Data Mining & Knowledge Discovery

Association Analysis - 01

Md. Biplob Hosen

Lecturer, IIT-JU

Email: biplob.hosen@juniv.edu

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

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},

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 $\min_{sup}$ 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**

Association Rules

- An association rule is an implication expression of the form X → 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.
- Example: {Milk, Diaper} → {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.

Continue…

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**

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

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

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

Support & Confidence

Why Use Support?

- Support is an important measure because a rule that has very low support might occur simply by chance.
- Also, from a business perspective a low support rule is unlikely to be interesting because it might not be profitable to promote items that customers seldom buy together.
- For these reasons, we are interested in finding rules whose support is greater than some user-defined threshold.

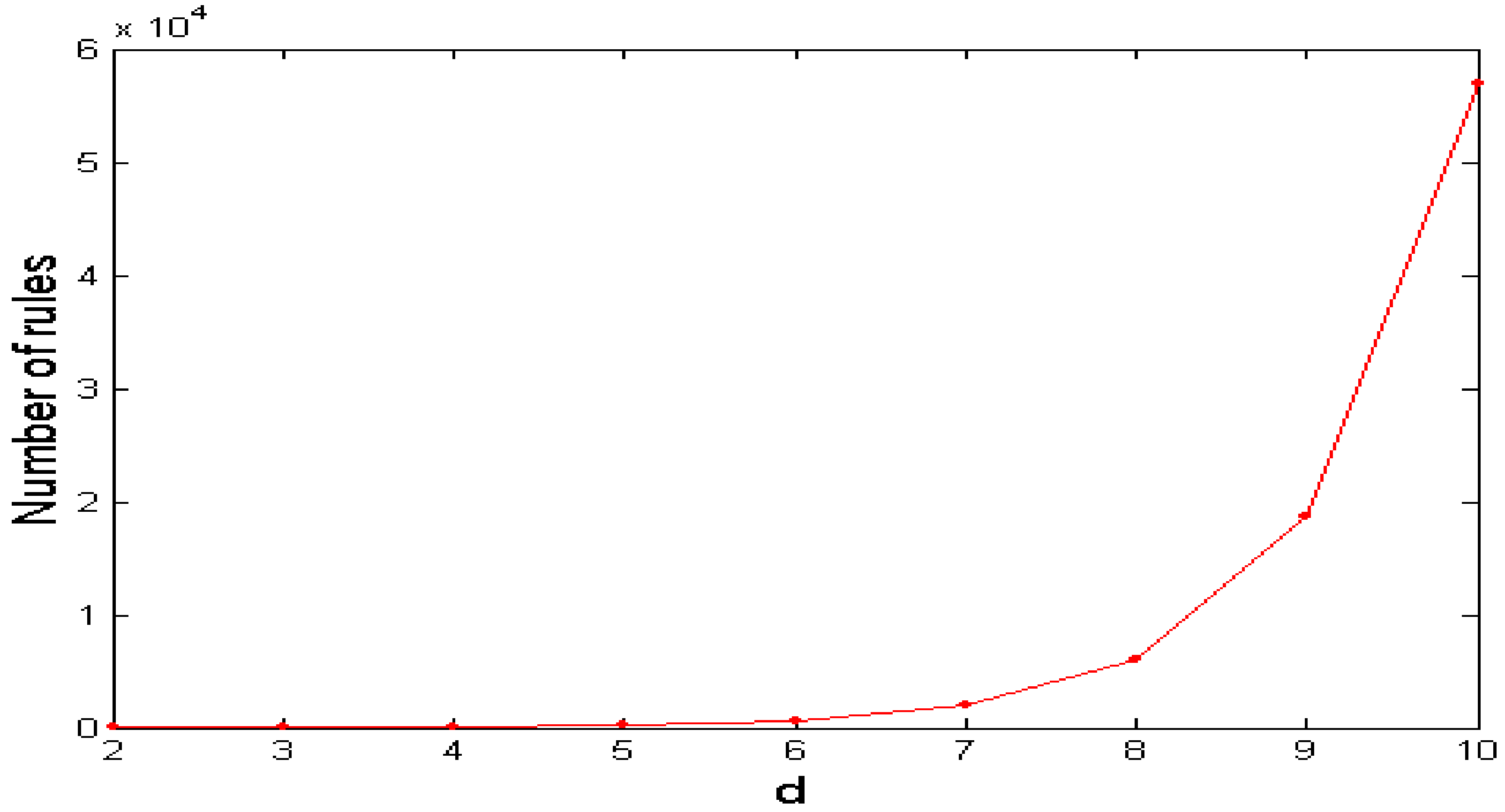
Why Use Confidence?

- Confidence, on the other hand, measures the reliability of the inference made by a rule.
- For a given rule $X \rightarrow Y$, the higher the confidence, the more likely it is for Y to be present in transactions that contain X.
- Confidence also provides an estimate of the conditional probability of Y given X.

Association Rule Discovery

- Given a set of transactions T, find all the rules having support ≥ min_{sup} and confidence ≥ min_{conf}, where, min_{sup} and min_{conf} 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.
- More specifically, assuming that neither the left nor the righthand side of the rule is an empty set, the total number of possible rules, R, extracted from a data set that contains d items is: $R = 3^d - 2^{d+1} + 1$.

Computational Complexity: Brute-force Approach



Mining Association Rules - Observations

- Even for the small data set shown the table, this approach requires us to compute the support and confidence for $3^6 - 2^7 + 1 = 602$ rules.
- More than 80% of the rules are discarded after applying $min_{sup} = 20\%$ and $min_{conf} = 50\%$, thus wasting most of the computations.
- 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.
- The support of a rule $X \rightarrow Y$ is the same as the support of its corresponding itemset, $X \cup Y$.

Mining Association Rules - Observations

- For example, the following rules have identical support because they involve items from the same itemset, **{Beer, Diaper, Milk}.**

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:

1. {Milk, Diaper} $\rightarrow$ {Beer} (s=0.4, c=0.67)
2. {Milk, Beer} $\rightarrow$ {Diaper} (s=0.4, c=1.0)
3. {Diaper, Beer} $\rightarrow$ {Milk} (s=0.4, c=0.67)
4. {Beer} $\rightarrow$ {Milk, Diaper} (s=0.4, c=0.67)
5. {Diaper} $\rightarrow$ {Milk, Beer} (s=0.4, c=0.5)
6. {Milk} $\rightarrow$ {Diaper, Beer} (s=0.4, c=0.5)

Mining Association Rules - Observations

Problems:

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

Solution:

- Therefore, a common strategy adopted by many association rule mining algorithms is to decompose the problem into two major subtasks by decoupling the support and confidence requirements.

Mining Association Rules

Two-step approach:

1. **Frequent Itemset Generation:**

- Generate all itemsets whose support $\geq$ min_{sup}.

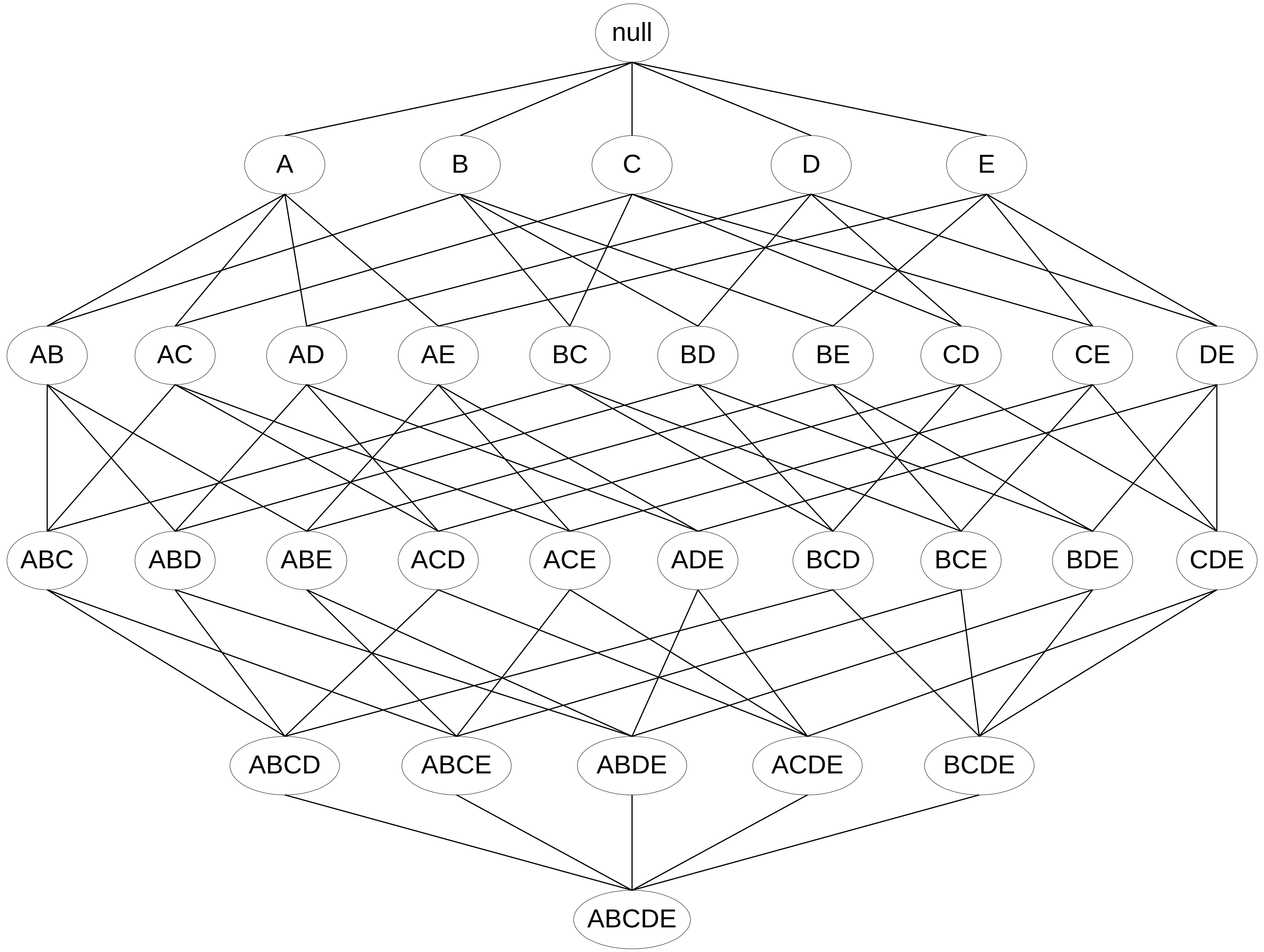
2. **Rule Generation:**

- 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 than those of rule generation.
- There are efficient ways to generate frequent itemsets.

Frequent Itemset Generation

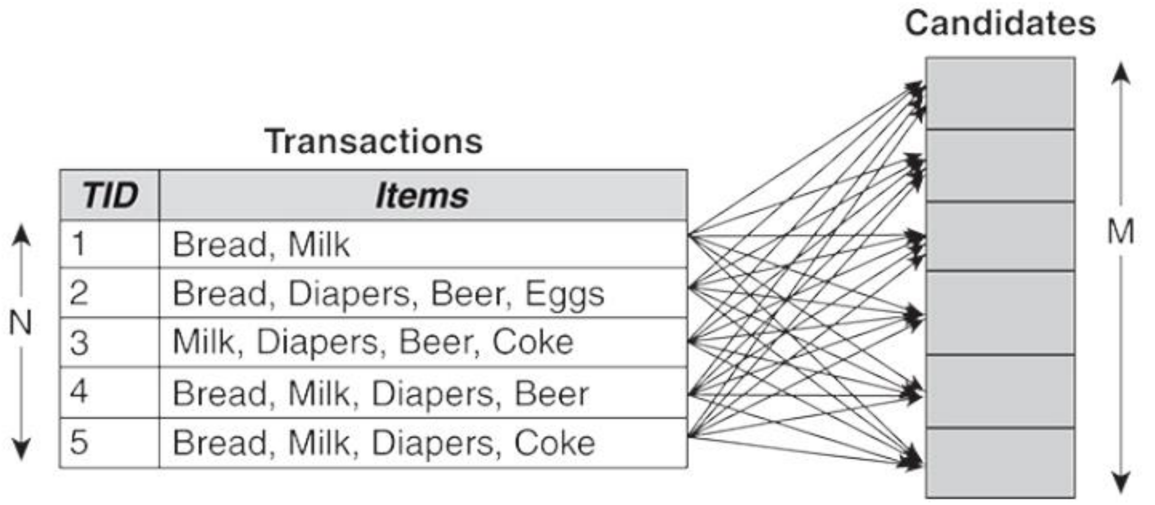
- A lattice structure can be used to enumerate the list of all possible itemsets.
- Given k items, there are 2^{k-1} possible frequent itemsets.



Continue…

Brute-force approach:

- Each itemset in the lattice is a candidate frequent itemset.



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

- Count the support of each candidate by scanning the database.
- If the candidate is contained in a transaction, its support count will be incremented. For example, the support for {Bread, Milk} is incremented three times because the itemset is contained in transactions 1, 4, and 5.
- Such an 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. Transaction width is the number of items present in a transaction.

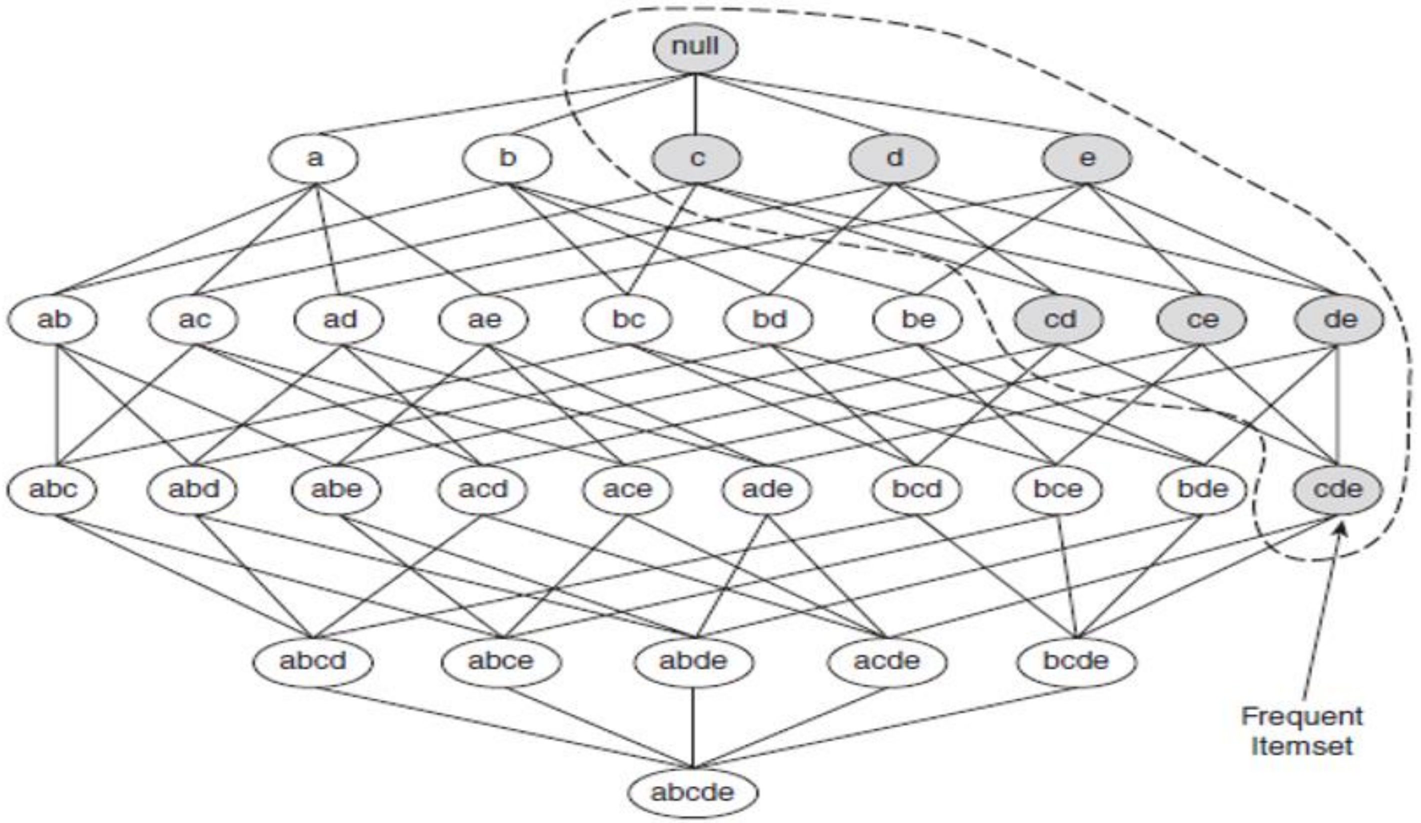
Frequent Itemset Generation Strategies

- There are three main approaches for reducing the computational complexity of frequent itemset generation.

1. **Reduce the number of candidates (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 (NM):** 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.
3. **Reduce the number of transactions (N):** As the size of candidate itemsets increases, fewer transactions will be supported by the itemsets. For instance, since the width of the first transaction in the above table is 2, it would be advantageous to remove this transaction before searching for frequent itemsets of size 3 and larger.

The Apriori Principle

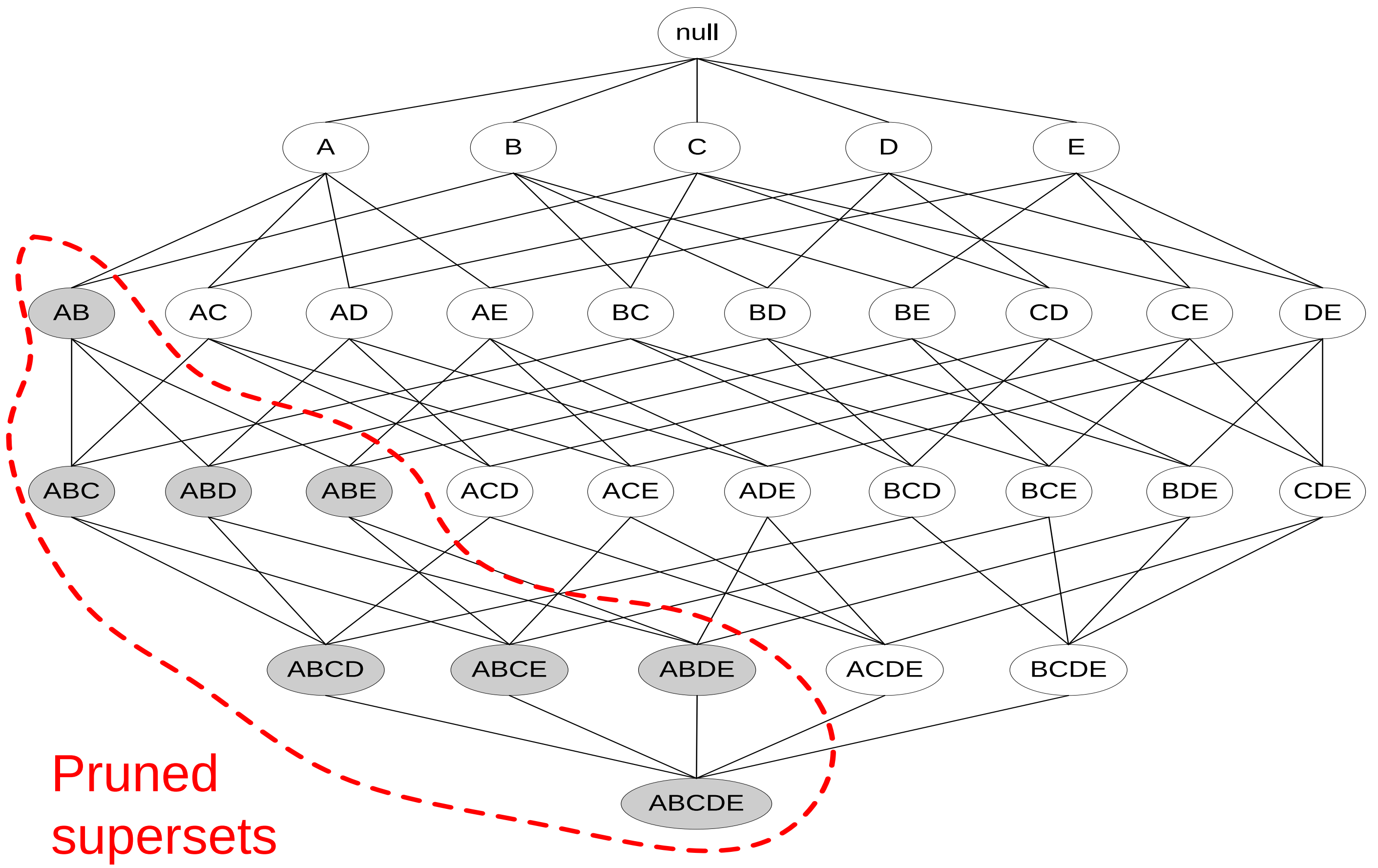
- The support measure can be used to reduce the number of candidate itemsets explored during frequent itemset generation.
- **Apriori principle:** If an itemset is frequent, then all of its subsets must also be frequent.



- To illustrate the idea behind the Apriori principle, consider the itemset lattice. 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} must also be frequent.

The Apriori Principle

- Conversely, if an itemset such as {a, b} is infrequent, then all of its supersets must be infrequent too.
- As illustrated in the figure, the entire subgraph containing the supersets of {a, b} can be pruned immediately once {a, b} is found to be infrequent.

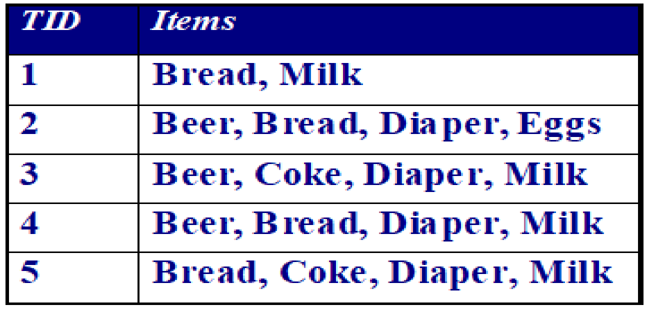


- This strategy of trimming the exponential search space based on the support measure is known as **support-based pruning**. Such a pruning strategy is made possible by a key property of the support measure, namely, that the support for an itemset never exceeds the support for its subsets. This property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

Frequent Itemset Generation in the Apriori Algorithm

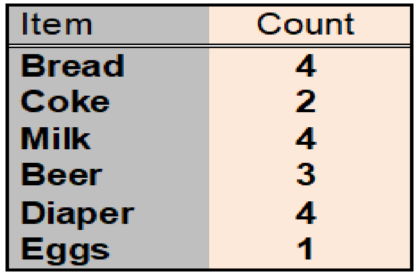
- We assume that the support threshold is 60%, which is equivalent to a minimum support count equal to 3.

Minimum Support = 3



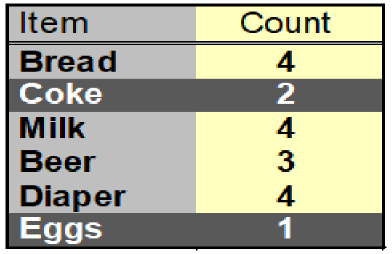
TID	*Items*
1	Bread, Milk
2	Beer, Bread, Diaper, Eggs
3	Beer, Coke, Diaper, Milk
4	Beer, Bread, Diaper, Milk
5	Bread, Coke, Diaper, Milk

Items (1-itemsets)

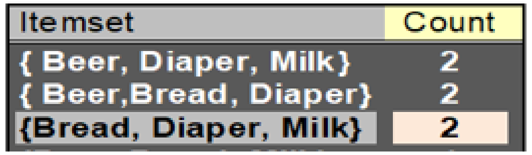


Item	Count
Bread	4
Coke	2
Milk	4
Beer	3
Diaper	4
Eggs	1

Items (1-itemsets)

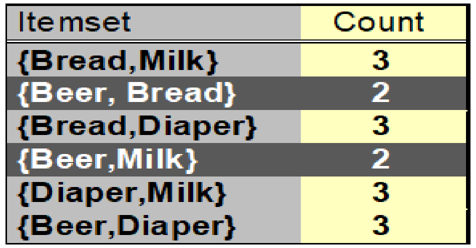


Item	Count
Bread	4
Coke	2
Milk	4
Beer	3
Diaper	4
Eggs	1



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

Triplets (3-itemsets)



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

Pairs (2-itemsets)

(No need to generate candidates involving Coke or Eggs)

Apriori Algorithm

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

Algorithm:

1. Let k=1.
2. Generate F_1 = {frequent 1-itemsets}.
3. 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 => $F_{k+1.}$

Practice

- Apply a priori algorithm for the following data sets for minimum support is 60%.

- **min$_{\textbf{support}}$** = 5*0.6 = 3

T_id	**Itemset**
T1	6, 7, 8, 5, 4, 10
T2	3, 8, 7, 5, 4, 10
T3	6, 1, 5, 4
T4	6, 9, 2, 5, 10
T5	2, 8, 5, 4

Solution

- **Table**

min$_{support}$ = 3

T_id	Itemset
T1	6, 7, 8, 5, 4, 10
T2	3, 8, 7, 5, 4, 10
T3	6, 1, 5, 4
T4	6, 9, 2, 5, 10
T5	2, 8, 8, 5, 4

L1

Item	Support
{1}	1
{2}	2
{3}	1
{4}	4
{5}	5
{6}	3
{7}	2
{8}	4
{9}	1
{10}	3

L2

Item	Support
{4,5}	4
{4,6}	2
{4,8}	3
{4,10}	2
{5,6}	3
{5,8}	3
{5,10}	3
{6,8}	1
{6,10}	2
{8,10}	2

L3

Item	Support
{4,5,8}	3
{4,5,6}	2
{4,5,10}	2
{5,6,8}	1
{5,6,10}	2
{5,8,10}	2

Thank You!