MODULE -3

ASSOCIATION ANALYSIS

Association Analysis: Association Analysis: Problem Definition, Frequent Item set Generation, Rule generation. Alternative Methods for Generating Frequent Item sets, FP Growth Algorithm, Evaluation of Association Patterns.

---------- ---------- ----------

Business enterprises accumulate large quantities of data from their day to- day operations. For example, huge amount of customer purchase data are collected daily at the checkout counters of grocery stores.

Table below illustrates an example of such data, commonly known as market basket transactions.

Table 6.1. An example of market basket transactions.

TID	Items
1	{Bread, Milk}
2	{Bread, Diapers, Beer, Eggs}
3	{Milk, Diapers, Beer, Cola}
4	{Bread, Milk, Diapers, Beer}
5	{Bread, Milk, Diapers, Cola}

- Each row in this table corresponds to a transaction, which contains a unique identifier labelled TID and a set of items bought by a given customer.
- Retailers are interested in analyzing the data to learn about the purchasing behaviour of their customers.
- Such valuable information can be used to support a variety of business-related applications such as marketing promotions, inventory management and customer relationship management.
- The uncovered relationships can be represented in the form of association rules or sets of frequent items.

  For example, the following rule can be extracted from the data set shown in Table 6.1: {Milk} → {Bread}.

The rule suggests that a strong relationship exists between the sale of Milk and Bread because many customers who buy Milk also buy Bread.

Retailers can use this type of rules to help them identify new opportunities for cross selling

their products to the customers.

Besides market basket data, association analysis is also applicable to other application domains such as bioinformatics, medical diagnosis, Web mining and scientific data analysis.

There are two key issues that need to be addressed when applying association analysis to market basket data.

1. Discovering patterns from a large transaction data set can be computationally expensive.
2. Some of the discovered patterns are potentially spurious because they may happen simply by chance.

6.1 Problem Definition

Binary Representation Market basket data can be represented in a binary format as shown in Table 6.2.

Table 6.2. A binary 0/1 representation of market basket data.

TID	Bread	Milk	Diapers	Beer	Eggs	Cola
1	1	1	0	0	0	0
2	1	0	1	1	1	0
3	0	1	1	1	0	1
4	1	1	1	1	0	0
5	1	1	1	0	0	1

Here each row corresponds to a transaction and each column corresponds to an item. An item can be treated as a binary variable whose value is one if the item is present in a transaction and zero otherwise. Because the presence of an item in a transaction is often considered more important than its absence, an item is an asymmetric binary variable.

Itemset and Support Count :

Let I = $\{i_1,i_2,...,i_d\}$ be the set of all items in a market basket data and T = $\{ t_1,t_2,...,t_d \}$ be the set of all transactions.

- Each transaction t_i contains a subset of items chosen from I.
- In association analysis, a collection of zero or more items is termed an itemset. If an itemset contains k items, it is called a k-itemset. {Bread, Diapers, Milk} is an example of a 3-itemset.
- The null (or empty) set is an itemset that does not contain any items. The transaction width is defined as the number of items present in a transaction.
- A transaction t_j is said to contain an itemset X if X is a subset of t_j.

For example, the second transaction shown in Table 6.2 contains the itemset {Bread, Diapers} but not {Bread, Milk}.

- An important property of an itemset is its **support count**, which refers to the number of transactions that contain a particular itemset.

Mathematically, the support count, σ(X), for an itemset X can be stated as follows:

$$\sigma(X) = \left|\{t_i | X \subseteq t_i,\ t_i \in T\}\right|,$$

where the symbol | . | denote the number of elements in a set.

Eg, In the data set shown in Table 6.2, the **support count** for {Beer, Diapers, Milk} is equal to **two** because there are only two transactions that contain all three items.

Association Rule

An association rule is an implication expression of the form X , Y, where X and Y are disjoint itemsets, i.e., X ∩Y = Ø.

The strength of an association rule can be measured in terms of its **support** and **confidence**.

- Support determines how often a rule is applicable to a given data set.
- Confidence determines how frequently items in Y appear in transactions that contain X.

  The formal definitions of these metrics are

$$\text{Support, } s(X \longrightarrow Y) = \frac{\sigma(X \cup Y)}{N}; \qquad (6.1)$$

$$\text{Confidence, } c(X \longrightarrow Y) = \frac{\sigma(X \cup Y)}{\sigma(X)}. \qquad (6.2)$$

Example 6.1. Consider the rule {**Milk, Diapers} → {Beer}.**

The **support count** for {Milk , Diapers, Beer} is **2** .

Total number of transactions are **5**,

The rule's support is **2 / 5 = 0.4**.

The rule's confidence is obtained by dividing the support count for {Milk, Diapers, Beer} by the support count for {Milk, Diapers}.

Since there are **3** transactions that contain milk and diapers,

The confidence for this rule is **2 / 3 = 0.67.**

Why Use Support and Confidence?

- Support is an important measure because a rule that has very low support may occur simply by chance.
- A low support rule is also likely to be uninteresting from a business perspective. Hence, support is often used to eliminate uninteresting rules.
- Confidence, measures the reliability of the inference made by a rule.
- For a given rule X > Y, the higher the confidence, the more likely it is for Y to be present in transactions that contain X.

Formulation of Association Rule Mining Problem

The association rule mining problem can be formally stated as follows:

Definition 6.1 (Association Rule Discovery) :

Given a set of transactions 7, find all the rules having **support > = minsup** and **confidence > = minconf** ,

where minsup and minconf are the corresponding **support and confidence thresholds.**

A brute-force approach for mining association rules is to compute the support and confidence for every possible rule.

This approach is prohibitively expensive because there are exponentially many rules that can be extracted from a data set.

Te total number of possible rules extracted from a data set that contains d items is :

$$R = 3^d - 2^{d+1} + 1.$$

Even for the small data set shown in Table 6.1, this approach requires us to compute the support and confidence for $\mathbf{3^6 - 2^{7+1} = 602}$ **rules.**

- More than 80% of the rules are discarded after applying minsup >= 20% and minconf >= 50% , thus making most of the computations become wasted.
- To avoid performing needless computations, it would be useful to prune the rules early without having to compute their support and confidence values.
- An initial step toward improving the performance of association rule mining algorithms is to decouple the support and confidence requirements.

From Equation 6.2, the support of a rule X -> Y depends only on the support of its corresponding itemset , X U Y.

For example, the following rules have identical support because they involve items from the same itemset, {Beer, Diapers, MiIk}:

{Beer, Diapers} --→ {Milk}, {Beer, Milk} ----- > { Diapers},

{Diapers, Milk} ---→ { Beer}, {Beer} --→ {Diapers, Milk},

{Milk} ---- > { Beer,Diapers}, {Diapers} --→ {Beer,Milk}.

If the itemset is infrequent, then all six candidate rules can be pruned immediately without our having to compute their confidence values.

Therefore, a common strategy adopted by many association rule mining algorithms is to decompose the problem into two major subtasks:

1**. Frequent Itemset Generation**, whose objective is to find all the itemsets that satisfy the minsup threshold. These itemsets are called frequent itemsets.

2. **Rule Generation**, whose objective is to extract all the high-confidence rules from the frequent itemsets found in the previous step. These rules are called strong rules.

6.2 Frequent Itemset Generation

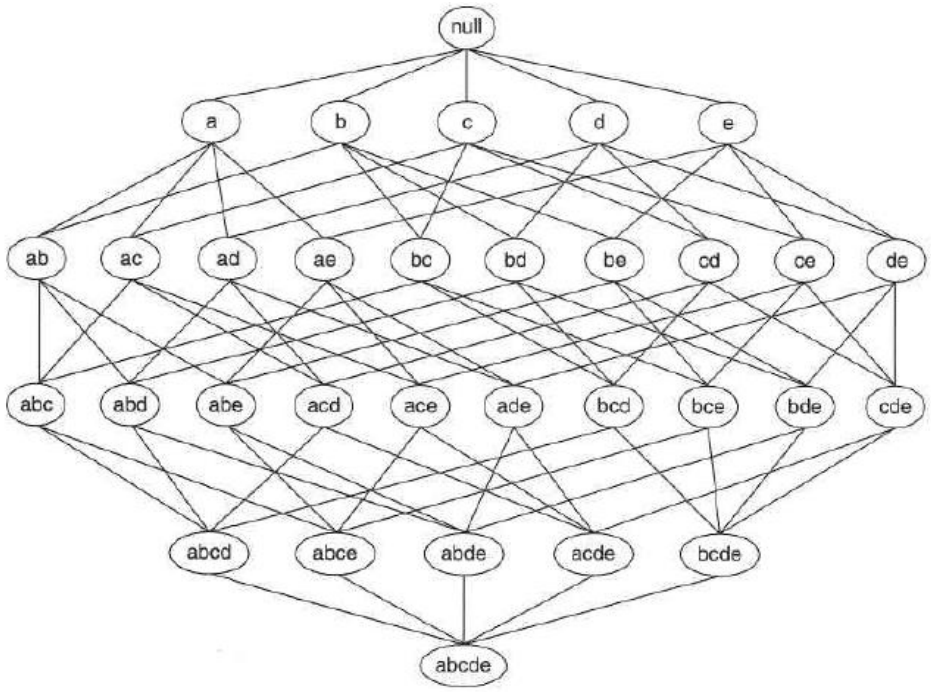


Figure 6.1. An itemset lattice.

A lattice structure can be used to enumerate the list of all possible itemsets. Figure 6.1 shows an itemset lattice for I= {a,b,c.,d,e}.

In general, a data set that contains k items can potentially generate up to 2^k - 7 frequent itemsets, excluding the null set.

A brute-force approach for finding frequent itemsets is to determine the support count for every candidate itemset in the lattice structure.

- Compare each candidate against every transaction, an operation that is shown in Figure 6.2.

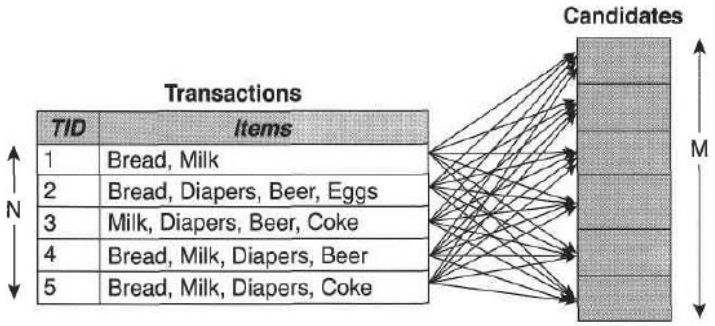


Figure 6.2. Counting the support of candidate itemsets.

- If the candidate is contained in a transaction, its support count will be incremented. Eg. The support for {Bread, Milk) is incremented **three** times because the itemset is contained in transactions 1, 4, and 5.
- Approach can be very expensive because it requires O(NMW) comparisons, where N is the number of transactions, $M = 2^k - 1$ is the number of candidate itemsets, and W is the maximum transaction width.

The computational complexity of frequent itemset generation can be reduced in the following way :

1. **Reduce the number of candidate itemsets (M).** The Apriori principle is an effective way to eliminate some of the candidate itemsets without counting their support values.
2. **Reduce the number of comparisons.** Instead of matching each candidate itemset against every transaction, we can reduce the number of comparisons by using more advanced data structures, either to store the candidate itemsets or to compress the data set.

6.2.1 The Apriori Principle

Eg. Consider the itemset lattice shown in Figure 6.3.

Suppose {c, d, e} is a frequent itemset. Any transaction that contains {c,d,e} must also contain its **subsets**, {c,d}, {c,e}, {d,e}, {c}, {d}, and {e}. As a result, if {c,d,e} is frequent, then all subsets of {c, d,e} (i.e., the shaded itemsets in this figure) must also be frequent.

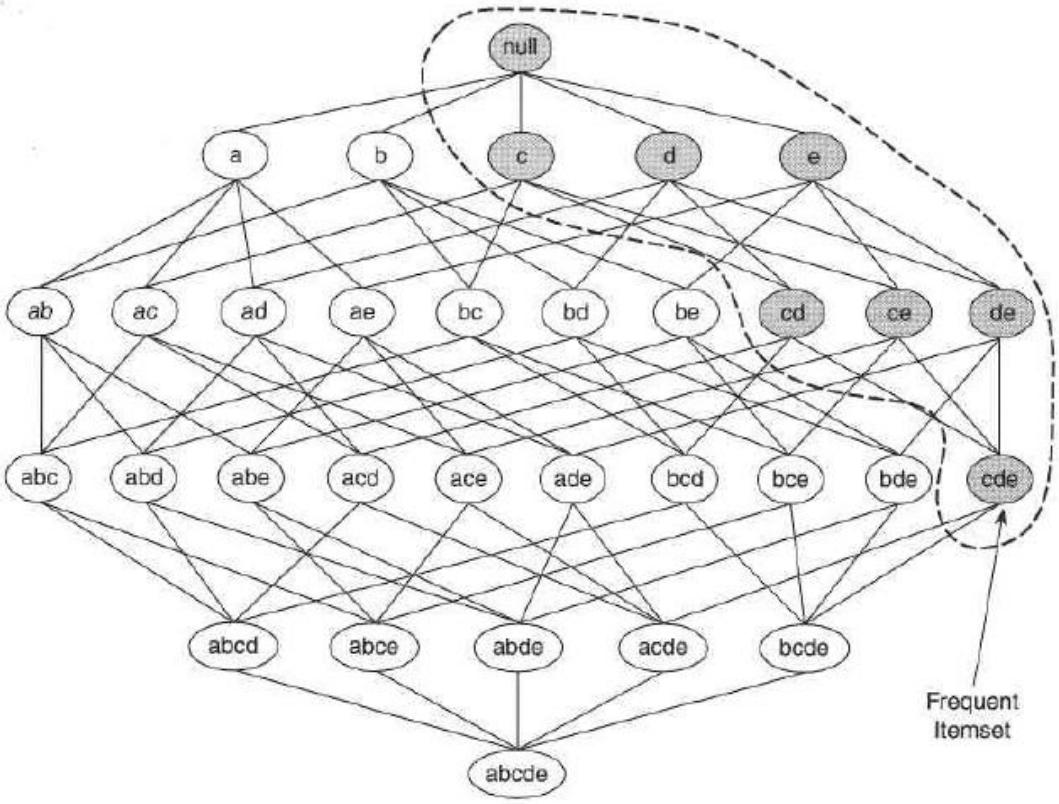


Figure 6.3. An illustration of the *Apriori* principle. If $\{c, d, e\}$ is frequent, then all subsets of this itemset are frequent.

Conversely, if an itemset such as {a, b} is **infrequent**, then all of its **supersets** must be **infrequent** too.

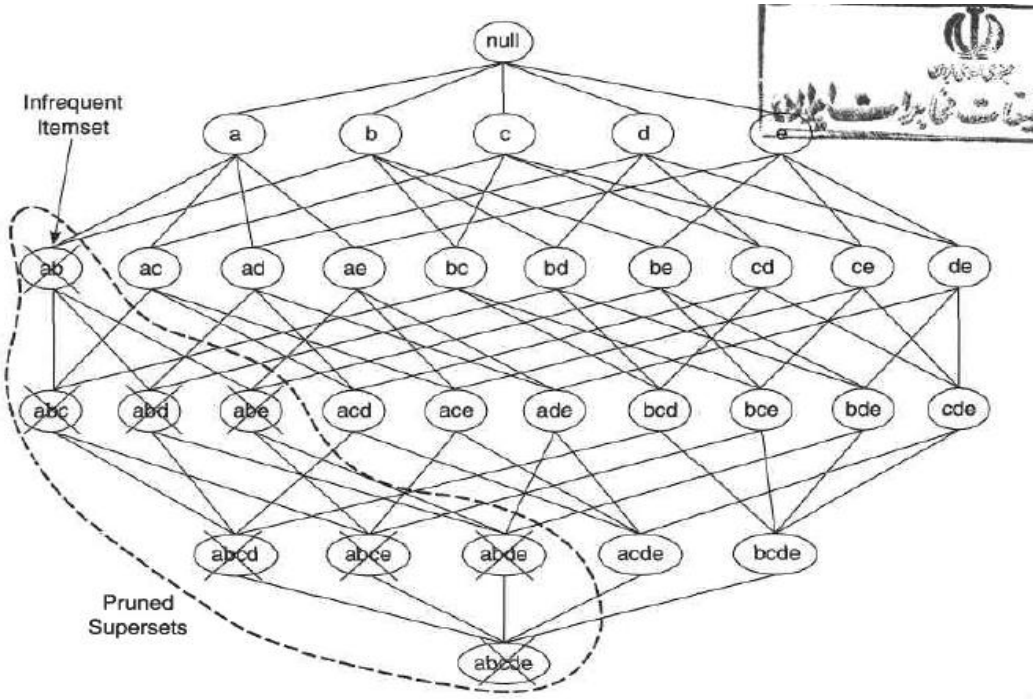


Figure 6.4. An illustration of support-based pruning. If $\{a, b\}$ is infrequent, then all supersets of $\{a, b\}$ are infrequent.

As illustrated in Figure 6.4, the entire subgraph containing the supersets of {a, b} can be pruned immediately once {a, b} is found to be infrequent.

This strategy of trimming the exponential search space based on the support measure is known as **support-based pruning**.

Such a pruning strategy is made possible by a key property of the support measure, namely, that the support for an itemset never exceeds the support for its subsets. This property is also known as the **anti-monotone property** of the support measure.

Definition 6.2 (Monotonicity Property). Let I be a set of items, and J =2^I be the power set of I. A measure f is monotone (or upward closed) if

$$\forall X, Y \in J: \ (X \subseteq Y) \longrightarrow f(X) \leq f(Y),$$

which means that if X is a subset of Y, then f(X) must not exceed f(Y).

f is anti-monotone (or downward closed) if

$$\forall X, Y \in J: \ (X \subseteq Y) \longrightarrow f(Y) \leq f(X),$$

which means that if X is a subset of Y, then f(Y) must not exceed f(X).

6.2.2 Frequent Itemset Generation in the Apriori Algorithm

Apriori, is the first association rule mining algorithm that pioneered the use of support-based pruning to systematically control the exponential growth of candidate itemsets.

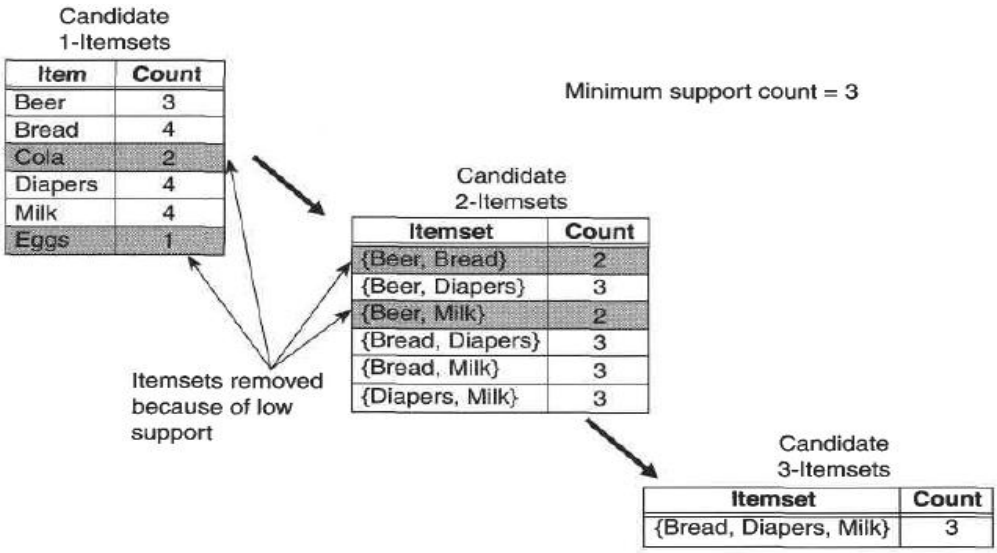


Figure 6.5. *Illustration of frequent itemset generation using the Apriori algorithm.*

Figure 6.5 provides a high-level illustration of the frequent itemset generation part of the Apriori algorithm for the transactions shown in Table 6.1.

We assume that the support threshold is 60%, which is equivalent to a **minimum support count** equal to **3**.

- Initially, every item is considered as a candidate l-itemset.
- After counting their supports, the candidate itemsets {Co1a} and {Eggs} are discarded
- because they appear in fewer than three transactions.
- In the next iteration, candidate 2-itemsets are generated using only the frequent 1-itemsets because the Apriori principle ensures that all supersets of the infrequent 1-itemsets must be infrequent.
- Because there are only four frequent 1-itemsets, the number of candidate 2-itemsets generated by the algorithm is $({}^{4}C_{2}) = 6$.
- Two of these six candidates, {Beer, Bread} and {Beer, Milk}, are subsequently found to be infrequent after computing their support values.
- The remaining four candidates are frequent, and thus will be used to generate candidate 3-itemsets.

- Without support-based. pruning, there are ($^{6}C_{3}$) =20 candidate 3-itemsets that can be formed using the six items given in this example.
- With the Apriori principle, we only need to keep candidate 3-itemsets whose subsets are frequent.
- The only candidate that has this property is {Bread, Diapers, Milk).

The effectiveness of the Apriori pruning strategy can be shown by counting the number of candidate itemsets generated.

Brute force strategy :

$$\binom{6}{1} + \binom{6}{2} + \binom{6}{3} = 6 + 15 + 20 = 41$$

The Apriori principle, decreases the candidates, which represents a 68% reduction in the number of candidate itemsets even in this simple example.

$$\binom{6}{1} + \binom{4}{2} + 1 = 6 + 6 + 1 = 13$$

The pseudocode for the frequent itemset generation part of the Apriori, algorithm is shown in Algorithm 6.1.

Algorithm 6.1 Frequent itemset generation of the *Apriori* algorithm.

1: $k = 1$.
2: $F_k = \{ i \mid i \in I \wedge \sigma(\{i\}) \geq N \times minsup\}$. {Find all frequent 1-itemsets}
3: **repeat**
4: $k = k + 1$.
5: $C_k = \text{apriori-gen}(F_{k-1})$. {Generate candidate itemsets}
6: **for** each transaction $t \in T$ **do**
7: $C_t = \text{subset}(C_k, t)$. {Identify all candidates that belong to t}
8: **for** each candidate itemset $c \in C_t$ **do**
9: $\sigma(c) = \sigma(c) + 1$. {Increment support count}
10: **end for**
11: **end for**
12: $F_k = \{ c \mid c \in C_k \wedge \sigma(c) \geq N \times minsup\}$. {Extract the frequent k-itemsets}
13: **until** $F_k = \emptyset$
14: Result $= \bigcup F_k$.

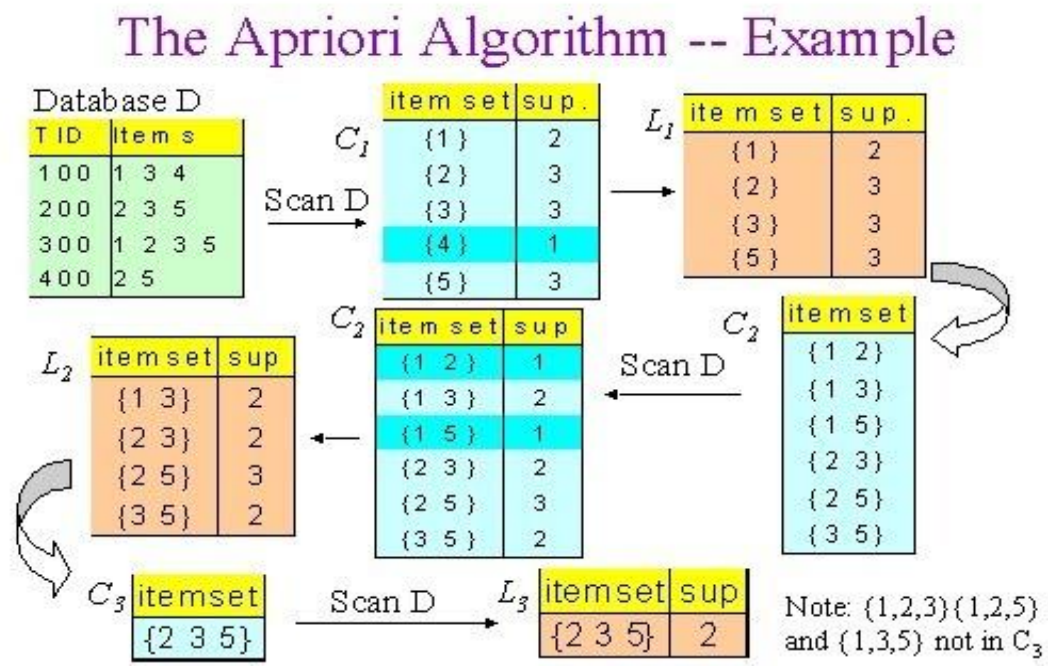
Let Cp denote the set of candidate k-itemsets and Fk denote the set of frequent k-itemsets:

- The algorithm initially makes a single pass over the data set to determine the support of each item. Upon completion of this step, the set of all frequent 1-itemsets, F_1, will be known (steps 1 and 2).
- Next, the algorithm will iteratively generate new candidate k-itemsets using the frequent (k - 1)-itemsets found in the previous iteration (step 5).
- To count the support of the candidates, the algorithm needs to make an additional pass over the data set (steps 6- 10). The subset function is used to determine all the candidate itemsets in Ca that are contained in each transaction f.
- After counting their supports, the algorithm eliminates all candidate itemsets whose support counts are less than minsup (step 12).
- The algorithm terminates when there are no new frequent itemsets generated, i.e., Fk=Ø (step 13).

The frequent itemset generation part of the Apriori algorithm has two important characteristics.

1. It is a **level-wise** algorithm; i.e., it traverses the itemset lattice one level at a time, from frequent 1-itemsets to the maximum size of frequent itemsets.
2. It employs a **generate-and-test** strategy for finding frequent itemsets. At each iteration, new candidate itemsets are generated from the frequent itemsets found in the previous iteration.

The support for each candidate is then counted and tested against the minsup threshold. The total number of iterations needed by the algorithm is kmax+1, where kmax is the maximum size of the frequent itemsets.



6.2.3 Candidate Generation and Pruning

The apriori-gen function shown in Step 5 of Algorithm 6.1 generates candidate itemsets by performing the following two operations:

1. **Candidate Generation**. This operation generates new candidate k-itemsets based on the frequent (k - l)-itemsets found in the previous iteration.

2. **Candidate Pruning**. This operation eliminates some of the candidate k-itemsets using the support-based pruning strategy.

Eg. For candidate pruning operation : Consider a candidate k-itemset, X= {i1,i2,...,ik}. The algorithm must determine whether all of its proper subsets, $X - \{i_j\}\ (\forall j = 1, 2, \ldots, k)$ are frequent.

If one of them is infrequent, then X is immediately pruned. This approach can effectively reduce the number of candidate itemsets considered during support counting. The complexity of this operation is O(k) for each candidate k-itemset.

The following is a list of requirements for an **effective candidate generation procedure**:

1. It should avoid generating too many unnecessary candidates. A candidate itemset is unnecessary if at least one of its subsets is infrequent.

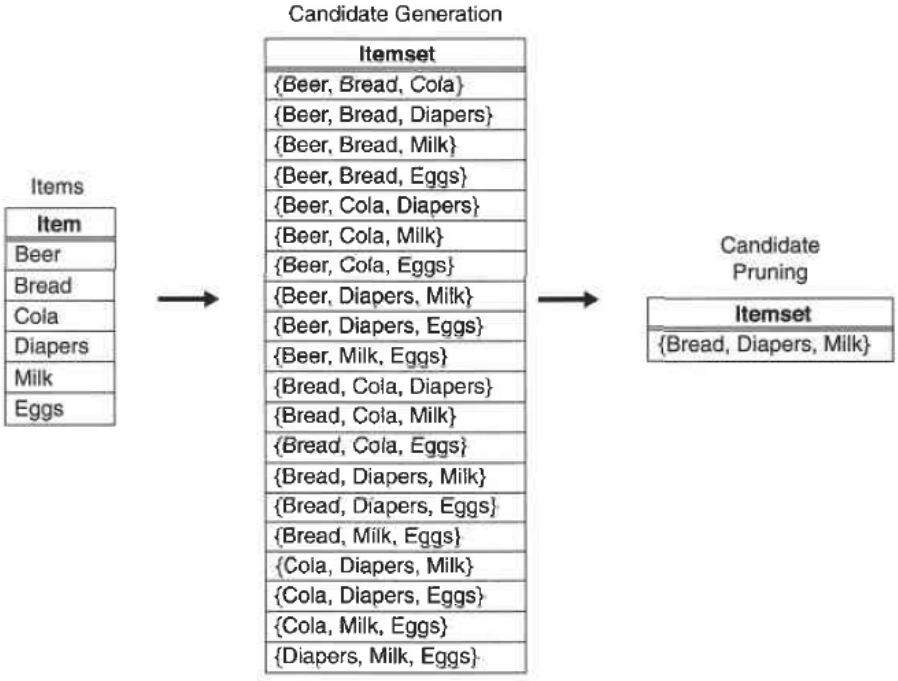
Such a candidate is guaranteed to be infrequent according to the antimonotone property of support.

2. It must ensure that the candidate set is complete, i.e., no frequent itemsets are left out by the candidate generation procedure. To ensure completeness, the set of candidate itemsets must subsume the set of all frequent itemsets, i.e., $\forall k : F_k \subseteq C_k$.

3. It should not generate the same candidate itemset more than once.

For example, the candidate itemset {a,b,c,d} can be generated in many ways-by merging {a,b, c} with {d}, {b, d} with {a,c}, {c} with {a,b,d}, etc.Generation of duplicate candidates leads to wasted computations and thus should be avoided for efficiency reasons.

Brute-Force Method : The brute-force method considers every k-itemset as a potential candidate and then applies the candidate pruning step to remove any unnecessary candidates. The number of candidate itemsets generated at level k is equal to (${}^{d}C_{k}$), where d is the total number of items.



Candidate pruning becomes extremely expensive because a large number of itemsets must be examined.

Given that the amount of computations needed for each candidate is O(k), the overall complexity of this method $O\left(\sum_{k=1}^{d} k \times \binom{d}{k}\right) = O\left(d \cdot 2^{d-1}\right)$.

An alternative method for candidate generation is to extend each frequent (k - 1)-itemset with other frequent items.

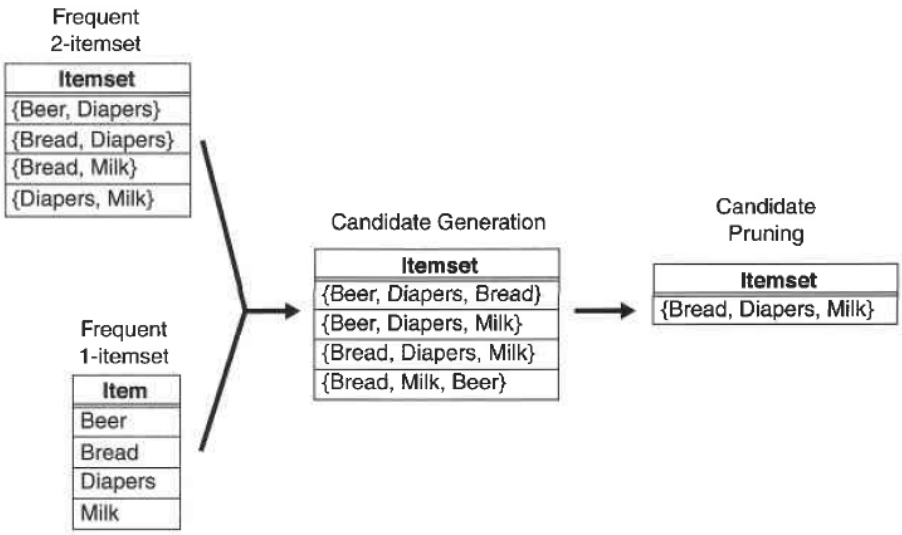


Figure illustrates how a frequent 2-itemset such as {Beer, Diapers} can be augmented with a frequent item such as Bread to produce a candidate 3-itemset {Beer, Diapers, Bread}. This method will produce O(l F_{k-1} 1 x l F_k l) candidate, k-itemsets, where l F j l is the number of frequent j-itemsets. The overall complexity of this step is $O(\sum_k k|F_{k-1}||F_1|)$.

The procedure is complete because every frequent k-itemset is composed of a frequent (k - 1)-itemset and a frequent 1-itemset. Therefore, all frequent k-itemsets are part of the candidate k-itemsets generated by this procedure.

This approach, does not prevent the same candidate itemset from being generated more than once. For instance, {Bread, Diapers, Mitk} can be generated by merging {Bread, Diapers} with {Milk}, {Bread, MiIk} with {Diapers} or {Diapers, MiIk} with {Bread}.

One way to avoid generating duplicate candidates is by ensuring that the items in each frequent itemset are kept sorted in their lexicographic order.

Each frequent (k-l)-itemset X is then extended with frequent items that are lexicographically larger than the items in X.

For example, the itemset {Bread, Diapers} can be augmented with {Milk} since Milk is lexicographically larger than Bread and Diapers.

{Diapers, MiIk} with {Bread} and {Bread, Milk} with {Diapers} should not be augmented because they violate the lexicographic ordering condition.

It still produces a large number of unnecessary candidates. For example, the candidate itemset obtained by merging {Beer, Diapers} with {Milk} is unnecessary because one of its subsets, {Beer, Milk}, is infrequent.

Other technique available to reduce the number of unnecessary candidates is for example, for every candidate k-itemset that survives the pruning step, every item in the candidate must be

contained in at least k - 1 of the frequent (k - 1)-itemsets. Otherwise, the candidate is guaranteed to be infrequent.

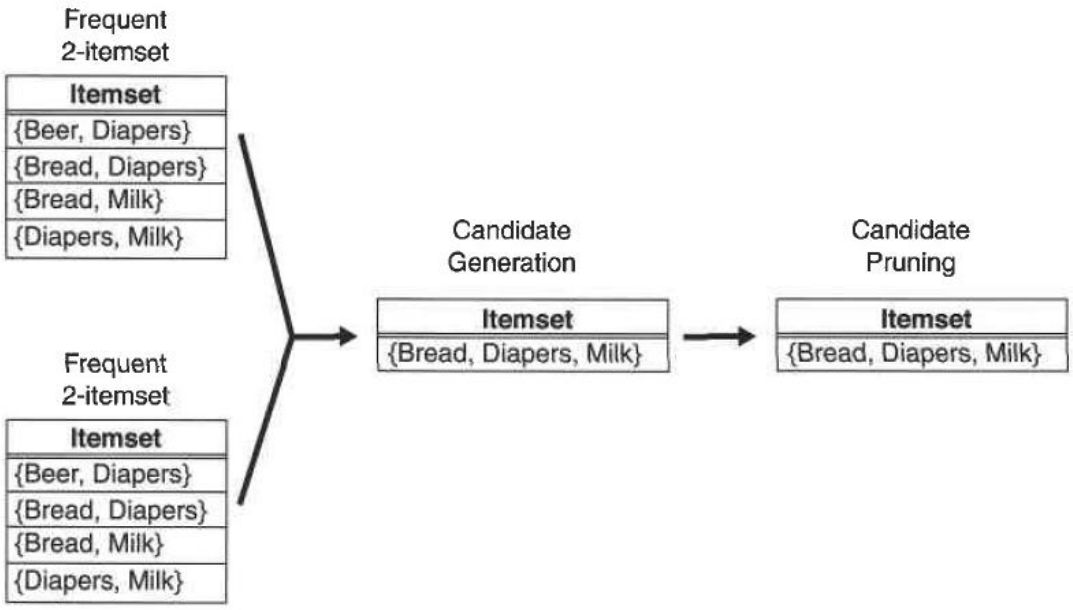
For example, {Beer, Diapers, Milk} is a viable candidate 3-itemset only if every item in the candidate, including Beer, is contained in at least two frequent 2-itemsets. Since there is only one frequent 2-itemset containing Beer, all candidate itemsets involving Beer must be infrequent.

F_{k-1} x F_{k-1} Method : The candidate generation procedure in the apriori-gen function merges a pair of frequent (k - l)-itemsets only if their first k - 2 items are identical.

Let

$$A = \{a_1, a_2, \ldots, a_{k-1}\} \text{ and } B = \{b_1, b_2, \ldots, b_{k-1}\}$$

be a pair of of frequent (k - l)-itemsets. A and B are merged if they satisfy the following conditions : $a_i = b_i \text{ (for } i = 1, 2, \ldots, k-2) \text{ and } a_{k-1} \neq b_{k-1}.$



In the Figure above, the frequent itemsets {Bread, Diapers} and {Bread, Milk} are merged to form a candidate 3-itemset {Bread, Diapers, Milk}.

The algorithm does not have to merge {Beer, Diapers} with {Diapers, Milk} because the first item in both itemsets is different.

If {Beer, Diapers, Milk} is a viable candidate, it would have been obtained by merging {Beer, Diapers} with {Beer, MiIk} instead.

This example illustrates both the completeness of the candidate generation procedure and the advantages of using lexicographic ordering to prevent duplicate candidates.

6.2.4 Support Counting

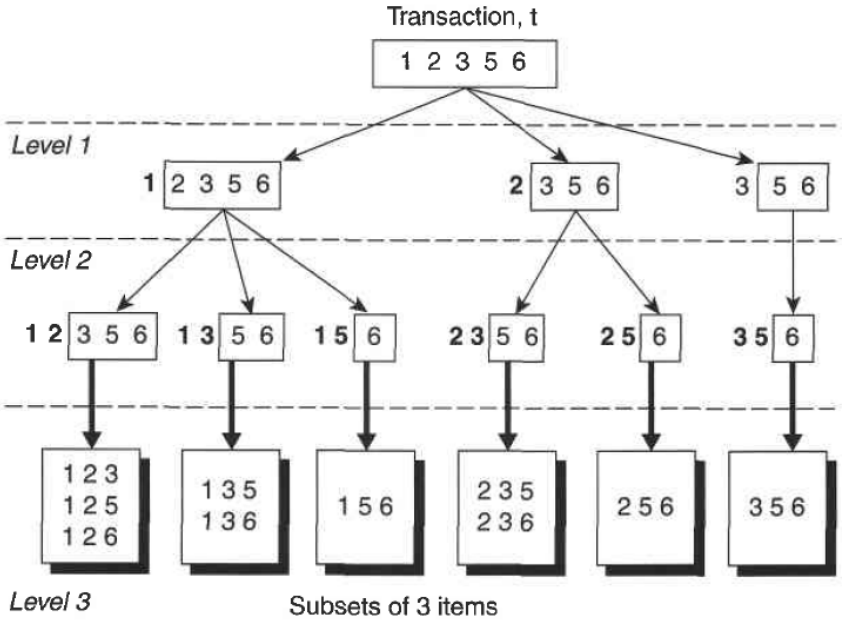
Support counting is the process of determining the frequency of occurrence for every candidate itemset that survives the candidate pruning step of the apriori-gen function.

Support counting is implemented in steps 6 through 11 of Algorithm 6.1.

One approach for doing this is to compare each transaction against every candidate itemset and to update the support counts of candidates contained in the transaction.

This approach is computationally expensive, especially when the numbers of transactions and candidate itemsets are large.

An alternative approach is to enumerate the itemsets contained in each transaction and use them to update the support counts of their respective candidate itemsets.



Example: Consider a transaction t that contains five items, {1,2,3,5,6}.

There are $^{5}C_{3} = 10$ itemsets of size 3 contained in this transaction.

Figure shows a systematic way for enumerating the 3-itemsets contained in l.

Assuming that each itemset keeps its items in increasing lexicographic order, an itemset can be enumerated by specifying the smallest item first, followed by the larger items.

For instance, given t : {L,2,3,5,6}, all the 3- itemsets contained in 't' must begin with item 1, 2, or 3.

It is not possible to construct a 3-itemset that begins with items 5 or 6 because there are only two items in 't' whose labels are greater than or equal to 5.

The number of ways to specify the first item of a 3-itemset contained in t is illustrated by the Level 1 prefix structures depicted in Figure above.

Eg $1\boxed{2\ 3\ 5\ 6}$ represents a 3-itemset that begins with item 1, followed by two more items chosen from the set { 2 , 3 , 5 , 6 } .

After fixing the first item, the prefix structures at Level 2 represent the number of ways to select the second item.

For example, $1\ 2\boxed{3\ 5\ 6}$ corresponds to itemsets that begin with prefix (1 2) and are followed by items 3, 5, or 6.

Finally, the prefix structures at Level 3 represent the complete set of 3-itemsets contained in 't'.

For example, the 3-itemsets that begin with prefix {1 2} are {1,2,3}, {1,2,5}, and {1,2,6}, while those that begin with prefix {2 3} are {2,3,5} and { 2 , 3 , 6 } .

The prefix structures shown in Figure above demonstrate how itemsets contained in a transaction can be systematically enumerated, i.e., by specifying their items one by one, from the leftmost item to the rightmost item.

Support Counting Using a Hash Tlee

In the Apriori, algorithm, candidate itemsets are partitioned into different buckets and stored in a hash tree.

During support counting, itemsets contained in each transaction are also hashed into their appropriate buckets.

Instead of comparing each itemset in the transaction with every candidate itemset, it is matched only against candidate itemsets that belong to the same bucket, as shown in Figure below.

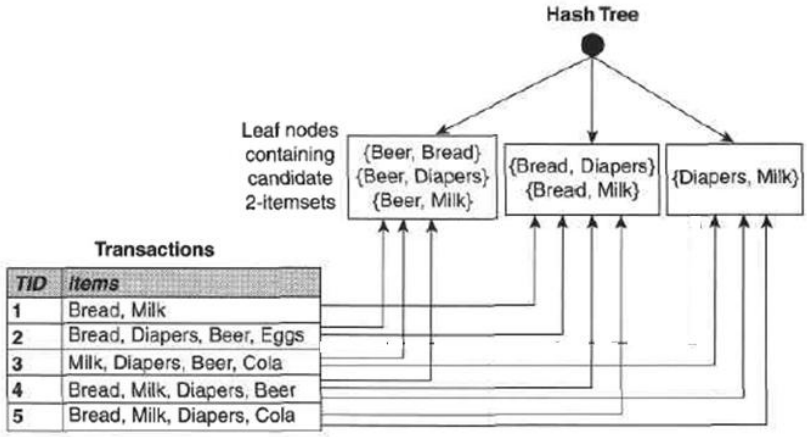
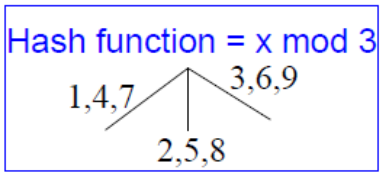


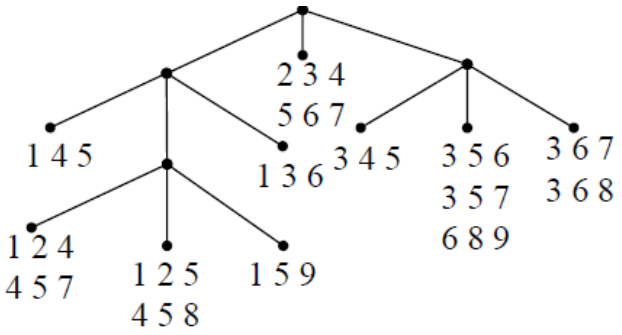
Figure 6.10. Counting the support of itemsets using hash structure.

Figure below shows an example of a hash tree structure.

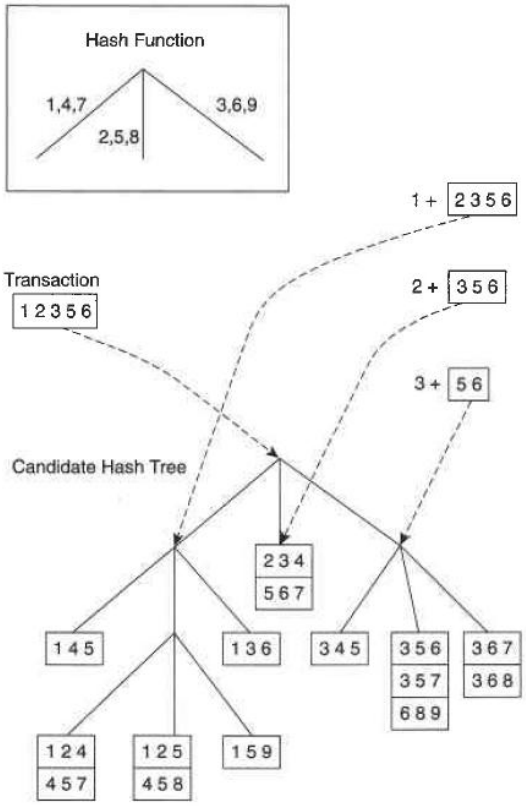
Consider an item set containing 15 candidate itemsets of length 3: {1 4 5}, {1 2 4}, {4 5 7}, {1 2 5}, {4 5 8}, {1 5 9}, {1 3 6}, {2 3 4}, {5 6 7}, {3 4 5}, {3 5 6}, {3 5 7}, {6 8 9}, {3 6 7}, {3 6 8}

We need Hash function and Leaf nodes stores the itemsets .



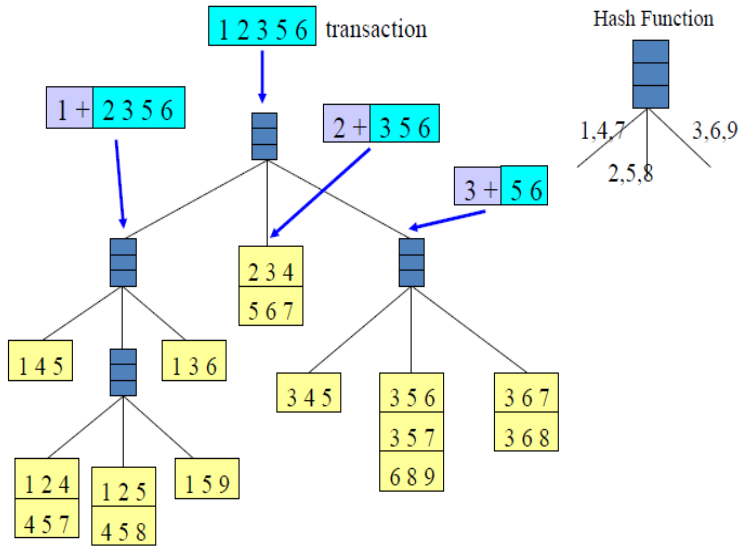


For a given transaction {1,2,3,5,6} , Hashing transaction at the root node of hash tree tree is :

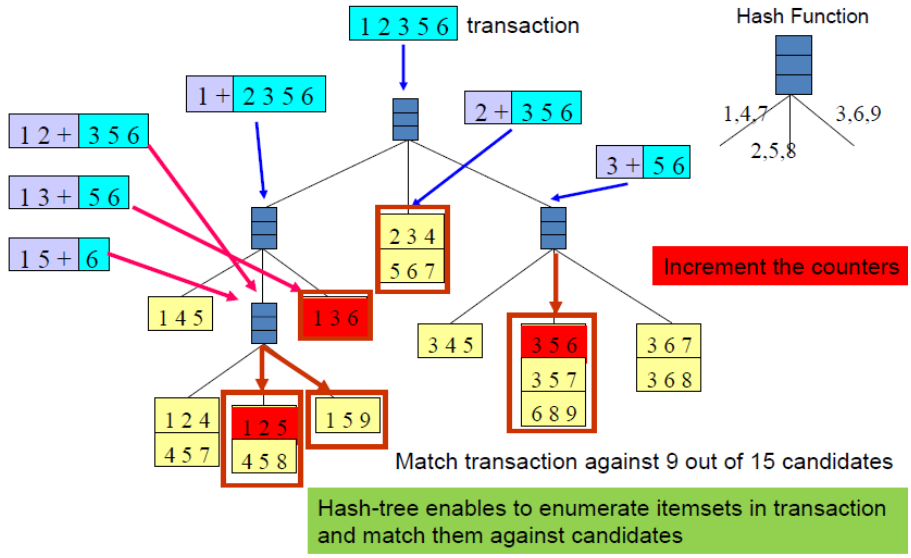


Subset Operation using Hash tree can be drawn as follows:

Figure (a)



Final resultant Hash tree is :



Each internal node of the tree uses the following hash function, h(p) = p mod 3, to determine which branch of the current node should be followed next.

For example, items 1, 4, and 7 are hashed to the same branch (i.e., the leftmost branch) because they have the same remainder after dividing the number by 3.

All candidate itemsets are stored at the leaf nodes of the hash tree. The hash tree shown in Figure above contains 15 candidate 3-itemsets, distributed across 9 leaf nodes.

Consider a transaction, t = {1,2,3,5,6}.

To update the support counts of the candidate itemsets, the hash tree must be traversed in such a way that all the leaf nodes containing candidate 3-itemsets belonging to t must be visited at least once.

The 3-itemsets contained in t must begin with items 1, 2,or 3, as indicated by the Level 1 prefix structures shown in the Figure in section 6.2.4.

Therefore, at the root node of the hash tree, the items 1, 2, and 3 of the transaction are hashed separately.

Item 1 is hashed to the left child ofthe root node.

Item 2 is hashed to the middle child.

Item 3 is hashed to the right child.

At the next level of the tree, the transaction is hashed on the second item listed in the Level 2 structures shown in the Figure in section 6.2.4.

For example, after hashing on item 1 at the root node, items 2, 3, and 5 of the transaction are hashed.

Items 2 and 5 are hashed to the middle child, while item 3 is hashed to the right child, as shown in Figure a.

This process continues until the leaf nodes of the hash tree are reached.

The candidate itemsets stored at the visited leaf nodes are compared against the transaction.

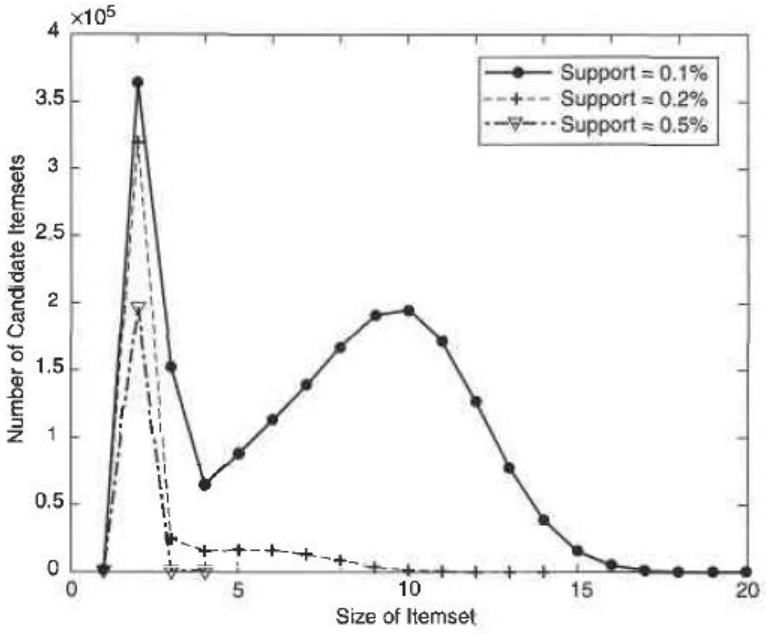
If a candidate is a subset of the transaction, its support count is incremented.

In the above example, 5 out of the 9 leaf nodes are visited and 9 out of the 15 itemsets are compared against the transaction.

6.2.5 Computational Complexity

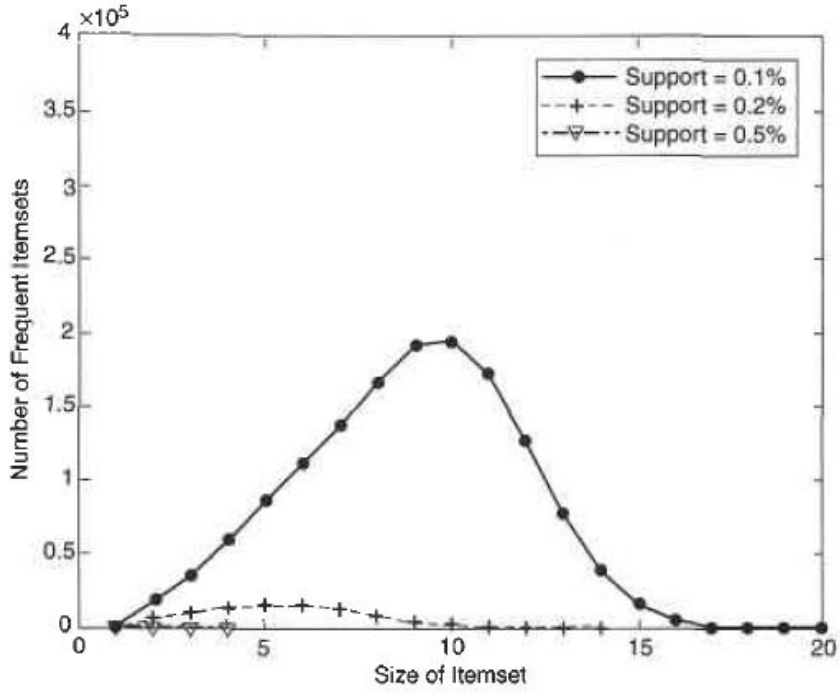
The computational complexity of the Apriori, algorithm can be affected by the following factors:

Support Threshold: Lowering the support threshold often results in more itemsets being declared as frequent. This has an adverse effect on the computational complexity of the algorithm because more candidate itemsets must be generated and counted, as shown in Figure.



(a) Number of candidate itemsets.

The maximum size of frequent itemsets also tends to increase with lower support thresholds. As the maximum size of the frequent itemsets increases, the algorithm will need to make more passes over the data set.



(b) Number of frequent itemsets.

Number of Items (Dimensionality): As the number of items increases, more space will be needed to store the support counts of items.

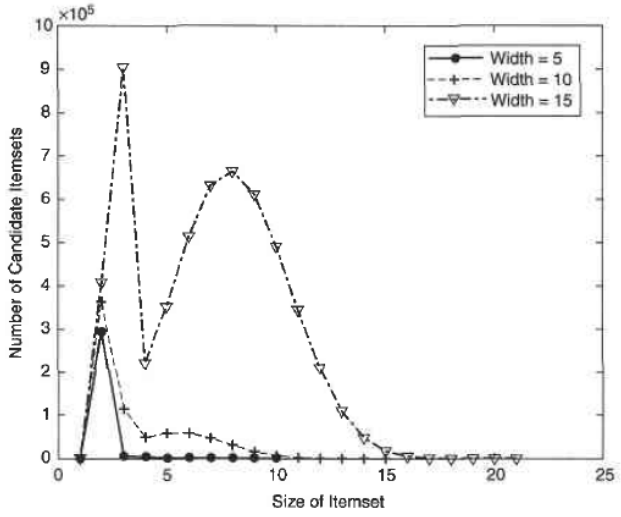
If the number of frequent items also grows with the dimensionality of the data, the computation and I/O costs will increase because of the larger number of candidate itemsets generated by the algorithm.

Number of Transactions: Since the Apriori, algorithm makes repeated passes over the data set, its run time increases with a larger number of transactions.

Average transaction Width: For dense data sets, the average transaction width can be very large.

This affects the complexity of the Apriori algorithm in two ways:

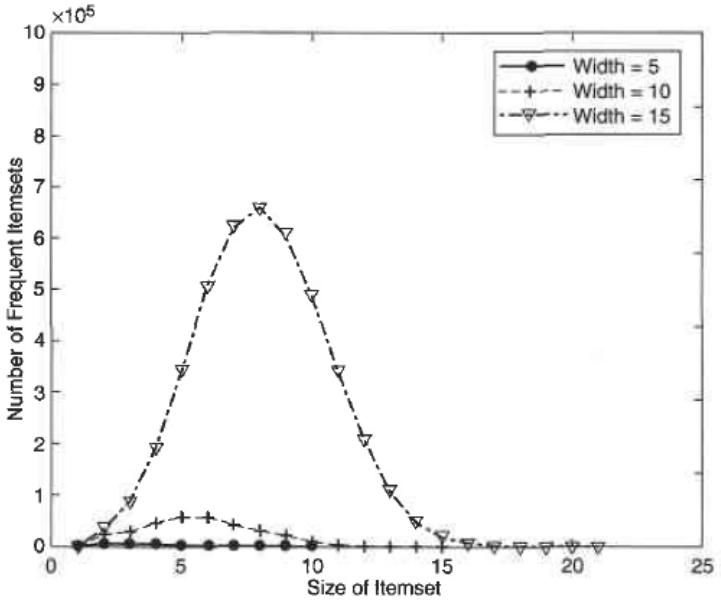
1. The maximum size of frequent itemsets tends to increase as the average transaction width increases. As a result, more candidate itemsets must be examined during candidate generation and support counting, as shown in Figure.
2. As the transaction width increases, more itemsets are contained in the transaction. This will increase the number of hash tree traversals performed during support counting.



(a) Number of candidate itemsets.

Generation of frequent l-itemsets: For each transaction, we need to update the support count for every item present in the transaction.

Assuming that w is the average transaction width, this operation requires O(Nw) time, where N is the total number of transactions.



(b) Number of Frequent Itemsets.

Candidate generation: To generate candidate k-itemsets, pairs of frequent (k - l)-itemsets are merged to determine whether they have at least k – 2 items in common.

Each merging operation requires at most k - 2 equality comparisons.

In the best-case scenario, every merging step produces a viable candidate k-itemset.

In the worst-case scenario, the algorithm must merge every pair of frequent (k - 1)-itemsets found in the previous iteration.

Therefore, the overall cost of merging frequent itemsets is

$$\sum_{k=2}^{w}(k-2)|C_k| < \text{ Cost of merging } < \sum_{k=2}^{w}(k-2)|F_{k-1}|^2.$$

A hash tree is also constructed during candidate generation to store the candidate itemsets. Because the maximum depth of the tree is k, the cost for populating the hash tree with candidate itemsets is $O\left(\sum_{k=2}^{w} k|C_k|\right)$.

During candidate pruning, we need to verify that the ,k - 2 subsets of every candidate k-itemset are frequent.

Since the cost for looking up a candidate in a hash tree is O(k), the candidate pruning step requires $O\left(\sum_{k=2}^{w} k(k-2)|C_k|\right)$ time.

Support counting : Each transaction of length l t l produces $^{l\,t\,l}$ C $_k$ itemsets of size k. This is also the effective number of hash tree traversals performed for each transaction.

The cost for support counting is $O\left(N \sum_k \binom{w}{k} \alpha_k\right)$.

where w is the maximum transaction width and α_k, is the cost for updating the support count of a candidate k-itemset in the hash tree.

6.3 Rule Generation

Each frequent k-itemset, Y, can produce up to 2^k - 2 association rules, ignoring rules that have empty antecedents or consequents (Ø--- >Y or Y→Ø). An association rule can be extracted by partitioning the itemset Y into two non-empty subsets, X and Y ->X, such that X ->Y-X satisfies the confidence threshold. All such rules must have already met the support threshold because they are generated from a frequent itemset.

Example: Let X = {1,2,3} be a frequent itemset. There are six candidate association rules that can be generated from X : {1,2} -> {3}, {1,3} -> {2}, {2,3}-> { 1}, {1} -> { 2,3}, {2}-->{ 1,3}, and {3}→{ 1,2}.

As each of their support is identical to the support for X, the rules must satisfy the support threshold.

Computing the confidence of an association rule does not require additional scans of the transaction data set.

Consider the rule {1,2} -> {3}, which is generated from the frequent itemset X : {1,2,3}. The confidence for this rule is ☐({1, 2,3}) / ☐({1, 2}). Because {1, 2, 3} is frequent, the anti-monotone property of support ensures that {1,2} must be frequent, too.

Since the support counts for both itemsets were already found during frequent itemset generation, there is no need to read the entire data set again.

6.3.1 Confidence-Based Pruning

The following theorem holds for the confidence measure.

Theorem 6.2: If a rule X →Y - X **does not** satisfy the confidence threshold, then any rule X' → Y – X' , where X' ,is a subset of X, **must not** satisfy the confidence threshold as well.

To prove this theorem, consider the following two rules: X' -→Y – X' and X ->Y-X,where X'C X. The confidence of the rules are ☐ (Y) / ☐ (X') and ☐ (Y) / ☐ (X), respectively. Since X' is a subset of X , ☐ (X') >= ☐ (X).

Therefore, the former rule cannot have a higher confidence than the latter rule.

6.3.2 Rule Generation in Apriori Algorithm

The Apriori algorithm uses a level-wise approach for generating association rules, where each level corresponds to the number of items that belong to the rule consequent.

Initially, all the high-confidence rules that have only one item in the rule consequent are extracted. These rules are then used to generate new candidate rules.

For example, if {acd}→ {b} and {abd} →{c} are high-confidence rules, then the candidate rule {a,d} →{,b c} is generated by merging the consequents of both rules.

Figure below shows a lattice structure for the association rules generated from the frequent itemset {a,b,c,d}.

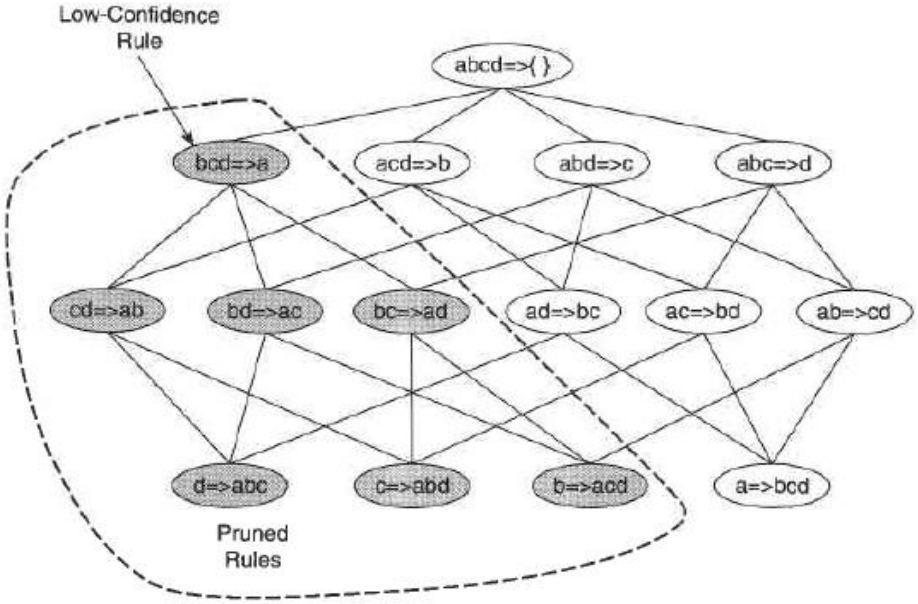


Figure 6.15. Pruning of association rules using the confidence measure.

If any node in the lattice has low confidence, then according to Theorem 6.2, the entire subgraph spanned by the node can be pruned immediately.

Suppose the confidence for {bcd} -> {a } is low. All the rules containing item a in its consequent, including {cd} -> {ab}, {bd}-> {a c}, {b c) ->{ a d}, and {d} -> {a b c} can be discarded.

The difference between Rule geneeration and Support counting algorithms is that, in rule generation, we do not have to make additional passes over the data set to compute the confidence of the candidate rules.

Here we determine the confidence of each rule by using the support counts computed during frequent itemset generation.

Algorithm 6.2 Rule generation of the *Apriori* algorithm.

1: **for** each frequent k-itemset f_k, $k \geq 2$ **do**
2: $H_1 = \{i \mid i \in f_k\}$ {1-item consequents of the rule.}
3: call ap-genrules$(f_k, H_1.)$
4: **end for**

Algorithm 6.3 Procedure ap-genrules(f_k, H_m).

1: $k = |f_k|$ {size of frequent itemset.}
2: $m = |H_m|$ {size of rule consequent.}
3: **if** $k > m + 1$ **then**
4: $H_{m+1} =$ apriori-gen(H_m).
5: **for** each $h_{m+1} \in H_{m+1}$ **do**
6: $conf = \sigma(f_k)/\sigma(f_k - h_{m+1})$.
7: **if** $conf \geq minconf$ **then**
8: **output** the rule $(f_k - h_{m+1}) \longrightarrow h_{m+1}$.
9: **else**
10: **delete** h_{m+1} from H_{m+1}.
11: **end if**
12: **end for**
13: call ap-genrules$(f_k, H_{m+1}.)$
14: **end if**

6.5 Alternative Methods for Generating Frequent Itemsets

General-to-Specific versus Specific-to-General: The Apriori algorithm uses a general-to-specifics search strategy, where pairs of frequent (k- l)-itemsets are merged to obtain candidate k-itemsets.

The general to-specific search strategy is effective, provided the maximum length of a frequent itemset is not too long.

The configuration of frequent itemsets that works best with this strategy is shown in Figure 6.19(a), where the darker nodes represent infrequent itemsets.

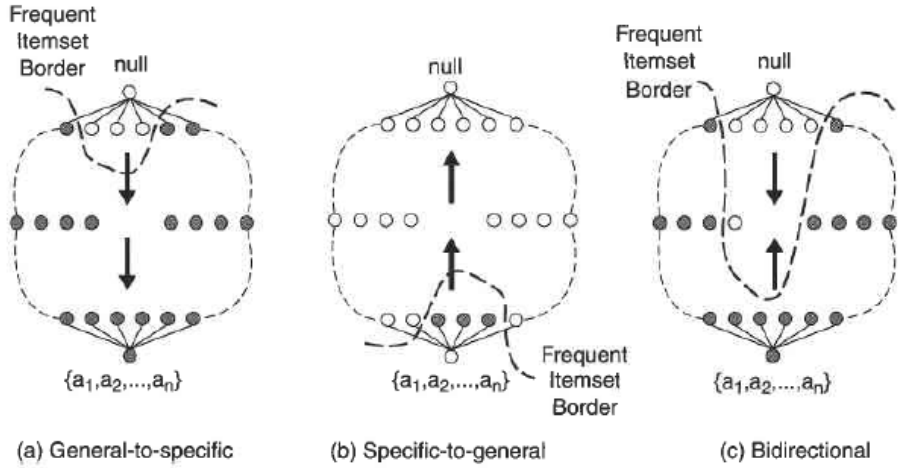


Figure 6.19. General-to-specific, specific-to-general, and bidirectional search.

Alternatively, a specific to- general search strategy looks for more specific frequent itemsets first, before finding the more general frequent itemsets. This strategy is useful to discover maximal frequent itemsets in dense transactions, where the frequent itemset border is located near the bottom of the lattice, as shown in Figure 6.19(b).

The Apriori, principle can be applied to prune all subsets of maximal frequent itemsets. Specifically, if a candidate k-itemset is maximal frequent, we do not have to examine any of its subsets of size k - l.

If the candidate k-itemset is infrequent, we need to check all of its k - 1 subsets in the next iteration.

Another approach is to combine both general-to-specific and specific-to-general search strategies. This bidirectional approach requires more space to store the candidate itemsets, but it can help to rapidly identify the frequent itemset border, given the configuration shown in Figure 6.19(c).

Equivalence Classes: Another way to envision the traversal is to first partition the lattice into disjoint groups of nodes (or equivalence classes).

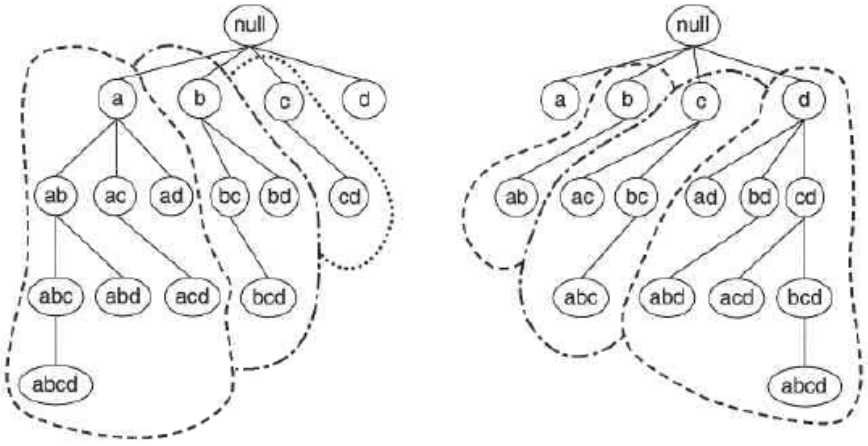
A frequent itemset generation algorithm searches for frequent itemsets within a particular equivalence class first before moving to another equivalence class.

Example, the level-wise strategy used in the Apriori, algorithm can be considered to be partitioning the lattice on the basis of itemset sizes; i.e., the algorithm discovers all frequent l-itemset first before proceeding to larger-sized itemsets.

Equivalence classes can also be defined according to the prefix or suffix labels of an itemset. In this case, two itemsets belong to the same equivalence class if they share a common prefix or suffix of length k.

In the prefix-based approach, the algorithm can search for frequent itemsets starting with the prefix a before looking for those starting with prefixes b) c) and so on.

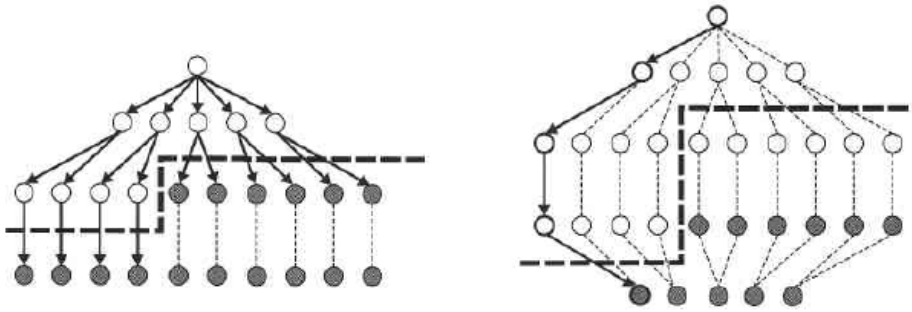
Both prefix-based and suffix-based equivalence classes can be demonstrated using the tree-like structure shown in Figure 6.20.



(a) Prefix tree. (b) Suffix tree.

Figure 6.20. Equivalence classes based on the prefix and suffix labels of itemsets.

Breadth-First versus Depth-First: The Apriori, algorithm traverses the lattice in a breadth-first manner) as shown in Figure 6.21(a).



(a) Breadth first (b) Depth first

Figure 6.21. Breadth-first and depth-first traversals.

It first discovers all the frequent 1-itemsets, followed by the frequent 2-itemsets, and so on, until no new frequent itemsets are generated. The itemset lattice can also be traversed in a depth-first manner, as shown in Figures 6.21(b) and 6.22.

The algorithm can start from, say, node a, in Figure 6.22, and count its support to determine whether it is frequent. If so, the algorithm progressively expands the next level of nodes, i.e., ab, abc, and so on, until an infrequent node is reached, say, abcd.

It then backtracks to another branch, say, abce, and continues the search from there.

The depth-first approach is often used by algorithms designed to find maximal frequent itemsets. This approach allows the frequent itemset border to be detected more quickly than using a breadth-first approach.

Once a maximal frequent itemset is found, substantial pruning can be performed on its subsets.

For example, if the node bcde shown in Figure 6.22 is maximal frequent, then the algorithm does not have to visit the subtrees rooted at bd,, be, c, d, and e because they will not contain any maximal frequent itemsets.

If abc is maximal frequent, only the nodes such as ac and bc arc not maximal frequent (but the subtrees of ac and bc may still contain maximal frequent itemsets).

The depth-first approach also allows a different kind of pruning based on the support of itemsets.

For example, suppose the support for {a,b,c} is identical to the support for {a, b}.

The subtrees rooted at abd and abe can be skipped because they are guaranteed not to have any maximal frequent itemsets.

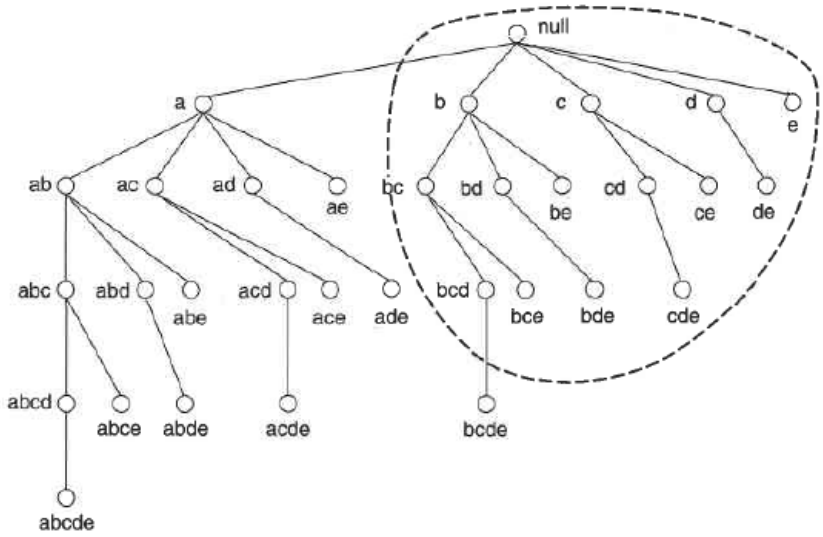


Figure 6.22. Generating candidate itemsets using the depth-first approach.

Representation of Transaction Data Set : There are many ways to represent a transaction data set. The choice of representation can affect the I/O costs incurred when computing the support of candidate itemsets.

Figure 6.23 shows two different ways of representing market basket transactions.

Horizontal Data Layout

TID	Items
1	a,b,e
2	b,c,d
3	c,e
4	a,c,d
5	a,b,c,d
6	a,e
7	a,b
8	a,b,c
9	a,c,d
10	b

Vertical Data Layout

a	b	c	d	e
1	1	2	2	1
4	2	3	4	3
5	5	4	5	6
6	7	8	9	
7	8	9		
8	10			
9				

Figure 6.23. Horizontal and vertical data format.

The representation on the left is called a horizontal data layout, which is adopted by many association rule mining algorithms, including Apriori.

Another possibility is to store the list of transaction identifiers (TID-list) associated with each item. Such a representation is known as the vertical data layout.

The support for each candidate itemset is obtained by intersecting the TlD-lists of its subset items.

The length of the TlD-lists shrinks as we progress to larger sized itemsets.

One problem with this approach is that the initial set of TlD-lists may be too large to fit into main memory, thus requiring more sophisticated techniques to compress the TlD-lists.

6.6 FP-Growth Algorithm

Algorithm encodes the data set using a compact data structure called an FP-tree and extracts frequent itemsets directly from this structure.

6.6.1 FP-Tree Representation

An FP-tree is a compressed representation of the input data. It is constructed by reading the data set one transaction at a time and mapping each transaction onto a path in the FP-tree.

As different transactions can have several items in common, their paths may overlap. The more the paths overlap with one another, the more compression we can achieve using the FP-tree structure.

If the size of the F.P-tree is small enough to fit into main memory, this will allow us to extract frequent itemsets directly from the structure in memory instead of making repeated passes over the data stored on disk.

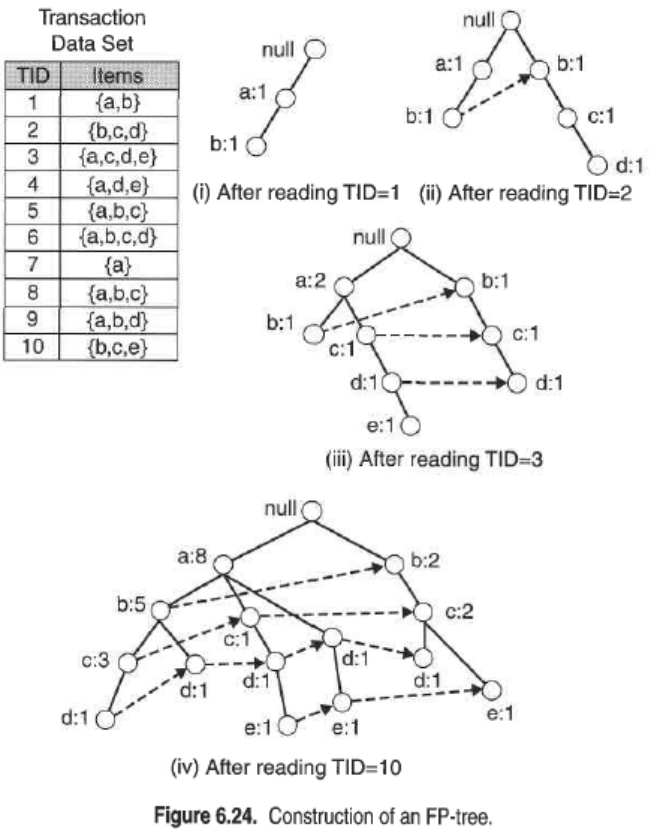


Figure 6.24. Construction of an FP-tree.

Figure 6.24 shows a data set that contains ten transactions and five items.

The structures of the FP-tree after reading the first three transactions are also depicted in the diagram.

Each node in the tree contains the label of an item along with a counter that shows the number of transactions mapped onto the given path.

Initially, the FP-tree contains only the root node represented by the null symbol.

The FP-tree is subsequently extended in the following way:

1. The data set is scanned once to determine the support count of each item.

Infrequent items are discarded, while the frequent items are sorted in decreasing support counts. For the data set shown in Figure 6.24, a is the most frequent item, followed by b, c, d, and e.

2. The algorithm makes a second pass over the data to construct the FPtree.

After reading the first transaction, {o,b), the nodes labelled as a and b are created. A path is then formed from nulI --+a '--+b to encode the transaction. Every node along the path has a frequency count of 1.

3. After reading the second transaction, {b,cd}, a new set of nodes is created for items b, c, arrd d.

A path is then formed to represent the transaction by connecting the nodes null -→b -→c -> d.

Every node along this path also has a frequency count equal to one. Although the first two transactions have an item in common, which is b, their paths are disjoint because the transactions do not share a common prefix.

4. The third transaction, {a,cd,e}, shares a common prefix item (which is α) with the first transaction.

As a result, the path for the third transaction, null →a -→c -→d → e, overlaps with the path for the first transaction, null -→a →b.

Because of their overlapping path, the frequency count for node α is incremented to two, while the frequency counts for the newly created nodes, c, d, and e) are equal to one.

5. This process continues until every transaction has been mapped onto one of the paths given in the FP-tree. The resulting FP-tree after reading all the transactions is shown at the bottom of Figure 6.24.

The size of an FP-tree is typically smaller than the size of the uncompressed data because many transactions in market basket data often share a few items in common.

In the best-case scenario, where all the transactions have the same set of iterns, the FP-tree contains only a single branch of nodes.

The worst-case scenario happens when every transaction has a unique set of items.

As none of the transactions have any items in common, the size of the FP-tree is effectively the same as the size of the original data.

The physical storage requirement for the FP-tree is higher because it requires additional space to store pointers between nodes and counters for each item.

The size of an FP-tree also depends on how the items are ordered. If the ordering scheme in the preceding example is reversed, i.e., from lowest to highest support item, the resulting FP-tree is shown in Figure 6.25.

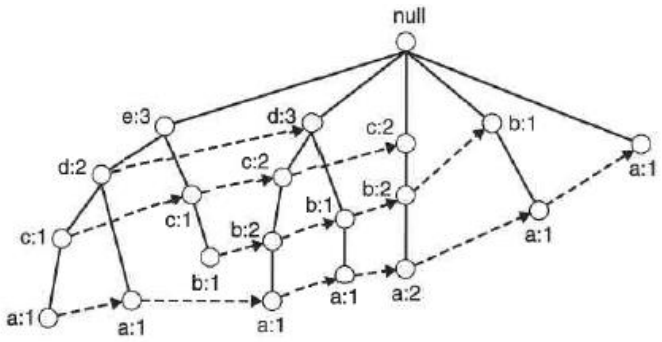


Figure 6.25. An FP-tree representation for the data set shown in Figure 6.24 with a different item ordering scheme.

The tree appears to be denser because the branching factor at the root node has increased from 2 to 5 and the number of nodes containing the high support items such as a and b has increased from 3 to 12.

Ordering by decreasing support counts does not always lead to the smallest tree.

For example, suppose we augment the data set given in Figure 6.24 with 100 transactions that contain {e}, 80 transactions that contain {d}, 60 transactions that contain {c}, and 40 transactions that contain {b}.

Item e is now most frequent, followed by d, c, b, and a. With the augmented transactions, ordering by decreasing support counts will result in an FP-tree similar to Figure 6.25, while a scheme based on increasing support counts produces a smaller FP-tree similar to Figure 6.24(iv).

An FP-tree also contains a list of pointers connecting between nodes that have the same items.

These pointers, represented as dashed lines in Figures 6.24 and 6.25, help to facilitate the rapid access of individual items in the tree.

6.6.2 Frequent Itemset Generation in FP-Growth Algorithm

FP-growth is an algorithm that generates frequent itemsets from an FP-tree by exploring the tree in a bottom-up fashion.

Given the example tree shown in Figure 6.24, the algorithm looks for frequent itemsets ending in e first, followed by d, c, b, and finally, a.

This bottom-up strategy for finding frequent itemsets ending with a particular item is equivalent to the suffix-based approach described in Section 6.5.

Since every transaction is mapped onto a path in the FP-tree, we can derive the frequent itemsets ending with a particular item, say, e, by examining only the paths containing node e. These paths can be accessed rapidly using the pointers associated with node e. The extracted paths are shown in Figure 6.26(a).

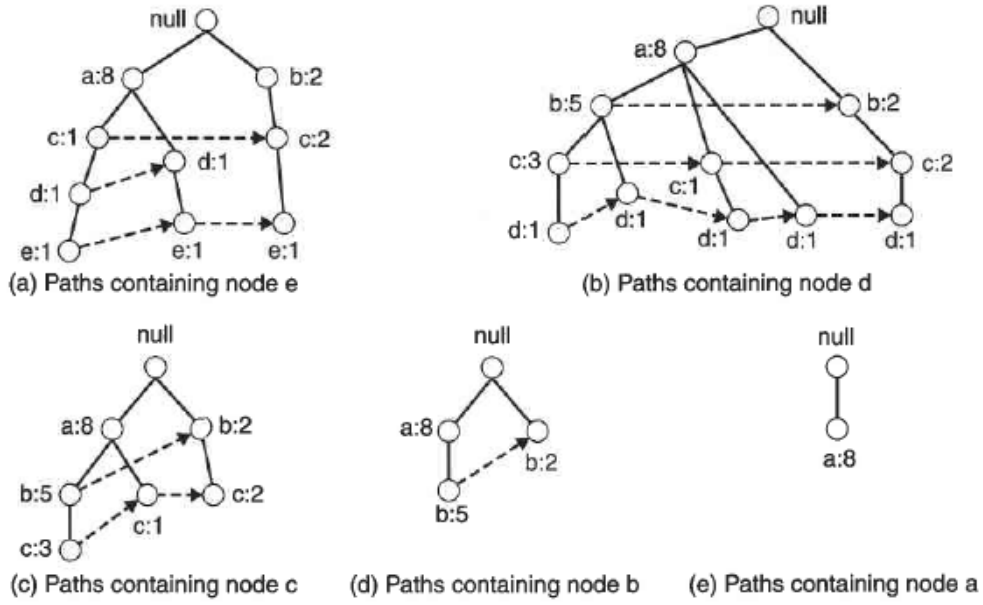


Figure 6.26. Decomposing the frequent itemset generation problem into multiple subproblems, where each subproblem involves finding frequent itemsets ending in e, d, c, b, and a.

After finding the frequent itemsets ending in e, the algorithm proceeds to look for frequent itemsets ending in d by processing the paths associated with node d.

The corresponding paths are shown in Figure 6.26(b). This process continues until all the paths associated with nodes c, b, and finally ,a, are processed.

The paths for these items are shown in Figures 6.26(c), (d), and (e), while their corresponding frequent itemsets are summarized in Table 6.6.

Table 6.6. The list of frequent itemsets ordered by their corresponding suffixes.

Suffix	Frequent Itemsets
e	{e}, {d,e}, {a,d,e}, {c,e},{a,e}
d	{d}, {c,d}, {b,c,d}, {a,c,d}, {b,d}, {a,b,d}, {a,d}
c	{c}, {b,c}, {a,b,c}, {a,c}
b	{b}, {a,b}
a	{a}

FP-growth finds all the frequent itemsets ending with a particular suffix by employing a divide-and-conquer strategy to split the problem into smaller subproblems.

Example 1: Suppose we are interested in finding all frequent itemsets ending in e.

To do this, we must first check whether the itemset {e} itself is frequent.

If it is frequent, we consider the subproblem of finding frequent itemsets ending in de, followed by ce, be, and ae.

In turn, each of these subproblems are further decomposed into smaller subproblems.

By merging the solutions obtained from the subproblems, all the frequent itemsets ending in e can be found.

This divide-and-conquer approach is the key strategy employed by the FP-growth algorithm.

Example 2. Consider the task of finding frequent itemsets ending with e.

1. The first step is to gather all the paths containing node e. These initial paths are called prefix paths and are shown in Figure 6.27(a).

2. From the prefix paths shown in Figure 6.27(a), the support count for e is obtained by adding the support counts associated with node e. Assuming that the minimum support count is 2, {e} is declared a frequent itemset because its support count is 3.

3.Because {e} is frequent, the algorithm has to solve the sub problems of finding frequent itemsets ending in de, ce, be, and ae.

Before solving these subproblems, it must first convert the prefix paths into a conditional FP-tree, which is structurally similar to an FP-tree, except it is used to find frequent itemsets ending with a particular suffix.

A conditional FP-tree is obtained in the following way:

(a) First, the support counts along the prefix paths must be updated because some of the counts include transactions that do not contain item e.

For example, the rightmost path shown in Figure 6.27(a),

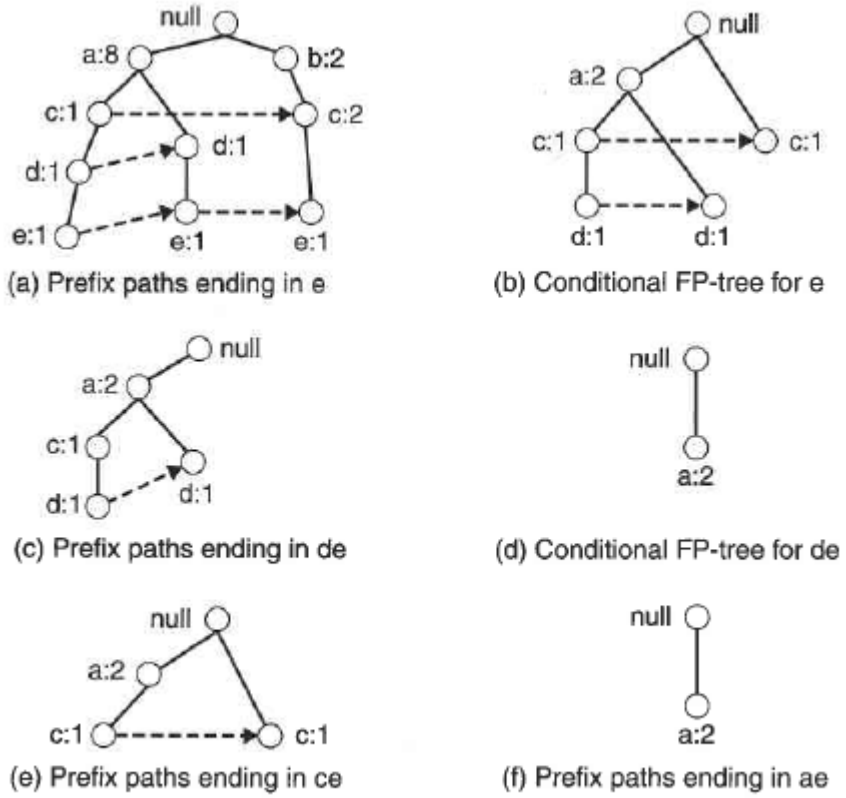


Figure 6.27. Example of applying the FP-growth algorithm to find frequent itemsets ending in e.

null-→b: 2 -→c:2 →e: 1, includes a transaction {b,c} that does not contain item e.

The counts along the prefix path must therefore be adjusted to 1 to reflect the actual number of transactions containing {b, c, e}.

(b) The prefix paths are truncated by removing the nodes for e. These nodes can be removed because the support counts along the prefix paths have been updated to reflect only transactions that contain e and the sub problems of finding frequent itemsets ending in de, ce, be, anrd ae no longer need information about node e.

(c) After updating the support counts along the prefix paths, some of the items may no longer be frequent.

For example, the node b appears only once and has a support count equal to 1, which means that there is only one transaction that contains both b and e. Item b can be safely ignored from subsequent analysis because all itemsets ending in be must be infrequent.

The conditional FP-tree for e is shown in Figure 6.27(b). The tree looks different than the original prefix paths because the frequency counts have been updated and the nodes b and e have been eliminated.

4.FP-growth uses the conditional FP-tree for e to solve the sub problems of finding frequent itemsets ending in de, ce, and ae.

To find the frequent itemsets ending in de, the prefix paths for d are gathered from the conditional FP-tree for e (Figure 6.27(c)).

By adding the frequency counts associated with node d, we obtain the support count for {d,e}.

Since the support count is equal to 2, {d,e} is declared a frequent itemset.

Next, the algorithm constructs the conditional FP-tree for de using the approach described in step 3.

After updating the support counts and removing the infrequent item c, the conditional FP-tree for de is shown in Figure 6.27(d).

Since the conditional FP-tree contains only one item, α, whose support is equal to minsup, the algorithm extracts the frequent itemset {a,d,e} and moves on to the next sub problem, which is to generate frequent itemsets ending in ce.

After processing the prefix paths for c, only {c, e} is found to be frequent. The algorithm proceeds to solve the next subprogram and found {a,e} to be the only frequent itemset remaining.

This example illustrates the divide-and-conquer approach used in the FPgrowth algorithm. At each recursive step, a conditional FP-tree is constructed by updating the frequency counts along the prefix paths and removing all infrequent items.

Because the subproblems are disjoint, FP-growth will not generate any duplicate itemsets. In addition, the counts associated with the nodes allow the algorithm to perform support counting while generating the common suffix itemsets.

6.7 Evaluation of Association Patterns

Association analysis algorithms have the potential to generate a large number of patterns. For example, although the data set shown in Table 6.1 contains only six items, it can produce up to hundreds of association rules at certain support and confidence thresholds.

As the size and dimensionality of real commercial databases can be very large, we could easily end up with thousands or even millions of patterns, many of which might not be interesting.

It is therefore important to establish a set of well-accepted criteria for evaluating the quality of association patterns.

1.The first set of criteria can be established through **statistical arguments.**

Patterns that involve a set of mutually independent items or cover very few transactions are considered uninteresting because they may capture spurious relationships in the data. Such

patterns can be eliminated by applying an objective interestingness measure that uses statistics derived from data to determine whether a pattern is interesting.

Examples of objective interestingness measures include support, confidence, and correlation.

2.The second set of criteria can be established through **subjective arguments.**

A pattern is considered subjectively uninteresting unless it reveals unexpected information about the data or provides useful knowledge that can lead to profitable actions.

For example, the rrle {Butter} - >{Bread} may not be interesting, despite having high support and confidence values, because the relationship represented by the rule may seem rather obvious.

On the other hand, the rule {Diapers} →{ Beer} is interesting because the relationship is quite unexpected and may suggest a new cross-selling opportunity for retailers.

Incorporating subjective knowledge into pattern evaluation is a difficult task because it requires a considerable amount of prior information from the domain experts.

The following are some of the approaches for incorporating subjective knowledge into the pattern discovery task.

Visualization : This approach requires a user-friendly environment to keep the human user in the loop.

It also allows the domain experts to interact with the data mining system by interpreting and verifying the discovered patterns.

Template-based approach This approach allows the users to constrain the type of patterns extracted by the mining algorithm.

Instead of reporting all the extracted rules, only rules that satisfy a user-specified template are returned to the users.

Subjective interestingness measure A subjective measure can be defined based on domain information such as concept hierarchy

The measure can then be used to filter patterns that are obvious and non-actionable.

6.7.1 Objective Measures of Interestingness

An objective measure is a data-driven approach for evaluating the quality of association patterns.

It is domain-independent and requires minimal input from the users, other than to specify a threshold for filtering low-quality patterns.

An objective measure is usually computed based on the frequency counts tabulated in a contingency table.

Table 6.7. A 2-way contingency table for variables A and B.

	B	$\overline{B}$	
A	f_{11}	f_{10}	f_{1+}
$\overline{A}$	f_{01}	f_{00}	f_{0+}
	f_{+1}	f_{+0}	N

Table 6.7 shows an example of a contingency table for a pair of binary variables, A and B. We use the notation $\overline{A}$ ($\overline{B}$) to indicate that ,A (B) is absent from a transaction.

Each entry fij in this 2 x 2 table denotes a frequency count.

For example, f11 is the number of times A and B appear together in the same transaction, while f01 is the number of transactions that contain B but not A.

The row sum f1+ represents the support count for A, while the column sum f+1 represents the support count for B.

Finally, even though focus is mainly on asymmetric binary variables, note that contingency tables are also applicable to other attribute types such as symmetric binary, nominal, and ordinal variables.

Limitations of the support-confidence Framework: Existing association rule mining formulation relies on the support and confidence measures to eliminate uninteresting patterns. The drawback of support is many potentially interesting patterns involving low support items might be eliminated by the support threshold.

The drawback of confidence is more subtle and is best demonstrated with the following example.

Example 6.3. Suppose we are interested in analyzing the relationship between people who drink tea and coffee. We may gather information about the beverage preferences among a group of people and summarize their responses into a table such as the one shown in Table 6.8.

Table 6.8. Beverage preferences among a group of 1000 people.

	$Coffee$	$\overline{Coffee}$	
Tea	150	50	200
$\overline{Tea}$	650	150	800
	800	200	1000

The information given in this table can be used to evaluate the association rule {tea,} →{Coffee}.

At first glance, it may appear that people who drink tea also tend to drink coffee because the rule's support (15%) and confidence (75%) values are reasonably high.

This argument would have been acceptable except that the fraction of people who drink coffee, regardless of whether they drink tea, is 80%, while the fraction of tea drinkers who drink coffee is only 75%.

Thus knowing that a person is a tea drinker actually decreases her probability of being a coffee drinker from 80% to 75%.

The rule {Tea) ->{Cof f ee} is therefore misleading despite its high confidence value.

Interest Factor The tea-coffee example shows that high-confidence rules can sometimes be misleading because the confidence measure ignores the support of the itemset appearing in the rule consequent.

One way to address this problem is by applying a metric known as lift:

$$Lift = \frac{c(A \longrightarrow B)}{s(B)},$$

which computes the ratio between the rule's confidence and the support of the itemset in the rule consequent.

For binary variables, lift is equivalent to another objective measure called interest factor, which is defined as follows:

$$I(A, B) = \frac{s(A, B)}{s(A) \times s(B)} = \frac{N f_{11}}{f_{1+} f_{+1}}.$$

Interest factor compares the frequency of a pattern against a baseline frequency computed under the statistical independence assumption. The baseline frequency for a pair of mutually independent variables is

$$\frac{f_{11}}{N} = \frac{f_{1+}}{N} \times \frac{f_{+1}}{N}, \quad \text{or equivalently,} \quad f_{11} = \frac{f_{1+} f_{+1}}{N}. \qquad (6.6)$$

Table 6.9. Contingency tables for the word pairs $\{p,q\}$ and $\{r,s\}$.

	p	$\bar{p}$	
q	880	50	930
$\bar{q}$	50	20	70
	930	70	1000

	r	$\bar{r}$	
s	20	50	70
$\bar{s}$	50	880	930
	70	930	1000

This equation follows from the standard approach of using simple fractions as estimates for probabilities. The fraction f11/N is an estimate for the joint probability P(A,B), while f1+/N and f+1/ N are the estimates for P(A) and P(B), respectively.

If A and B are statistically independent, then P(A,B): P(A) x P(B), thus leading to the formula shown in Equation 6.6. Using

Equations 6.5 and 6.6, we can interpret the measure as follows:

$$I(A,B)\begin{cases} = 1, & \text{if } A \text{ and } B \text{ are independent;} \\ > 1, & \text{if } A \text{ and } B \text{ are positively correlated;} \\ < 1, & \text{if } A \text{ and } B \text{ are negatively correlated.} \end{cases} \quad (6.7)$$

For the tea-coffee example shown in Table 6.8, $I = \frac{0.15}{0.2 \times 0.8} = 0.9375$, thus suggesting a slight negative correlation between tea drinkers and coffee drinkers.

Limitations of Interest Factor

In the text domain, it is reasonable to assume that the association between a pair of words depends on the number of documents that contain both words.

For example, because of their stronger association, we expect the words data and mining to appear together more frequently than the words compiler and mining in a collection of computer science articles.

Table 6.9 shows the frequency of occurrences between two pairs of words, {p,q} and {r,s}. Using the formula given in Equation 6.5, the interest factor for {p,q} is 1.02 and for {r, s} is 4.08.

These results are troubling for the following reasons:

Although p and q appear together in 88% of the documents, their interest factor is close to 1, which is the value when p and q are statistically independent.

On the other hand, the interest factor for {r, s} is higher than {p, q} even though r and s seldom appear together in the same document.

Confidence is perhaps the better choice in this situation because it considers the association between p and q (94.6%) to be much stronger than that between r and s (28.6%).

Correlation Analysis Correlation analysis is a statistical-based technique for analyzing relationships between a pair of variables.

For continuous variables, correlation is defined using Pearson's correlation. For binary variables, correlation can be measured using the Ø-coefficient. which is defined as

$$\phi = \frac{f_{11}f_{00} - f_{01}f_{10}}{\sqrt{f_{1+}f_{+1}f_{0+}f_{+0}}}. \quad (6.8)$$

The value of correlation ranges from -1 (perfect negative correlation) to +1 (perfect positive correlation).

If the variables are statistically independent, then Ø= 0. For example, the correlation between the tea and coffee drinkers given in Table 6.8 is -0.0625.

Limitations of Correlation Analysis The drawback of using correlation can be seen from the word association example given in Table 6.9.

Although the words p and q appear together more often than r and s, their Ø -coefficients are identical, i.e., Ø (p,q) = Ø (r,s) = 0.232.

This is because the Ø –coefficient gives equal importance to both co-presence and co-absence of items in a transaction.

It is therefore more suitable for analyzing symmetric binary variables.

IS Measure .IS is an alternative measure that has been proposed for handling asymmetric binary variables. The measure is defined as follows:

$$IS(A,B) = \sqrt{I(A,B) \times s(A,B)} = \frac{s(A,B)}{\sqrt{s(A)s(B)}}. \qquad (6.9)$$

IS is large when the interest factor and support of the pattern are large.

For example, the value of IS for the word pairs {p, q} and {r, s} shown in Table 6.9 are 0.946 and 0.286, respectively.

Contrary to the results given by interest factor and the Ø -coefficient, the IS measure suggests that the association between {p, q} is stronger than {r, s}, which agrees with what we expect from word associations in documents.

It is possible to show that IS is mathematically equivalent to the cosine measure for binary variables .

Table 6.10. Example of a contingency table for items p and q.

	q	$\overline{q}$	
p	800	100	900
$\overline{p}$	100	0	100
	900	100	1000

Consider A and B as a pair of bit vectors, A **.** B = s(A,B) the dot product between the vectors**,** and l**A**l = √s(A) the magnitude of vector A.

Therefore:

$$IS(A,B) = \frac{s(A,B)}{\sqrt{s(A) \times s(B)}} = \frac{\mathbf{A} \bullet \mathbf{B}}{|\mathbf{A}| \times |\mathbf{B}|} = cosine(\mathbf{A}, \mathbf{B}). \qquad (6.10)$$

The IS measure can also be expressed as the geometric mean between the confidence of association rules extracted from a pair of binary variables:

$$IS(A,B) = \sqrt{\frac{s(A,B)}{s(A)} \times \frac{s(A,B)}{s(B)}} = \sqrt{c(A \rightarrow B) \times c(B \rightarrow A)}. \qquad (6.11)$$

Because the geometric mean between any two numbers is always closer to the smaller number, the IS value of an itemset {p, q} is low whenever one of its rules, p → Q or q-> p, has low confidence.

Limitations of IS Measure The IS value for a pair of independent itemsets, A and B, is

$$IS_{\text{indep}}(A,B) = \frac{s(A,B)}{\sqrt{s(A) \times s(B)}} = \frac{s(A) \times s(B)}{\sqrt{s(A) \times s(B)}} = \sqrt{s(A) \times s(B)}.$$

since the value depends on s(A) and s(B), IS shares a similar problem as the confidence measure-that the value of the measure can be quite large, even for uncorrelated and negatively correlated patterns.

For example, despite the large IS value between items p and q given in Table 6.10 (0.889), it is still less than the expected value when the items are statistically independent (IS_{indep} : 0.9).

Alternative Objective Interestingness Measures

These measures can be divided into two categories, symmetric and asymmetric measures.

A measure M is symmetric if M(A →B) = M(B →A).

For example, interest factor is a symmetric measure because its value is identical for the rules A → B and B →A.

In contrast, confidence is an asymmetric measure since the confidence for A → B and B->A, may not be the same.

Symmetric measures are generally used for evaluating itemsets, while asymmetric measures are more suitable for analyzing association rules.

Tables 6.11 and 6.12 provide the definitions for some of these measures in terms of the frequency counts of a 2 x 2 contingency table.

Table 6.11. Examples of symmetric objective measures for the itemset $\{A, B\}$.

Measure (Symbol)	Definition
Correlation (ϕ)	$\frac{Nf_{11}-f_{1+}f_{+1}}{\sqrt{f_{1+}f_{+1}f_{0+}f_{+0}}}$
Odds ratio (α)	$(f_{11}f_{00})/(f_{10}f_{01})$
Kappa (κ)	$\frac{Nf_{11}+Nf_{00}-f_{1+}f_{+1}-f_{0+}f_{+0}}{N^2-f_{1+}f_{+1}-f_{0+}f_{+0}}$
Interest (I)	$(Nf_{11})/(f_{1+}f_{+1})$
Cosine (IS)	$(f_{11})/(\sqrt{f_{1+}f_{+1}})$
Piatetsky-Shapiro (PS)	$\frac{f_{11}}{N} - \frac{f_{1+}f_{+1}}{N^2}$
Collective strength (S)	$\frac{f_{11}+f_{00}}{f_{1+}f_{+1}+f_{0+}f_{+0}} \times \frac{N-f_{1+}f_{+1}-f_{0+}f_{+0}}{N-f_{11}-f_{00}}$
Jaccard (ζ)	$f_{11}/(f_{1+}+f_{+1}-f_{11})$
All-confidence (h)	$\min\left[\frac{f_{11}}{f_{1+}}, \frac{f_{11}}{f_{+1}}\right]$

Table 6.12. Examples of asymmetric objective measures for the rule $A \longrightarrow B$.

Measure (Symbol)	Definition
Goodman-Kruskal (λ)	$\left(\sum_j \max_k f_{jk} - max_k f_{+k}\right)/\left(N - \max_k f_{+k}\right)$
Mutual Information (M)	$\left(\sum_i \sum_j \frac{f_{ij}}{N} \log \frac{Nf_{ij}}{f_{i+}f_{+j}}\right)/\left(-\sum_i \frac{f_{i+}}{N} \log \frac{f_{i+}}{N}\right)$
J-Measure (J)	$\frac{f_{11}}{N} \log \frac{Nf_{11}}{f_{1+}f_{+1}} + \frac{f_{10}}{N} \log \frac{Nf_{10}}{f_{1+}f_{+0}}$
Gini index (G)	$\frac{f_{1+}}{N} \times \left(\frac{f_{11}}{f_{1+}}\right)^2 + \left(\frac{f_{10}}{f_{1+}}\right)^2] - \left(\frac{f_{+1}}{N}\right)^2$ $+ \frac{f_{0+}}{N} \times [\left(\frac{f_{01}}{f_{0+}}\right)^2 + \left(\frac{f_{00}}{f_{0+}}\right)^2] - \left(\frac{f_{+0}}{N}\right)^2$
Laplace (L)	$(f_{11}+1)/(f_{1+}+2)$
Conviction (V)	$(f_{1+}f_{+0})/(Nf_{10})$
Certainty factor (F)	$\left(\frac{f_{11}}{f_{1+}} - \frac{f_{+1}}{N}\right)/\left(1 - \frac{f_{+1}}{N}\right)$
Added Value (AV)	$\frac{f_{11}}{f_{1+}} - \frac{f_{+1}}{N}$

Consistency among Objective Measures

If the measures are consistent, then we can choose any one of them as our evaluation metric. Otherwise, it is important to understand what their differences are in order to determine which measure is more suitable for analyzing certain types of patterns.

Suppose the symmetric and asymmetric measures are applied to rank the ten contingency tables shown in Table 6.13.

Table 6.13. Example of contingency tables.

Example	f_{11}	f_{10}	f_{01}	f_{00}
E_1	8123	83	424	1370
E_2	8330	2	622	1046
E_3	3954	3080	5	2961
E_4	2886	1363	1320	4431
E_5	1500	2000	500	6000
E_6	4000	2000	1000	3000
E_7	9481	298	127	94
E_8	4000	2000	2000	2000
E_9	7450	2483	4	63
E_{10}	61	2483	4	7452

These contingency tables are chosen to illustrate the differences among the existing measures.

The ordering produced by these measures are shown in Tables 6.14 and 6.15, respectively (with 1 as the most interesting and 10 as the least interesting table).

Table 6.14. Rankings of contingency tables using the symmetric measures given in Table 6.11.

	ϕ	α	κ	I	IS	PS	S	ζ	h
E_1	1	3	1	6	2	2	1	2	2
E_2	2	1	2	7	3	5	2	3	3
E_3	3	2	4	4	5	1	3	6	8
E_4	4	8	3	3	7	3	4	7	5
E_5	5	7	6	2	9	6	6	9	9
E_6	6	9	5	5	6	4	5	5	7
E_7	7	6	7	9	1	8	7	1	1
E_8	8	10	8	8	8	7	8	8	7
E_9	9	4	9	10	4	9	9	4	4
E_{10}	10	5	10	1	10	10	10	10	10

Table 6.15. Rankings of contingency tables using the asymmetric measures given in Table 6.12.

	λ	M	J	G	L	V	F	AV
E_1	1	1	1	1	4	2	2	5
E_2	2	2	2	3	5	1	1	6
E_3	5	3	5	2	2	6	6	4
E_4	4	6	3	4	9	3	3	1
E_5	9	7	4	6	8	5	5	2
E_6	3	8	6	5	7	4	4	3
E_7	7	5	9	8	3	7	7	9
E_8	8	9	7	7	10	8	8	7
E_9	6	4	10	9	1	9	9	10
E_{10}	10	10	8	10	6	10	10	8

Some of the measures appear to be consistent with each other, there are certain measures that produce quite different ordering results.

For example, the rankings given by the Ø-coefficient agree with those provided by k and collective strength, but are somewhat different than the rankings produced by interest factor and odds ratio.

Furthermore, a contingency table such as $-E_{10}$ is is ranked lowest according to the Ø-coefficient, but highest according to interest factor.

Properties of Objective Measures

Inversion Property Consider the bit vectors shown in Figure 6.28.

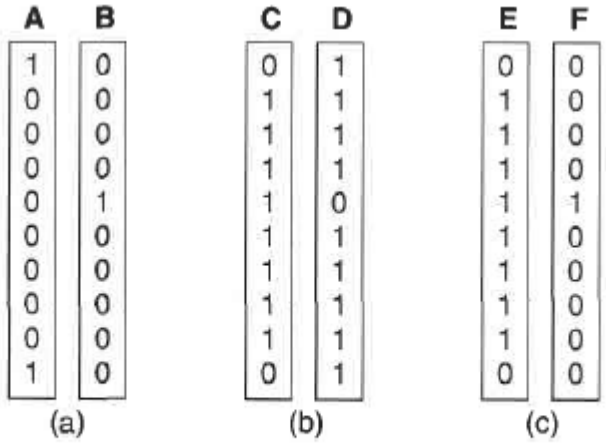


Figure 6.28. Effect of the inversion operation. The vectors C and E are inversions of vector A, while the vector D is an inversion of vectors B and F.

The 0/1 bit in each column vector indicates whether a transaction (row) contains a particular item (column).

For example, the vector A indicates that item α belongs to the first and last transactions, whereas the vector B indicates that item b is contained only in the fifth transaction.

The vectors C and E are in fact related to the vector A-their bits have been inverted from 0's (absence) to l's (presence), and vice versa.

Similarly, D is related to vectors B and F by inverting their bits.

The process of flipping a bit vector is called inversion.

If a measure is invariant under the inversion operation, then its value for the vector pair (C, D) should be identical to its value for (A, B).

The inversion property of a measure can be tested as follows.

Definition 6.6 (Inversion Property). An objective measure M is invariant under the inversion operation if its value remains the same when exchanging the frequency counts f11 with f00and f10 with f01.

Among the measures that remain invariant under this operation include the Ø-coefficient odds ratio, k, and collective strength.

These measures may not be suitable for analyzing asymmetric binary data.

For example, the Ø -coefficient between C and D is identical to the @-coefficient between A and B, even though items c and d appear together more frequently than a and b.

Furthermore, the d-coefficient between C and D is less than that between E and F even though items e and f appear together only once.

For asymmetric binary data, measures that do not remain invariant under the inversion operation are preferred. Some of the non-invariant measures include interest factor, IS, PS, and the Jaccard coefficient .

Null Addition Property Suppose we are interested in analyzing the relationship between a pair of words, such as data and rnining, in a set of documents.

This process of adding unrelated data (in this case, documents) to a given data set is known as the null addition operation.

Deffnition 6.7 (Null Addition Property). An objective measure M is invariant under the null addition operation if it is not affected by increasing f00, while all other frequencies in the contingency table stay the same.

For applications such as document analysis or market basket analysis, the measure is expected to remain invariant under the null addition operation.

Otherwise, the relationship between words may disappear simply by adding enough documents that do not contain both words.

Examples of measures that satisfy this property include cosine (IS) and Jaccard (ξ) measures, while those that violate this property include interest factor, PS, odds ratio, and the Ø-coefficient.

Scaling Property Table 6.16 shows the contingency tables for gender and the grades achieved by students enrolled in a particular course in 1993 and 2004.

Table 6.16. The grade-gender example.

	Male	Female	
High	30	20	50
Low	40	10	50
	70	30	100

(a) Sample data from 1993.

	Male	Female	
High	60	60	120
Low	80	30	110
	140	90	230

(b) Sample data from 2004.

The data in these tables showed that the number of male students has doubled since 1993, while the number of female students has increased by a factor of 3.

The male students in 2004 are not performing any better than those in 1993 because the ratio of male students who achieve a high grade to those who achieve a low grade is still the same, i.e., 3:4.

Similarly, the female students in 2004 are performing no better than those in 1993. The association between grade and gender is expected to remain unchanged despite changes in the sampling distribution.

Table 6.17. Properties of symmetric measures.

Symbol	Measure	Inversion	Null Addition	Scaling
ϕ	ϕ-coefficient	Yes	No	No
α	odds ratio	Yes	No	Yes
κ	Cohen's	Yes	No	No
I	Interest	No	No	No
IS	Cosine	No	Yes	No
PS	Piatetsky-Shapiro's	Yes	No	No
S	Collective strength	Yes	No	No
ζ	Jaccard	No	Yes	No
h	All-confidence	No	No	No
s	Support	No	No	No

Definition 6.8 (Scaling Invariance Property). An objective measure M is invariant under the row/column scaling operation if M(T) = M(T'), where T is a contingency table with frequency counts $[f_{11};\ f_{10};\ f_{01};\ f_{00}]$, T' is a contingency table with scaled frequency counts $[k_1k_3f_{11};\ k_2k_3f_{10};\ k_1k_4f_{01};$, and k1, kz, ks, k4 are positive constants.

From Table 6.17,only the odds ratio (a) is invariant under the row and column scaling operations.

All other measures such as the f coefficient, n, IS, interest factor, and collective strength (S) change their values when the rows and columns of the contingency table are rescaled.

6.7.2 Measures beyond Pairs of Binary Variables

An example of a three-dimensional contingency table for a, b, and c is shown in Table 6.18.

Table 6.18. Example of a three-dimensional contingency table.

c	b	$\bar{b}$	
a	f_{111}	f_{101}	f_{1+1}
$\bar{a}$	f_{011}	f_{001}	f_{0+1}
	f_{+11}	f_{+01}	f_{++1}

$\bar{c}$	b	$\bar{b}$	
a	f_{110}	f_{100}	f_{1+0}
$\bar{a}$	f_{010}	f_{000}	f_{0+0}
	f_{+10}	f_{+00}	f_{++0}

Each entry fijk in this table represents the number of transactions that contain a particular combination of items a, b, and c. For example, f_{101} is the number of transactions that contain a and c, but not b.

On the other hand, a marginal frequency such as ,f_{1+1} is the number of transactions that contain a and c, irrespective of whether b is present in the transaction.

Given a k-itemset {i1,i2,i3,i4,....,in}, the condition for statistical independence can be stated as follows

$$f_{i_1 i_2 \ldots i_k} = \frac{f_{i_1+\ldots+} \times f_{+i_2\ldots+} \times \ldots \times f_{++\ldots i_k}}{N^{k-1}}. \qquad (6.12)$$

Above definition can be extended to objective measures such as interest factor and P,S, which are based on deviations from statistical independence' to more than two variables:

$$I = \frac{N^{k-1} \times f_{i_1 i_2 \ldots i_k}}{f_{i_1+\ldots+} \times f_{+i_2\ldots+} \times \ldots \times f_{++\ldots i_k}}$$

$$PS = \frac{f_{i_1 i_2 \ldots i_k}}{N} - \frac{f_{i_1+\ldots+} \times f_{+i_2\ldots+} \times \ldots \times f_{++\ldots i_k}}{N^k}$$

Another approach is to define the objective measure as the maximum, minimum, or average value for the associations between pairs of items in a pattern.

For example, given a k-itemset X= {i1,i2,i3,i4,....,in},we may define the

Ø-coefficient for X as the average Ø -coefficient between every pair of items

(ip,iq) in X.

Since the measure considers only pairwise associations, it may not capture all the underlying relationships within a pattern.

Analysis of multidimensional contingency tables is more complicated because of the presence of partial associations in the data.

For example, some associations may appear or disappear when conditioned upon the value of certain variables. This problem is known as Simpson's paradox

6.7.3 Simpson's Paradox

It is important to exercise caution when interpreting the association between variables because the observed relationship may be influenced by the presence of other confounding factors, i.e., hidden variables that are not included in the analysis.

In some cases, the hidden variables may cause the observed relationship between a pair of variables to disappear or reverse its direction, a phenomenon that is known as **Simpson's paradox.**

Example: Consider the relationship between the sale of high-definition television (HDTV) and exercise machine, as shown in Table 6.19.

Table 6.19. A two-way contingency table between the sale of high-definition television and exercise machine.

Buy HDTV	Buy Exercise Machine		
	Yes	No	
Yes	99	81	180
No	54	66	120
	153	147	300

Table 6.20. Example of a three-way contingency table.

Customer Group	Buy HDTV	Buy Exercise Machine		Total
		Yes	No	
College Students	Yes	1	9	10
	No	4	30	34
Working Adult	Yes	98	72	170
	No	50	36	86

The rule {HDTV=Yes}→{Exercise machine = Yes} has a confidence of 99/180 = 55% and the rule {HDTV = No} →{ Exercise machine = Yes} has a confidence of 54/120 = 45%.

Together, these rules suggest that customers who buy high-definition televisions are more likely to buy exercise machines than those who do not buy high-definition televisions.

Sales of these items depend on whether the customer is a college student or a working adult.

Table 6.20 summarizes the relationship between the sale of HDTVs and exercise machines among college students and working adults.

The support counts given in the table for college students and working adults sum up to the frequencies shown in Table 6.19.

For college students:

$$c(\{\text{HDTV=Yes}\} \longrightarrow \{\text{Exercise machine=Yes}\}) = 1/10 = 10\%,$$
$$c(\{\text{HDTV=No}\} \longrightarrow \{\text{Exercise machine=Yes}\}) = 4/34 = 11.8\%,$$

For working adults:

$$c(\{\text{HDTV=Yes}\} \longrightarrow \{\text{Exercise machine=Yes}\}) = 98/170 = 57.7\%,$$
$$c(\{\text{HDTV=No}\} \longrightarrow \{\text{Exercise machine=Yes}\}) = 50/86 = 58.1\%.$$

The rules suggest that, for each group, customers who do not buy high definition televisions are more likely to buy exercise machines, which contradict the previous conclusion when data from the two customer groups are pooled together.

Even if alternative measures such as correlation, odds ratio or interest are applied, we still find that the sale of HDTV and exercise machine is positively correlated in the combined data but is negatively correlated in the stratified data.

The reversal in the direction of association is known as Simpson's paradox.

The paradox can be explained in the following way:

Most customers who buy HDTVs are working adults.

Working adults are also the largest group of customers who buy exercise machines.

Nearly 85% of the customers are working adults, the observed relationship between HDTV and exercise machine turns out to be stronger in the combined data than what it would have been if the data is stratified.

This can also be illustrated mathematically as follows:

Suppose

$$a/b < c/d \text{ and } p/q < r/s,$$

where a/b and p/q may represent the confidence of the rule A → B in two different strata, while c/d and r/ s may represent the confidence of the rule $\overline{A} \longrightarrow B$ in the two strata.

When the data is pooled together' the confidence values of the rules in the combined data are (a+p) / (b+q) and (c+r) / (d+s) respectively.

Simpson's paradox occurs when

$$\frac{a+p}{b+q} > \frac{c+r}{d+s},$$

thus leading to the wrong conclusion about the relationship between the variables.

Proper stratification is needed to avoid generating spurious patterns resulting from Simpson's paradox.

For example, market basket data from a major supermarket chain should be stratified according to store locations, while medical records from various patients should be stratified according to confounding factors such as age and gender.

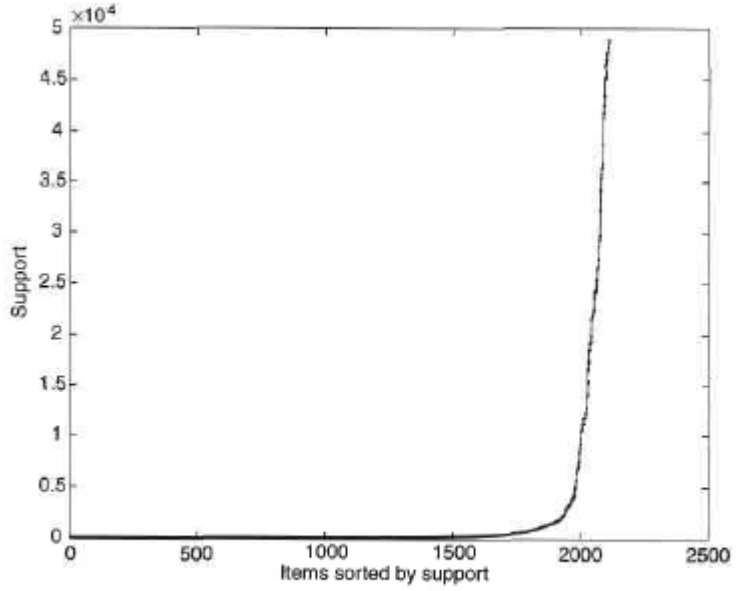


Figure 6.29. Support distribution of items in the census data set.