Mining Frequent Patterns, Associations, and Correlations

- ***Frequent patterns are patterns (such as item sets, 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 item sets 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.***

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 item-sets in transactional and relational databases.

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 businesses decision-making processes, such as catalog design, cross-marketing, and customer shopping behavior analysis 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.***

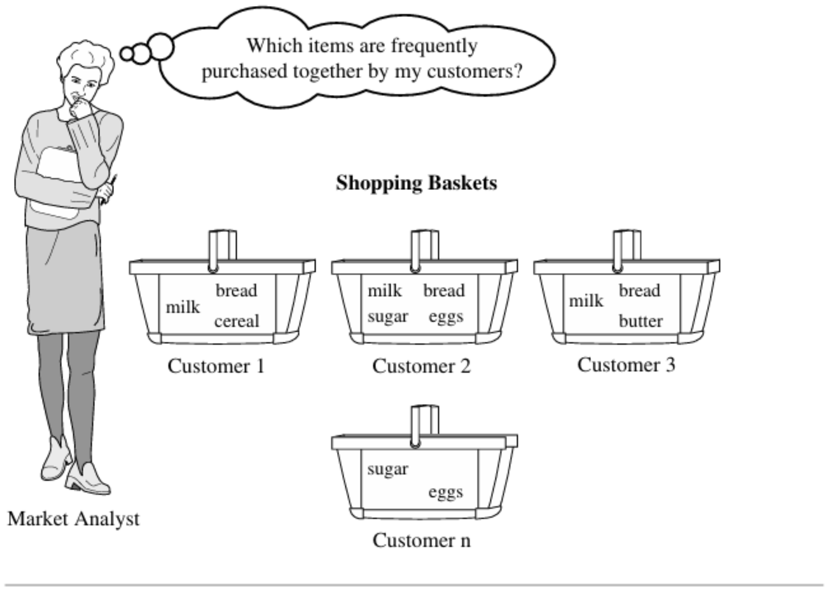


Figure 5.1 Market basket analysis.

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

Market basket analysis.

- ***Suppose, as manager of an All Electronics 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 use 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 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 increase the sales of both 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 toward the software dis- play to purchase antivirus software and may decide to purchase a home security system as well. Market basket analysis can also help retailers 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: computer ⇒ antivirus software [support = 2%, confidence = 60%] (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 Associ- ation 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 $I = \{I_1, I_2, \ldots, I_m\}$ 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 $T \subseteq I$. 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 $A \subset I$, $B \subset I$, 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, $P(A \cup B)$.[1] 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,

$$support(A \Rightarrow B) = P(A \cup B)$$

$$confidence(A \Rightarrow B) = P(B|A).$$

- ***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.[2] An itemset that contains k items is a k-itemset. The set {computer and antivirus software} is a 2-item set. 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 prespecified minimum support threshold (i.e., the absolute support of I satisfies the corresponding minimum support count threshold), then I is a frequent itemset.[3] The set of frequent k-itemsets is commonly denoted by L_k.[4]***

From Equation (5.3), we have

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

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

[1]Notice that the notation *P*(*AuB)* 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 or B*), which indicates the probability that a transaction contains either *A* or *B*.

[2]In the data mining research literature, "itemset" is more commonly used than "item set."

[3]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 fre- quency of occurrence of the set. Hence, we use the more recent term **frequent**.

[4]Although the term **frequent** is preferred over **large**, for historical reasons frequent *k*-itemsets are still denoted as L_k.

found, it is straightforward to derive the corresponding association rules *A* -> *B* and *B*->*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 predetermined 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. Because 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 $\{a_1, a_2, \ldots, a_{100}\}$, contains $\binom{100}{1} = 100$ frequent 1-itemsets: $a_1, a_2, \ldots, a_{100}$, $\binom{100}{2}$ frequent 2-itemsets: $(a_1, a_2), (a_1, a_3), \ldots, (a_{99}, a_{100})$, and so on. The total number of frequent itemsets that it contains is thus,

$$\binom{100}{1} + \binom{100}{2} + \cdots + \binom{100}{100} = 2^{100} - 1 \approx 1.27 \times 10^{30}. \tag{5.5}$$

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[5] *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 *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 C be the set of closed frequent itemsets for a data set *S* satisfying a minimum support threshold, *min sup*. Let M be the set of maximal frequent itemsets for *S* satisfying *min sup*. Suppose that we have the support count of each itemset in C and M . Notice that C and its count information can be used to derive the whole set of frequent item- sets. Thus we say that C contains complete information regarding its corresponding fre- quent itemsets. On the other hand, M registers only the support of the maximal itemsets.

[5] 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.

It usually does not contain the complete support information regarding its correspond- ing 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: $\{\langle a_1, a_2, \ldots, a_{100}\rangle; \langle a_1, a_2, \ldots, a_{50}\rangle\}$. Let the minimum support count threshold be *min_sup* = 1. We find two closed frequent itemsets and their support counts, that is, $C = \{\{a_1, a_2, \ldots, a_{100}\} : 1; \{a_1, a_2, \ldots, a_{50}\} : 2\}$. There is one maximal frequent itemset: $\mathcal{M} = \{\{a_1, a_2, \ldots, a_{100}\} : 1\}$. (We cannot include $\{a_1, a_2, \ldots, a_{50}\}$ as a maximal frequent itemset because it has a frequent super-set, $\{a_1, a_2, \ldots, a_{100}\}$.) 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!

The set of closed frequent itemsets contains complete information regarding the frequent itemsets. For example, from C, we can derive, say, (1) $\{a_2, a_{45} : 2\}$ since $\{a_2, a_{45}\}$ is a sub-itemset of the itemset $\{a_1, a_2, \ldots, a_{50} : 2\}$; and (2) $\{a_8, a_{55} : 1\}$ since $\{a_8, a_{55}\}$ is not a sub-itemset of the previous itemset but of the itemset $\{a_1, a_2, \ldots, a_{100} : 1\}$. 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.

5.1.1 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 pat- tern 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 item- sets), **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 associa- tion 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:

- $buys(X, \text{“computer”}) \Rightarrow buys(X, \text{“HP\_printer”})$ (5.6)
- $buys(X, \text{“laptop\_computer”}) \Rightarrow buys(X, \text{“HP\_printer”})$ (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**.

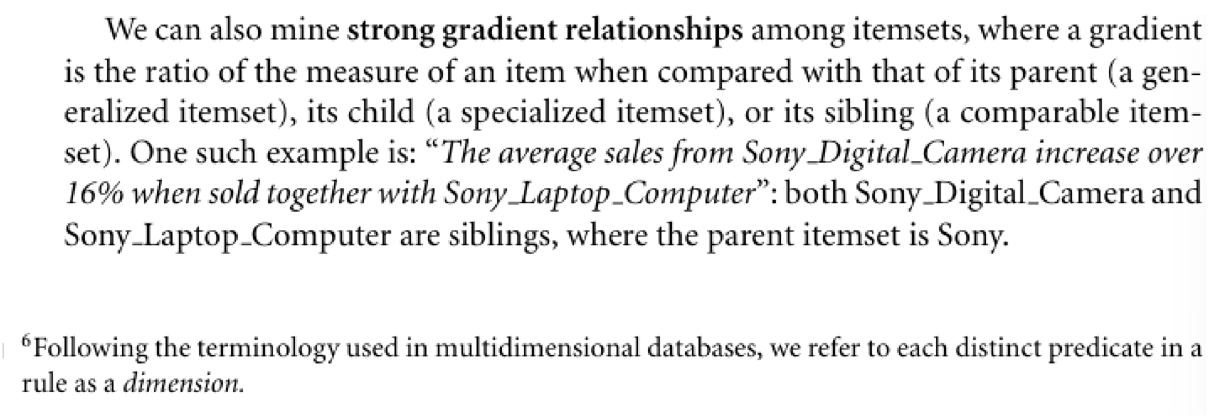
- **Based on the *number of data dimensions* involved in the rule:** If the items or attributes in an association rule reference only one dimension, then it is a **single-dimensional association rule**. Note that Rule (5.1), for example, could be rewritten as Rule (5.8):
- $buys(X, \text{“computer”}) \Rightarrow buys(X, \text{“antivirus software”})$ (5.8)

- Rules (5.6), (5.7), and (5.8) are single-dimensional association rules because they each refer to only one dimension, *buys*.[6]

- 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:
$age(X, \text{“30...39”}) \wedge income(X, \text{“42K...48K”}) \Rightarrow buys(X, \text{“high resolution TV”})$.
- (5.9)
- **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 bas- ket 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) 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 vari- ous 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 is: "*The average sales from Sony_Digital_Camera increase over 16% when sold together with Sony_Laptop_Computer*": both Sony_Digital_Camera and Sony_Laptop_Computer are siblings, where the parent itemset is Sony.

[6]Following the terminology used in multidimensional databases, we refer to each distinct predicate in a rule as a *dimension.*

- **Based on the *kinds of patterns* to be mined:** Many kinds of frequent patterns can be mined from different kinds of data sets. For this chapter, our focus is on **frequent item- set 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 the memory card.
- **Structured pattern mining** searches for frequent *sub- structures* 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, and so on, and such containment relationships can be defined recursively. Therefore, structured pattern mining can be considered as the most general form of frequent pattern mining.

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 dis- cussed for market basket analysis in Section 5.1.1. We begin by presenting **Apriori**, the 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.

5.1.1 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 following.

- Apriori employs an iterative approach known as a *level-wise* search, where *k*-itemsets are used to explore($k + 1$)-item sets. 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.

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.[7]

"*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 jth 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 $l_i[1] < l_i[2] < \ldots < l_i[k-1]$. 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 $(l_1[1] = l_2[1]) \wedge (l_1[2] = l_2[2]) \wedge \ldots \wedge (l_1[k-2] = l_2[k-2]) \wedge (l_1[k-1] < l_2[k-1])$. 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 $l_1[1], l_1[2], \ldots, l_1[k-2], l_1[k-1], l_2[k-1]$.

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

[7]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.

TID	*List of item_IDs*
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

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.

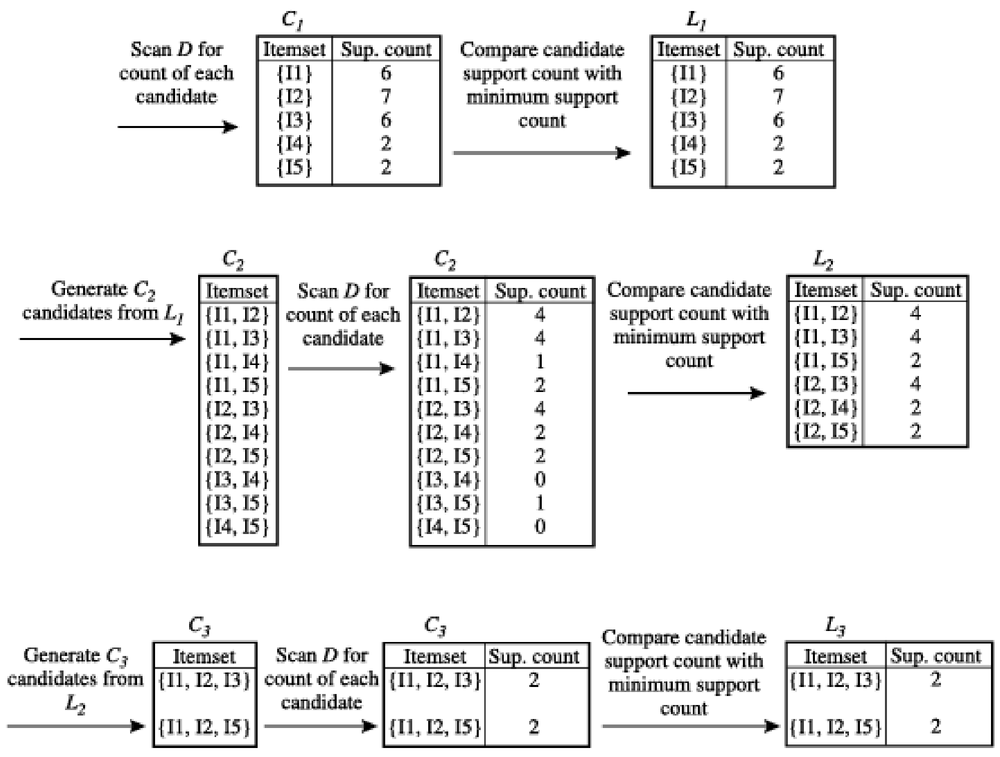
Example 5.3 **Apriori.** Let's look at a concrete example, based on the *AllElectronics* transaction database, D, of Table 5.1. There are nine transactions in this database, that is, $|D| = 9$. We use Figure 5.2 to illustrate the Apriori algorithm for finding frequent itemsets in D.

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, that is, $min\_sup = 2$. (Here, we are referring to *absolute* support because we are using a support count. The corresponding relative support is 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.[8] C_2 consists of $\binom{|L_1|}{2}$ 2-itemsets. Note that no candidates are removed from C_2 during the prune step because each subset of the candidates is also frequent.

[8] $L_1 \bowtie L_1$ 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.


Scan *D* for count of each candidate
Compare candidate support count with minimum support count
Generate C_2 candidates from L_1
Scan *D* for count of each candidate
Compare candidate support count with minimum support count
Generate C_3 candidates from L_2
Scan *D* for count of each candidate
Compare candidate support count with minimum support count

C_1

Itemset	Sup. count
{I1}	6
{I2}	7
{I3}	6
{I4}	2
{I5}	2

L_1

Itemset	Sup. count
{I1}	6
{I2}	7
{I3}	6
{I4}	2
{I5}	2

C_2

Itemset
{I1, I2}
{I1, I3}
{I1, I4}
{I1, I5}
{I2, I3}
{I2, I4}
{I2, I5}
{I3, I4}
{I3, I5}
{I4, I5}

C_2

Itemset	Sup. count
{I1, I2}	4
{I1, I3}	4
{I1, I4}	1
{I1, I5}	2
{I2, I3}	4
{I2, I4}	2
{I2, I5}	2
{I3, I4}	0
{I3, I5}	1
{I4, I5}	0

L_2

Itemset	Sup. count
{I1, I2}	4
{I1, I3}	4
{I1, I5}	2
{I2, I3}	4
{I2, I4}	2
{I2, I5}	2

C_3

Itemset
{I1, I2, I3}
{I1, I2, I5}

C_3

Itemset	Sup. count
{I1, I2, I3}	2
{I1, I2, I5}	2

L_3

Itemset	Sup. count
{I1, I2, I3}	2
{I1, I2, I5}	2

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

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).

(a) Join: $C_3 = L_2$ **ɒ** L_2 = {{I1, I2}, {I1, I3}, {I1, I5}, {I2, I3}, {I2, I4}, {I2, I5}} **ɒ** {{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.

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 because 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 pseudo-code for the Apriori algorithm and its related procedures. Step 1 of Apriori finds the frequent 1-itemsets, L_1. In steps 2 to 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 that is not frequent (step 3). This procedure is described below. Once all of 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 of those candidates satisfying minimum support (step 9) form the set of frequent itemsets, L (step 11). 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 to 4). The prune component (steps 5 to 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.

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)   $L_1$ = find_frequent_1-itemsets(D);
(2)   for ($k = 2; L_{k-1} \neq \phi; k{+}{+}$) {
(3)       $C_k$ = apriori_gen($L_{k-1}$);
(4)       for each transaction $t \in D$ { // scan $D$ for counts
(5)           $C_t$ = subset($C_k, t$); // get the subsets of $t$ that are candidates
(6)           for each candidate $c \in C_t$
(7)               c.count++;
(8)       }
(9)       $L_k = \{c \in C_k | c.count \geq min\_sup\}$
(10)  }
(11)  return $L = \cup_k L_k$;

procedure 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 $(l_1[1] = l_2[1]) \wedge (l_1[2] = l_2[2]) \wedge \ldots \wedge (l_1[k-2] = l_2[k-2]) \wedge (l_1[k-1] < l_2[k-1])$ then {
(4)               $c = l_1 \bowtie l_2$; // join step: generate candidates
(5)               if 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 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.

5.1.1 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:

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

- The conditional probability is expressed in terms of itemset support count, where support count(AuB) is the number of transactions containing the itemsets (AuB), and 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 $\frac{support\_count(l)}{support\_count(s)} \geq min\_conf$, where min_conf is the minimum confidence threshold.

- Because 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 association rules.** Let's try an example based on the transactional data for *AllElectronics* 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:

$$\begin{array}{ll}
I1 \wedge I2 \Rightarrow I5, & confidence = 2/4 = 50\% \\
I1 \wedge I5 \Rightarrow I2, & confidence = 2/2 = 100\% \\
I2 \wedge I5 \Rightarrow I1, & confidence = 2/2 = 100\% \\
I1 \Rightarrow I2 \wedge I5, & confidence = 2/6 = 33\% \\
I2 \Rightarrow I1 \wedge I5, & confidence = 2/7 = 29\% \\
I5 \Rightarrow I1 \wedge I2, & confidence = 2/2 = 100\%
\end{array}$$

If the minimum confidence threshold is, say, 70%, then only the second, third, and last rules above are output, because 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.

5.1.1 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 $\{a_1, \ldots, a_{100}\}$, 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 re-examine 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.

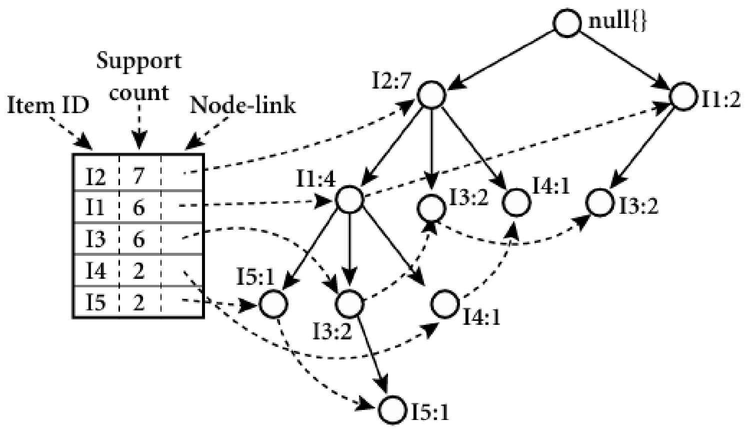


Figure 5.7 An FP-tree registers compressed, frequent pattern information.

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

Item	*Conditional Pattern Base*	*Conditional FP-tree*	*Frequent Patterns Generated*
I5	{{I2, I1: 1}, {I2, I1, I3: 1}}	⟨I2: 2, I1: 2⟩	{I2, I5: 2}, {I1, I5: 2}, {I2, I1, I5: 2}
I4	{{I2, I1: 1}, {I2: 1}}	⟨I2: 2⟩	{I2, I4: 2}
I3	{{I2, I1: 2}, {I2: 2}, {I1: 2}}	⟨I2: 4, I1: 2⟩, ⟨I1: 2⟩	{I2, I3: 4}, {I1, I3: 4}, {I2, I1, I3: 2}
I1	{{I2: 4}}	⟨I2: 4⟩	{I2, I1: 4}

- 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.

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 ⟨I2, I1, I5: 1⟩ and ⟨I2, I1, I3, I5: 1⟩. Therefore, considering I5 as a suffix, its corresponding two prefix paths are ⟨I2, I1: 1⟩ and ⟨I2, I1, I3: 1⟩, which form its conditional pattern base. Its conditional FP-tree contains only a single path, ⟨I2: 2, I1: 2⟩; 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}.

For I4, its two prefix paths form the conditional pattern base, {{I2 I1: 1}, {I2: 1}}, which generates a single-node conditional FP-tree, ⟨I2: 2⟩, and derives one frequent

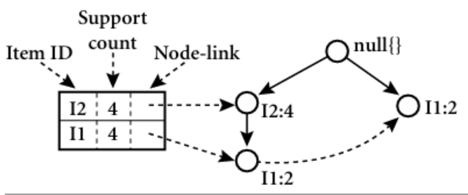


Figure 5.8 The conditional FP-tree associated with the conditional node I3.

pattern, {I2, I1: 2}. Notice that although I5 follows I4 in the first branch, there is no need to include I5 in the analysis here because any frequent pattern involving I5 is analyzed in the examination of I5.

Similar to the above analysis, I3's conditional pattern base is {{I2, I1: 2}, {I2: 2}, {I1: 2}}. Its conditional FP-tree has two branches, ⟨I2: 4, I1: 2⟩ and ⟨I1: 2⟩, as shown in Figure 5.8, which generates the set of patterns, {{I2, I3: 4}, {I1, I3: 4}, {I2, I1, I3: 2}}. Finally, I1's conditional pattern base is {{I2: 4}}, whose FP-tree contains only one node, ⟨I2: 4⟩, 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. 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** (Equivalence CLASS Transformation) algorithm developed by Zaki [Zak00].

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 $N.item\text{-}name = p.item\text{-}name$, 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.

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

itemset	*TID_set*
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}

Example 5.6 Mining frequent itemsets using vertical data format. Consider the horizontal data for- mat 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.

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. Because 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 because the itemsets {I1, I4} and {I3, I5} each contain only one transaction, they do not belong to the set of frequent 2-itemsets.

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

Table 5.4 The 2-itemsets in vertical data format.

itemset	*TID_set*
{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.

itemset	*TID_set*
{I1, I2, I3}	{T800, T900}
{I1, I2, I5}	{T100, T800}

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 is *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.

5.1.1 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[9] 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 identifythe 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

[9]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.

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 support_count(X) = 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: $\{\langle a_1, a_2, \ldots, a_{100}\rangle, \langle a_1, a_2, \ldots, a_{50}\rangle\}$, and the minimum support count is *min_sup* = 2. The projection on the first item, a_1, derives the frequent itemset, $\{a_1, a_2, \ldots, a_{50} : 2\}$, based on the *itemset merging* optimization. Because support($\{a_2\}$) = support ($\{a_1, a_2, \ldots, a_{50}\}$) = 2, and $\{a_2\}$ is a proper subset of $\{a_1, a_2, \ldots, a_{50}\}$, there is no need to examine a_2 and its projected database. Similar pruning can be done for $a_3, \ldots, a_{50}$ 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 fre- quent 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.*

$\{\langle a_1, a_2, \ldots, a_{100}\rangle, \langle a_1, a_2, \ldots, a_{50}\rangle\}$, where *min_sup* = 2. Because 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 $a_3, \ldots, a_{50}$. There is no need to mine anything more after mining a_1's projected database.

Besides pruning the search space in the closed itemset mining process, another impor- tant optimization is to perform efficient checking of a newly derived frequent itemset to see whether it is closed, because the mining process 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 whether 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 frame- work, 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 main- tain 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 correspond- ing 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 struc- ture** 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 item- sets. *"Can we extend these methods for efficient mining of maximal frequent itemsets?"* Because maximal frequent itemsets share many similarities with closed frequent item- sets, many of the optimization techniques developed here can be extended to mining maximal frequent itemsets. However, we leave this method as an exercise for interested readers.

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 sup- plemented 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 mea- sures, based on the statistics "behind" the data, can be used as one step toward 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 transac- tions 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 6,000 of the customer transactions included computer games, while 7,500 included videos, and 4,000 included both computer games and videos. Suppose that a data mining program for dis- covering 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:

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

Rule (5.21) is a strong association rule and would therefore be reported, since its support value of $\frac{4,000}{10,000} = 40\%$ and confidence value of $\frac{4,000}{6,000} = 66\%$ satisfy the minimum support and minimum confidence thresholds, respectively. However, Rule (5.21) is misleading because 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 $A => B$ 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 impli- cation between A and B. Hence, alternatives to the support-confidence framework can be useful in mining interesting data relationships.