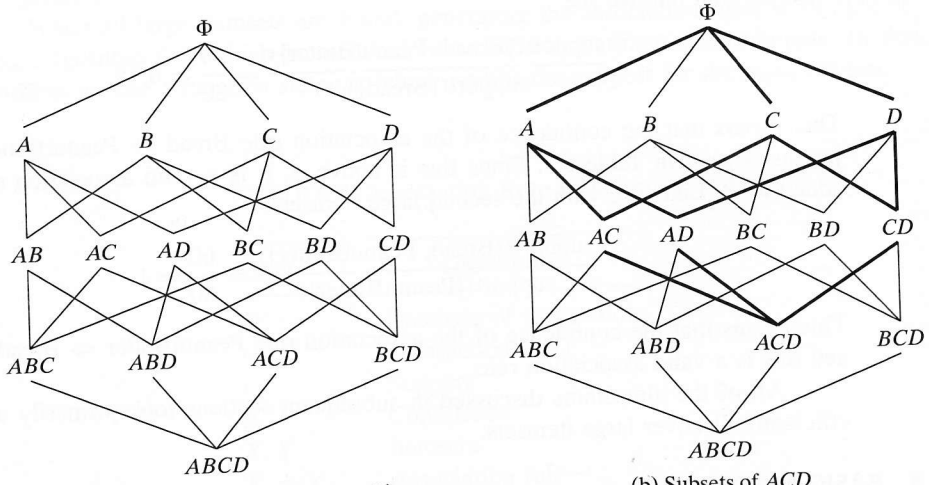
The large itemsets are also said to be *downward closed* because if an itemset satisfies the minimum support requirements, so do all of its subsets. Looking at the contrapositive of this, if we know that an itemset is small, we need not generate any supersets of it as candidates because they also must be small. We use the lattice shown in Figure 6.1(a) to illustrate the concept of this important property. In this case there are four items $\{A, B, C, D\}$. The lines in the lattice represent the subset relationship, so the large itemset property says that any set in a path above an itemset must be large if the original itemset is large. In Figure 6.1 (b) the nonempty subsets of ACD[1] are seen as $\{AC, AD, CD, A, C, D\}$. If ACD is large, so is each of these subsets. If any one of these subsets is small, then so is ACD.

The basic idea of the Apriori algorithm is to generate candidate itemsets of a particular size and then scan the database to count these to see if they are large. During scan i, candidates of size i, C_i are counted. Only those candidates that are large are used to generate candidates for the next pass. That is L_i are used to generate C_{i+1}. An itemset is considered as a candidate only if all its subsets also are large. To generate candidates of size $i+1$, joins are made of large itemsets found in the previous pass. Table 6.5 shows the process using the data found in Table 6.1 with $s = 30\%$ and $\alpha = 50\%$. There are no candidates of size three because there is only one large itemset of size two.

An algorithm called Apriori-Gen is used to generate the candidate itemsets for each pass after the first. All singleton itemsets are used as candidates in the first pass. Here the set of large itemsets of the previous pass, L_{i-1}, is joined with itself to determine the candidates. Individual itemsets must have all but one item in common in order to be combined. Example 6.3 further illustrates the concept. After the first scan, every large itemset is combined with every other large itemset.



(a) Lattice of itemsets for $\{A, B, C, D\}$

(b) Subsets of ACD

FIGURE 6.1: Downward closure.

[1]Following the usual convention with association rule discussions, we simply list the items in the set rather than using the traditional set notation. So here we use ACD to mean $\{A, C, D\}$.

TABLE 6.5: Using Apriori with Transactions in Table 6.1

Pass	Candidates	Large Itemsets
1	{Beer}, {Bread}, {Jelly}, {Milk}, {PeanutButter}	{Beer}, {Bread}, {Milk}, {PeanutButter}
2	{Beer, Bread}, {Beer, Milk}, {Beer, PeanutButter}, {Bread, Milk}, {Bread, PeanutButter}, {Milk, PeanutButter}	{Bread, PeanutButter}

TABLE 6.6: Sample Clothing Transactions

Transaction	Items	Transaction	Items
t_1	Blouse	t_{11}	TShirt
t_2	Shoes, Skirt, TShirt	t_{12}	Blouse, Jeans, Shoes, Skirt, TShirt
t_3	Jeans, TShirt	t_{13}	Jeans, Shoes, Shorts, TShirt
t_4	Jeans, Shoes, TShirt	t_{14}	Shoes, Skirt, TShirt
t_5	Jeans, Shorts	t_{15}	Jeans, TShirt
t_6	Shoes, TShirt	t_{16}	Skirt, TShirt
t_7	Jeans, Skirt	t_{17}	Blouse, Jeans, Skirt
t_8	Jeans, Shoes, Shorts, TShirt	t_{18}	Jeans, Shoes, Shorts, TShirt
t_9	Jeans	t_{19}	Jeans
t_{10}	Jeans, Shoes, TShirt	t_{20}	Jeans, Shoes, Shorts, TShirt

EXAMPLE 6.3

A woman's clothing store has 10 cash register transactions during one day, as shown in Table 6.6. When Apriori is applied to the data, during scan one, we have six candidate itemsets, as seen in Table 6.7. Of these, 5 candidates are large. When Apriori-Gen is applied to these 5 candidates, we combine every one with all the other 5. Thus, we get a total of $4 + 3 + 2 + 1 = 10$ candidates during scan two. Of these, 7 candidates are large. When we apply Apriori-Gen at this level, we join any set with another set that has one item in common. Thus, {Jeans, Shoes} is joined with {Jeans, Shorts} but not with {Shorts, TShirt}. {Jeans, Shoes} will be joined with any other itemset containing either Jeans or Shoes. When it is joined, the new item is added to it. There are four large itemsets after scan four. When we go to join these we must match on two of the three attributes. For example {Jeans, Shoes, Shorts} After scan four, there is only one large itemset. So we obtain no new itemsets of size five to count in the next pass. joins with {Jeans, Shoes, TShirt} to yield new candidate {Jeans, Shoes, Shorts, TShirt}.

The Apriori-Gen algorithm is shown in Algorithm 6.2. Apriori-Gen is guaranteed to generate a superset of the large itemsets of size i, $C_i \supset L_i$, when input L_{i-1}. A

TABLE 6.7: Apriori-Gen Example

Scan	Candidates	Large Itemsets
1	{Blouse}, {Jeans}, {Shoes}, {Shorts}, {Skirt}, {TShirt}	{Jeans}, {Shoes}, {Shorts} {Skirt}, {Tshirt}
2	{Jeans, Shoes}, {Jeans, Shorts}, {Jeans, Skirt}, {Jeans, TShirt}, {Shoes, Shorts}, {Shoes, Skirt}, {Shoes, TShirt}, {Shorts, Skirt}, {Shorts, TShirt}, {Skirt, TShirt}	{Jeans, Shoes}, {Jeans, Shorts}, {Jeans, TShirt}, {Shoes, Shorts}, {Shoes, TShirt}, {Shorts, TShirt}, {Skirt, TShirt}
3	{Jeans, Shoes, Shorts}, {Jeans, Shoes, TShirt}, {Jeans, Shorts, TShirt}, {Jeans, Skirt, TShirt}, {Shoes, Shorts, TShirt}, {Shoes, Skirt, TShirt}, {Shorts, Skirt, TShirt}	{Jeans, Shoes, Shorts}, {Jeans, Shoes, TShirt}, {Jeans, Shorts, TShirt}, {Shoes, Shorts, TShirt}
4	{Jeans, Shoes, Shorts, TShirt}	{Jeans, Shoes, Shorts, TShirt}
5	Ø	Ø

pruning step, not shown, could be added at the end of this algorithm to prune away any candidates that have subsets of size $i-1$ that are not large.

6.3.2

ALGORITHM 6.2

```
Input:
   L_{i-1}   //Large itemsets of size i-1
Output:
   C_i       //Candidates of size i
Apriori-gen algorithm:
   C_i = Ø;
   for each I ∈ L_{i-1} do
      for each J ≠ I ∈ L_{i-1} do
         if i-2 of the elements in I and J are equal then
            C_i = C_i ∪ {I ∪ J};
```

Given the large itemset property and Apriori-Gen, the Apriori algorithm itself (see Algorithm 6.3) is rather straightforward. In this algorithm we use c_i to be the count for item $I_i \in I$.

ALGORITHM 6.3

```
Input:
   I       //Itemsets
   D       //Database of transactions
   s       //Support
Output:
   L       //Large itemsets
Apriori algorithm:
   k = 0; //k is used as the scan number.
   L = Ø;
```

```
C1 = I;          //Initial candidates are set to be the items.
repeat
   k = k + 1;
   Lk = ∅;
   for each Ii ∈ Ck do
      ci = 0;    // Initial counts for each itemset are 0.
   for each tj ∈ D do
      for each Ii ∈ CK do
         if Ii ∈ tj then
            ci = ci + 1;
   for each Ii ∈ Ck do
      if ci ≥ (s× | D |) do
         Lk = Lk ∪ Ii;
   L = L ∪ Lk;
   Ck+1 = Apriori-Gen(Lk)
until Ck+1 = ∅;
```

The Apriori algorithm assumes that the database is memory-resident. The maximum number of database scans is one more than the cardinality of the largest large itemset. This potentially large number of database scans is a weakness of the Apriori approach.

6.3.2 Sampling Algorithm

To facilitate efficient counting of itemsets with large databases, sampling of the database may be used. The original sampling algorithm reduces the number of database scans to one in the best case and two in the worst case. The database sample is drawn such that it can be memory-resident. Then any algorithm, such as Apriori, is used to find the large itemsets for the sample. These are viewed as *potentially large* (*PL*) itemsets and used as candidates to be counted using the entire database. Additional candidates are determined by applying the *negative border* function, BD^-, against the large itemsets from the sample. The entire set of candidates is then $C = BD^-(PL) \cup PL$. The negative border function is a generalization of the Apriori-Gen algorithm. It is defined as the minimal set of itemsets that are not in *PL*, but whose subsets are all in *PL*. Example 6.4 illustrates the idea.

EXAMPLE 6.4

Suppose the set of items is $\{A, B, C, D\}$. The set of large itemsets found to exist in a sample of the database is $PL = \{A, C, D, CD\}$. The first scan of the entire database, then, generates the set of candidates as follows: $C = BD^-(PL) \cup PL = \{B, AC, AD\} \cup \{A, C, D, CD\}$. Here we add AC because both A and C are in *PL*. Likewise we add AD. We could not have added ACD because neither AC nor AD is in *PL*. When looking at the lattice (Figure 6.2), we add only sets where all subsets are already in *PL*. Note that we add B because all its subsets are vacuously in *PL*.

Algorithm 6.4 shows the sampling algorithm. Here the Apriori algorithm is shown to find the large itemsets in the sample, but any large itemset algorithm could be used. Any algorithm to obtain a sample of the database could be used as well. The set of large