Association Rule Mining

Rina BUOY

Introduction to Data Mining, 2nd Edition

Association Rule Mining

- Association rule mining is a technique used to uncover hidden relationships between variables in large datasets.
- It is a popular method in data mining and machine learning and has a wide range of applications in various fields, such as market basket analysis, customer segmentation, and fraud detection.
- The goal of association rule mining is to uncover rules that describe the relationships between different items in a large dataset.

Association Rule Mining

- Customer Segmentation: Association rule mining can also be used to segment customers based on their purchasing habits.
- Fraud Detection: You can also use association rule mining to detect fraudulent activity.
- Market Basket Analysis: A retailer might use association rule mining to discover that customers who purchase diapers are also likely to purchase baby formula.
- Recommendation systems: Association rule mining can be used to suggest items that a customer might be interested in based on their past purchases or browsing history.

Definition: Frequent Itemset

- **Itemset**
  - A collection of one or more items
    - Example: {Milk, Bread, Diaper}
  - k-itemset
    - An itemset that contains k items
- **Support count (σ)**
  - Frequency of occurrence of an itemset
  - E.g. σ({Milk, Bread,Diaper}) = 2
- **Support**
  - Fraction of transactions that contain an itemset
  - E.g. s({Milk, Bread, Diaper}) = 2/5
- **Frequent Itemset**
  - An itemset whose support is greater than or equal to a *minsup* threshold

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**

Definition: Association Rule

- Association Rule
  - An implication expression of the form X $\rightarrow$ Y, where X and Y are itemsets
  - Example:
    {Milk, Diaper} $\rightarrow$ {Beer}

- Rule Evaluation Metrics
  - Support (s)
    - Fraction of transactions that contain both X and Y
  - Confidence (c)
    - Measures how often items in Y appear in transactions that contain X

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:

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

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

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

Association Rule Mining Task

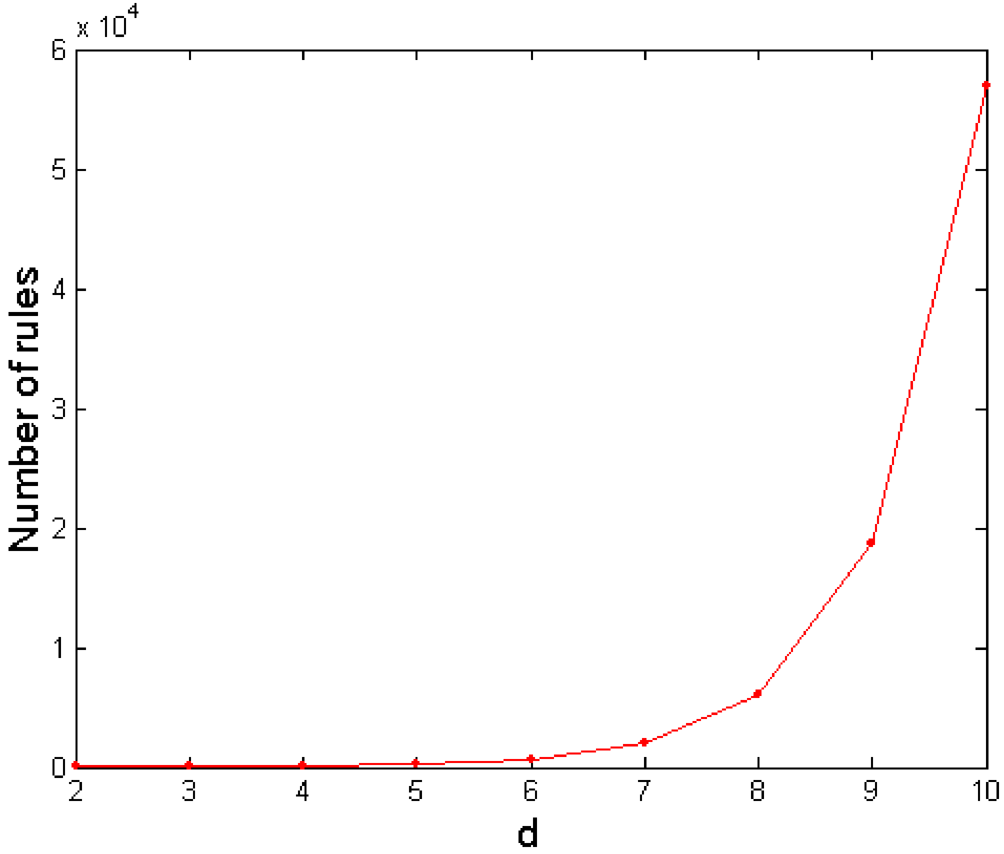
- Given a set of transactions T, the goal of association rule mining is to find all rules having
  - support ≥ *minsup* threshold
  - confidence ≥ *minconf* threshold

- 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

  $\Rightarrow$ Computationally prohibitive!

Computational Complexity

- Given d unique items:
  - Total number of itemsets = 2^d
  - Total number of possible association rules:



$$R = \sum_{k=1}^{d-1}\left[\binom{d}{k} \times \sum_{j=1}^{d-k}\binom{d-k}{j}\right]$$
$$= 3^d - 2^{d+1} + 1$$

If d=6, R = 602 rules

Mining Association Rules

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 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)

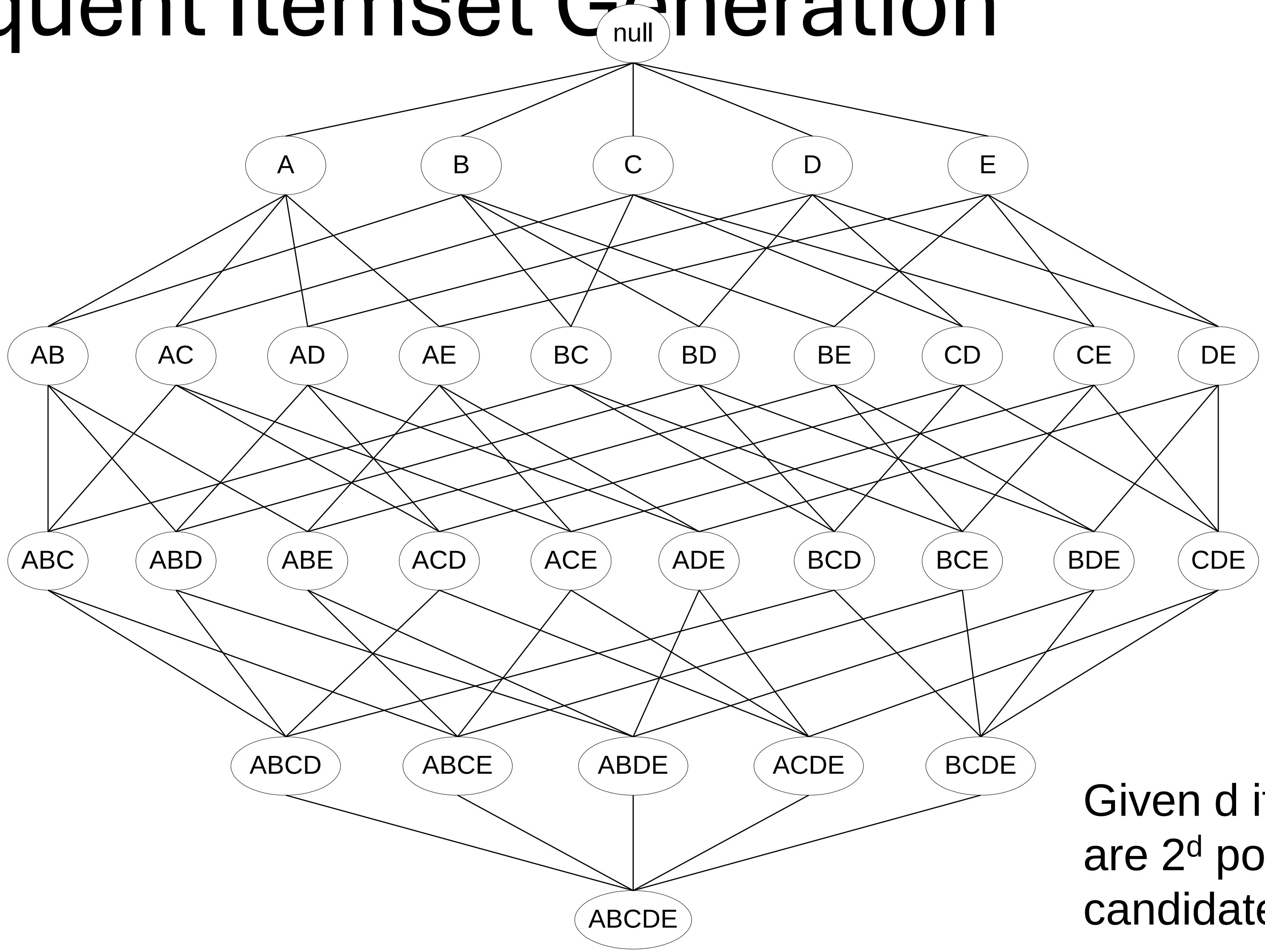
Observations:

- 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

Mining Association Rules

- Two-step approach:
  1. Frequent Itemset Generation
     - Generate all itemsets whose support $\geq$ 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

Frequent Itemset Generation



Given d items, there are 2^d possible candidate itemsets

Frequent Itemset Generation

- Brute-force approach:
  - Each itemset in the lattice is a candidate frequent itemset
  - Count the support of each candidate by scanning the database

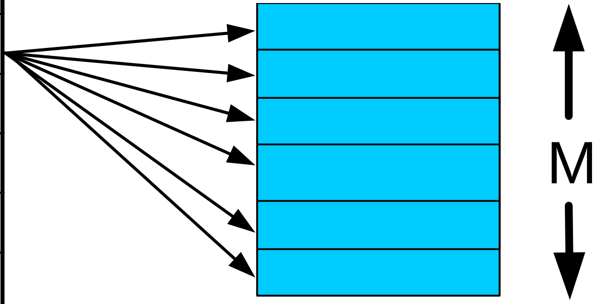
Transactions

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

N

w

List of Candidates



  - Match each transaction against every candidate
  - Complexity ~ O(NMw) => Expensive since $M = 2^d$!!!

Frequent Itemset Generation Strategies

- 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

Reducing Number of Candidates

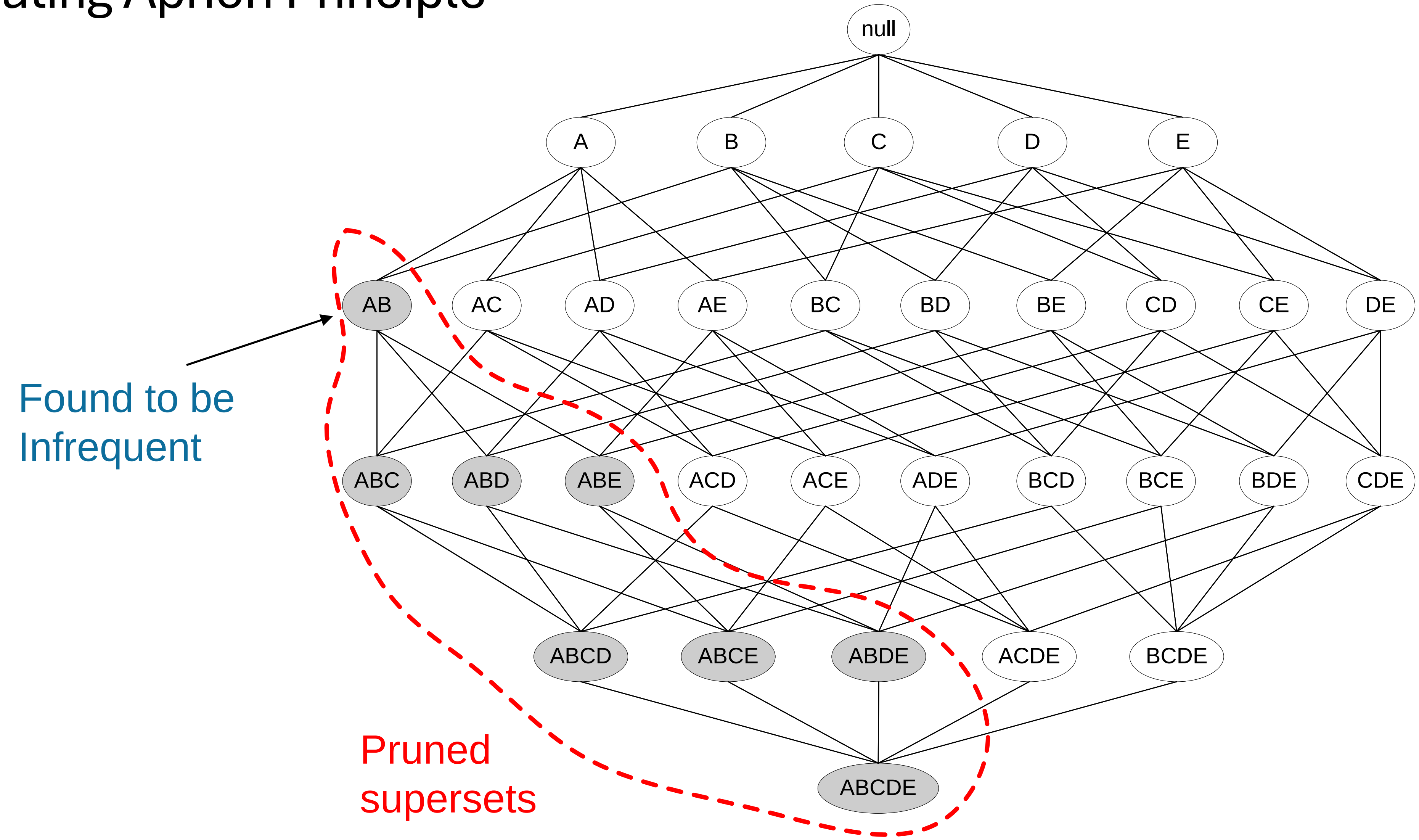
- Apriori principle:
  - If an itemset is frequent, then all of its subsets must also be frequent

- Apriori principle holds due to the following property of the support measure:

$$\forall X, Y : (X \subseteq Y) \Rightarrow s(X) \geq s(Y)$$

  - Support of an itemset never exceeds the support of its subsets
  - This is known as the anti-monotone property of support

Illustrating Apriori Principle



Illustrating Apriori Principle

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

Items (1-itemsets)

Item	Count
Bread	4
Coke	2
Milk	4
Beer	3
Diaper	4
Eggs	1

Minimum Support = 3

If every subset is considered,

$^6C_1 + {}^6C_2 + {}^6C_3$

$6 + 15 + 20 = 41$

With support-based pruning,

$6 + 6 + 4 = 16$

Illustrating Apriori Principle

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

Items (1-itemsets)

Item	Count
Bread	**4**
Coke	**2**
Milk	**4**
Beer	**3**
Diaper	**4**
Eggs	**1**

Minimum Support = 3

If every subset is considered,

$^6C_1 + {}^6C_2 + {}^6C_3$

$6 + 15 + 20 = 41$

With support-based pruning,

$6 + 6 + 4 = 16$

Illustrating Apriori Principle

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

Item	Count
Bread	**4**
Coke	**2**
Milk	**4**
Beer	**3**
Diaper	**4**
Eggs	**1**

Items (1-itemsets)

Itemset
{Bread,Milk}
{Bread, Beer }
{Bread,Diaper}
{Beer, Milk}
{Diaper, Milk}
{Beer,Diaper}

Pairs (2-itemsets)

(No need to generate candidates involving Coke or Eggs)

Minimum Support = 3

If every subset is considered,

$^6C_1 + {}^6C_2 + {}^6C_3$

$6 + 15 + 20 = 41$

With support-based pruning,

$6 + 6 + 4 = 16$

Illustrating Apriori Principle

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

Items (1-itemsets)

Item	Count
Bread	**4**
Coke	**2**
Milk	**4**
Beer	**3**
Diaper	**4**
Eggs	**1**

Pairs (2-itemsets)

Itemset	Count
{Bread,Milk}	**3**
{Beer, Bread}	**2**
{Bread,Diaper}	**3**
{Beer,Milk}	**2**
{Diaper,Milk}	**3**
{Beer,Diaper}	**3**

(No need to generate candidates involving Coke or Eggs)

Minimum Support = 3

If every subset is considered,

$^6C_1 + {}^6C_2 + {}^6C_3$

$6 + 15 + 20 = 41$

With support-based pruning,

$6 + 6 + 4 = 16$

Illustrating Apriori Principle

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

Items (1-itemsets)

Item	Count
Bread	**4**
Coke	**2**
Milk	**4**
Beer	**3**
Diaper	**4**
Eggs	**1**

Pairs (2-itemsets)

(No need to generate candidates involving Coke or Eggs)

Itemset	Count
{Bread,Milk}	**3**
{Bread,Beer}	**2**
{Bread,Diaper}	**3**
{Milk,Beer}	**2**
{Milk,Diaper}	**3**
{Beer,Diaper}	**3**

Minimum Support = 3

Triplets (3-itemsets)

Itemset
{ Beer, Diaper, Milk}
{ Beer,Bread,Diaper}
{Bread,Diaper,Milk}
{ Beer, Bread, Milk}

If every subset is considered,

$$^6C_1 + {}^6C_2 + {}^6C_3$$

$$6 + 15 + 20 = 41$$

With support-based pruning,

$$6 + 6 + 4 = 16$$

Illustrating Apriori Principle

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

Items (1-itemsets)

Item	Count
Bread	4
Coke	2
Milk	4
Beer	3
Diaper	4
Eggs	1

Pairs (2-itemsets)

(No need to generate candidates involving Coke or Eggs)

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

Minimum Support = 3

Triplets (3-itemsets)

Itemset	Count
{ Beer, Diaper, Milk}	2
{ Beer,Bread, Diaper}	2
{Bread, Diaper, Milk}	2
{Beer, Bread, Milk}	1

If every subset is considered,

$^6C_1 + {}^6C_2 + {}^6C_3$

$6 + 15 + 20 = 41$

With support-based pruning,

$6 + 6 + 4 = 16$

Illustrating Apriori Principle

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

Items (1-itemsets)

Item	Count
Bread	4
Coke	2
Milk	4
Beer	3
Diaper	4
Eggs	1

Pairs (2-itemsets)

(No need to generate candidates involving Coke or Eggs)

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

Minimum Support = 3

Triplets (3-itemsets)

Itemset	Count
{ Beer, Diaper, Milk}	2
{ Beer,Bread, Diaper}	2
{Bread, Diaper, Milk}	2
{Beer, Bread, Milk}	1

If every subset is considered,

$^6C_1 + {}^6C_2 + {}^6C_3$

$6 + 15 + 20 = 41$

With support-based pruning,

$6 + 6 + 4 = 16$

$6 + 6 + 1 = 13$

Apriori Algorithm

- F_k: frequent k-itemsets
- L_k: candidate k-itemsets

- Algorithm
  - Let k=1
  - Generate F_1 = {frequent 1-itemsets}
  - Repeat until F_k is empty
    - **Candidate Generation**: Generate L_{k+1} from F_k
    - **Candidate Pruning**: Prune candidate itemsets in L_{k+1} containing subsets of length k that are infrequent
    - **Support Counting**: Count the support of each candidate in L_{k+1} by scanning the DB
    - **Candidate Elimination**: Eliminate candidates in L_{k+1} that are infrequent, leaving only those that are frequent => F_{k+1}

Candidate Generation: Brute-force method

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

Items

Item
Beer
Bread
Cola
Diapers
Eggs
Milk

Candidate Generation

Itemset
{Beer, Bread, Cola}
{Beer, Bread, Diapers}
{Beer, Bread, Eggs}
{Beer, Bread, Milk}
{Beer, Cola, Diapers}
{Beer, Cola, Eggs}
{Beer, Cola, Milk}
{Beer, Diapers, Eggs}
{Beer, Diapers, Milk}
{Beer, Eggs, Milk}
{Bread, Cola, Diapers}
{Bread, Cola, Eggs}
{Bread, Cola, Milk}
{Bread, Diapers, Eggs}
{Bread, Diapers, Milk}
{Bread, Eggs, Milk}
{Cola, Diapers, Eggs}
{Cola, Diapers, Milk}
{Cola, Eggs, Milk}
{Diapers, Eggs, Milk}

Candidate Pruning

Itemset
{Bread, Diapers, Milk}

Frequent 2-itemset

Itemset
{Beer, Diapers}
{Bread, Diapers}
{Bread, Milk}
{Diapers, Milk}

Figure 5.6. A brute-force method for generating candidate 3-itemsets.

Candidate Generation: Merge Fk-1 and F1 itemsets

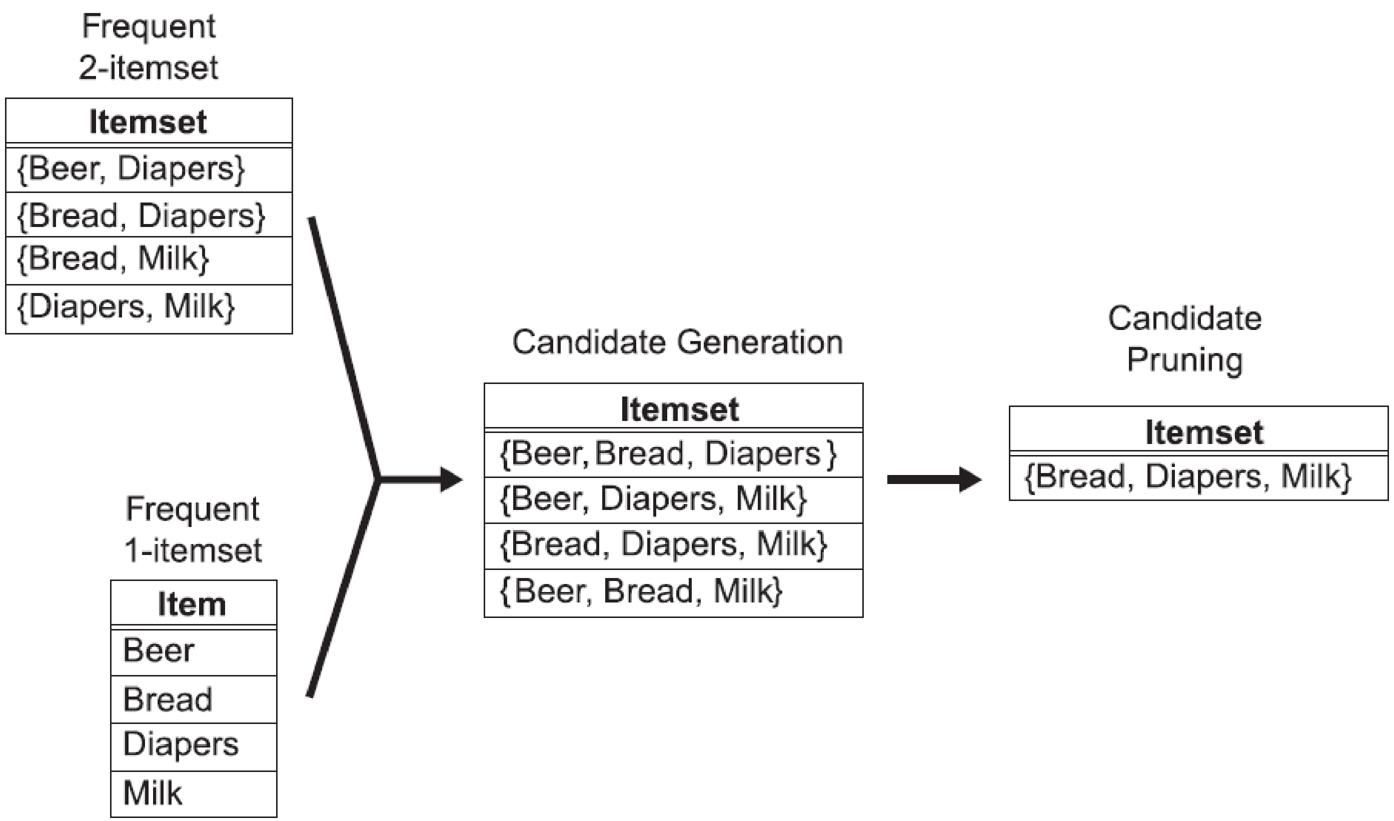


Figure 5.7. Generating and pruning candidate k-itemsets by merging a frequent $(k-1)$-itemset with a frequent item. Note that some of the candidates are unnecessary because their subsets are infrequent.

Candidate Generation: F_{k-1} x F_{k-1} Method

- Merge two frequent (k-1)-itemsets if their first (k-2) items are identical

- F_3 = {ABC,ABD,ABE,ACD,BCD,BDE,CDE}
  - Merge(**AB**C, **AB**D) = **AB**CD
  - Merge(**AB**C, **AB**E) = **AB**CE
  - Merge(**AB**D, **AB**E) = **AB**DE

  - Do not merge(**A**BD,**A**CD) because they share only prefix of length 1 instead of length 2

Candidate Pruning

- Let F_3 = {ABC,ABD,ABE,ACD,BCD,BDE,CDE} be the set of frequent 3-itemsets

- L_4 = {ABCD,ABCE,ABDE} is the set of candidate 4-itemsets generated (from previous slide)

- Candidate pruning
  - Prune ABCE because ACE and BCE are infrequent
  - Prune ABDE because ADE is infrequent

- After candidate pruning: L_4 = {ABCD}

Candidate Generation: Fk-1 x Fk-1 Method

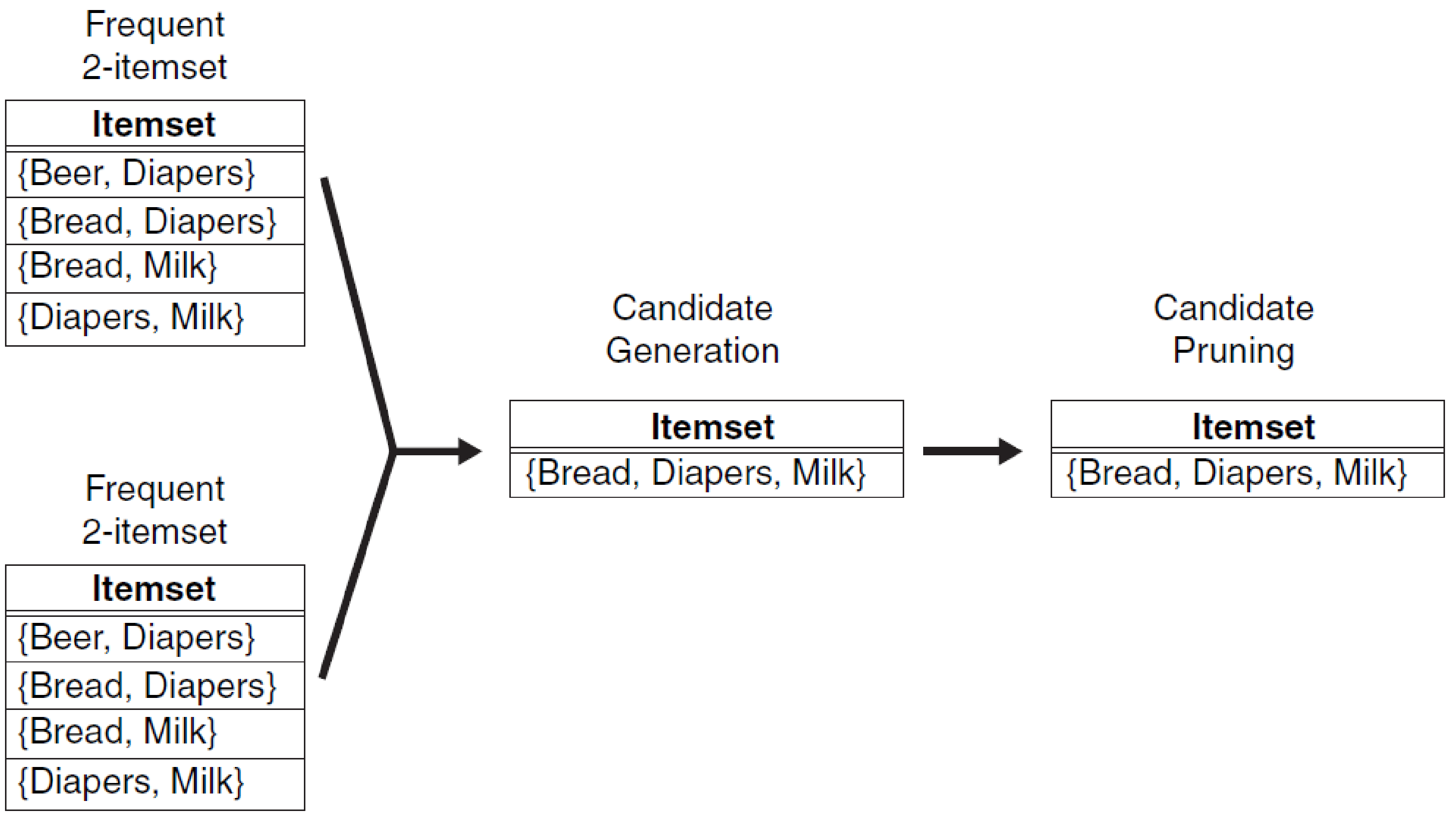


Figure 5.8. Generating and pruning candidate k-itemsets by merging pairs of frequent $(k-1)$-itemsets.

Illustrating Apriori Principle

Item	Count
Bread	4
Coke	2
Milk	4
Beer	3
Diaper	4
Eggs	1

Items (1-itemsets)

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

Pairs (2-itemsets)

(No need to generate candidates involving Coke or Eggs)

Minimum Support = 3

Triplets (3-itemsets)

Itemset	Count
{Bread, Diaper, Milk}	2

If every subset is considered,

$^6C_1 + {}^6C_2 + {}^6C_3$

$6 + 15 + 20 = 41$

With support-based pruning,

$6 + 6 + 1 = 13$

Use of $F_{k-1}xF_{k-1}$ method for candidate generation results in only one 3-itemset. This is eliminated after the support counting step.

Alternate F_{k-1} x F_{k-1} Method

- Merge two frequent (k-1)-itemsets if the last (k-2) items of the first one is identical to the first (k-2) items of the second.

- F_3 = {ABC,ABD,ABE,ACD,BCD,BDE,CDE}
  - Merge(A<u>**BC**</u>, <u>**BC**</u>D) = A<u>**BC**</u>D
  - Merge(A<u>**BD**</u>, <u>**BD**</u>E) = A<u>**BD**</u>E
  - Merge(A<u>**CD**</u>, <u>**CD**</u>E) = A<u>**CD**</u>E
  - Merge(B<u>**CD**</u>, <u>**CD**</u>E) = B<u>**CD**</u>E

Candidate Pruning for Alternate F_{k-1} x F_{k-1} Method

- Let F_3 = {ABC,ABD,ABE,ACD,BCD,BDE,CDE} be the set of frequent 3-itemsets

- L_4 = {ABCD,ABDE,ACDE,BCDE} is the set of candidate 4-itemsets generated (from previous slide)
- Candidate pruning
  - Prune ABDE because ADE is infrequent
  - Prune ACDE because ACE and ADE are infrequent
  - Prune BCDE because BCE
- After candidate pruning: L_4 = {ABCD}

Support Counting of Candidate Itemsets

- Scan the database of transactions to determine the support of each candidate itemset
  - Must match every candidate itemset against every transaction, which is an expensive operation

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

Itemset
{ Beer, Diaper, Milk}
{ Beer,Bread,Diaper}
{Bread, Diaper, Milk}
{ Beer, Bread, Milk}

Support Counting of Candidate Itemsets

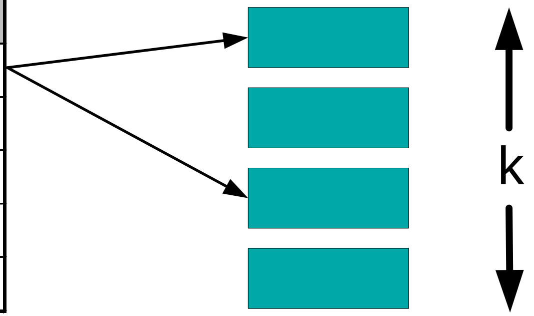
- To reduce number of comparisons, store the candidate itemsets in a hash structure
  - Instead of matching each transaction against every candidate, match it against candidates contained in the hashed buckets

Transactions

N

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

Hash Structure



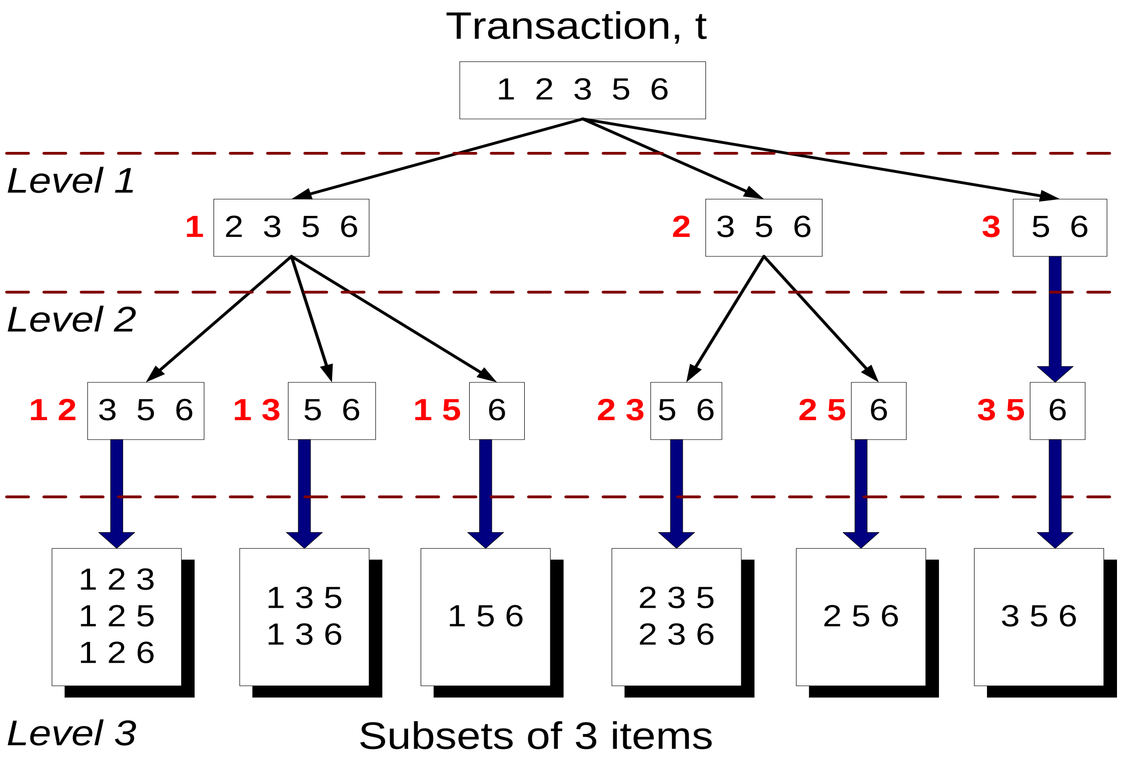
Buckets

Support Counting: An Example

Suppose you have 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}

How many of these itemsets are supported by transaction (1,2,3,5,6)?



Support Counting Using a Hash Tree

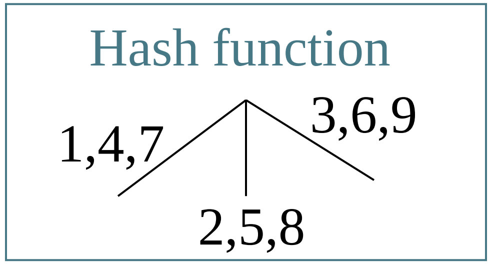
Suppose you have 15 candidate itemsets of length 3:

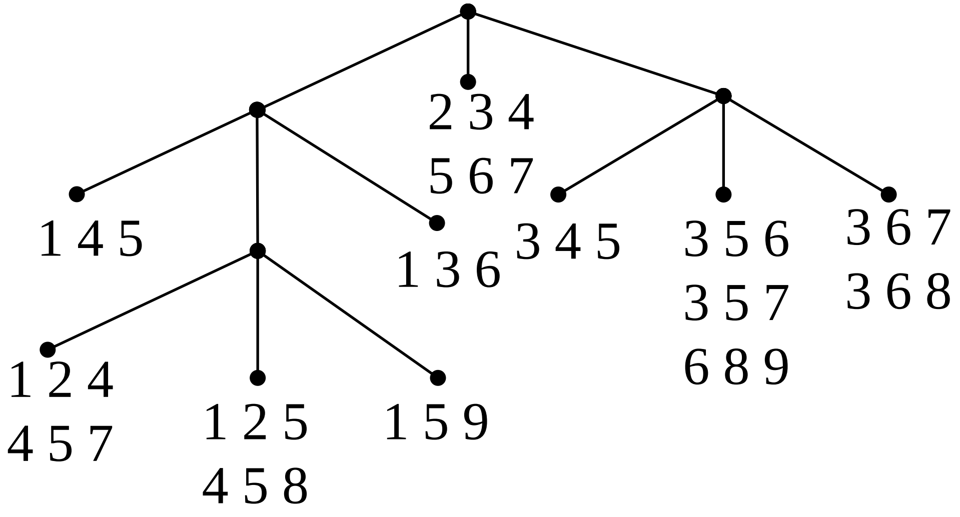
{1 4 5}, {1 2 4}, {4 5 7}, {1 2 5}, {4 5 8}, {1 5 9}, {1 3 6}, {2 3 4}, {5 6 7}, {3 4 5}, {3 5 6}, {3 5 7}, {6 8 9}, {3 6 7}, {3 6 8}

You need:

• Hash function

• Max leaf size: max number of itemsets stored in a leaf node (if number of candidate itemsets exceeds max leaf size, split the node)





Rule Generation

- Given a frequent itemset L, find all non-empty subsets $f \subset L$ such that $f \rightarrow L - f$ 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 $L \rightarrow \emptyset$ and $\emptyset \rightarrow L$)

Rule Generation

- In general, confidence does not have an anti-monotone property

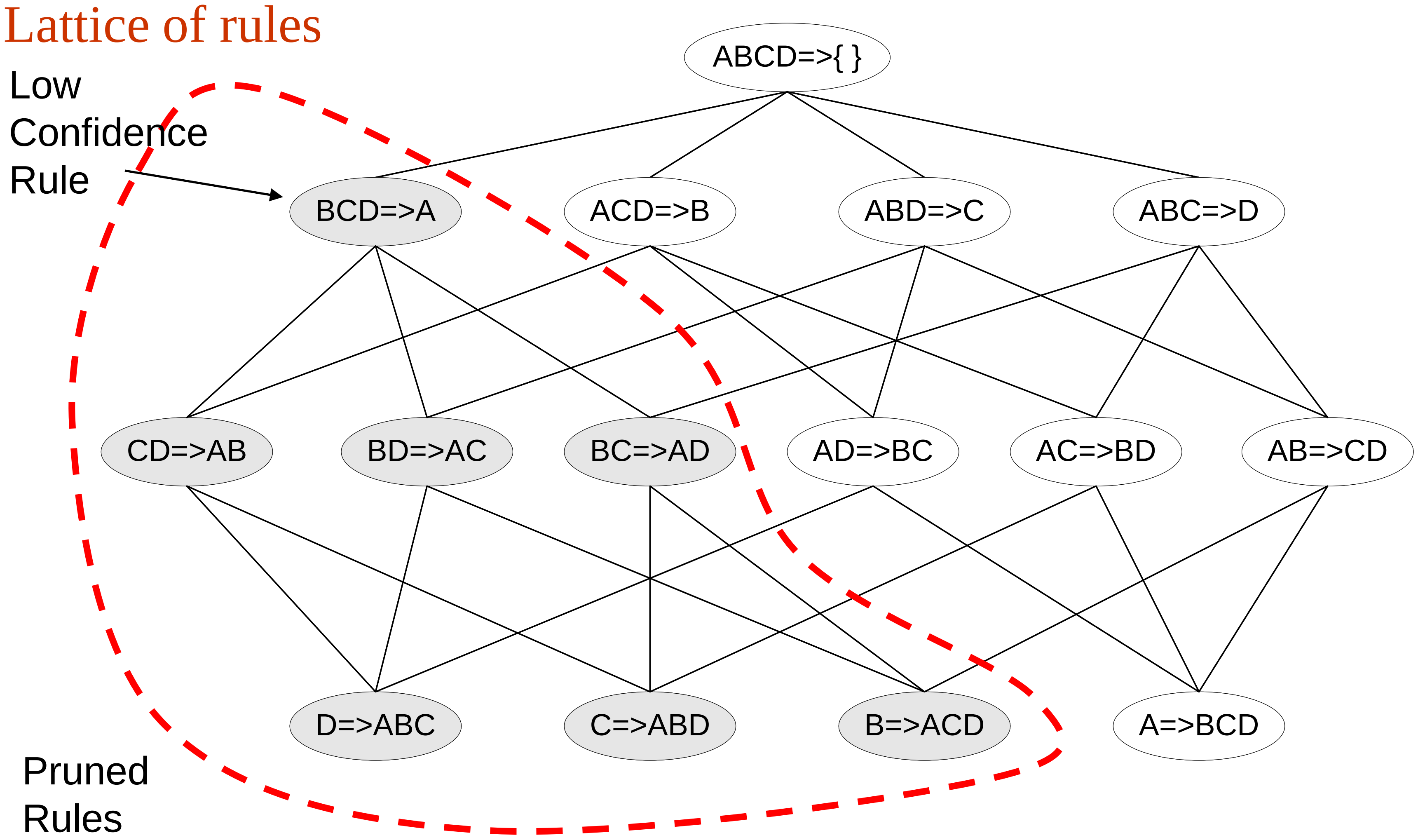
  $c(ABC \rightarrow D)$ can be larger or smaller than $c(AB \rightarrow D)$

- But confidence of rules generated from the same itemset has an anti-monotone property
  - E.g., Suppose {A,B,C,D} is a frequent 4-itemset:

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

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

Rule Generation for Apriori Algorithm



Practice