MODULE-3 ASSOCIATION ANALYSIS

3.1 Introduction

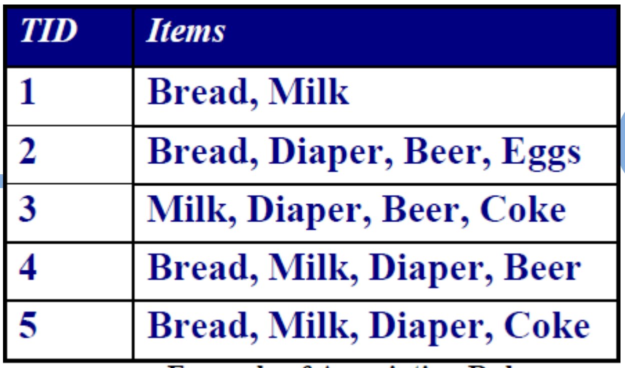
Data Mining Association Analysis: Basic Concepts and Algorithms

Association Rule Mining

Given a set of transactions, find rules that will predict the occurrence of an item based on the occurrences of other items in the transaction

Market-Basket transactions

VTU om



TID	*Items*
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

Example of Association Rules

{Diaper} ->{Beer},

{Milk, Bread} ->{Eggs,Coke},

{Beer, Bread} -> {Milk},

Definition: Item Set and support count

Itemset and Support Count Let I ={i1,i2,….id} be the set of all itemsin a market basket data and T : {t1,t2,..-,tN} be the set of all transactions. Each transaction ti contains a subset of items chosen from I
In association analysis, a collection of zero or more items is termed an itemset. If an itemsetcontains k-items, it is called a k-itemset

Example: {Milk, Bread, Diaper}

Association Rule An association rule is an implication expression of the form $X \longrightarrow Y$, where X and Y are disjoint itemsets, i.e., $X \cap Y = \emptyset$. 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, while confidence determines how frequently items in Y appear in transactions that contain X. The formal definitions of these metrics are

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

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

Rule Evaluation Metrics:
Support count (σ)
– Frequency of occurrence of an itemset

– E.g. σ({Milk, Bread, Diaper}) = 2/5 Support(s)

– Fraction of transactions that contain an itemset

– E.g. s({Milk, Bread, Diaper}) = 2

Frequent Itemset

– An itemset whose support is greater than or equal to a *minsup* threshold .

VTUPulse.com

Example:

$$\{\text{Milk}, \text{Diaper}\} \Rightarrow \text{Beer}$$

$$s = \frac{\sigma(\text{Milk}, \text{Diaper}, \text{Beer})}{|\text{T}|} = \frac{2}{5} = 0.4$$

$$c = \frac{\sigma(\text{Milk}, \text{Diaper}, \text{Beer})}{\sigma(\text{Milk}, \text{Diaper})} = \frac{2}{3} = 0.67$$

3.2 Association Rule Mining Task

Given a set of transactions T, the goal of association rule mining is to find all rules having
– support ≥ *minsup*threshold

– confidence ≥ *minconf*threshold

Brute-force approach:

– List all possible association rules

– Compute the support and confidence for each rule

– Prune rules that fail the *minsup* and *minconf* thresholds

▯ Computationally prohibitive!

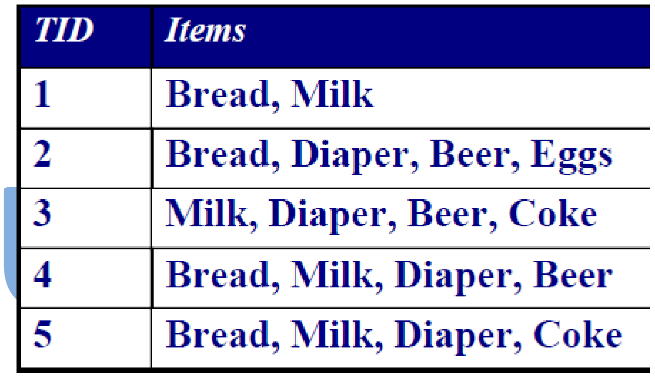
More specifically, the total number of possible rules extracted from a data set that contains d items is

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

Even for the small data set with 6 items, this approach requires us to compute the support and confidence for 36 - 27 * 1 = 602 rules.

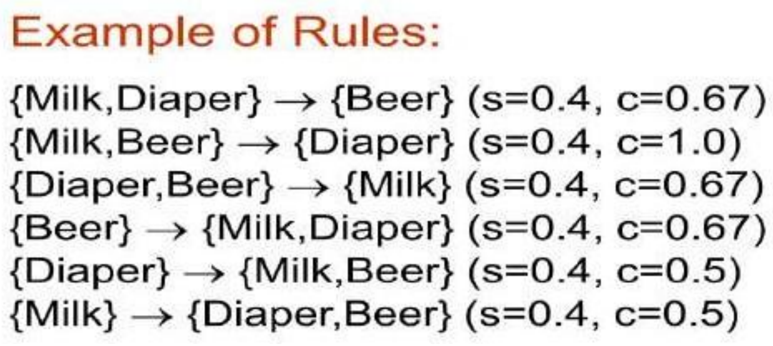
More than 80% of the rules are discarded after applying minsup : 20Vo andminconf : 5070, thus making most of the computations become wasted.

To avoid performing needless computations, it would be useful to prune the rulesearly without having to compute their support and confidence values.



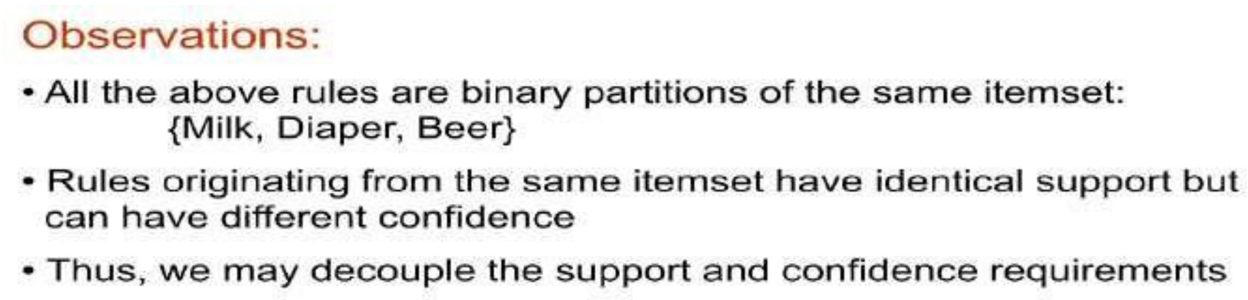
TID	*Items*
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

VTU om



Example of Rules:

$\{Milk,Diaper\} \rightarrow \{Beer\}$ (s=0.4, c=0.67)
$\{Milk,Beer\} \rightarrow \{Diaper\}$ (s=0.4, c=1.0)
$\{Diaper,Beer\} \rightarrow \{Milk\}$ (s=0.4, c=0.67)
$\{Beer\} \rightarrow \{Milk,Diaper\}$ (s=0.4, c=0.67)
$\{Diaper\} \rightarrow \{Milk,Beer\}$ (s=0.4, c=0.5)
$\{Milk\} \rightarrow \{Diaper,Beer\}$ (s=0.4, c=0.5)



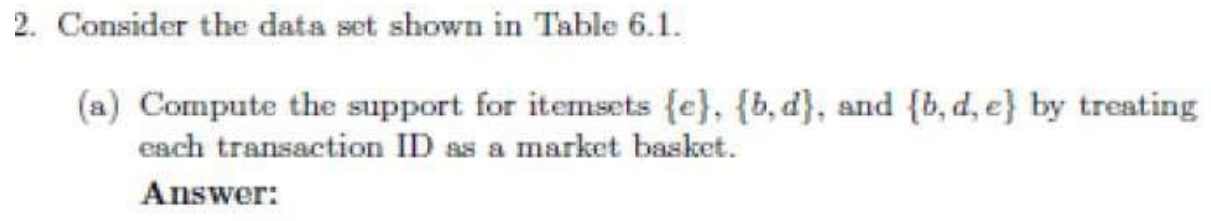
Observations:

- All the above rules are binary partitions of the same itemset: {Milk, Diaper, Beer}
- Rules originating from the same itemset have identical support but can have different confidence
- Thus, we may decouple the support and confidence requirements

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
– Generate all itemsets whose support >= minsup

2. Rule Generation
– Generate high confidence rules from each frequent itemset, where each rule is a binary partitioning of a frequent itemset .
Frequent itemset generation is still computationally expensive.



2. Consider the data set shown in Table 6.1.

   (a) Compute the support for itemsets $\{e\}$, $\{b,d\}$, and $\{b,d,e\}$ by treating each transaction ID as a market basket.

   Answer:

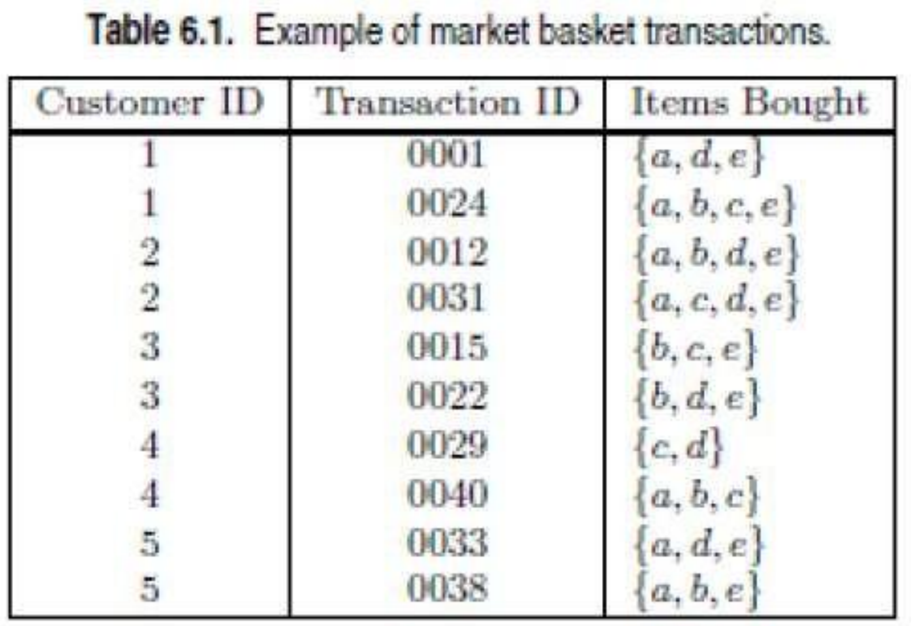


Table 6.1. Example of market basket transactions.

Customer ID	Transaction ID	Items Bought
1	0001	$\{a, d, e\}$
1	0024	$\{a, b, c, e\}$
2	0012	$\{a, b, d, e\}$
2	0031	$\{a, c, d, e\}$
3	0015	$\{b, c, e\}$
3	0022	$\{b, d, e\}$
4	0029	$\{c, d\}$
4	0040	$\{a, b, c\}$
5	0033	$\{a, d, e\}$
5	0038	$\{a, b, e\}$

$$\begin{aligned} s(\{e\}) &= \frac{8}{10} = 0.8 \\ s(\{b, d\}) &= \frac{2}{10} = 0.2 \\ s(\{b, d, e\}) &= \frac{2}{10} = 0.2 \end{aligned}$$

com

(b) Use the results in part (a) to compute the confidence for the association rules $\{b,d\} \longrightarrow \{e\}$ and $\{e\} \longrightarrow \{b,d\}$. Is confidence a symmetric measure?

Answer:

$$\begin{aligned} c(bd \longrightarrow e) &= \frac{0.2}{0.2} = 100\% \\ c(e \longrightarrow bd) &= \frac{0.2}{0.8} = 25\% \end{aligned}$$

No, confidence is not a symmetric measure.

(c) Repeat part (a) by treating each customer ID as a market basket. Each item should be treated as a binary variable (1 if an item appears in at least one transaction bought by the customer, and 0 otherwise.)

Answer:

$$\begin{aligned} s(\{e\}) &= \frac{4}{5} = 0.8 \\ s(\{b,d\}) &= \frac{5}{5} = 1 \\ s(\{b,d,e\}) &= \frac{4}{5} = 0.8 \end{aligned}$$

(d) Use the results in part (c) to compute the confidence for the as
rules $\{b,d\} \longrightarrow \{e\}$ and $\{e\} \longrightarrow \{b,d\}$.

Answer:

$$\begin{aligned} c(bd \longrightarrow e) &= \frac{0.8}{1} = 80\% \\ c(e \longrightarrow bd) &= \frac{0.8}{0.8} = 100\% \end{aligned}$$

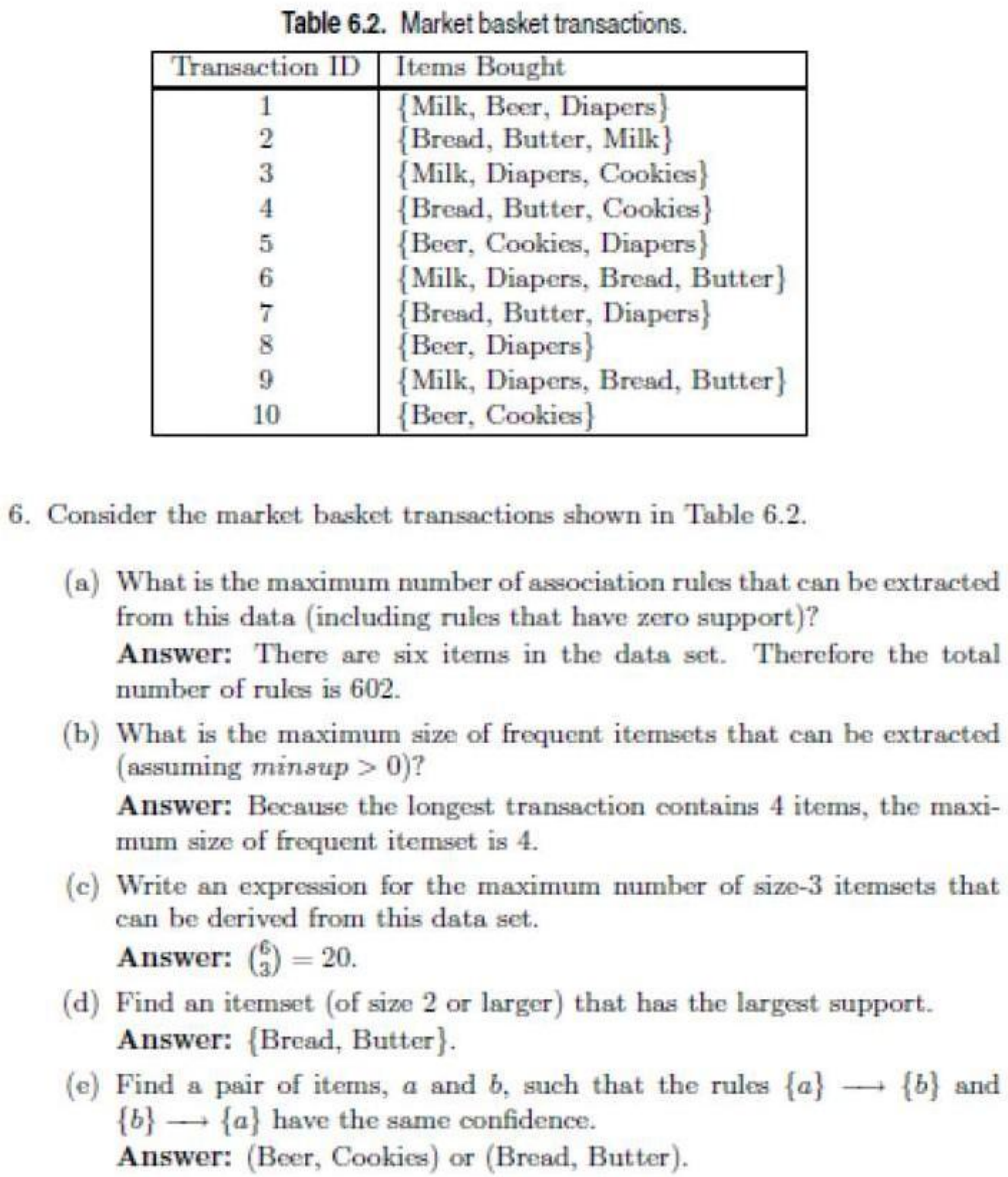


Table 6.2. Market basket transactions.

Transaction ID	Items Bought
1	{Milk, Beer, Diapers}
2	{Bread, Butter, Milk}
3	{Milk, Diapers, Cookies}
4	{Bread, Butter, Cookies}
5	{Beer, Cookies, Diapers}
6	{Milk, Diapers, Bread, Butter}
7	{Bread, Butter, Diapers}
8	{Beer, Diapers}
9	{Milk, Diapers, Bread, Butter}
10	{Beer, Cookies}

6. Consider the market basket transactions shown in Table 6.2.

   (a) What is the maximum number of association rules that can be extracted from this data (including rules that have zero support)?
   Answer: There are six items in the data set. Therefore the total number of rules is 602.

   (b) What is the maximum size of frequent itemsets that can be extracted (assuming *minsup* > 0)?
   Answer: Because the longest transaction contains 4 items, the maximum size of frequent itemset is 4.

   (c) Write an expression for the maximum number of size-3 itemsets that can be derived from this data set.
   Answer: $\binom{6}{3} = 20$.

   (d) Find an itemset (of size 2 or larger) that has the largest support.
   Answer: {Bread, Butter}.

   (e) Find a pair of items, a and b, such that the rules $\{a\} \longrightarrow \{b\}$ and $\{b\} \longrightarrow \{a\}$ have the same confidence.
   Answer: (Beer, Cookies) or (Bread, Butter).

3.3 Frequent Itemset Generation:

A lattice structure can be used to enumerate the list of all possible item sets.

Figure 6.1 shows an item set lattice for 1: {a,b,c.,d,e}.In general, a data setthat contains k items can potentially generate up to 2^k- 1 frequent itemsets ,excluding the null set. Because k can be very large in many practical applications, the search space of item sets that need to be explored is exponentially Large.

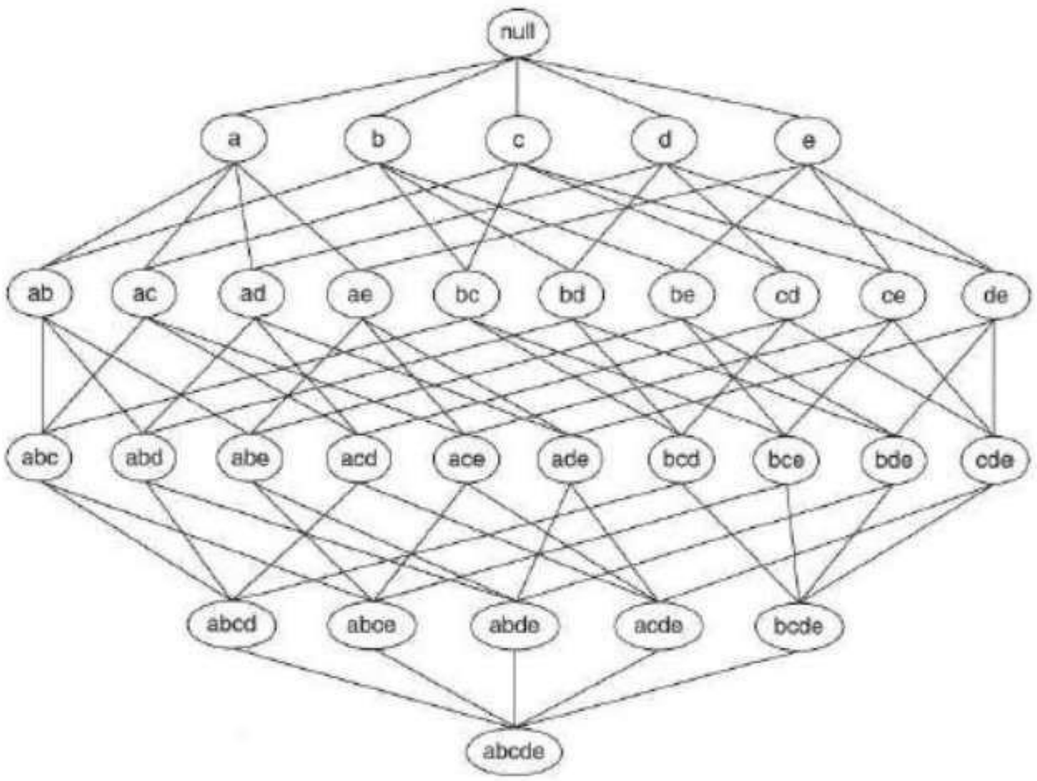


Figure 6.1. An itemset lattice.

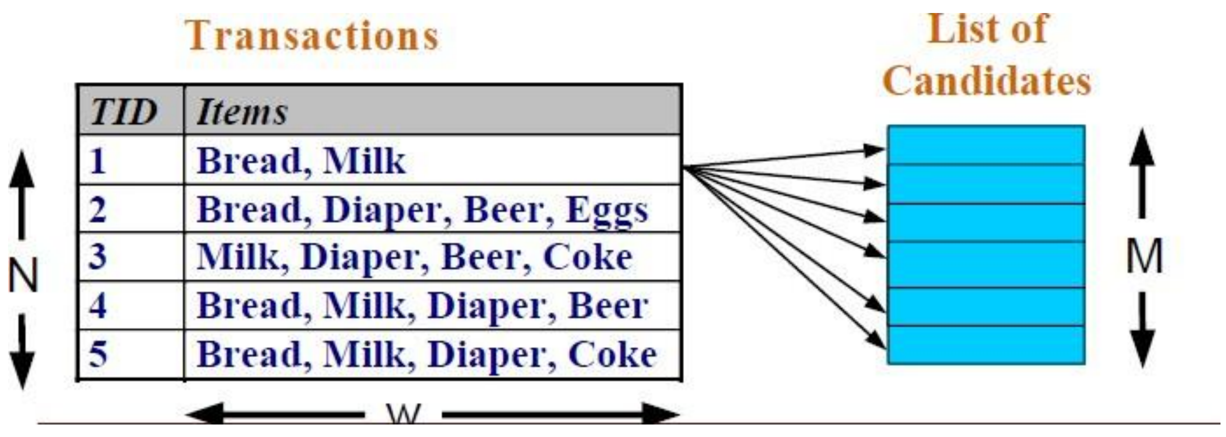
VTUPulse.com

Brute-force approach:

– Each itemset in the lattice is a candidate frequent itemset

– Count the support of each candidate by scanning the database

Such an approach can be very expensive because it requires O(N Mw) 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.



TID	Items
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

There are several ways to reduce the computational complexity of frequent itemset generation.

Reduce the number of candidates (M)

– Complete search: $M=2^d$
– Use pruning techniques to reduce M

Reduce the number of transactions (N)
– Reduce size of N as the size of itemset increases

Reduce the number of comparisons (NM)

– Use efficient data structures to store the candidates or transactions
– No need to match every candidate against every transaction.

Apriori principle : Reducing Number of Candidates

Apriory principle: If an itemset is frequent, then all of its subsets must also be frequent

To illustrate the idea behind the Apriory principle, consider the itemset Iattice shown in Figure 6.3. Suppose {c, d, e} is a frequent itemset. Clearly,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.

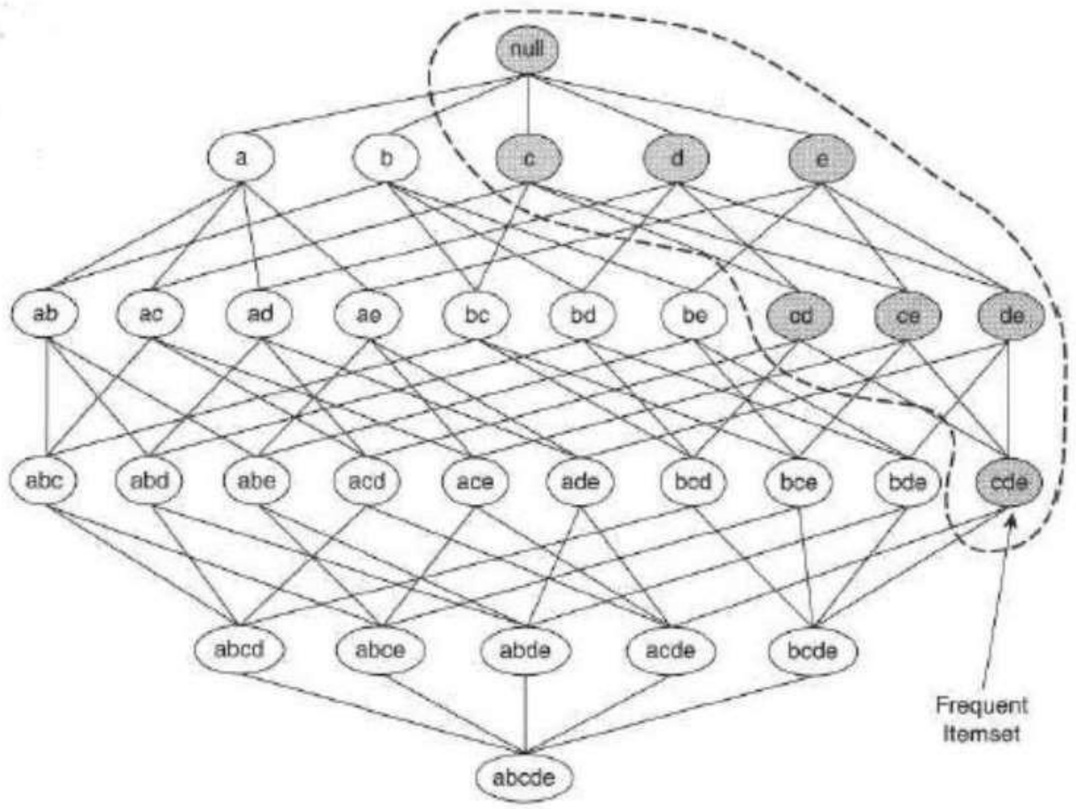


Figure 6.3. An illustration of the *Apriori* principle. If $\{c, d, e\}$ is frequent, then all subsets of this itemset are frequent.

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

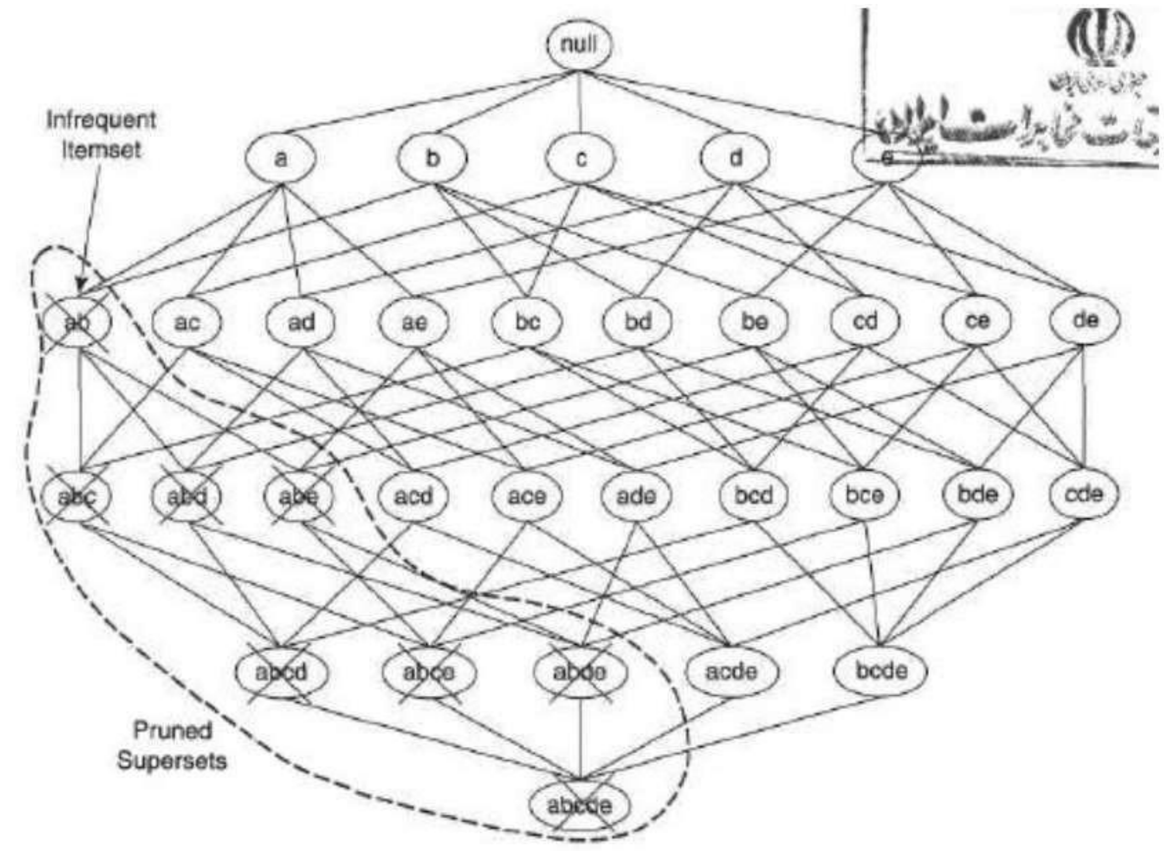
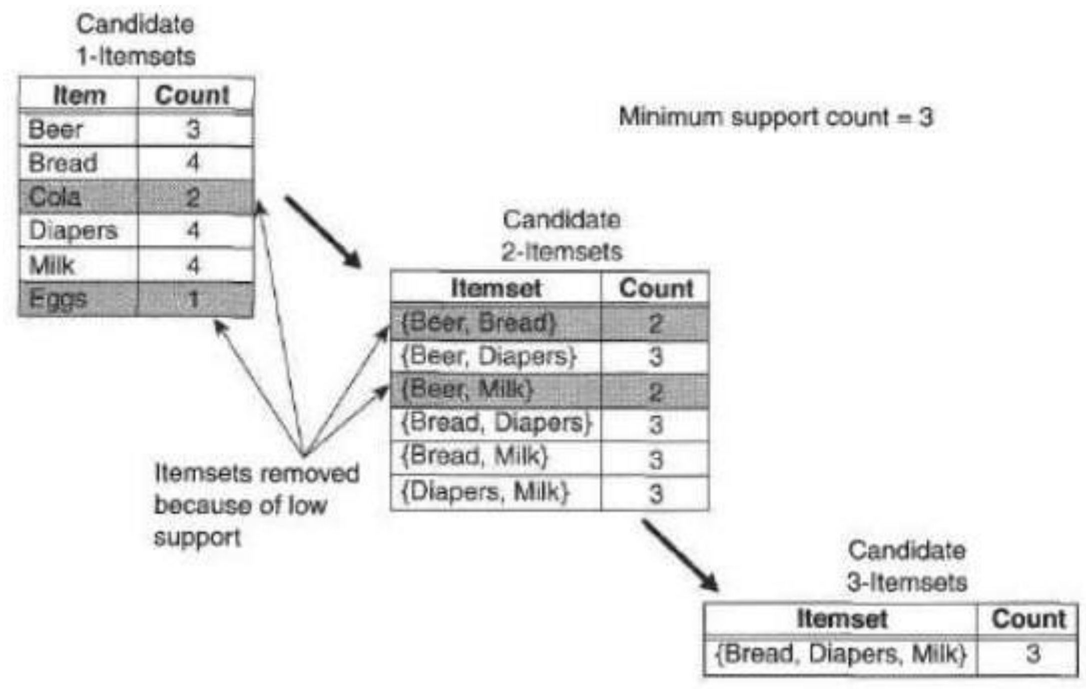


Figure 6.4. An illustration of support-based pruning. If $\{a, b\}$ is infrequent, then all supersets of $\{a, b\}$ are infrequent.

Frequent Itemset Generation in the Apriori Algorithm: Illustration with example.

Figure 6.5 provides a high-level illustration of the frequent item set generation part of the Apriori algorithm for the transactions shown inTable 6.1. We assume that the support threshold is 60 To, which is equivalent to a minimum support count equal to 3.


Candidate 1-Itemsets
Minimum support count = 3
Candidate 2-Itemsets
Itemsets removed because of low support
Candidate 3-Itemsets

Item	Count
Beer	3
Bread	4
Cola	2
Diapers	4
Milk	4
Eggs	1

Itemset	Count
{Beer, Bread}	2
{Beer, Diapers}	3
{Beer, Milk}	2
{Bread, Diapers}	3
{Bread, Milk}	3
{Diapers, Milk}	3

Itemset	Count
{Bread, Diapers, Milk}	3

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

Initially, every item is considered as a candidate l-itemset. Aftercounting 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 becausethe Apriory 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 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 20 candidate3-itemsets that can be formed using the six items given in this example. With the Apriory principle, we only need to keep candidate 3- itemsets whose subsets are frequent. The only candidate that has this property is {Bread, Diapers, Milk).

If every subset is considered,
$^6C_1 + {}^6C_2 + {}^6C_3 = 41$
With support-based pruning,
$6 + 6 + 1 = 13$

The effectiveness of the Apriory pruning strategy can be shown by counting the number of candidate itemsets generated.
A brute-force strategy of enumerating all itemsets(up to size3) as candidates w ill produce 41 candidates.

VTUPulse.com

With the Apriory principle, this number decreases t o candidates, which epresents a 68% reduction in the number of candidate itemsets even in this simple example.

Apriori Algorithm:

Input: set of items I, set of transactions T, number of transactions N, minimum support minsup.
Output: frequent k-itemsets Fk, k=1…
Method:
K=1

- Compute support for each 1-itemset (item) by scanning the transactions
- F1 = items that have support above minsup
- Repeat until no new frequent itemsets are identified

1. Ck+1 = candidate k+1 -itemsets generated from length k frequent itemsets Fk
2. Compute the support of each candidate in Ck+1 by scanning the transactions T
3. Fk+1 = Candidates in Ck+1 that have support above minsup

VTUPulse.com

8. The *Apriori* algorithm uses a generate-and-count strategy for deriving frequent itemsets. Candidate itemsets of size $k+1$ are created by joining a pair of frequent itemsets of size k (this is known as the candidate generation step). A candidate is discarded if any one of its subsets is found to be infrequent during the candidate pruning step. Suppose the *Apriori* algorithm is applied to the data set shown in Table 6.3 with $minsup = 30\%$, i.e., any itemset occurring in less than 3 transactions is considered to be infrequent.

Table 6.3. Example of market basket transactions.

Transaction ID	Items Bought
1	$\{a, b, d, e\}$
2	$\{b, c, d\}$
3	$\{a, b, d, e\}$
4	$\{a, c, d, e\}$
5	$\{b, c, d, e\}$
6	$\{b, d, e\}$
7	$\{c, d\}$
8	$\{a, b, c\}$
9	$\{a, d, e\}$
10	$\{b, d\}$

(a) Draw an itemset lattice representing the data set given in Table 6.3. Label each node in the lattice with the following letter(s):

- **N**: If the itemset is not considered to be a candidate itemset by the *Apriori* algorithm. There are two reasons for an itemset not to be considered as a candidate itemset: (1) it is not generated at all during the candidate generation step, or (2) it is generated during the candidate generation step but is subsequently removed during the candidate pruning step because one of its subsets is found to be infrequent.
- **F**: If the candidate itemset is found to be frequent by the *Apriori* algorithm.
- **I**: If the candidate itemset is found to be infrequent after support counting.

Answer:

The lattice structure is shown below.

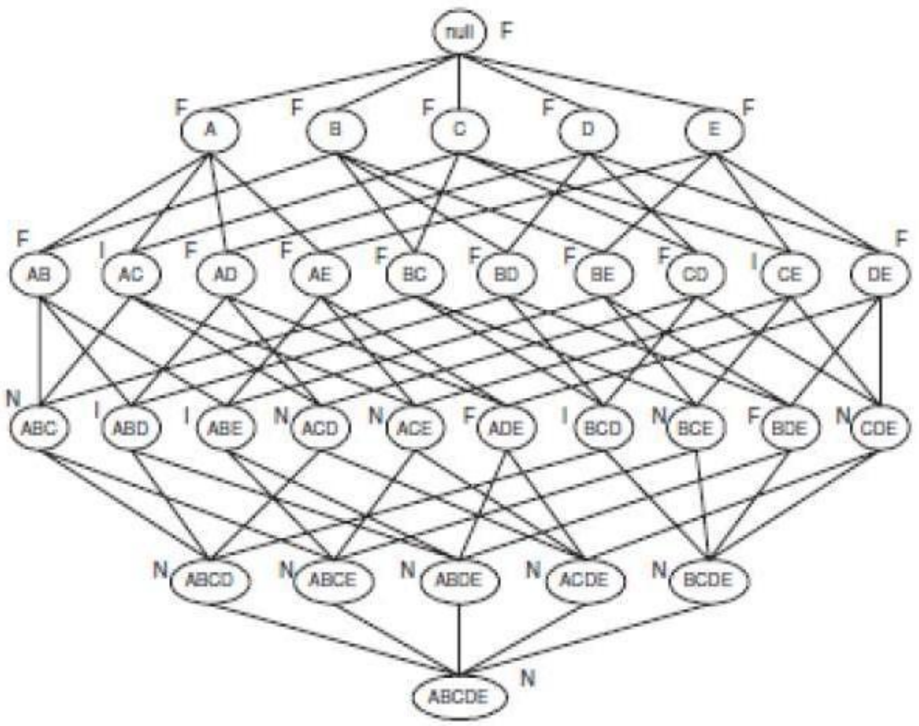


Figure 6.1. Solution.

(b) What is the percentage of frequent itemsets (with respect to all itemsets in the lattice)?

Answer:

Percentage of frequent itemsets = 16/32 = 50.0% (including the null set).

(c) What is the pruning ratio of the *Apriori* algorithm on this data set? (Pruning ratio is defined as the percentage of itemsets not considered to be a candidate because (1) they are not generated during candidate generation or (2) they are pruned during the candidate pruning step.)

Answer:

Pruning ratio is the ratio of N to the total number of itemsets. Since the count of $N = 11$, therefore pruning ratio is $11/32 = 34.4\%$.

(d) What is the false alarm rate (i.e, percentage of candidate itemsets that are found to be infrequent after performing support counting)?

Answer:

False alarm rate is the ratio of I to the total number of itemsets. Since the count of $I = 5$, therefore the false alarm rate is $5/32 = 15.6\%$.

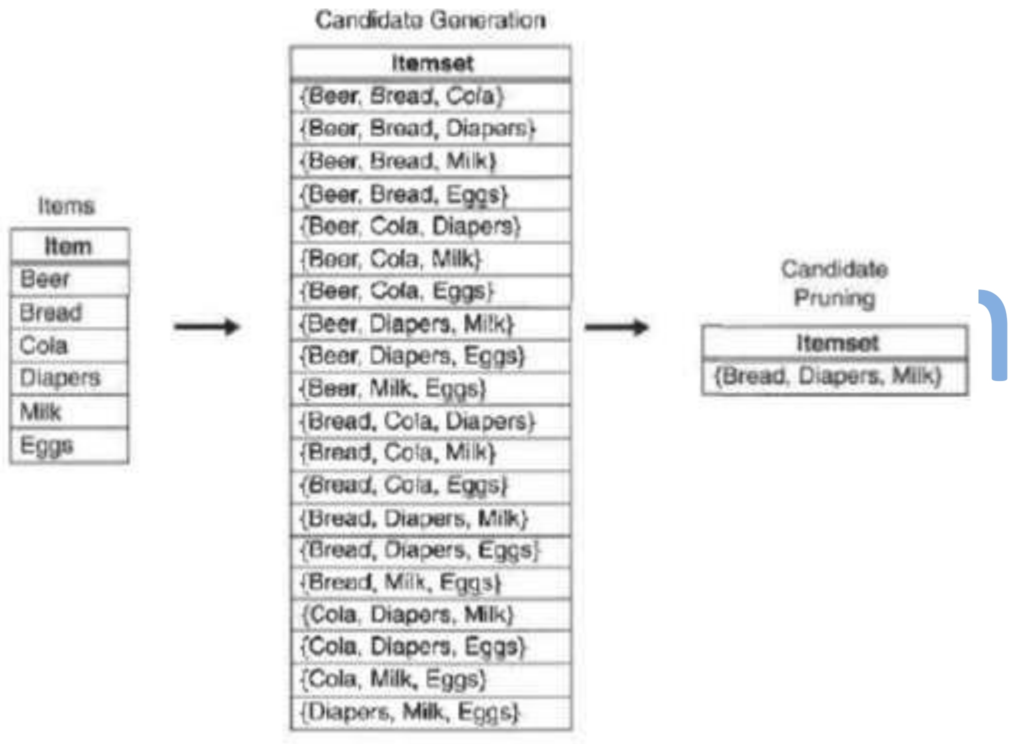
Candidate Generation and Pruning

Candidate Generation: This operation generates new candidate kitemsets based on the frequent (k - l)-itemsets found in the previous iteration.
Candidate Pruning: This operation eliminates some of the candidate k-itemsets using the support- based pruning strategy.
The folIowing is a list of requirements for an effective candidate generation procedure:

- It should avoid generating too many unnecessary candidates.
- It must ensure that the candidate set is complete, i.e., no frequent itemsetsare left out by the candidate generation procedure.
- It should not generate the same candidate itemset more than once. e.g. {a,b,c,d} can be generated by merging {a,b,c} with {d} or {b,d} with {a,c}, {a,b} with {c,d}

Several candidate generation strategies are discussed below.

Brute-Force Method: The brute-force method considers every k-itemset asa potential candidate and then applies the candidate pruning step to removeany unnecessary candidates (see Figure



6.6).

Figure 6.6. A brute-force method for generating candidate 3-itemsets.

Fk-1 x F1 Method:
Combine frequent k-1 –itemsets with frequent 1- itemsets
Figure 6.7 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}.

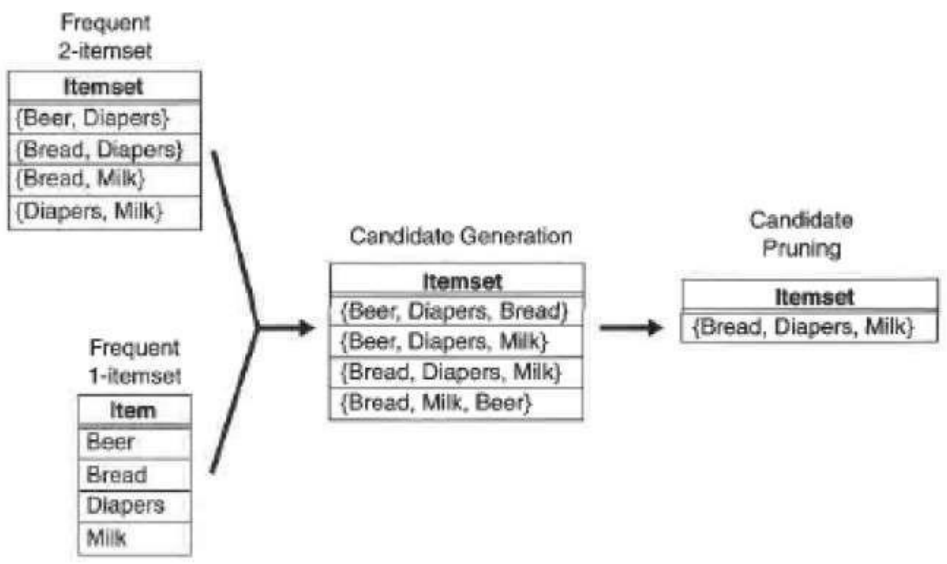


Figure 6.7. Generating and pruning candidate k-itemsets by merging a frequent $(k-1)$-itemset with a frequent item. Note that some of the candidates are unnecessary because their subsets are infrequent.

Satisfaction of our requirements

1) while many k-itemsets are left ungenerated, can still generate unnecessary candidates
e.g. merging {Beer, Diapers} with {Milk} is unnecessary, since {Beer, Milk} is infrequent.

2) Method is complete: each frequent itemset consists of a frequent k-1 –itemset and a frequent 1-itemset.

3) Can generate the same set twice
e.g. {Bread, Diapers, Milk} can be generated by merging {Bread,Diapers} with {Milk} or {Bread,Milk} with {Diapers} or {Diapers, Milk} with {Bread}
This can be circumvented by keeping all frequent itemsets in their lexicographical order (\
- e.g. {Bread,Diapers} can be merged with {Milk} as 'Milk' comes after 'Bread' and 'Diapers' in lexicographical order
- {Diapers, Milk} is not merged with {Bread}, {Bread, Milk} is not merged with {Diapers} as that would violate the lexicographical ordering

VTUPulse.com

Fk-1 x Fk-1 Method:

- Combine a frequent *k-1* –itemset with another frequent *k-1* -itemset
- Items are stored in lexicographical order in the itemset
- When considering for merging, only pairs that share first k-2 items are considered

o `e.g. {Bread, Diapers} is merged with {Bread,Milk}`

o if the pairs share fewer than k-2 items, the resulting itemset would be larger than k, so we do not need to generate it yet.

- The resulting *k*-itemset has *k* subsets of size *k-1*, which will be checked against support threshold

- the merging ensures that at least two of the subsets are frequent
- An additional check is made that the remaining *k-2* subsets are frequent as well.

In Figure 6.8, the frequent itemsets {Bread, Diapers} and {Bread, Milk} are merged to form a candidate 3-itemset {Bread, Diapers, Milk}.

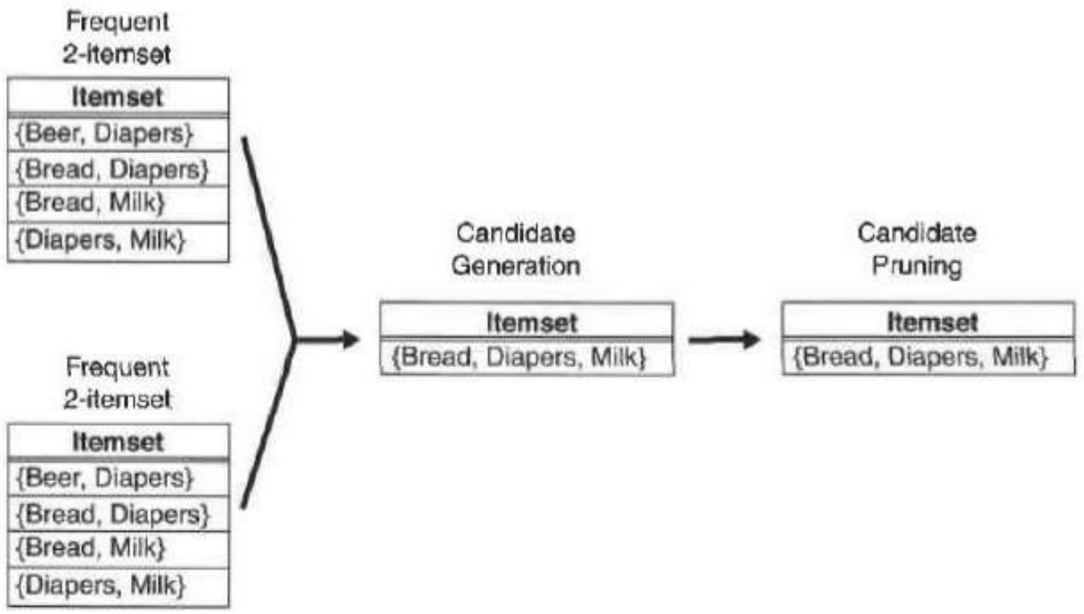
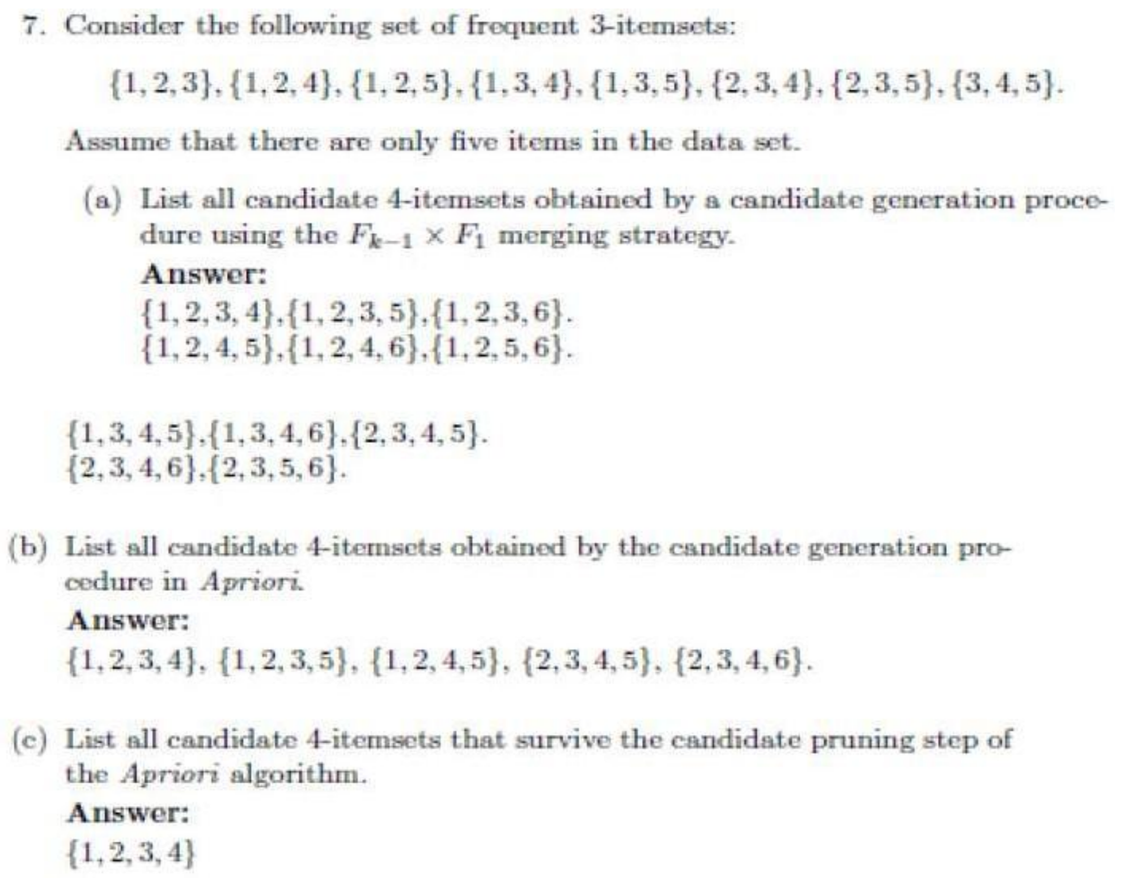


Figure 6.8. Generating and pruning candidate k-itemsets by merging pairs of frequent $(k-1)$-itemsets.

VTUPulse.com

Satisfaction of our requirements

1) Avoids the generation of many unnecessary candidates that are generated by the Fk-1 x F1 method e.g. will not generate {Beer, Diapers, Milk} as {Beer,Milk} is infrequent
2) Method is complete: every frequent k-itemset can be formed of two frequent k-1 –itemsets differing in their last item.
3) Each candidate itemset is generated only once.


7. Consider the following set of frequent 3-itemsets:

$\{1,2,3\}, \{1,2,4\}, \{1,2,5\}, \{1,3,4\}, \{1,3,5\}, \{2,3,4\}, \{2,3,5\}, \{3,4,5\}$.

Assume that there are only five items in the data set.

(a) List all candidate 4-itemsets obtained by a candidate generation procedure using the $F_{k-1} \times F_1$ merging strategy.

Answer:

$\{1,2,3,4\}$,$\{1,2,3,5\}$,$\{1,2,3,6\}$.
$\{1,2,4,5\}$,$\{1,2,4,6\}$,$\{1,2,5,6\}$.

$\{1,3,4,5\}$,$\{1,3,4,6\}$,$\{2,3,4,5\}$.
$\{2,3,4,6\}$,$\{2,3,5,6\}$.

(b) List all candidate 4-itemsets obtained by the candidate generation procedure in *Apriori*.

Answer:

$\{1,2,3,4\}$, $\{1,2,3,5\}$, $\{1,2,4,5\}$, $\{2,3,4,5\}$, $\{2,3,4,6\}$.

(c) List all candidate 4-itemsets that survive the candidate pruning step of the *Apriori* algorithm.

Answer:

$\{1,2,3,4\}$

VTUPulse.com

Support counting using hash tree:

Given the candidate itemsets Ck and the set of transactions T, we need to compute the support counts σ(X) for each itemset X in Ck.

Brute-force algorithm would compare each transaction against each itemset.

- large amount of comparisons.

An alternative approach

- Divide the candidate itemsets Ck into buckets by using a hash function for each transaction t.
- Hash the itemsets contained in t into buckets using the same hash function.
- Compare the corresponding buckets of candidates and the transaction.
- Increment the support counts of each matching candidate itemset.
- A hash tree is used to implement the hash function.

An alternative approach is to enumerate the itemsets contained in each transaction and use them to update the support counts oftheir respective candidate itemsets. To illustrate, consider a transaction t that contains five items, {1,2,3,5,6}.

Figure 6.9 shows a systematic way for enumerating the 3-itemsets contained in t. 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 : {1,2,3,5,6}, all the 3- itemsets contained in f must begin with item 1, 2, or 3.

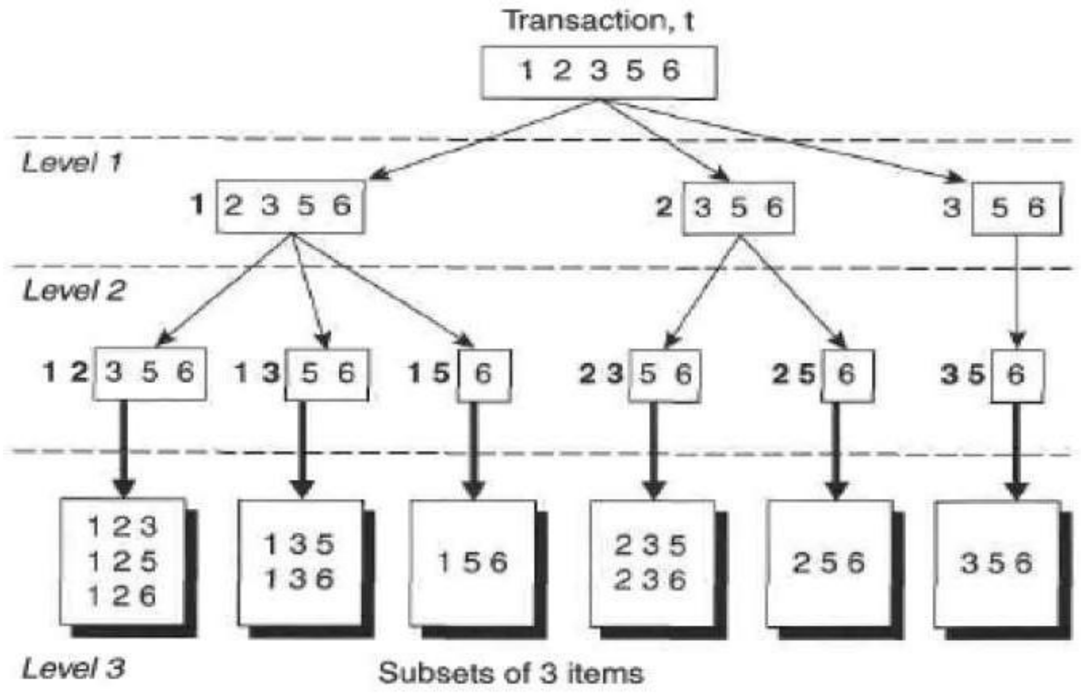
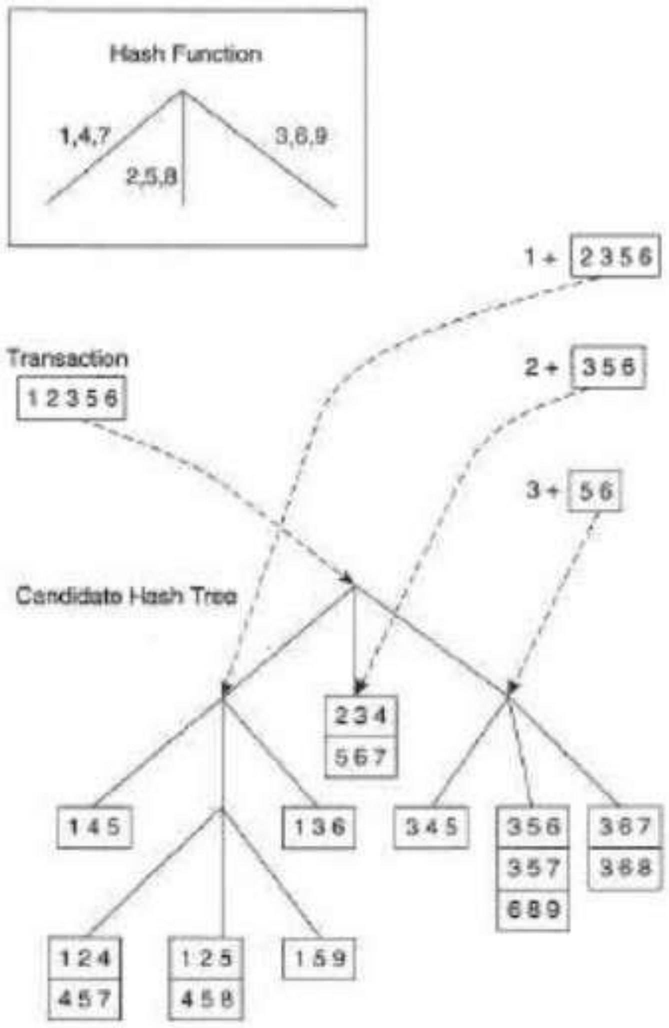


Figure 6.9. Enumerating subsets of three items from a transaction *t*.

Figure 6.11 shows an example of a hash tree structure.
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 6.11 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.
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 of the root node, item 2 is hashed to the middle child, and 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 Figure 6.9.
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 6.12. This process continues until the leaf nodes of the hash tree are reached.
The candidate item sets 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 this example, 5 out of the 9 leaf nodes are visited and 9 out of the 15 item sets are compared against the transaction.

VTUPulse.com



.com

Figure 6.11. Hashing a transaction at the root node of a hash tree.

9. The *Apriori* algorithm uses a hash tree data structure to efficiently count the support of candidate itemsets. Consider the hash tree for candidate 3-itemsets shown in Figure 6.2.

   (a) Given a transaction that contains items $\{1, 3, 4, 5, 8\}$, which of the hash tree leaf nodes will be visited when finding the candidates of the transaction?

   Answer:

   The leaf nodes visited are L1, L3, L5, L9, and L11.

   (b) Use the visited leaf nodes in part (b) to determine the candidate itemsets that are contained in the transaction $\{1, 3, 4, 5, 8\}$.

   Answer:

   The candidates contained in the transaction are $\{1, 4, 5\}$, $\{1, 5, 8\}$, and $\{4, 5, 8\}$.

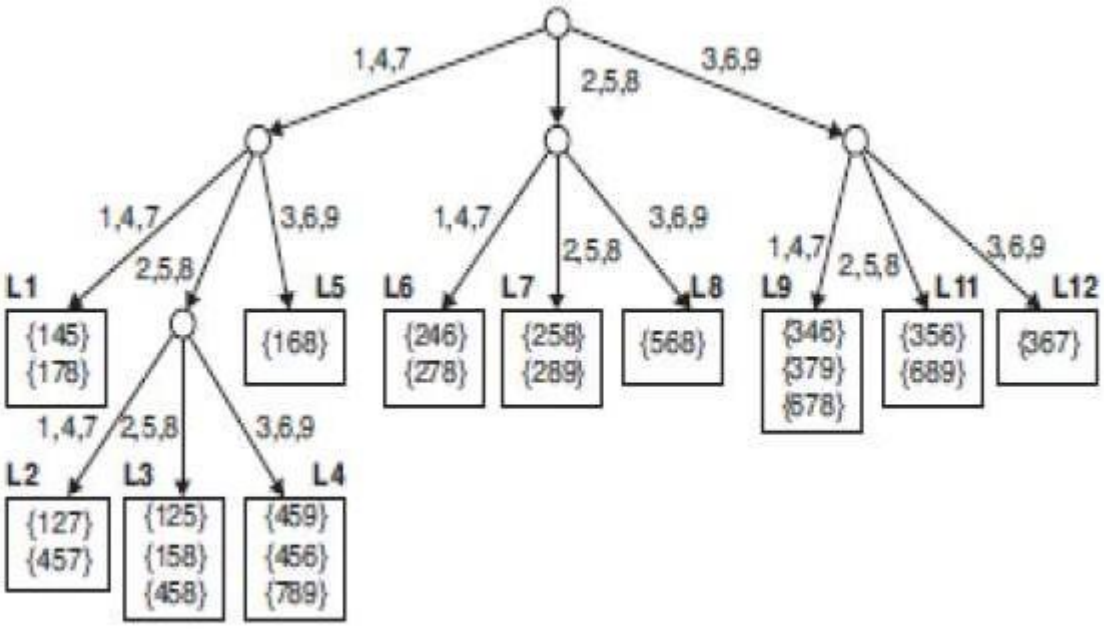


Figure 6.2. An example of a hash tree structure.

– If {A,B,C,D} is a frequent itemset, candidate rules:

Rule Generation

Given a frequent itemset L, find all non-empty subsets f □ L such that f □ L – f satisfies the minimum confidence requirement

– If {A,B,C,D} is a frequent itemset, candidate rules:

$ABC \rightarrow D$,	$ABD \rightarrow C$,	$ACD \rightarrow B$,	$BCD \rightarrow A$,
$A \rightarrow BCD$,	$B \rightarrow ACD$,	$C \rightarrow ABD$,	$D \rightarrow ABC$
$AB \rightarrow CD$,	$AC \rightarrow BD$,	$AD \rightarrow BC$,	$BC \rightarrow AD$,
$BD \rightarrow AC$,	$CD \rightarrow AB$,		

If $|L| = k$, then there are $2^k - 2$ candidate association rules (ignoring $L \rightarrow \varnothing$ and $\varnothing \rightarrow L$)

– In general, confidence does not have an anti-monotone property
How to efficiently generate rules from frequent itemsets? c(ABC->D) can be larger or smaller than c(AB->D)
– But confidence of rules generated from the same itemset has an anti-monotone property
e.g., L = {A,B,C,D}:

$$c(ABC \rightarrow D) \geq c(AB \rightarrow CD) \geq c(A \rightarrow BCD)$$

◆ Confidence is anti-monotone w.r.t. number of items on the RHS of the rule

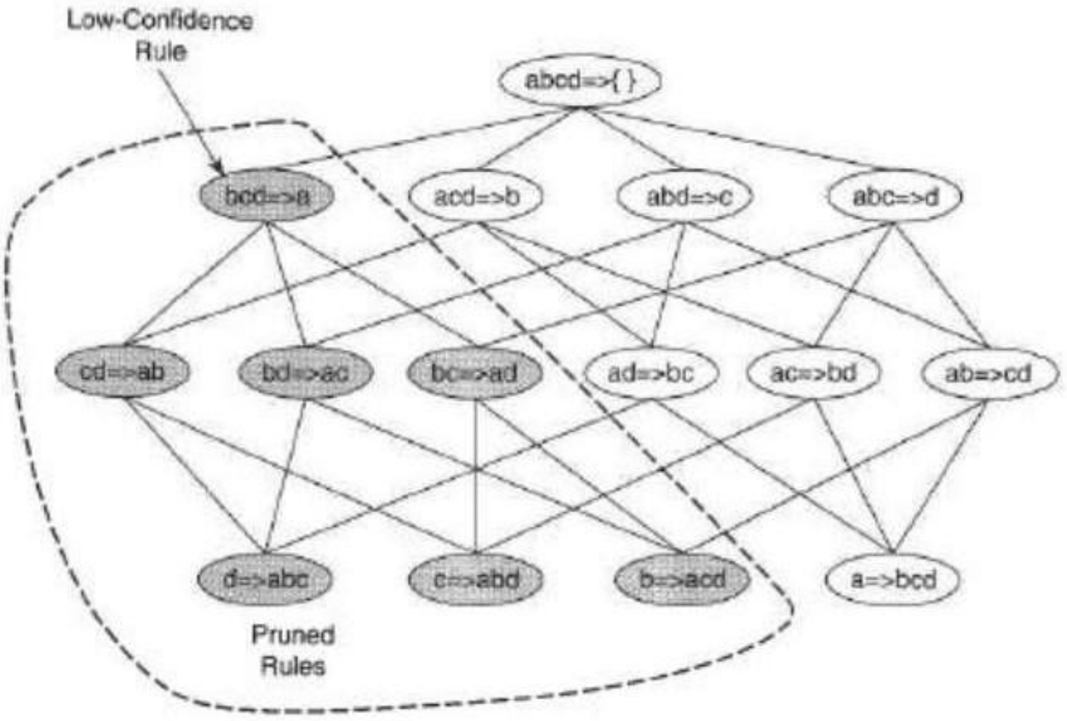


Figure 6.15. Pruning of association rules using the confidence measure.

Candidate rule is generated by merging two rules that share the same prefix in the rule consequent join(CD=>AB,BD=>AC) would produce the candidate rule D => ABC Prune rule D=>ABC if its subset AD=>BC does not have high confidence.

3.4 Alternative Methods for Generating Frequent Itemsets

Traversal of Itemset Lattice::A search for frequent itemsets can be conceptually viewed as a traversal on the itemset lattice.

The search strategy employed by an algorithm dictates how the lattice structure is traversed during the frequent itemset generation process. Some search strategies are better than others, depending on the configuration of frequent itemsets in the lattice.

Equivalence classes : 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.

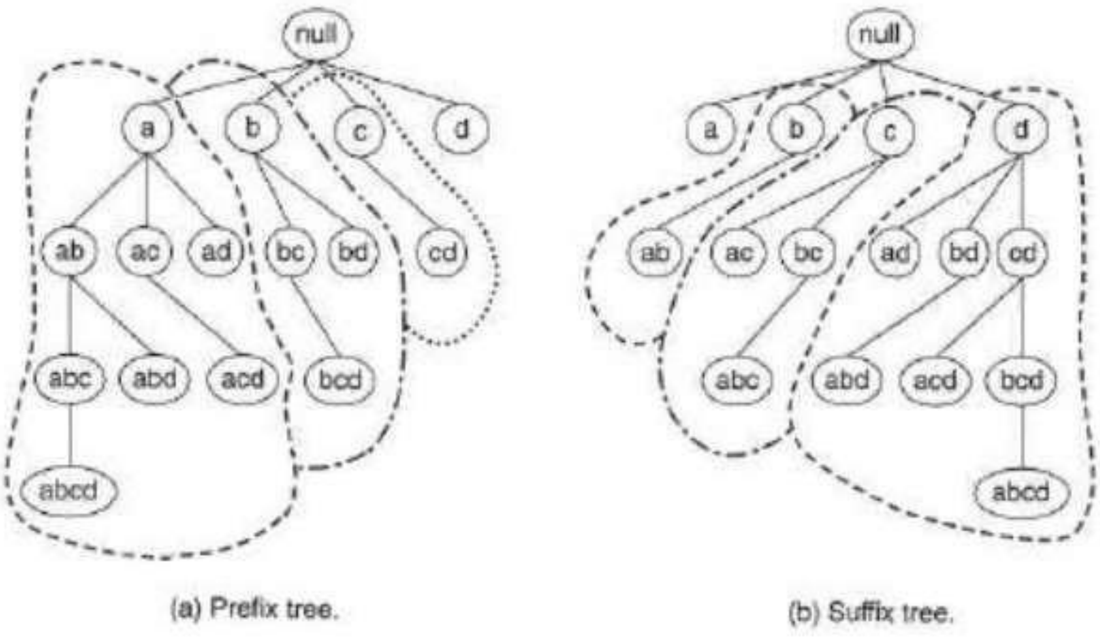


Figure 6.20. Equivalence classes based on the prefix and suffix labels of itemsets.

Breadth-First versus Depth-First: The Apriori, algorithm traverses the lattice in a breadth-first manner) as shown in Figure 6.2L(a). 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 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.

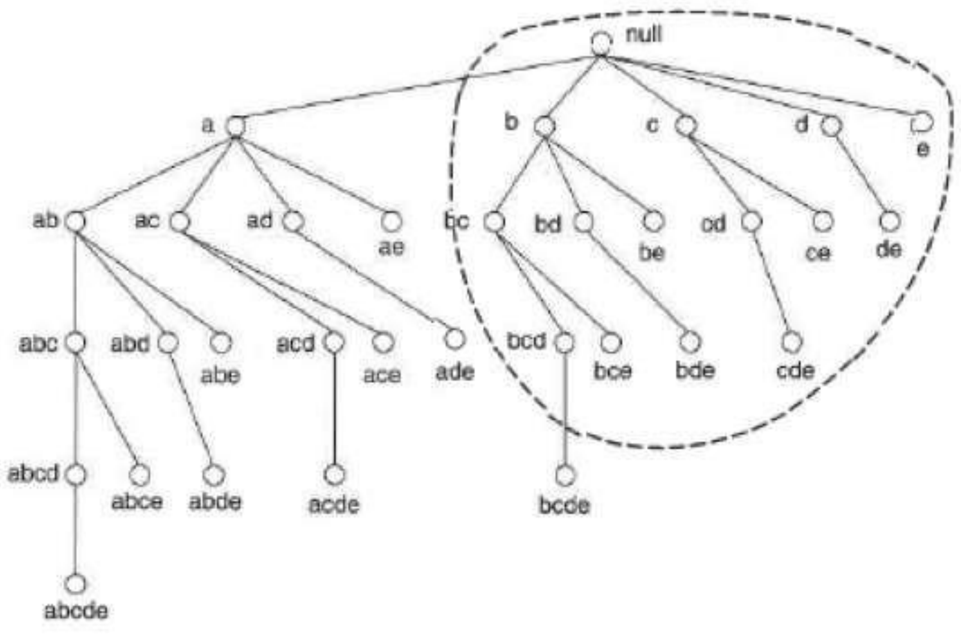


Figure 6.22. Generating candidate itemsets using the depth-first approach.

3.5 FP-Growth Algorithm

Apriori: uses a generate-and-test approach – generates candidate itemsets and tests if they are frequent – Generation of candidate itemsets is expensive(in both space and time)

– Support counting is expensive

- Subset checking (computationally expensive) Multiple Database scans

VTUPulse.com

○ FP-Growth: allows frequent itemset discovery without candidate itemset generation. Two step approach: –

Step 1: Build a compact data structure called the FP-tree

- Built using 2 passes over the data-set.

Step 2: Extracts frequent itemsets directly from the FP-tree

Step 1: FP-Tree Construction

FP-Tree is constructed using 2 passes over the data-set:

Pass 1: Scan data and find support for each

– Sort frequent items in decreasing order based on their support.

Use this order when building the FP-Tree, so common prefixes can be shared. Pass 2:

Nodes correspond to items and have a counter

1. FP-Growth reads 1 transaction at a time and maps it to a path 2. Fixed order is used, so paths can overlap when transactions share items (when they have the same prfix). – In this case, counters are incremented

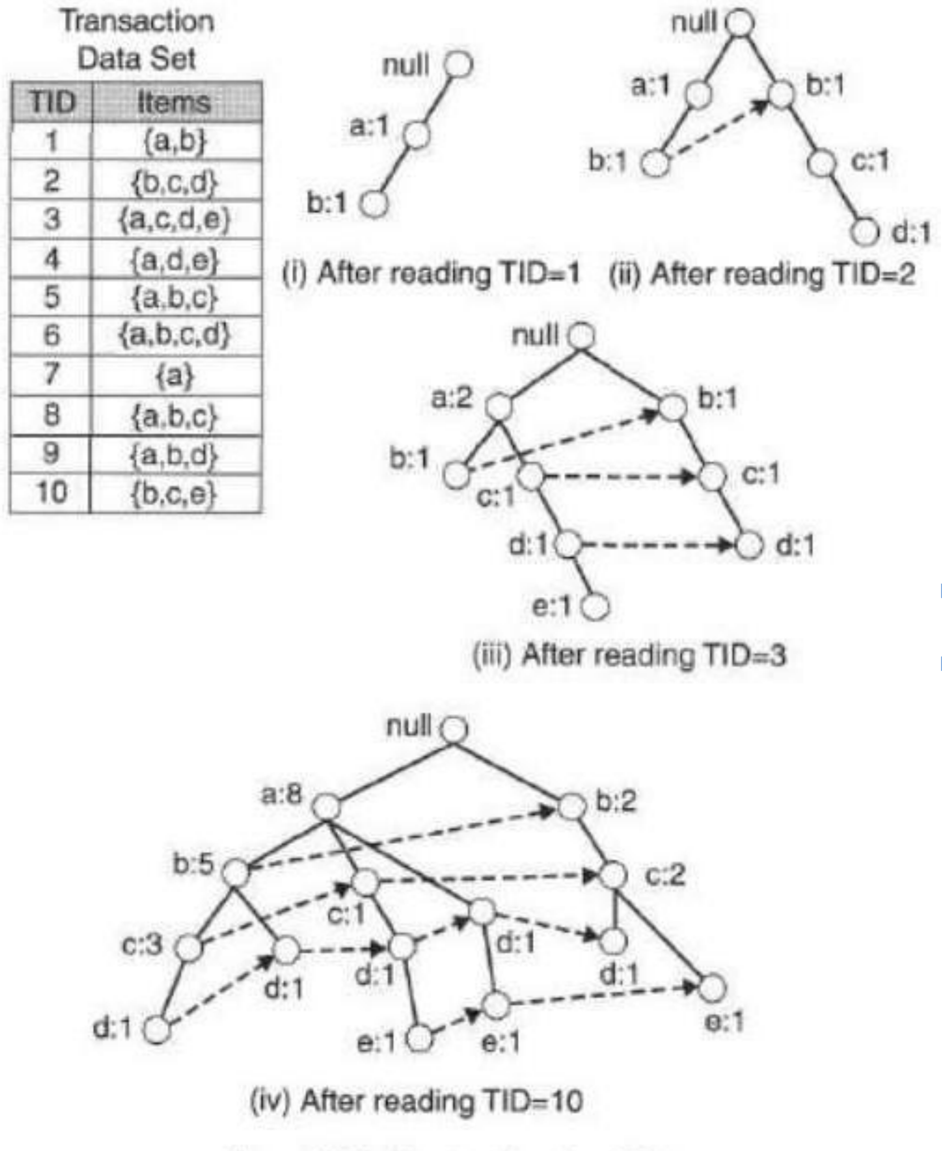
3. Pointers are maintained between nodes containing the same item, creating singly linked lists (dotted lines) – The more paths that overlap, the higher the compression. FP-tree may fit in memory.

4. Frequent itemsets extracted from the FP-Tree.

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

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 FP tree. After reading the first transaction, {a,b), the nodes labeled 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.



Transaction Data Set

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

:om

Figure 6.24. Construction of an FP-tree.

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

The third transaction, {a,c,d,e}, shares a common prefix item (which is a) 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, nuI,a ,b. Because of their overlapping path, the frequency count for node a is incremented to two, while the frequency counts for the newly created nodes, c, d, and e) are equal to one.
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 6.25.

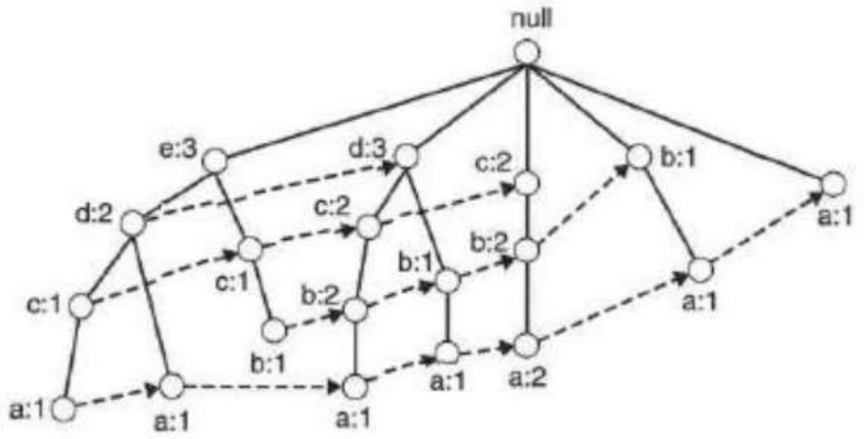


Figure 6.25. An FP-tree representation for the data set shown in Figure 6.24 with a different item ordering scheme.

Step 2: Frequent Itemset Generation
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.
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).
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.

VTUPulse.com

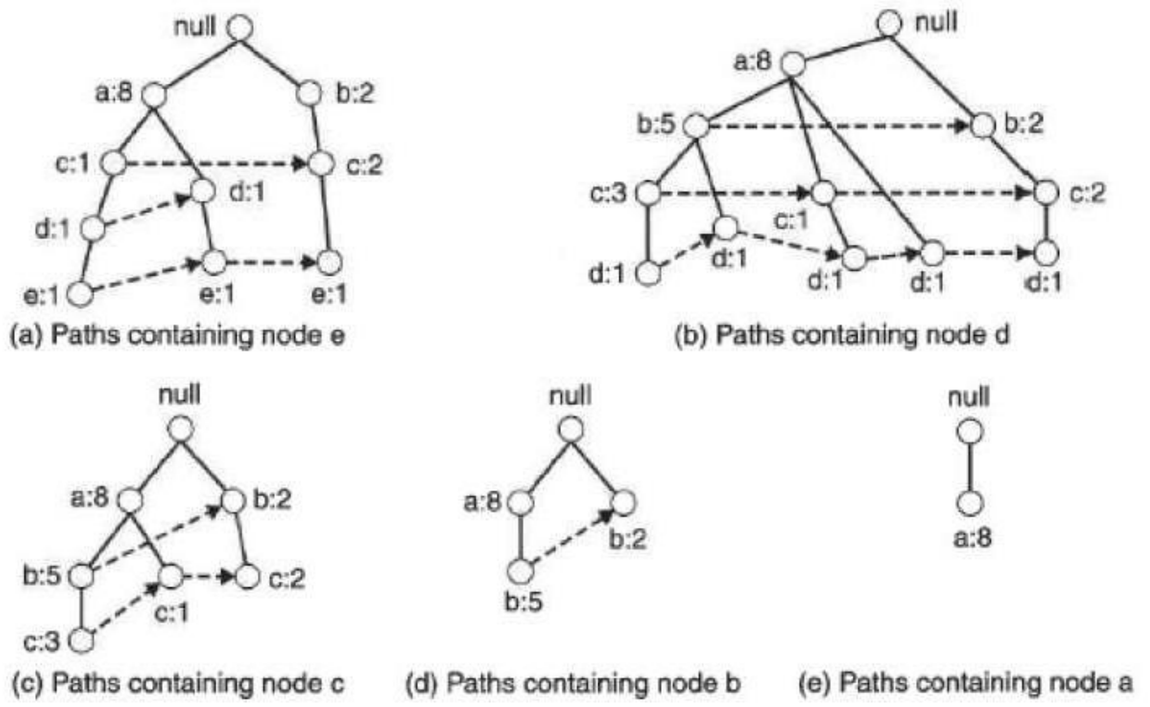
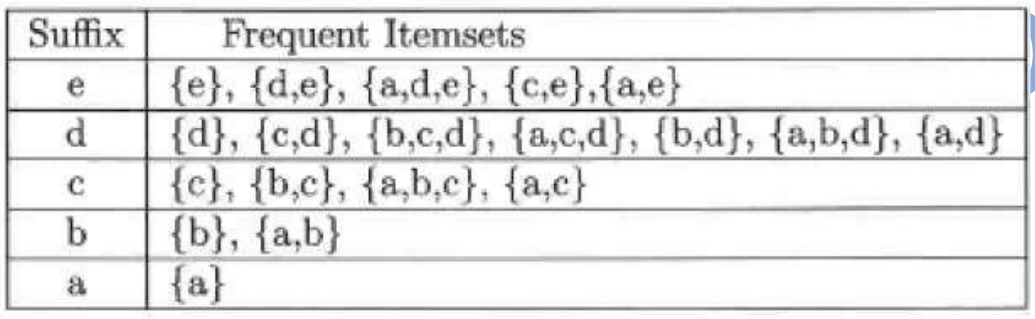

null
a:8
b:2
c:1
c:2
d:1
e:1
(a) Paths containing node e
b:5
c:3
(b) Paths containing node d
(c) Paths containing node c
(d) Paths containing node b
(e) Paths containing node a

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.

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}

om

3.7 Evaluation of Association Pattern

Association rule algorithms tend to produce too many rules
– many of them are uninteresting or redundant
– Redundant if {A,B,C} -> {D} and {A,B} -> {D} have same support & confidence
Interestingness measures can be used to prune/rank the derived patterns

Objective measures of interestingness

Given a rule X -> Y, information needed to compute rule interestingness can be obtained from a contingency table
Contingency table for X -> Y

	Y	Y	
X	f_{11}	f_{10}	f_{1+}
X	f_{01}	f_{00}	f_{o+}
	f_{+1}	f_{+0}	N

f_{11}: support of X and Y
f_{10}: support of X and Y
f_{01}: support of X and Y
f_{00}: support of X and Y

Interest Factor

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

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

.com

Example:

	Coffee	Coffee	
Tea	15	5	20
Tea	75	5	80
	90	10	100

For the tea-coffee example shown in Table 6.8, $I = \frac{0.15}{0.2\times0.8} = 0.9375$, thus suggesting a slight negative correlation between tea drinkers and coffee drinkers.

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

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

Correlation Analysis
For binary variables, correlation can be measured using the d-coefficient. which is defined as

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

The value of correlation ranges from -1 (perfect negative correlation) to +1 (perfect positive correlation). If the variables are statistically independent, then it is 0.

12. The original association rule mining formulation uses the support and confidence measures to prune uninteresting rules.

(a) Draw a contingency table for each of the following rules using the transactions shown in Table 6.4.

Rules: $\{b\} \longrightarrow \{c\}$, $\{a\} \longrightarrow \{d\}$, $\{b\} \longrightarrow \{d\}$, $\{e\} \longrightarrow \{c\}$, $\{c\} \longrightarrow \{a\}$.

Answer:

	c	$\overline{c}$
b	3	4
$\overline{b}$	2	1

	d	$\overline{d}$
a	4	1
$\overline{a}$	5	0

	d	$\overline{d}$
b	6	1
$\overline{b}$	3	0

	c	$\overline{c}$
e	2	4
$\overline{e}$	3	1

	a	$\overline{a}$
c	2	3
$\overline{c}$	3	2

(b) Use the contingency tables in part (a) to compute and rank the rules in decreasing order according to the following measures.

Table 6.4. Example of market basket transactions.

Transaction ID	Items Bought
1	$\{a, b, d, e\}$
2	$\{b, c, d\}$
3	$\{a, b, d, e\}$
4	$\{a, c, d, e\}$
5	$\{b, c, d, e\}$
6	$\{b, d, e\}$
7	$\{c, d\}$
8	$\{a, b, c\}$
9	$\{a, d, e\}$
10	$\{b, d\}$

i. Support.
Answer:

Rules	Support	Rank
$b \longrightarrow c$	0.3	3
$a \longrightarrow d$	0.4	2
$b \longrightarrow d$	0.6	1
$e \longrightarrow c$	0.2	4
$c \longrightarrow a$	0.2	4

ii. Confidence.
Answer:

Rules	Confidence	Rank
$b \longrightarrow c$	3/7	3
$a \longrightarrow d$	4/5	2
$b \longrightarrow d$	6/7	1
$e \longrightarrow c$	2/6	5
$c \longrightarrow a$	2/5	4

iii. Interest$(X \longrightarrow Y) = \frac{P(X,Y)}{P(X)}P(Y)$.
Answer:

Rules	Interest	Rank
$b \longrightarrow c$	0.214	3
$a \longrightarrow d$	0.72	2
$b \longrightarrow d$	0.771	1
$e \longrightarrow c$	0.167	5
$c \longrightarrow a$	0.2	4

ulse.com

iv. IS$(X \longrightarrow Y) = \frac{P(X,Y)}{\sqrt{P(X)P(Y)}}$.
Answer:

Rules	IS	Rank
$b \longrightarrow c$	0.507	3
$a \longrightarrow d$	0.596	2
$b \longrightarrow d$	0.756	1
$e \longrightarrow c$	0.365	5
$c \longrightarrow a$	0.4	4

17. Suppose we have market basket data consisting of 100 transactions and 20 items. If the support for item a is 25%, the support for item b is 90% and the support for itemset $\{a, b\}$ is 20%. Let the support and confidence thresholds be 10% and 60%, respectively.

    (a) Compute the confidence of the association rule $\{a\} \rightarrow \{b\}$. Is the rule interesting according to the confidence measure?

    Answer:

    Confidence is $0.2/0.25 = 80\%$. The rule is interesting because it exceeds the confidence threshold.

    (b) Compute the interest measure for the association pattern $\{a, b\}$. Describe the nature of the relationship between item a and item b in terms of the interest measure.

    Answer:

    The interest measure is $0.2/(0.25 \times 0.9) = 0.889$. The items are negatively correlated according to interest measure.

3.8 Important Questions

1. What is association analysis? Define support and confidence with an example.
2. Develop the appriori algorithm for frequent itemset generation, with an example.
3. Explain the various measure of evaluating association patterns.
4. Explain in detail frequent itemset generation and rule generation with reference to appriori along with an example.
5. Define following: a) Support b) Confidence.
6. Explain FP growth algorithm for discovering frequent item sets. What are its limitation.
7. Consider following transaction data set

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

VTUPulse.com

Construct the FP tress by showing the tress separately after reading each transaction.
8. Illustrate the limitations of support confidence framework for evaluation of an association rule

9. Define cross support pattern. Suppose the support for milk is 70%, support for sugar is 10% and support for bread is 0.04%. given h_c= 0.01. is the frequent item set {milk, sugar, bread} the cross- support pattern?

10. Which are the factors affecting the computational complexity of appriori algorithm? Explain them.

11. Define a frequent pattern tree. Discuss the method of computing a FP-Tree, with an algorithm.

12. Give an example to show that items in a strong association rule may actually be negatively corelated.

13. A database has five transactions. Let min-sup = 60% and min-conf = 80%

TID	ITEM
T1	{M, O, N, K, E, Y}
T2	{D, O, N, K, E, Y}
T3	{M, A, K, E}
T4	{M, U, C, K, Y}
T5	{C, O, O, K, I, E}

Find all frequent item sets using appriori and FP growth respectively,
14. Explain various alternative methods for generating frequent item sets.

15. A database has four transactions. Let min-sup = 40% and min-conf = 60%

TID	DATE	ITEM
T1	01/01/10	{K, A, D, B}
T2	01/01/10	{D, A, C, E, B}
T3	01/15/10	{C, A, B, E}
T4	01/22/10	{B, A, D}

vtuDse.com

Find all frequent item sets using appriori and FP growth algorithms. Compare the efficiency of two measuring process.
16. Explain various Candidate Generation and Pruning techniques.

17. Explain the various properties of objective measures.

18. Comprehend the Simpson's Paradox.

19. Illustrate the nature of Simpson's paradox for the following two-way contingency table

Buy HDTV	Buy Exercise machine		
	yes	no	
yes	99	81	180
no	54	66	120
	153	147	300

20. What is appriori algorithm? Give an example. A database has six transactions of purchase of books from a book shop as given below

TID	ITEM
T1	{ANN, CC, JC, CG}
T2	{CC, D, CG}
T3	{ANN, D, CC, TC}
T4	{ANN, CC, D, CG}
T5	{ANN, CC, D, TC, CG}
T6	{C, D, TC}

Let X= {CC, TC} and Y= {ANN, TC, CC} find confidence and support of the association rule X→Y and inverse rule Y→X

21. Consider the following transaction data set:

VTUPulse.com

TID	ITEM
T100	I_1, I_2, I_5
T200	I_2, I_4
T300	I_2, I_3
T400	I_1, I_2, I_4
T500	I_1, I_3
T600	I_2, I_3
T700	I_1, I_3
T800	I_1, I_2, I_3, I_5
T900	I_1, I_2, I_3

Construct FP Tree.Generate List of frequent item set ordered by their corresponding suffixes.

22. Consider following set of frequent 3 item sets

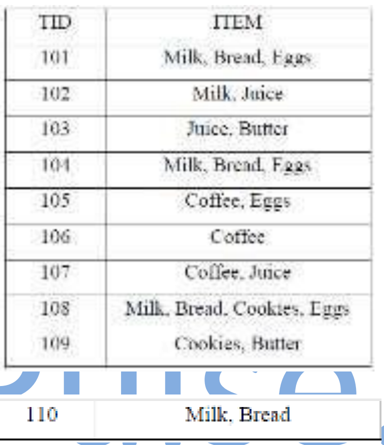
{1, 2, 3}	{1, 3, 5}
{1, 2, 4}	{2, 3, 4}
{1, 2, 5}	{2, 3, 5}
{1, 3, 4}	{3, 4, 5}

Assume that there are only 5 items in data set.

a) List all candidate 4 item sets obtained by a candidate generation procedure using F_{k-1} X F_1 merging strategy

b) List all candidate 4 item sets obtained by the candidate generation procedure in appriori,

23. Apply appriori algorithm for



TID	ITEM
101	Milk, Bread, Eggs
102	Milk, Juice
103	Juice, Butter
104	Milk, Bread, Eggs
105	Coffee, Eggs
106	Coffee
107	Coffee, Juice
108	Milk, Bread, Cookies, Eggs
109	Cookies, Butter
110	Milk, Bread

VTUPulse.com

Item set = {Milk, Bread, Eggs, Cookies, Coffee, Butter, Juice}, use 0.2 for min-sup.