourth column of Table 5.12.<sup>10</sup>

The fourth category is **convertible constraints**. Some constraints belong to none of the above three categories. However, if the items in the itemset are arranged in a particular order, the constraint may become monotonic or antimonotonic with regard to the frequent itemset mining process. For example, the constraint “ $\text{avg}(I.\text{price}) \leq 100$ ” is neither antimonotonic nor monotonic. However, if items in a transaction are added to an itemset in price-ascending order, the constraint becomes *antimonotonic*, because if an itemset  $I$  violates the constraint (i.e., with an average price greater than \$100), further addition of more expensive items into the itemset will never make it satisfy the constraint. Similarly, if items in a transaction are added to an itemset in price-descending order, it becomes *monotonic*, because if the itemset satisfies the constraint (i.e., with an average price no greater than \$100), adding cheaper items into the current itemset will still make the average price no greater than \$100. Aside from “ $\text{avg}(S) \leq v$ ,” and “ $\text{avg}(S) \geq v$ ,” given in Table 5.12, there are many other convertible constraints, such as “ $\text{variance}(S) \geq v$ ,” “ $\text{standard\_deviation}(S) \geq v$ ,” and so on.

Note that the above discussion does not imply that every constraint is convertible. For example, “ $\text{sum}(S) \theta v$ ,” where  $\theta \in \{\leq, \geq\}$  and each element in  $S$  could be of any real value, is not convertible. Therefore, there is yet a fifth category of constraints, called **inconvertible constraints**. The good news is that although there still exist some tough constraints that are not convertible, most simple SQL expressions with built-in SQL aggregates belong to one of the first four categories to which efficient constraint mining methods can be applied.

## 5.6 Summary

- The discovery of frequent patterns, association and correlation relationships among huge amounts of data is useful in selective marketing, decision analysis, and business management. A popular area of application

<sup>10</sup>For constraint  $\text{count}(S) \leq v$  (and similarly for  $\text{count}(S) \geq v$ ), we can have a member generation function based on a cardinality constraint, i.e.,  $\{X \mid X \subseteq \text{Itemset} \wedge |X| \leq v\}$ . Member generation in this manner takes a different flavor and thus is called *weakly succinct*.

is **market basket analysis**, which studies the buying habits of customers by searching for sets of items that are frequently purchased together (or in sequence). **Association rule mining** consists of first finding **frequent itemsets** (set of items, such as  $A$  and  $B$ , satisfying a *minimum support threshold*, or percentage of the task-relevant tuples), from which **strong** association rules in the form of  $A \Rightarrow B$  are generated. These rules also satisfy a *minimum confidence threshold* (a prespecified probability of satisfying  $B$  under the condition that  $A$  is satisfied). Associations can be further analyzed to uncover **correlation rules**, which convey statistical correlations between itemsets  $A$  and  $B$ .

- **Frequent pattern mining** can be categorized in many different ways according to various criteria, such as the following:
  1. Based on the **completeness** of patterns to be mined, categories of frequent pattern mining include mining the *complete set of frequent itemsets*, the *closed frequent itemsets*, the *maximal frequent itemsets*, *constrained frequent itemsets*, and so on.
  2. Based on the **levels** and **dimensions** of data involved in the rule, categories can include the mining of *single-level association rules*, *multilevel association rules*, *single-dimensional association rules*, and *multidimensional association rules*.
  3. Based on the **types of values** handled in the rule, the categories can include mining *Boolean association rules*, and *quantitative association rules*.
  4. Based on the **kinds of rules** to be mined, categories include mining *association rules*, and *correlation rules*.
  5. Based on the **kinds of patterns** to be mined, frequent pattern mining can be classified into *frequent itemset mining*, *sequential pattern mining*, *structured pattern mining*, and so on. This chapter has focused on frequent itemset mining.
- Many efficient and scalable algorithms have been developed for **frequent itemset mining**, from which association and correlation rules can be derived. These algorithms can be classified into three categories: (1) *Apriori-like algorithms*, (2) *frequent-pattern growth*-based algorithms, such as FP-growth, and (3) *algorithms that use the vertical data format*.
- The **Apriori algorithm** is a seminal algorithm for mining frequent itemsets for Boolean association rules. It explores the level-wise mining Apriori property that *all nonempty subsets of a frequent itemset must also be frequent*. At the  $k$ th iteration (for  $k \geq 2$ ), it forms frequent  $k$ -itemset candidates based on the frequent  $(k - 1)$ -itemsets, and scans the database once to find the *complete* set of frequent  $k$ -itemsets,  $L_k$ .  
Variations involving hashing and transaction reduction can be used to make the procedure more efficient. Other variations include partitioning the data (mining on each partition and then combining the results), and sampling the data (mining on a subset of the data). These variations can reduce the number of data scans required to as little as two or one.
- **Frequent pattern growth (FP-growth)** is a method of mining frequent itemsets without candidate generation. It constructs a highly compact data structure (an *FP-tree*) to compress the original transaction database. Rather than employing the generate-and-test strategy of Apriori-like methods, it focuses on frequent pattern (fragment) growth, which avoids costly candidate generation, resulting in greater efficiency.
- **Mining frequent itemsets using vertical data format (Eclat)** is a method that transforms a given data set of transactions in the horizontal data format of *TID-itemset* into the vertical data format of *item-TID\_set*. It mines the transformed data set by TID\_set intersections based on the Apriori property and additional optimization techniques, such as *diffset*.
- Methods for mining frequent itemsets can be extended for the mining of **closed frequent itemsets** (from which the set of frequent itemsets can easily be derived). These incorporate additional optimization techniques, such as *item merging*, *sub-itemset pruning* and *item skipping*, as well as efficient *subset checking* of generated itemsets in a *pattern-tree*.

- Mining frequent itemsets and associations has been **extended in various ways** to include mining *multilevel association rules* and *multidimensional association rules*.
- **Multilevel association rules** can be mined using several strategies, based on how minimum support thresholds are defined at each level of abstraction, such as *uniform support*, *reduced support* and *group-based support*. Redundant multilevel (descendent) association rules can be eliminated if their support and confidence are close to their expected values, based on their corresponding ancestor rules.
- Techniques for mining **multidimensional association rules** can be categorized according to their treatment of quantitative attributes. First, quantitative attributes may be *discretized statically*, based on predefined concept hierarchies. Data cubes are well suited to this approach, since both the data cube and quantitative attributes can make use of concept hierarchies. Second, **quantitative association rules** can be mined where quantitative attributes are discretized dynamically based on binning and/or clustering, where “adjacent” association rules may be further combined by clustering to generate concise and meaningful rules.
- Not all strong association rules are interesting. It is more effective to mine items that are statistically correlated. Therefore, association rules should be augmented with a correlation measure to generate more meaningful **correlation rules**. There are several correlation measures to choose from, including **lift**,  $\chi^2$ , **all\_confidence**, and **cosine**. A measure is **null-invariant** if its value is free from the influence of **null-transactions** (i.e., *transactions that do not contain any of the itemsets being examined*). Since large databases typically have numerous null-transactions, a null-invariant correlation measure should be used, such as *all\_confidence* or *cosine*. When interpreting correlation measure values, it is important to understand their implications and limitations.
- **Constraint-based rule mining** allow users to focus the search for rules by providing metarules (i.e., pattern templates) and additional mining constraints. Such mining is facilitated with the use of a declarative data mining query language and user interface, and poses great challenges for mining query optimization. Rule constraints can be classified into five categories: **antimonotonic**, **monotonic**, **succinct**, **convertible**, and **inconvertible**. Constraints belonging to the first four of these categories can be used during frequent itemset mining to guide the process, leading to more efficient and effective mining.
- **Association rules should not be used directly for prediction** without further analysis or domain knowledge. They do not necessarily indicate causality. They are, however, a helpful starting point for further exploration, making them a popular tool for understanding data. The application of frequent patterns to classification, cluster analysis, and other data mining tasks will be discussed in subsequent chapters.

## 5.7 Exercises

1. The Apriori algorithm makes use of *prior knowledge* of subset support properties.
  - (a) Prove that all nonempty subsets of a frequent itemset must also be frequent.
  - (b) Prove that the support of any nonempty subset  $s'$  of itemset  $s$  must be at least as great as the support of  $s$ .
  - (c) Given frequent itemset  $l$  and subset  $s$  of  $l$ , prove that the confidence of the rule “ $s' \Rightarrow (l - s')$ ” cannot be more than the confidence of “ $s \Rightarrow (l - s)$ ”, where  $s'$  is a subset of  $s$ .
  - (d) A *partitioning* variation of Apriori subdivides the transactions of a database  $D$  into  $n$  nonoverlapping partitions. Prove that any itemset that is frequent in  $D$  must be frequent in at least one partition of  $D$ .
2. Section 5.2.2 describes a method for *generating association rules* from frequent itemsets. Propose a more efficient method. Explain why it is more efficient than the one proposed in Section 5.2.2. (*Hint*: Consider incorporating the properties of Exercise 5.1(b) and 5.1(c) into your design.)
3. A database has 5 transactions. Let  $min\_sup = 60\%$  and  $min\_conf = 80\%$ .

<i>TID</i>	<i>items_bought</i>
T100	{M, O, N, K, E, Y}
T200	{D, O, N, K, E, Y}
T300	{M, A, K, E}
T400	{M, U, C, K, Y}
T500	{C, O, O, K, I, E}

- (a) Find all frequent itemsets using Apriori and FP-growth, respectively. Compare the efficiency of the two mining processes.
- (b) List all of the *strong* association rules (with support  $s$  and confidence  $c$ ) matching the following metarule, where  $X$  is a variable representing customers, and  $item_i$  denotes variables representing items (e.g., “A”, “B”, etc.):

$$\forall x \in \text{transaction}, \text{buys}(X, item_1) \wedge \text{buys}(X, item_2) \Rightarrow \text{buys}(X, item_3) \quad [s, c]$$

4. (**Implementation project**) Implement three *frequent itemset mining* algorithms introduced in this chapter: (1) Apriori [AS94b], (2) FP-growth [HPY00], and (3) Eclat [Zak00] (mining using vertical data format), using a programming language that you are familiar with, such as C++ or Java. Compare the performance of each algorithm with various kinds of large data sets. Write a report to analyze the situations (such as data size, data distribution, minimal support threshold setting, and pattern density) where one algorithm may perform better than the others, and state why.
5. A database has four transactions. Let  $min\_sup = 60\%$  and  $min\_conf = 80\%$ .

<i>cust_ID</i>	<i>TID</i>	<i>items_bought</i> (in the form of <i>brand-item_category</i> )
01	T100	{King’s-Crab, Sunset-Milk, Dairyland-Cheese, Best-Bread}
02	T200	{Best-Cheese, Dairyland-Milk, Goldenfarm-Apple, Tasty-Pie, Wonder-Bread}
01	T300	{Westcoast-Apple, Dairyland-Milk, Wonder-Bread, Tasty-Pie}
03	T400	{Wonder-Bread, Sunset-Milk, Dairyland-Cheese}

- (a) At the granularity of *item\_category* (e.g.,  $item_i$  could be “Milk”), for the following rule template,

$$\forall X \in \text{transaction}, \text{buys}(X, item_1) \wedge \text{buys}(X, item_2) \Rightarrow \text{buys}(X, item_3) \quad [s, c]$$

list the frequent  $k$ -itemset for the largest  $k$ , and *all* of the *strong* association rules (with their support  $s$  and confidence  $c$ ) containing the frequent  $k$ -itemset for the largest  $k$ .

- (b) At the granularity of *brand-item\_category* (e.g.,  $item_i$  could be “Sunset-Milk”), for the following rule template,

$$\forall X \in \text{customer}, \text{buys}(X, item_1) \wedge \text{buys}(X, item_2) \Rightarrow \text{buys}(X, item_3)$$

list the frequent  $k$ -itemset for the largest  $k$  (but do not print any rules).

6. Suppose that a large store has a transaction database that is *distributed* among four locations. Transactions in each component database have the same format, namely  $T_j : \{i_1, \dots, i_m\}$ , where  $T_j$  is a transaction identifier, and  $i_k$  ( $1 \leq k \leq m$ ) is the identifier of an item purchased in the transaction. Propose an efficient algorithm to mine global association rules (without considering multilevel associations). You may present your algorithm in the form of an outline. Your algorithm should not require shipping all of the data to one site and should not cause excessive network communication overhead.
7. Suppose that frequent itemsets are saved for a large transaction database,  $DB$ . Discuss how to efficiently mine the (global) association rules under the same minimum support threshold, if a set of new transactions, denoted as  $\Delta DB$ , is (*incrementally*) added in?
8. [*Contributed by Tao Cheng*] Most frequent pattern mining algorithms consider only distinct items in a transaction. However, multiple occurrences of an item in the same shopping basket, such as four cakes and three jugs of milk, can be important in transaction data analysis. How can one mine frequent itemsets efficiently considering multiple occurrences of items? Propose modifications to the well-known algorithms, such as Apriori and FP-growth, to adapt to such a situation.

9. (**Implementation project**) Implement three *closed frequent itemset mining* methods (1) A-Close [PBT99] (based on an extension of Apriori [AS94b]), (2) CLOSET+ [WHP03] (based on an extension of FP-growth [HPY00]), and (3) CHARM [ZH02] (based on an extension of Eclat [Zak00]). Compare their performance with various kinds of large data sets. Write a report to answer the following questions:
- Why is mining the set of closed frequent itemsets often more desirable than mining the complete set of frequent itemsets (based on your experiments on the same data set as Exercise 4)?
  - Analyze at what situations (such as data size, data distribution, minimal support threshold setting, and pattern density) and why that one algorithm performs better than the others.
10. Suppose that a data relation describing students at *Big-University* has been generalized to the generalized relation  $R$  in Table 5.13.

Table 5.13: Generalized relation for Exercise 5.9.

major	status	age	nationality	gpa	count
French	M.A	over_30	Canada	2.8_3.2	3
cs	junior	16...20	Europe	3.2_3.6	29
physics	M.S	26...30	Latin_America	3.2_3.6	18
engineering	Ph.D	26...30	Asia	3.6_4.0	78
philosophy	Ph.D	26...30	Europe	3.2_3.6	5
French	senior	16...20	Canada	3.2_3.6	40
chemistry	junior	21...25	USA	3.6_4.0	25
cs	senior	16...20	Canada	3.2_3.6	70
philosophy	M.S	over_30	Canada	3.6_4.0	15
French	junior	16...20	USA	2.8_3.2	8
philosophy	junior	26...30	Canada	2.8_3.2	9
philosophy	M.S	26...30	Asia	3.2_3.6	9
French	junior	16...20	Canada	3.2_3.6	52
math	senior	16...20	USA	3.6_4.0	32
cs	junior	16...20	Canada	3.2_3.6	76
philosophy	Ph.D	26...30	Canada	3.6_4.0	14
philosophy	senior	26...30	Canada	2.8_3.2	19
French	Ph.D	over_30	Canada	2.8_3.2	1
engineering	junior	21...25	Europe	3.2_3.6	71
math	Ph.D	26...30	Latin_America	3.2_3.6	7
chemistry	junior	16...20	USA	3.6_4.0	46
engineering	junior	21...25	Canada	3.2_3.6	96
French	M.S	over_30	Latin_America	3.2_3.6	4
philosophy	junior	21...25	USA	2.8_3.2	8
math	junior	16...20	Canada	3.6_4.0	59

Let the concept hierarchies be as follows:

$status :$              $\{freshman, sophomore, junior, senior\} \in undergraduate.$   
                           $\{M.Sc., M.A., Ph.D.\} \in graduate.$   
 $major :$              $\{physics, chemistry, math\} \in science.$   
                           $\{cs, engineering\} \in appl\_sciences.$   
                           $\{French, philosophy\} \in arts.$   
 $age :$                  $\{16...20, 21...25\} \in young.$   
                           $\{26...30, over\_30\} \in old.$   
 $nationality :$        $\{Asia, Europe, Latin\_America\} \in foreign.$

$\{U.S.A., Canada\} \in North\_America.$

Let the minimum support threshold be 20% and the minimum confidence threshold be 50% (at each of the levels).

- (a) Draw the concept hierarchies for *status*, *major*, *age*, and *nationality*.
- (b) Write a program to find the set of strong multilevel association rules in  $R$  using *uniform support* for all levels, for the following rule template,

$$\forall S \in R, P(S, x) \wedge Q(S, y) \Rightarrow gpa(S, z) \quad [s, c]$$

where  $P, Q \in \{status, major, age, nationality\}$ .

- (c) Use the program to find the set of strong multilevel association rules in  $R$  using *level-cross filtering by single items*. In this strategy, an item at the  $i$ th level is examined if and only if its parent node at the  $(i - 1)$ th level in the concept hierarchy is frequent. That is, if a node is frequent, its children will be examined; otherwise, its descendants are pruned from the search. Use a reduced support of 10% for the lowest abstraction level, for the preceding rule template.
11. Propose and outline a **level-shared mining** approach to mining multilevel association rules in which each item is encoded by its level position, and an initial scan of the database collects the count for each item *at each concept level*, identifying frequent and subfrequent items. Comment on the processing cost of mining multilevel associations with this method in comparison to mining single-level associations.
  12. (**Implementation project**) Many techniques have been proposed to further improve the performance of frequent-itemset mining algorithms. Taking FP-tree-based frequent pattern-growth algorithms, such as FP-growth, as an example, implement one of the following optimization techniques, and compare the performance of your new implementation with the one that does not incorporate such optimization.
    - (a) The previously proposed frequent pattern mining with FP-tree generates conditional pattern bases using a bottom-up projection technique, i.e., project on the prefix path of an item  $p$ . However, one can develop a **top-down projection** technique, i.e., project on the suffix path of an item  $p$  in the generation of a conditional pattern-base. Design and implement such a top-down FP-tree mining method and compare your performance with the bottom-up projection method.
    - (b) Nodes and pointers are used uniformly in FP-tree in the design of the FP-growth algorithm. However, such a structure may consume a lot of space when the data are sparse. One possible alternative design is to explore **array-and pointer- based hybrid implementation**, where a node may store multiple items when a node contains no splitting point to multiple sub-branches. Develop such an implementation and compare it with the original one.
    - (c) It is time- and space- consuming to generate numerous conditional pattern bases during pattern-growth mining. One interesting alternative is to **push right** the branches that have been mined for a particular item  $p$ , that is, to push them to the remaining branch(es) of the FP-tree. This is done so that fewer conditional pattern bases have to be generated and additional sharing can be explored when mining the remaining branches of the FPtree. Design and implement such a method and conduct a performance study on it.
  13. Give a short example to show that items in a strong association rule may actually be *negatively correlated*.
  14. The following contingency table summarizes supermarket transaction data, where *hot dogs* refers to the transactions containing hot dogs, *hotdogs* refers to the transactions that do not contain hot dogs, *hamburgers* refers to the transactions containing hamburgers, and *hamburgers* refers to the transactions that do not contain hamburgers.

	<i>hot dogs</i>	<i>hotdogs</i>	$\Sigma_{row}$
<i>hamburgers</i>	2000	500	2500
<i>hamburgers</i>	1000	1500	2500
$\Sigma_{col}$	3000	2000	5000

- (a) Suppose that the association rule “*hot dogs*  $\Rightarrow$  *hamburgers*” is mined. Given a minimum support threshold of 25% and a minimum confidence threshold of 50%, is this association rule strong?
- (b) Based on the given data, is the purchase of *hot dogs* independent of the purchase of *hamburgers*? If not, what kind of *correlation* relationship exists between the two?
15. In multidimensional data analysis, it is interesting to extract pairs of *similar* cell characteristics associated with substantial changes in measure in a data cube, where cells are considered *similar* if they are related by roll-up (i.e., *ancestors*), drill-down (i.e., *descendants*), or 1-dimensional mutation (i.e., *siblings*) operations. Such an analysis is called **cube gradient analysis**. Suppose the measure of the cube is *average*. A user poses a set of *probe cells* and would like to find their corresponding sets of *gradient cells* each of which satisfies a certain gradient threshold. For example, find the set of corresponding gradient cells whose average sale price is greater than 20% of that of the given probe cells. Develop an algorithm than mines the set of constrained gradient cells efficiently in a large data cube.
16. Association rule mining often generates a large number of rules. Discuss effective methods that can be used to reduce the number of rules generated while still preserving most of the interesting rules.
17. Sequential patterns can be mined in methods similar to the mining of association rules. Design an efficient algorithm to mine **multilevel sequential patterns** from a transaction database. An example of such a pattern is the following: “*A customer who buys a PC will buy Microsoft software within three months*”, on which one may drill down to find a more refined version of the pattern, such as “*A customer who buys a Pentium PC will buy Microsoft Office within three months*”.
18. Prove that each entry in the following table correctly characterizes its corresponding rule constraint for frequent itemset mining.

	Rule constraint	Antimonotonic	Monotonic	Succinct
a)	$v \in S$	no	yes	yes
b)	$S \subseteq V$	yes	no	yes
c)	$\min(S) \leq v$	no	yes	yes
d)	$\text{range}(S) \leq v$	yes	no	no
e)	$\text{variance}(S) \leq v$	convertible	convertible	no

19. The price of each item in a store is nonnegative. The store manager is only interested in rules of the form: “*one free item may trigger \$200 total purchases in the same transaction*”. State how to mine such rules *efficiently*.
20. The price of each item in a store is nonnegative. For each of the following cases, identify the kinds of constraint they represent and briefly discuss how to mine such association rules *efficiently*.
- (a) Containing at least one Nintendo game
- (b) Containing items whose sum of the prices is less than \$150
- (c) Containing one free item and other items whose sum of the prices is at least \$200
- (d) Where the average price of all the items is between \$100 and \$500

## 5.8 Bibliographic Notes

Association rule mining was first proposed by Agrawal, Imielinski, and Swami [AIS93]. The Apriori algorithm discussed in Section 5.2.1 for frequent itemset mining was presented in Agrawal and Srikant [AS94b]. A variation of the algorithm using a similar pruning heuristic was developed independently by Mannila, Toivonen, and Verkamo [MTV94]. A joint publication combining these works later appeared in Agrawal, Mannila, Srikant, Toivonen, and Verkamo [AMS<sup>+</sup>96]. A method for generating association rules from frequent itemsets is described in Agrawal and Srikant [AS94a].

References for the variations of Apriori described in Section 5.2.3 include the following. The use of hash tables to improve association mining efficiency was studied by Park, Chen, and Yu [PCY95a]. Transaction reduction techniques are described in Agrawal and Srikant [AS94b], Han and Fu [HF95], and Park, Chen, and Yu [PCY95a]. The partitioning technique was proposed by Savasere, Omiecinski, and Navathe [SON95]. The sampling approach is discussed in Toivonen [Toi96]. A dynamic itemset counting approach is given in Brin, Motwani, Ullman, and Tsur [BMUT97]. An efficient incremental updating of mined association rules was proposed by Cheung, Han, Ng, and Wong [CHNW96]. Parallel and distributed association data mining under the Apriori framework was studied by Park, Chen, and Yu [PCY95b], Agrawal and Shafer [AS96], and Cheung, Han, Ng, et al. [CHN<sup>+</sup>96]. Another parallel association mining method, which explores itemset clustering using a vertical database layout, was proposed in Zaki, Parthasarathy, Ogihara, and Li [ZPOL97].

Other scalable frequent itemset mining methods have been proposed as alternatives to the Apriori-based approach. FP-growth, a pattern-growth approach for mining frequent itemsets without candidate generation, was proposed by Han, Pei, and Yin [HPY00] (Section 5.2.4). An exploration of hyper-structure mining of frequent patterns, called H-Mine, was proposed by Pei, Han, Lu, Nishio, Tang, and Yang [PHMA<sup>+</sup>01]. OP, a method that integrates top-down and bottom-up traversal of FP-trees in pattern-growth mining, was proposed by Liu, Pan, Wang, and Han [LPWH02]. An array-based implementation of prefix-tree-structure for efficient pattern growth mining was proposed by Grahne and Zhu [GZ03b]. Eclat, an approach for mining frequent itemsets by exploring the vertical data format, was proposed by Zaki [Zak00]. A depth-first generation of frequent itemsets was proposed by Agarwal, Aggarwal, and Prasad [AAP01].

The mining of frequent closed itemsets was proposed in Pasquier, Bastide, Taouil, and Lakhil [PBTL99], where an Apriori-based algorithm called A-Close for such mining was presented. CLOSET, an efficient closed itemset mining algorithm based on the frequent pattern growth method, was proposed by Pei, Han, and Mao [PHM00], and further refined as CLOSET+ in Wang, Han and Pei [WHP03]. A prefix-tree-based algorithm, called FPClose, for mining closed itemsets using pattern growth approach, was proposed by Grahne and Zhu [GZ03b]. An extension for mining closed frequent itemsets with the vertical data format, called CHARM, was proposed by Zaki and Hsiao [ZH02]. Mining max-patterns was first studied by Bayardo [Bay98]. Another efficient method for mining maximal frequent itemsets using vertical data format, called MAFIA, was proposed by Burdick, Calimlim, and Gehrke [BCG01]. AFOPT, a method that explores a *right push* operation on FP-trees during the mining process, was proposed by Liu, Lu, Lou and Yu [LLLY03]. Pan, Cong, Tung, et al. [PCT<sup>+</sup>03] proposed CARPENTER, a method for finding closed patterns in long biological datasets, which integrates the advantages of vertical data formats and pattern growth methods. A FIMI (Frequent Itemset Mining Implementation) workshop dedicated to the implementation methods of frequent itemset mining was reported by Goethals and Zaki [GZ03a].

Frequent itemset mining has various extensions, including sequential pattern mining (Agrawal and Srikant [AS95]), episodes mining (Mannila, Toivonen, and Verkamo [MTV97]), spatial association rule mining (Koperski and Han [KH95]), cyclic association rule mining (Özden, Ramaswamy, and Silberschatz [ORS98]), negative association rule mining (Savasere, Omiecinski and Navathe [SON98]), intertransaction association rule mining (Lu, Han, and Feng [LHF98]), and calendric market basket analysis (Ramaswamy, Mahajan, and Silberschatz [RMS98]). Multilevel association mining was studied in Han and Fu [HF95], and Srikant and Agrawal [SA95]. In Srikant and Agrawal [SA95], such mining was studied in the context of *generalized association rules*, and an R-interest measure was proposed for removing redundant rules. A non-grid-based technique for mining quantitative association rules, which uses a measure of partial completeness, was proposed by Srikant and Agrawal [SA96]. The ARCS system for mining quantitative association rules based on rule clustering was proposed by Lent, Swami, and Widom [LSW97]. Techniques for mining quantitative rules based on x-monotone and rectilinear regions were presented by Fukuda,

Morimoto, Morishita, and Tokuyama [FMMT96], and Yoda, Fukuda, Morimoto, et al. [YFM<sup>+</sup>97]. Mining multi-dimensional association rules using static discretization of quantitative attributes and data cubes was studied by Kamber, Han, and Chiang [KHC97]. Mining (distance-based) association rules over interval data was proposed by Miller and Yang [MY97]. Mining quantitative association rules based on a statistical theory to present only those that deviate substantially from normal data was studied by Aumann and Lindell [AL99].

The problem of mining interesting rules has been studied by many researchers. The statistical independence of rules in data mining was studied by Piatetski-Shapiro [PS91]. The interestingness problem of strong association rules is discussed in Chen, Han, and Yu [CHY96], Brin, Motwani, and Silverstein [BMS97], and Aggarwal and Yu [AY99], which cover several interestingness measures including *lift*. An efficient method for generalizing associations to correlations is given in Brin, Motwani, and Silverstein [BMS97]. Other alternatives to the support-confidence framework for assessing the interestingness of association rules are proposed in Brin, Motwani, Ullman, and Tsur [BMUT97] and Ahmed, El-Makky, and Taha [AEMT00]. A method for mining strong gradient relationships among itemsets was proposed by Imielinski, Khachiyan, and Abdulghani [IKA02]. Silverstein, Brin, Motwani, and Ullman [SBMU98] studied the problem of mining causal structures over transaction databases. Some comparative studies of different interestingness measures were done by Hilderman and Hamilton [HH01] and by Tan, Kumar and Srivastava [TKS02]. The use of *all\_confidence* as a correlation measure for generating interesting association rules was studied by Omiecinski [Omi03] and by Lee, Kim, Cai and Han [LKCH03].

To reduce the huge set of frequent patterns generated in data mining, recent studies have been working on mining compressed set of frequent patterns. Mining closed patterns can be viewed as lossless compression of frequent patterns. Lossy compression of patterns include maximal patterns by Bayardo [Bay98]), top-*k* patterns by Wang, Han, Lu, and Tsvetkov [WHLT05], and error-tolerant patterns by Yang, Fayyad, and Bradley [YFB01]. Afrati, Gionis, and Mannila [AGM04] proposed to use *K* itemsets to cover a collection of frequent itemsets. Yan, Cheng, Xin and Han proposed a profile-based approach [YCXH05] and Xin, Han, Yan, and Cheng proposed a clustering-based approach [XHYC05] for frequent itemset compression.

The use of metarules as syntactic or semantic filters defining the form of interesting single-dimensional association rules was proposed in Klemettinen, Mannila, Ronkainen, et al. [KMR<sup>+</sup>94]. Metarule-guided mining, where the metarule consequent specifies an action (such as Bayesian clustering or plotting) to be applied to the data satisfying the metarule antecedent, was proposed in Shen, Ong, Mitbander, and Zaniolo [SOMZ96]. A relation-based approach to metarule-guided mining of association rules was studied in Fu and Han [FH95]. Methods for constraint-based association rule mining discussed in this chapter were studied by Ng, Lakshmanan, Han, and Pang [NLHP98], Lakshmanan, Ng, Han, and Pang [LNHP99], and Pei, Han, and Lakshmanan [PHL01]. An efficient method for mining constrained correlated sets was given in Grahne, Lakshmanan, and Wang [GLW00]. A dual mining approach was proposed by Bucila, Gehrke, Kifer, and White [BGKW03]. Other ideas involving the use of templates or predicate constraints in mining have been discussed in [AK93], [DT93], [HK91], [LHC97], [ST96], [SVA97].

The association mining language presented in this chapter was based on an extension of the data mining query language, DMQL, proposed in Han, Fu, Wang, et al. [HFW<sup>+</sup>96], by incorporation of the spirit of the SQL-like operator for mining single-dimensional association rules proposed by Meo, Psaila, and Ceri [MPC96]. MSQL, a query language for mining flexible association rules, was proposed by Imielinski and Virmani [IV99]. *OLE DB for Data Mining (DM)*, a data mining query language that includes association mining modules was proposed by Microsoft Corporation [Cor00].

# Bibliography

- [AAP01] R. Agarwal, C. C. Aggarwal, and V. V. V. Prasad. A tree projection algorithm for generation of frequent itemsets. *Journal of Parallel and Distributed Computing*, 61:350–371, 2001.
- [AEMT00] K. M. Ahmed, N. M. El-Makky, and Y. Taha. A note on “beyond market basket: Generalizing association rules to correlations”. *SIGKDD Explorations*, 1:46–48, 2000.
- [AGM04] F. N. Afrati, A. Gionis, and H. Mannila. Approximating a collection of frequent sets. In *Proc. 2004 ACM SIGKDD Int. Conf. Knowledge Discovery in Databases (KDD’04)*, pages 12–19, Seattle, WA, Aug. 2004.
- [AIS93] 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 (SIGMOD’93)*, pages 207–216, Washington, DC, May 1993.
- [AK93] T. Anand and G. Kahn. Opportunity explorer: Navigating large databases using knowledge discovery templates. In *Proc. AAAI-93 Workshop Knowledge Discovery in Databases*, pages 45–51, Washington, DC, July 1993.
- [AL99] Y. Aumann and Y. Lindell. A statistical theory for quantitative association rules. In *Proc. 1999 Int. Conf. Knowledge Discovery and Data Mining (KDD’99)*, San Diego, CA, Aug. 1999.
- [AMS<sup>+</sup>96] 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, Aug. 1996.
- [AS94a] R. Agrawal and R. Srikant. Fast algorithm for mining association rules in large databases. In *Research Report RJ 9839*, IBM Almaden Research Center, San Jose, CA, June 1994.
- [AS94b] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. 1994 Int. Conf. Very Large Data Bases (VLDB’94)*, pages 487–499, Santiago, Chile, Sept. 1994.
- [AS95] R. Agrawal and R. Srikant. Mining sequential patterns. In *Proc. 1995 Int. Conf. Data Engineering (ICDE’95)*, pages 3–14, Taipei, Taiwan, Mar. 1995.
- [AS96] R. Agrawal and J. C. Shafer. Parallel mining of association rules: Design, implementation, and experience. *IEEE Trans. Knowledge and Data Engineering*, 8:962–969, 1996.
- [AY99] C. C. Aggarwal and P. S. Yu. A new framework for itemset generation. In *Proc. 1998 ACM Symp. Principles of Database Systems (PODS’98)*, pages 18–24, Seattle, WA, June 1999.
- [Bay98] R. J. Bayardo. Efficiently mining long patterns from databases. In *Proc. 1998 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD’98)*, pages 85–93, Seattle, WA, June 1998.
- [BCG01] D. Burdick, M. Calimlim, and J. Gehrke. MAFIA: A maximal frequent itemset algorithm for transactional databases. In *Proc. 2001 Int. Conf. Data Engineering (ICDE’01)*, pages 443–452, Heidelberg, Germany, April 2001.

- [BGKW03] C. Bucila, J. Gehrke, D. Kifer, and W. White. DualMiner: A dual-pruning algorithm for itemsets with constraints. *Data Mining and Knowledge Discovery*, 7:241–272, 2003.
- [BMS97] 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 (SIGMOD'97)*, pages 265–276, Tucson, Arizona, May 1997.
- [BMUT97] S. Brin, R. Motwani, J. D. Ullman, and S. Tsur. Dynamic itemset counting and implication rules for market basket analysis. In *Proc. 1997 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'97)*, pages 255–264, Tucson, Arizona, May 1997.
- [CHN<sup>+</sup>96] D. W. Cheung, J. Han, V. Ng, A. Fu, and Y. Fu. A fast distributed algorithm for mining association rules. In *Proc. 1996 Int. Conf. Parallel and Distributed Information Systems*, pages 31–44, Miami Beach, Florida, Dec. 1996.
- [CHNW96] 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 (ICDE'96)*, pages 106–114, New Orleans, Louisiana, Feb. 1996.
- [CHY96] M. S. Chen, J. Han, and P. S. Yu. Data mining: An overview from a database perspective. *IEEE Trans. Knowledge and Data Engineering*, 8:866–883, 1996.
- [Cor00] Microsoft Corporation. OLEDB for Data Mining draft specification, version 0.9. In <http://www.microsoft.com/data/oledb/dm>, Feb. 2000.
- [DT93] V. Dhar and A. Tuzhilin. Abstract-driven pattern discovery in databases. *IEEE Trans. Knowledge and Data Engineering*, 5:926–938, 1993.
- [FH95] Y. Fu and J. Han. Meta-rule-guided mining of association rules in relational databases. In *Proc. 1995 Int. Workshop Integration of Knowledge Discovery with Deductive and Object-Oriented Databases (KDOOD'95)*, pages 39–46, Singapore, Dec. 1995.
- [FMMT96] 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 (SIGMOD'96)*, pages 13–23, Montreal, Canada, June 1996.
- [GZ03a] B. Goethals and M. Zaki. An introduction to workshop on frequent itemset mining implementations. In *Proc. of ICDM-FIMI Workshop on Frequent Itemset Mining Implementations*, Melbourne, Florida, Nov. 2003.
- [GZ03b] G. Grahne and J. Zhu. Efficiently using prefix-trees in mining frequent itemsets. In *Proc. ICDM'03 Int. Workshop on Frequent Itemset Mining Implementations (FIMI'03)*, Melbourne, FL, Nov. 2003.
- [HF95] J. Han and Y. Fu. Discovery of multiple-level association rules from large databases. In *Proc. 1995 Int. Conf. Very Large Data Bases (VLDB'95)*, pages 420–431, Zurich, Switzerland, Sept. 1995.
- [HFW<sup>+</sup>96] J. Han, Y. Fu, W. Wang, J. Chiang, W. Gong, K. Koperski, D. Li, Y. Lu, A. Rajan, N. Stefanovic, B. Xia, and O. R. Zaïane. DBMiner: A system for mining knowledge in large relational databases. In *Proc. 1996 Int. Conf. Data Mining and Knowledge Discovery (KDD'96)*, pages 250–255, Portland, Oregon, Aug. 1996.
- [HH01] R. J. Hilderman and H. J. Hamilton. *Knowledge Discovery and Measures of Interest*. Kluwer Academic, 2001.
- [HK91] P. Hoschka and W. Klösgen. A support system for interpreting statistical data. In G. Piattetsky-Shapiro and W. J. Frawley, editors, *Knowledge Discovery in Databases*, pages 325–346. AAAI/MIT Press, 1991.
- [HPY00] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *Proc. 2000 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'00)*, pages 1–12, Dallas, TX, May 2000.

- [IKA02] T. Imielinski, L. Khachiyan, and A. Abdulghani. Cubegrades: Generalizing association rules. *Data Mining and Knowledge Discovery*, 6:219–258, 2002.
- [IV99] T. Imielinski and A. Virmani. MSQL: A query language for database mining. *Data Mining and Knowledge Discovery*, 3:373–408, 1999.
- [KH95] K. Koperski and J. Han. Discovery of spatial association rules in geographic information databases. In *Proc. 1995 Int. Symp. Large Spatial Databases (SSD'95)*, pages 47–66, Portland, Maine, Aug. 1995.
- [KHC97] M. Kamber, J. Han, and J. Y. Chiang. Metarule-guided mining of multi-dimensional association rules using data cubes. In *Proc. 1997 Int. Conf. Knowledge Discovery and Data Mining (KDD'97)*, pages 207–210, Newport Beach, CA, Aug. 1997.
- [KMR<sup>+</sup>94] 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.
- [LHC97] B. Liu, W. Hsu, and S. Chen. Using general impressions to analyze discovered classification rules. In *Proc. 1997 Int. Conf. Knowledge Discovery and Data Mining (KDD'97)*, pages 31–36, Newport Beach, CA, Aug. 1997.
- [LHF98] H. Lu, J. Han, and L. Feng. Stock movement and n-dimensional inter-transaction association rules. In *Proc. 1998 SIGMOD Workshop Research Issues on Data Mining and Knowledge Discovery (DMKD'98)*, pages 12:1–12:7, Seattle, WA, June 1998.
- [LKCH03] Y.-K. Lee, W.-Y. Kim, Y. D. Cai, and J. Han. CoMine: Efficient mining of correlated patterns. In *Proc. 2003 Int. Conf. Data Mining (ICDM'03)*, Melbourne, FL, Nov. 2003.
- [LLLY03] G. Liu, H. Lu, W. Lou, and J. X. Yu. On computing, storing and querying frequent patterns. In *Proc. 2003 ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD'03)*, pages 607–612, Washington, D.C., Aug. 2003.
- [LNHP99] L. V. S. Lakshmanan, R. Ng, J. Han, and A. Pang. Optimization of constrained frequent set queries with 2-variable constraints. In *Proc. 1999 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'99)*, pages 157–168, Philadelphia, PA, June 1999.
- [LPWH02] J. Liu, Y. Pan, K. Wang, and J. Han. Mining frequent item sets by opportunistic projection. In *Proc. 2002 ACM SIGKDD Int. Conf. Knowledge Discovery in Databases (KDD'02)*, pages 239–248, Edmonton, Canada, July 2002.
- [LSW97] 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.
- [MPC96] 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 (VLDB'96)*, pages 122–133, Bombay, India, Sept. 1996.
- [MTV94] H. Mannila, H. Toivonen, and A. I. Verkamo. Efficient algorithms for discovering association rules. In *Proc. AAAI'94 Workshop Knowledge Discovery in Databases (KDD'94)*, pages 181–192, Seattle, WA, July 1994.
- [MTV97] H. Mannila, H. Toivonen, and A. I. Verkamo. Discovery of frequent episodes in event sequences. *Data Mining and Knowledge Discovery*, 1:259–289, 1997.
- [MY97] R. J. Miller and Y. Yang. Association rules over interval data. In *Proc. 1997 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'97)*, pages 452–461, Tucson, Arizona, May 1997.
- [NLHP98] 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 (SIGMOD'98)*, pages 13–24, Seattle, WA, June 1998.

- [Omi03] E. Omiecinski. Alternative interest measures for mining associations. *IEEE Trans. Knowledge and Data Engineering*, 15:57–69, 2003.
- [ORS98] 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.
- [PBTL99] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Discovering frequent closed itemsets for association rules. In *Proc. 7th Int. Conf. Database Theory (ICDT'99)*, pages 398–416, Jerusalem, Israel, Jan. 1999.
- [PCT<sup>+</sup>03] F. Pan, G. Cong, A. K. H. Tung, J. Yang, and M. Zaki. CARPENTER: Finding closed patterns in long biological datasets. In *Proc. 2003 ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD'03)*, Washington, D.C., Aug. 2003.
- [PCY95a] 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 (SIGMOD'95)*, pages 175–186, San Jose, CA, May 1995.
- [PCY95b] 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.
- [PHL01] J. Pei, J. Han, and L. V. S. Lakshmanan. Mining frequent itemsets with convertible constraints. In *Proc. 2001 Int. Conf. Data Engineering (ICDE'01)*, pages 433–332, Heidelberg, Germany, April 2001.
- [PHM00] J. Pei, J. Han, and R. Mao. CLOSET: An efficient algorithm for mining frequent closed itemsets. In *Proc. 2000 ACM-SIGMOD Int. Workshop Data Mining and Knowledge Discovery (DMKD'00)*, pages 11–20, Dallas, TX, May 2000.
- [PHMA<sup>+</sup>01] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and M.-C. Hsu. PrefixSpan: Mining sequential patterns efficiently by prefix-projected pattern growth. In *Proc. 2001 Int. Conf. Data Engineering (ICDE'01)*, pages 215–224, Heidelberg, Germany, April 2001.
- [PS91] G. Piatetsky-Shapiro. *Notes of AAAI'91 Workshop Knowledge Discovery in Databases (KDD'91)*. Anaheim, CA, July 1991.
- [RMS98] 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 (VLDB'98)*, pages 368–379, New York, NY, Aug. 1998.
- [SA95] R. Srikant and R. Agrawal. Mining generalized association rules. In *Proc. 1995 Int. Conf. Very Large Data Bases (VLDB'95)*, pages 407–419, Zurich, Switzerland, Sept. 1995.
- [SA96] R. Srikant and R. Agrawal. Mining sequential patterns: Generalizations and performance improvements. In *Proc. 5th Int. Conf. Extending Database Technology (EDBT'96)*, pages 3–17, Avignon, France, Mar. 1996.
- [SOMZ96] W. Shen, K. Ong, B. Mitbender, and C. Zaniolo. Metaqueries for data mining. In U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 375–398. AAAI/MIT Press, 1996.
- [SON95] 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 (VLDB'95)*, pages 432–443, Zurich, Switzerland, Sept. 1995.
- [SON98] A. Savasere, E. Omiecinski, and S. Navathe. Mining for strong negative associations in a large database of customer transactions. In *Proc. 1998 Int. Conf. Data Engineering (ICDE'98)*, pages 494–502, Orlando, FL, Feb. 1998.
- [ST96] A. Silberschatz and A. Tuzhilin. What makes patterns interesting in knowledge discovery systems. *IEEE Trans. on Knowledge and Data Engineering*, 8:970–974, Dec. 1996.

- [SVA97] R. Srikant, Q. Vu, and R. Agrawal. Mining association rules with item constraints. In *Proc. 1997 Int. Conf. Knowledge Discovery and Data Mining (KDD'97)*, pages 67–73, Newport Beach, CA, Aug. 1997.
- [TKS02] P.-N. Tan, V. Kumar, and J. Srivastava. Selecting the right interestingness measure for association patterns. In *Proc. 2002 ACM SIGKDD Int. Conf. Knowledge Discovery in Databases (KDD'02)*, pages 32–41, Edmonton, Canada, July 2002.
- [Toi96] H. Toivonen. Sampling large databases for association rules. In *Proc. 1996 Int. Conf. Very Large Data Bases (VLDB'96)*, pages 134–145, Bombay, India, Sept. 1996.
- [WHLT05] J. Wang, J. Han, Y. Lu, and P. Tzvetkov. TFP: An efficient algorithm for mining top-k frequent closed itemsets. *IEEE Trans. Knowledge and Data Engineering*, 17:652–664, 2005.
- [WHP03] J. Wang, J. Han, and J. Pei. CLOSET+: Searching for the best strategies for mining frequent closed itemsets. In *Proc. 2003 ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD'03)*, pages 236–245, Washington, D.C., Aug. 2003.
- [XHYC05] D. Xin, J. Han, X. Yan, and H. Cheng. Mining compressed frequent-pattern sets. In *Proc. 2005 Int. Conf. Very Large Data Bases (VLDB'05)*, Trondheim, Norway, Aug. 2005.
- [YCXH05] X. Yan, H. Cheng, D. Xin, and J. Han. Summarizing itemset patterns: A profile-based approach. *In submitted for publication*, Feb. 2005.
- [YFB01] C. Yang, U. Fayyad, and P. S. Bradley. Efficient discovery of error-tolerant frequent itemsets in high dimensions. In *Proc. 2001 ACM SIGKDD Int. Conf. Knowledge Discovery in Databases (KDD'01)*, pages 194–203, San Francisco, CA, Aug. 2001.
- [YFM<sup>+</sup>97] K. Yoda, T. Fukuda, Y. Morimoto, S. Morishita, and T. Tokuyama. Computing optimized rectilinear regions for association rules. In *Proc. 1997 Int. Conf. Knowledge Discovery and Data Mining (KDD'97)*, pages 96–103, Newport Beach, CA, Aug. 1997.
- [Zak00] M. J. Zaki. Scalable algorithms for association mining. *IEEE Trans. Knowledge and Data Engineering*, 12:372–390, 2000.
- [ZH02] M. J. Zaki and C. J. Hsiao. CHARM: An efficient algorithm for closed itemset mining. In *Proc. 2002 SIAM Int. Conf. Data Mining (SDM'02)*, pages 457–473, Arlington, VA, April 2002.
- [ZPOL97] M. J. Zaki, S. Parthasarathy, M. Ogihara, and W. Li. Parallel algorithm for discovery of association rules. *Data Mining and Knowledge Discovery*, 1:343–374, 1997.