POLITECNICO DI MILANO

Association Rules

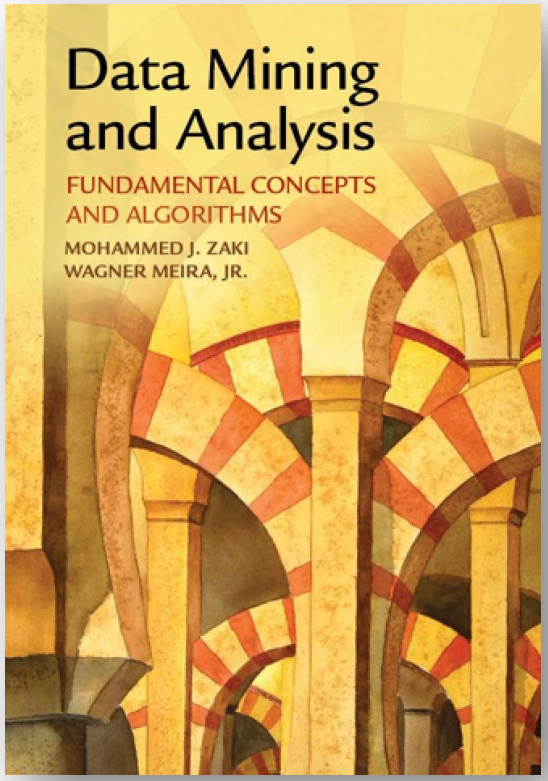
Data Mining and Text Mining (UIC 583 @ Politecnico di Milano)

Prof. Pier Luca Lanzi

Readings

- "Data Mining and Analysis" by Zaki & Meira
  - Chapter 8
  - Section 9.1
  - Chapter 10 up to Section 10.2.1 included
  - Section 12.1

- http://www.dataminingbook.info




GranDessert
Panna

RICARD
ANGOSTURA
aromatic bitters
ANGOSTURA
aromatic bitters
Since 1824
BREEZER
7,99
5,98
4,11

Market-Basket Transactions

Bread Peanuts Milk Fruit Jam	Bread Jam Soda Chips Milk Fruit	Steak Jam Soda Chips Bread	Jam Soda Peanuts Milk Fruit
Jam Soda Chips Milk Bread	Fruit Soda Chips Milk	Fruit Soda Peanuts Milk	Fruit Peanuts Cheese Yogurt

Association Rules

What is Association Rule Mining?

- Finding frequent patterns, associations, correlations, or causal structures among sets of items or objects in transaction databases, relational databases, and other information repositories

- Applications
  - Basket data analysis
  - Cross-marketing
  - Catalog design
  - …

What is 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

- Examples
  - {bread} $\Rightarrow$ {milk}
  - {soda} $\Rightarrow$ {chips}
  - {bread} $\Rightarrow$ {jam}

TID	Items
1	Bread, Peanuts, Milk, Fruit, Jam
2	Bread, Jam, Soda, Chips, Milk, Fruit
3	Steak, Jam, Soda, Chips, Bread
4	Jam, Soda, Peanuts, Milk, Fruit
5	Jam, Soda, Chips, Milk, Bread
6	Fruit, Soda, Chips, Milk
7	Fruit, Soda, Peanuts, Milk
8	Fruit, Peanuts, Cheese, Yogurt

Frequent Itemset

- Itemset
  - A collection of one or more items, e.g., {milk, bread, jam}
  - k-itemset, an itemset that contains k items
- Support count (σ)
  - Frequency of occurrence of an itemset
  - σ({Milk, Bread}) = 3
    σ({Soda, Chips}) = 4
- Support
  - Fraction of transactions that contain an itemset
  - s({Milk, Bread}) = 3/8
    s({Soda, Chips}) = 4/8
- Frequent Itemset
  - An itemset whose support is greater than or equal to a minsup threshold

TID	Items
1	Bread, Peanuts, Milk, Fruit, Jam
2	Bread, Jam, Soda, Chips, Milk, Fruit
3	Steak, Jam, Soda, Chips, Bread
4	Jam, Soda, Peanuts, Milk, Fruit
5	Jam, Soda, Chips, Milk, Bread
6	Fruit, Soda, Chips, Milk
7	Fruit, Soda, Peanuts, Milk
8	Fruit, Peanuts, Cheese, Yogurt

What is An Association Rule?

- Implication of the form X $\Rightarrow$ Y, where X and Y are itemsets
- Example, {bread} $\Rightarrow$ {milk}
- Rule Evaluation Metrics, Suppor & Confidence
- 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

$$s = \frac{\sigma(\{\text{Bread, Milk}\})}{\#\text{ of transactions}} = 0.38$$

$$c = \frac{\sigma(\{\text{Bread, Milk}\})}{\sigma(\{\text{Bread}\})} = 0.75$$

What is the Goal?

- 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
- Brute-force approach is computationally prohibitive!

Mining Association Rules

- {Bread, Jam} ⇒ {Milk} s=0.4 c=0.75
- {Milk, Jam} ⇒ {Bread} s=0.4 c=0.75
- {Bread} ⇒ {Milk, Jam} s=0.4 c=0.75
- {Jam} ⇒ {Bread, Milk} s=0.4 c=0.6
- {Milk} ⇒ {Bread, Jam} s=0.4 c=0.5

- All the above rules are binary partitions of the same itemset {Milk, Bread, Jam}

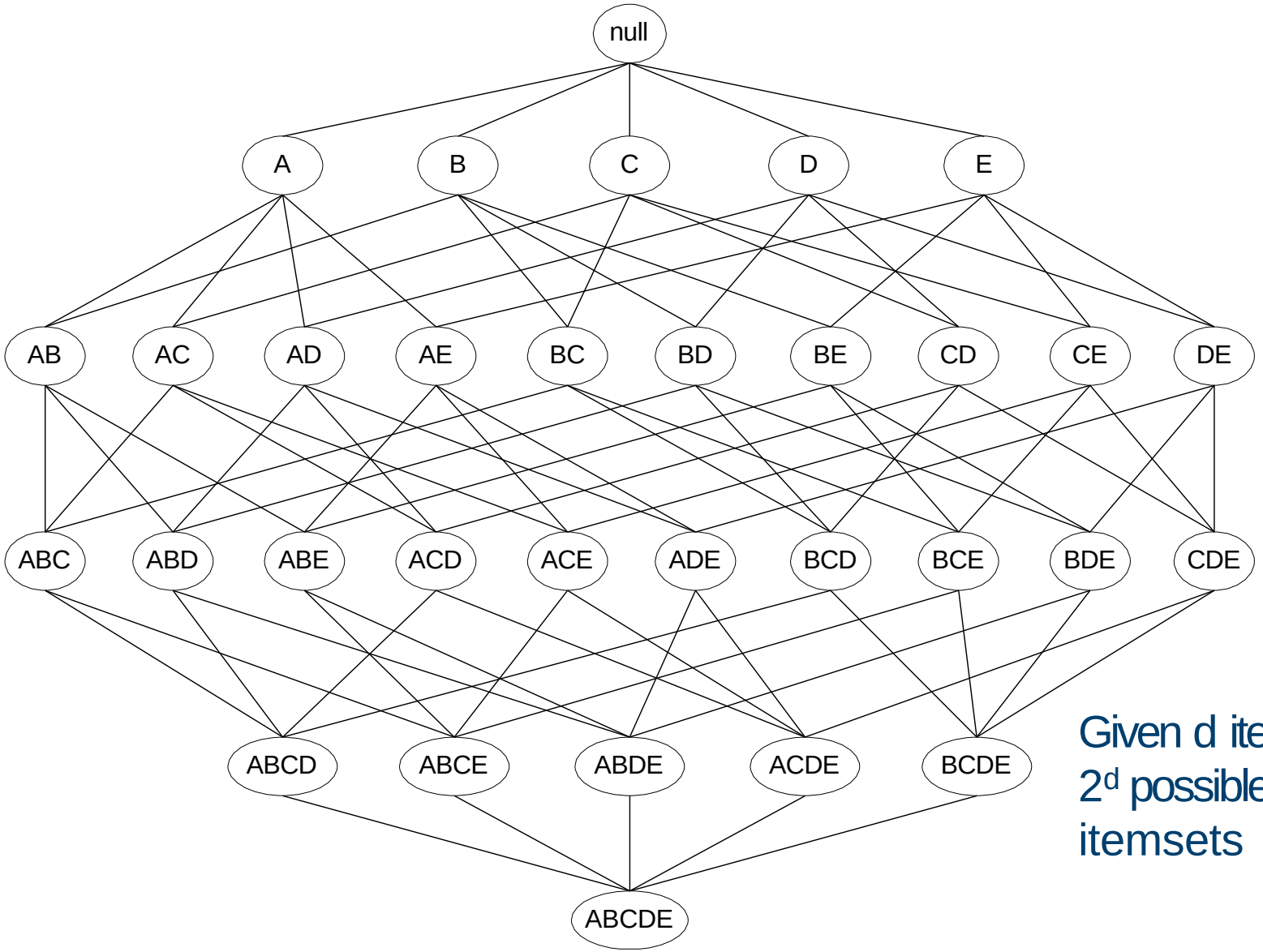
- Rules originating from the same itemset have identical support but can have different confidence
- We can decouple the support and confidence requirements!

Mining Association Rules

Mining Association Rules: Two Step Approach

- Frequent Itemset Generation
  - Generate all itemsets whose support $\geq$ minsup
- Rule Generation
  - Generate high confidence rules from frequent itemset
  - Each rule is a binary partitioning of a frequent itemset
- Frequent itemset generation is computationally expensive

Frequent Itemset Generation



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

Brute Force

BRUTEFORCE (D, $\mathcal{I}$, *minsup*):

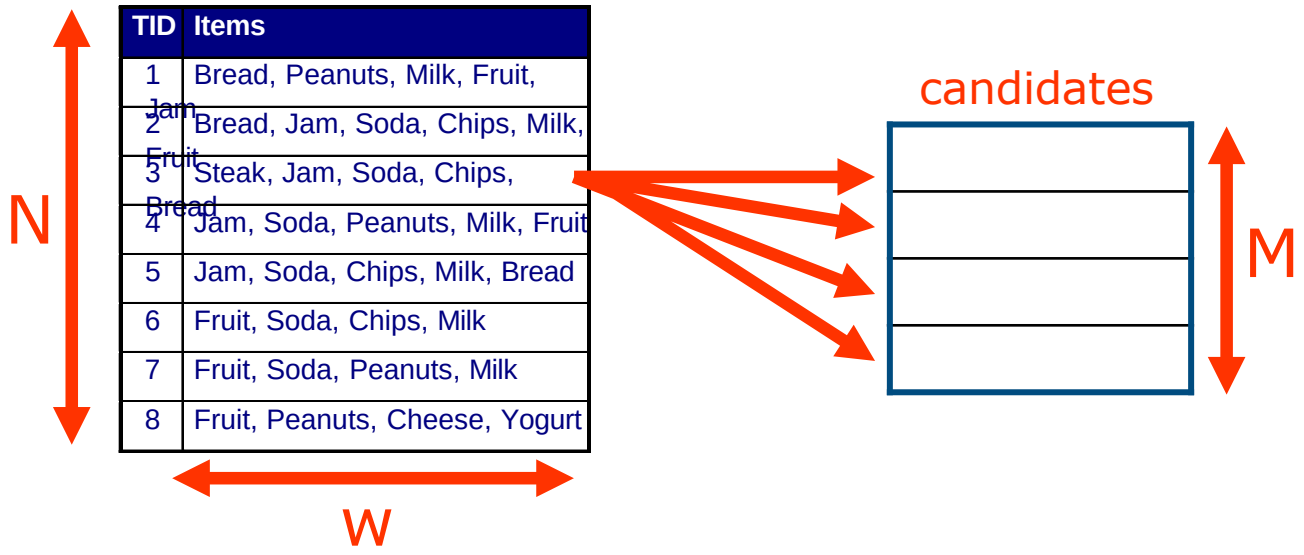
```
1 F ← ∅ // set of frequent itemsets
2 foreach X ⊆ I do
3     sup(X) ← COMPUTESUPPORT (X, D)
4     if sup(X) ≥ minsup then
5         F ← F ∪ {(X, sup(X))}
6 return F
```

COMPUTESUPPORT (X, D):

```
1 sup(X) ← 0
2 foreach ⟨t, i(t)⟩ ∈ D do
3     if X ⊆ i(t) then
4         sup(X) ← sup(X) + 1
5 return sup(X)
```

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



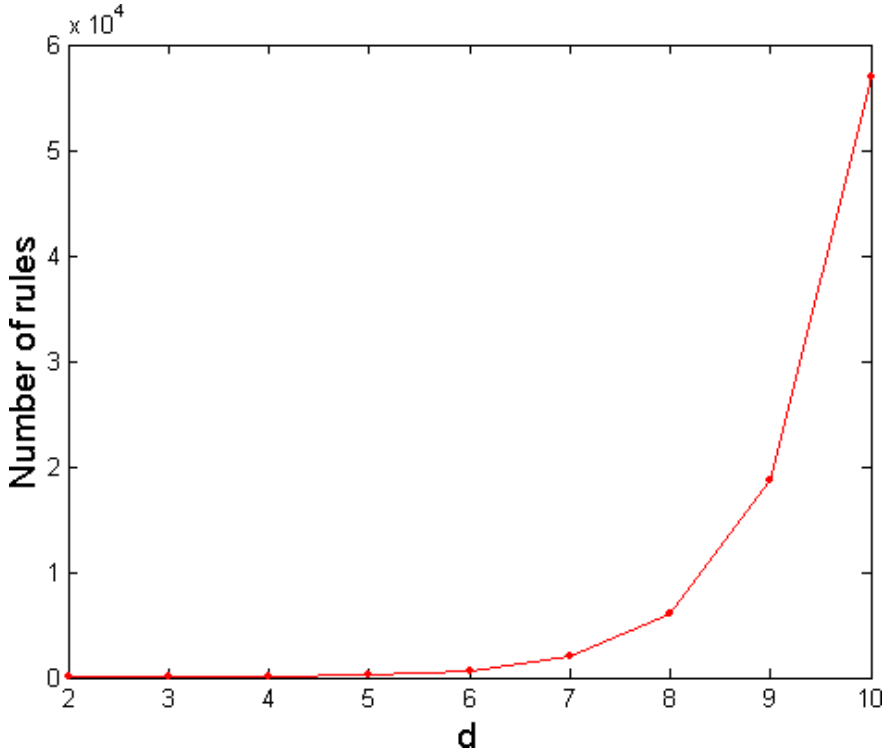
  - Match each transaction against every candidate
  - Complexity ~ O(NMw) => Expensive since M = 2^d

Computational Complexity

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

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

- For d=6, there are 602 rules



Frequent Itemset Generation Strategies

- Reduce the number of candidates (M)
  - Complete search has $M=2^d$
  - 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

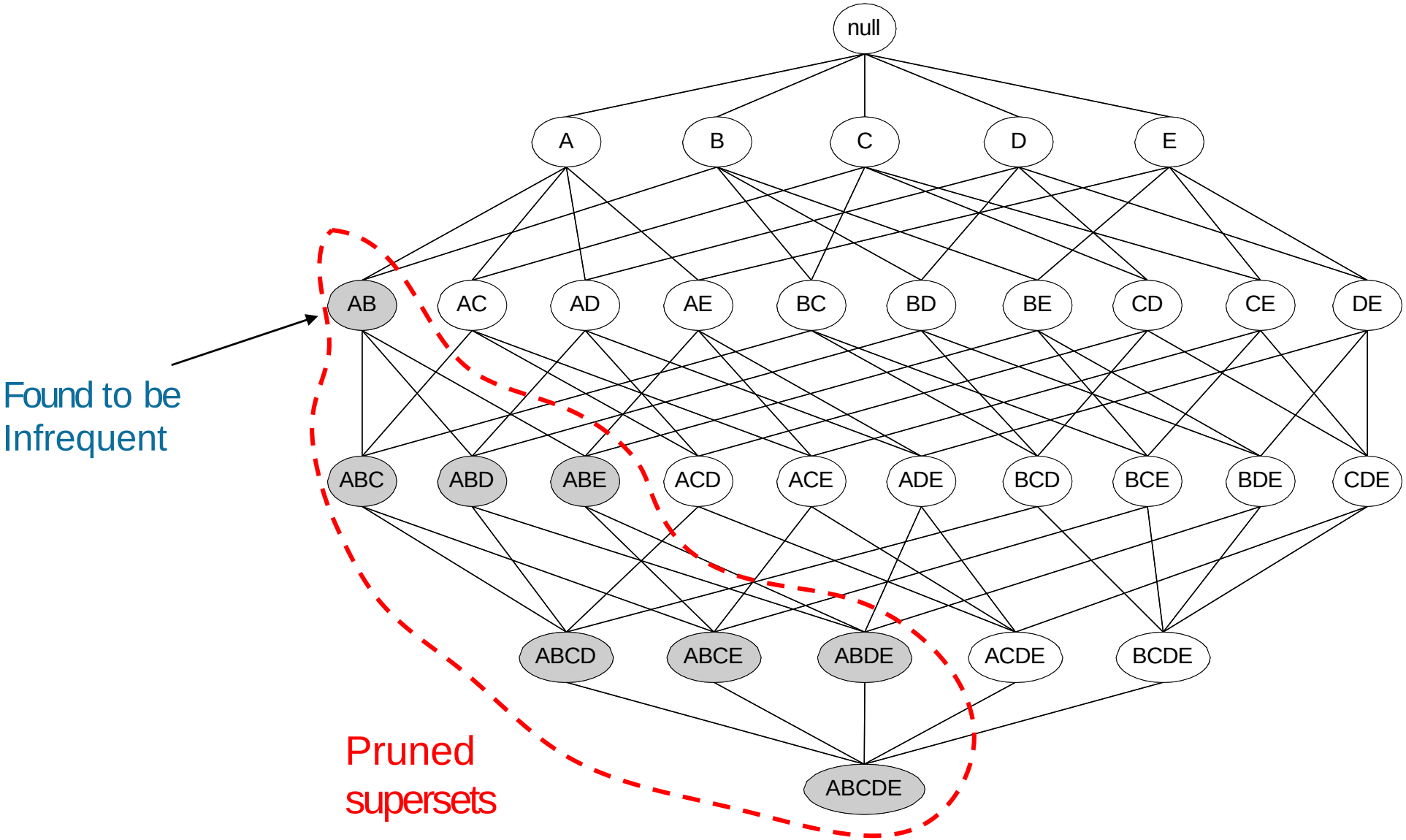
Reducing the Number of Candidates

- 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



How does the Apriori principle work?

Item	Count
Bread	4
Peanuts	4
Milk	6
Fruit	6
Jam	5
Soda	6
Chips	4
Steak	1
Cheese	1
Yogurt	1

1-itemsets

Minimum Support = 4

2-Itemset	Count
Bread, Jam	4
Peanuts, Fruit	4
Milk, Fruit	5
Milk, Jam	4
Milk, Soda	5
Fruit, Soda	4
Jam, Soda	4
Soda, Chips	4

2-itemsets

3-Itemset	Count
Milk, Fruit, Soda	4

3-itemsets

Example

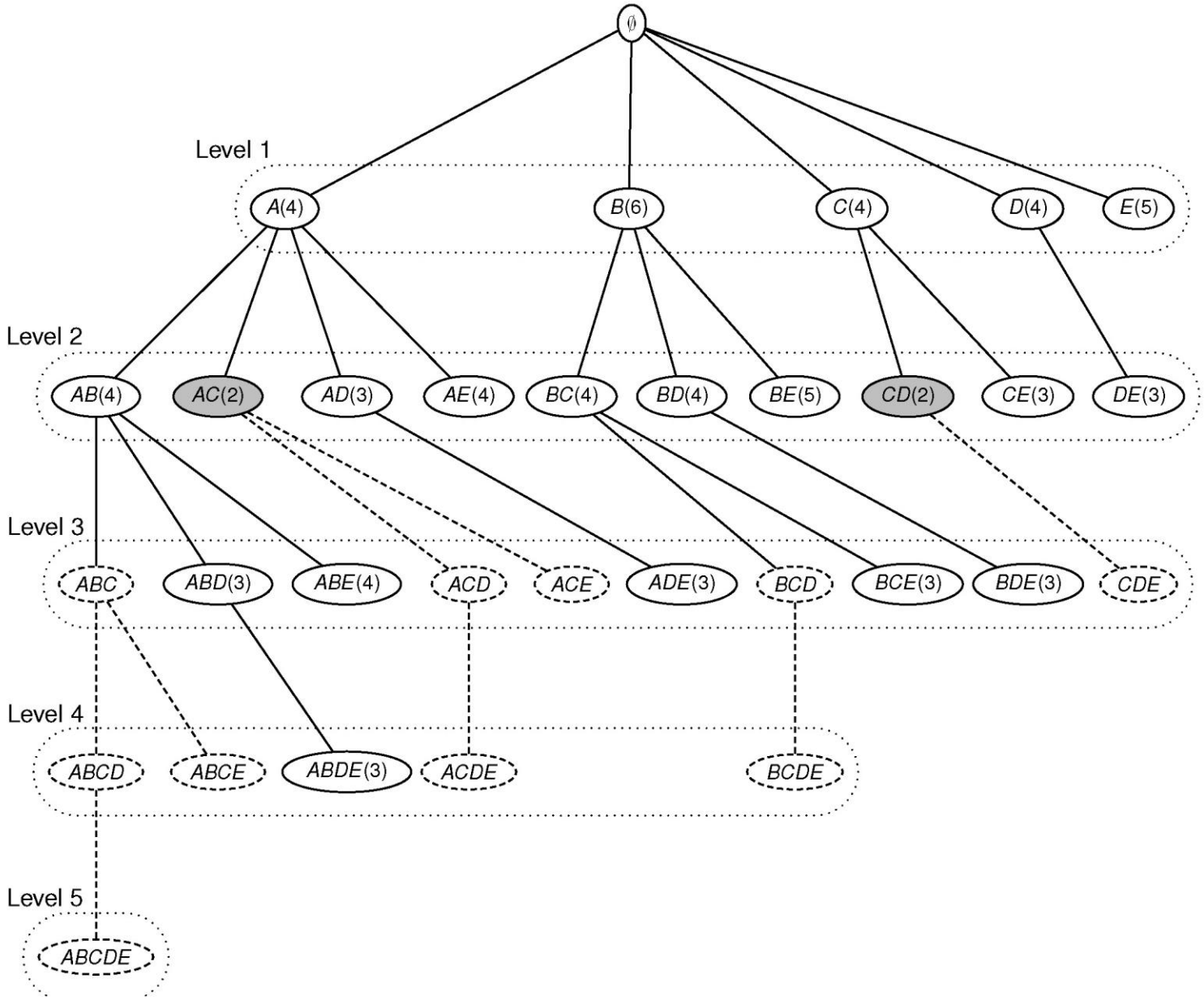
- Given the following database and a min support of 3, generate all the frequent itemsets

D	*A*	*B*	*C*	*D*	*E*
1	1	1	0	1	1
2	0	1	1	0	1
3	1	1	0	1	1
4	1	1	1	0	1
5	1	1	1	1	1
6	0	1	1	1	0

Binary Database

t	**i**(*t*)
1	*ABDE*
2	*BCE*
3	*ABDE*
4	*ABCE*
5	*ABCDE*
6	*BCD*

Transaction Database


∅
Level 1
A(4)
B(6)
C(4)
D(4)
E(5)
Level 2
AB(4)
AC(2)
AD(3)
AE(4)
BC(4)
BD(4)
BE(5)
CD(2)
CE(3)
DE(3)
Level 3
ABC
ABD(3)
ABE(4)
ACD
ACE
ADE(3)
BCD
BCE(3)
BDE(3)
CDE
Level 4
ABCD
ABCE
ABDE(3)
ACDE
BCDE
Level 5
ABCDE

The Apriori Algorithm

- Let k=1
- Generate frequent itemsets of length 1
- Repeat until no new frequent itemsets are identified
  - Generate length (k+1) candidate itemsets from length k frequent itemsets
  - Prune candidate itemsets containing subsets of length k that are infrequent
  - Count the support of each candidate by scanning the DB
  - Eliminate candidates that are infrequent, leaving only those that are frequent

Apriori Algorithm

APRIORI (D, $\mathcal{I}$, *minsup*):

1 $\mathcal{F} \leftarrow \emptyset$
2 $\mathcal{C}^{(1)} \leftarrow \{\emptyset\}$ // Initial prefix tree with single items
3 **foreach** $i \in \mathcal{I}$ **do** Add i as child of $\emptyset$ in $\mathcal{C}^{(1)}$ with $sup(i) \leftarrow 0$
4 $k \leftarrow 1$ // k denotes the level
5 **while** $\mathcal{C}^{(k)} \neq \emptyset$ **do**
6 COMPUTESUPPORT ($\mathcal{C}^{(k)}$, **D**)
7 **foreach** *leaf* $X \in \mathcal{C}^{(k)}$ **do**
8 **if** $sup(X) \geq minsup$ **then** $\mathcal{F} \leftarrow \mathcal{F} \cup \{(X, sup(X))\}$
9 **else** remove X from $\mathcal{C}^{(k)}$
10 $\mathcal{C}^{(k+1)} \leftarrow$ EXTENDPREFIXTREE ($\mathcal{C}^{(k)}$)
11 $k \leftarrow k + 1$
12 **return** $\mathcal{F}^{(k)}$

Apriori Algorithm

COMPUTESUPPORT ($\mathcal{C}^{(k)}$, **D**):

foreach $\langle t, \mathbf{i}(t) \rangle \in \mathbf{D}$ **do**
  foreach *k-subset* $X \subseteq \mathbf{i}(t)$ **do**
    if $X \in \mathcal{C}^{(k)}$ **then** $sup(X) \leftarrow sup(X) + 1$

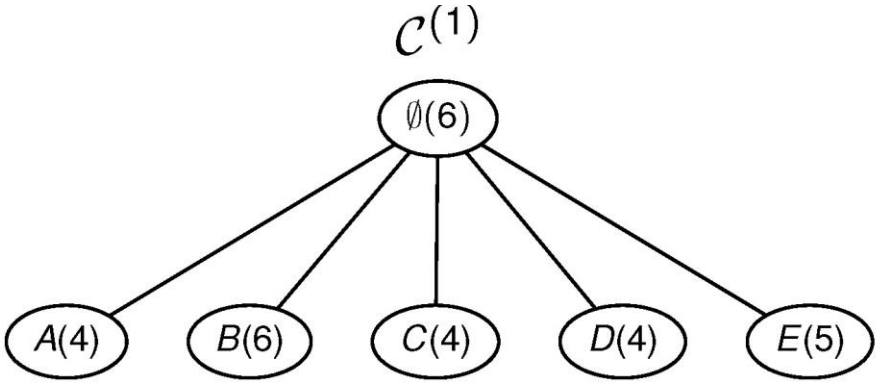
EXTENDPREFIXTREE ($\mathcal{C}^{(k)}$):

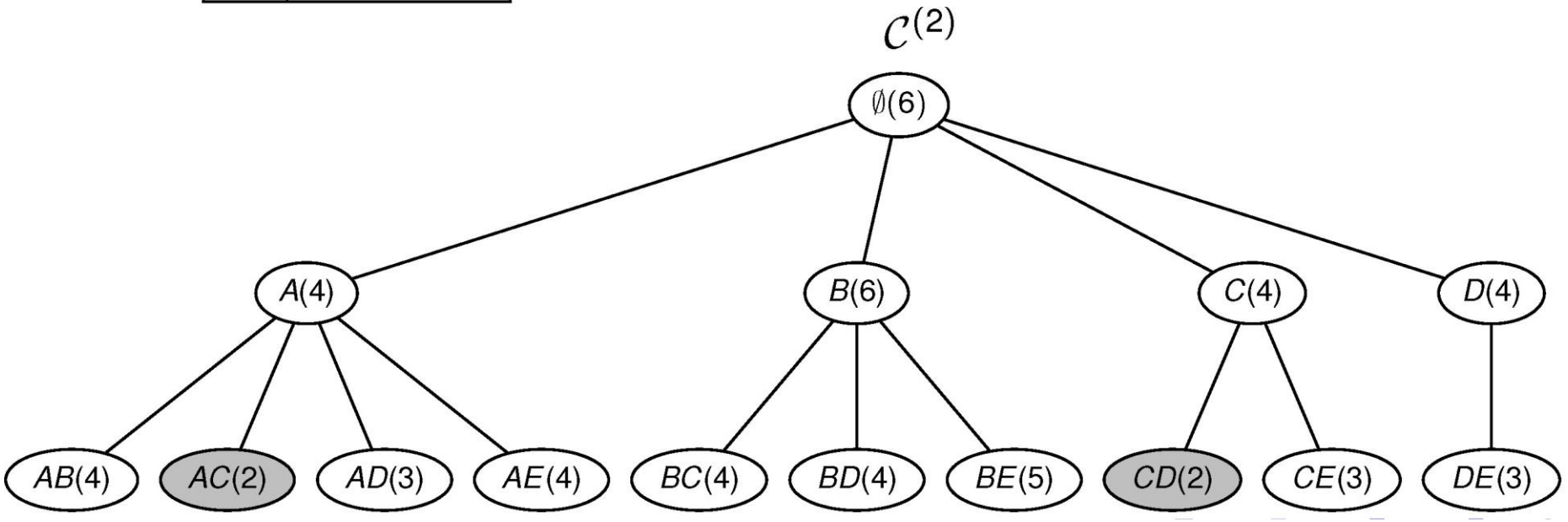
foreach *leaf* $X_a \in \mathcal{C}^{(k)}$ **do**
  foreach *leaf* $X_b \in \text{SIBLING}(X_a)$, *such that* $b > a$ **do**
    $X_{ab} \leftarrow X_a \cup X_b$
    `// prune candidate if there are any infrequent subsets`
    if $X_j \in \mathcal{C}^{(k)}$, ***for all*** $X_j \subset X_{ab}$, *such that* $|X_j| = |X_{ab}| - 1$ **then**
      Add X_{ab} as child of X_a with $sup(X_{ab}) \leftarrow 0$
  if *no extensions from* X_a **then**
    remove X_a, and all ancestors of X_a with no extensions, from $\mathcal{C}^{(k)}$
return $\mathcal{C}^{(k)}$

Apriori Algorithm Example

D

t	**i**(t)
1	*ABDE*
2	*BCE*
3	*ABDE*
4	*ABCE*
5	*ABCDE*
6	*BCD*





Eclat Algorithm

The Eclat Algorithm

- Leverages the tidsets directly for support computation.
- The support of a candidate itemset can be computed by intersecting the tidsets of suitably chosen subsets.
- Given t(X) and t(Y) for any two frequent itemsets X and Y, then $t(XY)=t(X) \wedge t(Y)$
- And $sup(XY) = |t(XY)|$

D	A	B	C	D	E
1	1	1	0	1	1
2	0	1	1	0	1
3	1	1	0	1	1
4	1	1	1	0	1
5	1	1	1	1	1
6	0	1	1	1	0

Binary Database

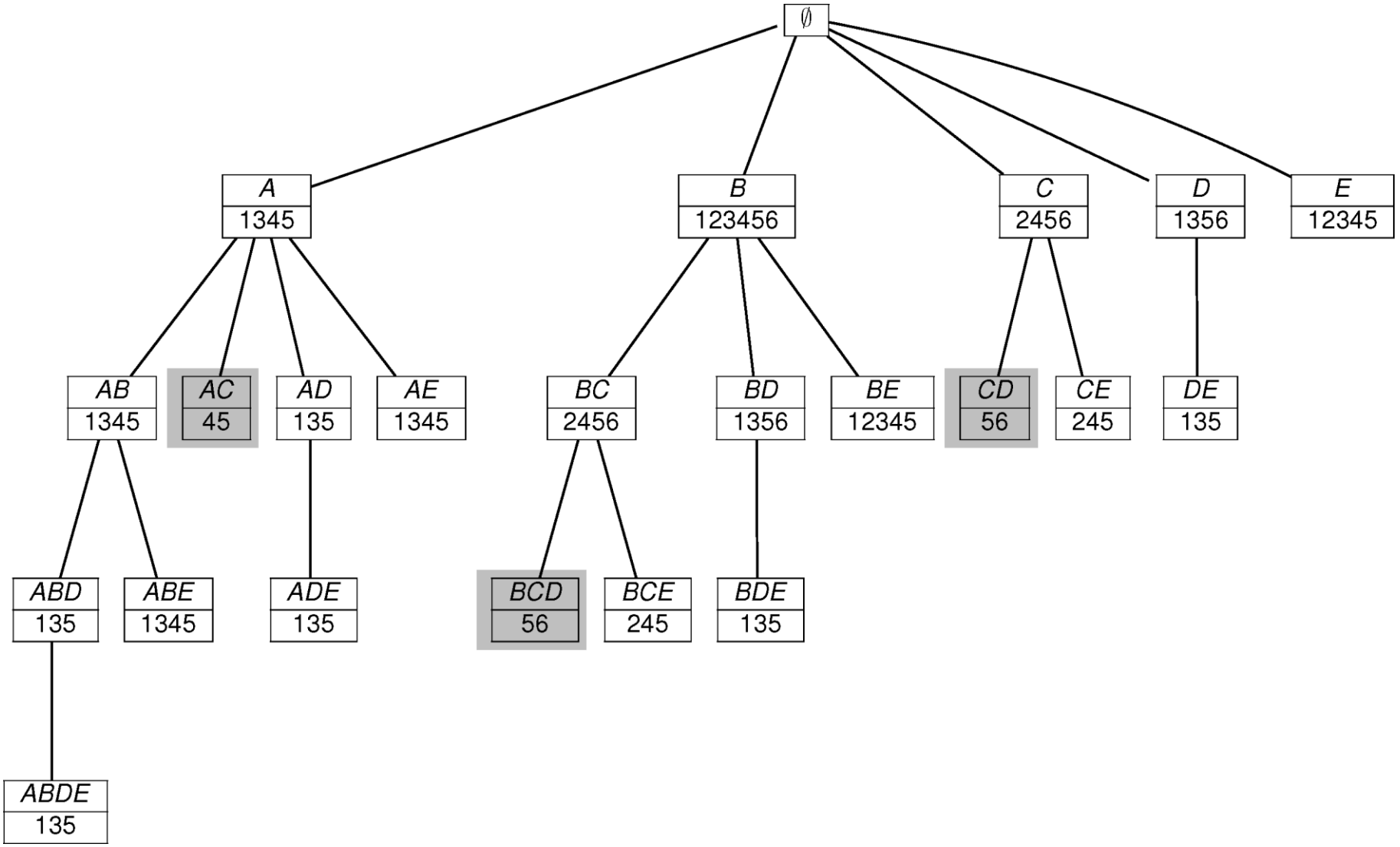
t	**i**(*t*)
1	*ABDE*
2	*BCE*
3	*ABDE*
4	*ABCE*
5	*ABCDE*
6	*BCD*

Transaction Database

t(*x*)				
A	*B*	*C*	*D*	*E*
1	1	2	1	1
3	2	4	3	2
4	3	5	5	3
5	4	6	6	4
	5			5
	6			

Vertial Database

Example


∅
A
1345
B
123456
C
2456
D
1356
E
12345
AB
1345
AC
45
AD
135
AE
1345
BC
2456
BD
1356
BE
12345
CD
56
CE
245
DE
135
ABD
135
ABE
1345
ADE
135
BCD
56
BCE
245
BDE
135
ABDE
135

```
// Initial Call:  F ← ∅, P ← {⟨i, t(i)⟩ | i ∈ I, |t(i)| ≥ minsup}
ECLAT (P, minsup, F):
1 foreach ⟨X_a, t(X_a)⟩ ∈ P do
2     F ← F ∪ {(X_a, sup(X_a))}
3     P_a ← ∅
4     foreach ⟨X_b, t(X_b)⟩ ∈ P, with X_b > X_a do
5         X_ab = X_a ∪ X_b
6         t(X_ab) = t(X_a) ∩ t(X_b)
7         if sup(X_ab) ≥ minsup then
8             P_a ← P_a ∪ {⟨X_ab, t(X_ab)⟩}
9     if P_a ≠ ∅ then ECLAT (P_a, minsup, F)
```

Frequent Patterns Mining Without Candidate Generation

Apriori's Performance Bottlenecks

- The core of the Apriori algorithm
  - Use frequent (k–1)-itemsets to generate candidate frequent k-itemsets
  - Use database scan and pattern matching to collect counts for the candidate itemsets
- Huge candidate sets:
  - 10^4 frequent 1-itemset will generate 10^7 candidate 2-itemsets
  - To discover a frequent pattern of size 100, e.g., $\{a_1, a_2, \ldots, a_{100}\}$, needs to generate $2^{100} \sim 10^{30}$ candidates.
  - Multiple scans of database, it needs (n+1) scans, n is the length of the longest pattern

Mining Frequent Patterns Without Candidate Generation

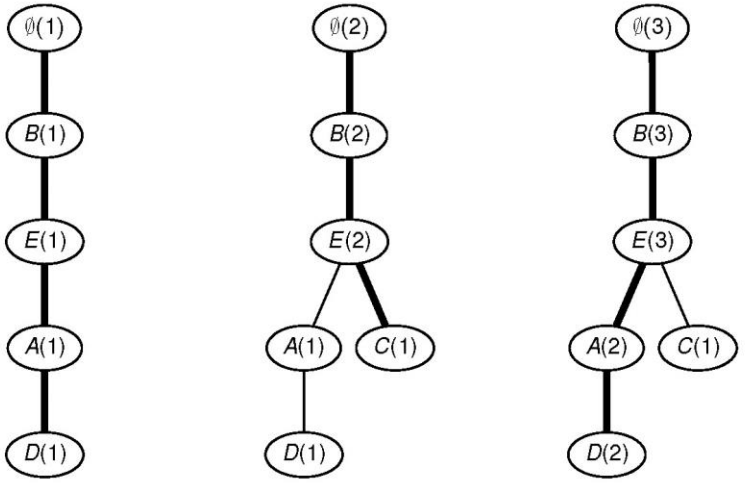
- Compress a large database into a compact, Frequent-Pattern tree (FP-tree) structure
  - Highly condensed, but complete for frequent pattern mining
  - Avoid costly database scans
- Use an efficient, FP-tree-based frequent pattern mining method
- A divide-and-conquer methodology: decompose mining tasks into smaller ones
- Avoid candidate generation: sub-database test only

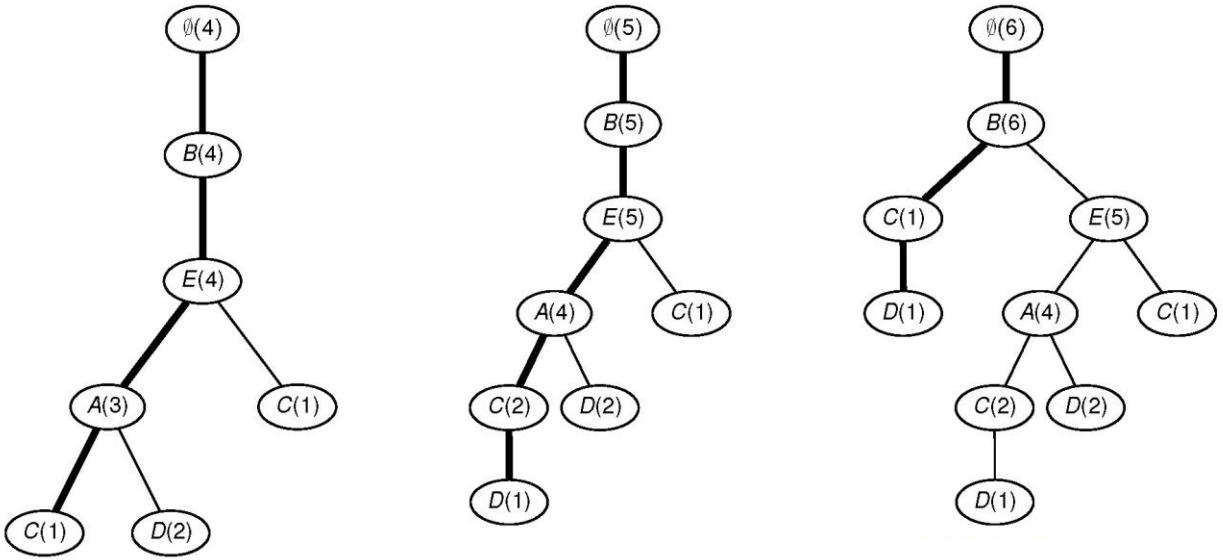
The FP-Growth Algorithm

- Leave the generate-and-test paradigm of Apriori
- Data sets are encoded using a compact structure, the FP-tree
- Frequent itemsets are extracted directly from the FP-tree

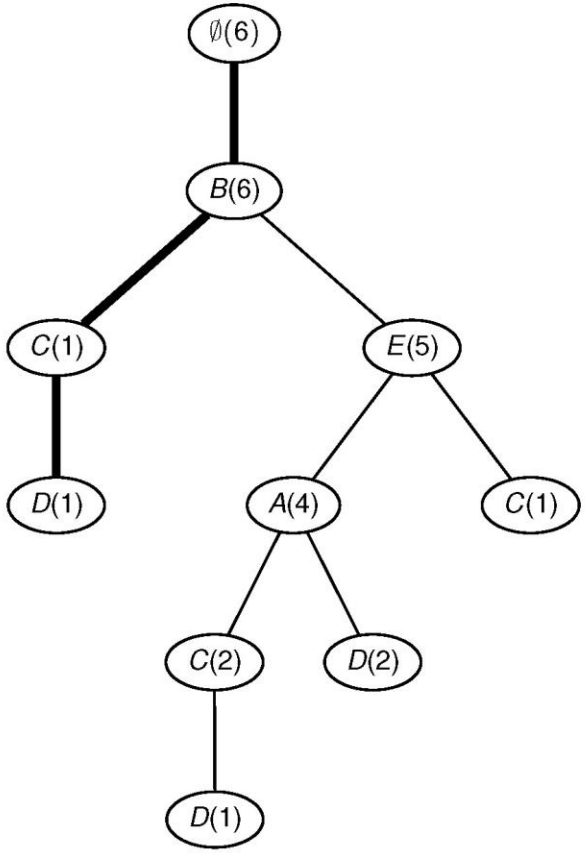
- Major Steps to mine FP-tree
  - Construct conditional pattern base for each node in the FP-tree
  - Construct conditional FP-tree from each conditional pattern-base
  - Recursively mine conditional FP-trees and grow frequent patterns obtained so far
  - If the conditional FP-tree contains a single path, simply enumerate all the patterns

Transactions
BEAD
BEC
BEAD
BEAC
BEACD
BCD

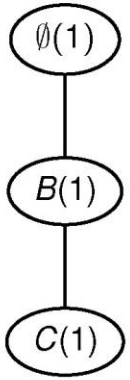




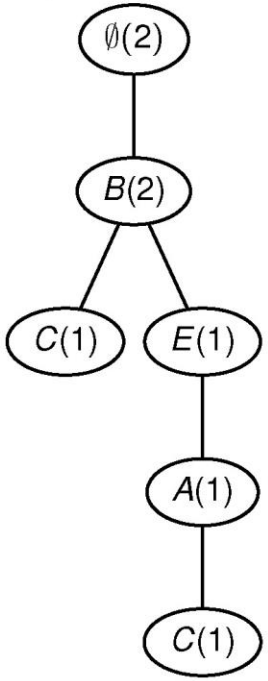
FP-Tree



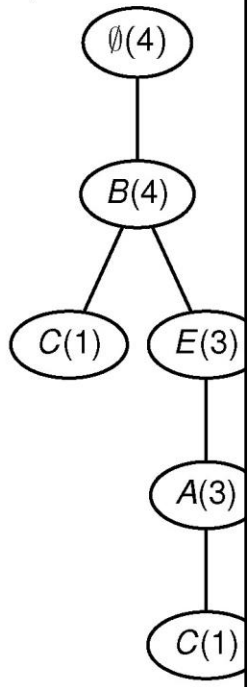
Add BC, $cnt = 1$

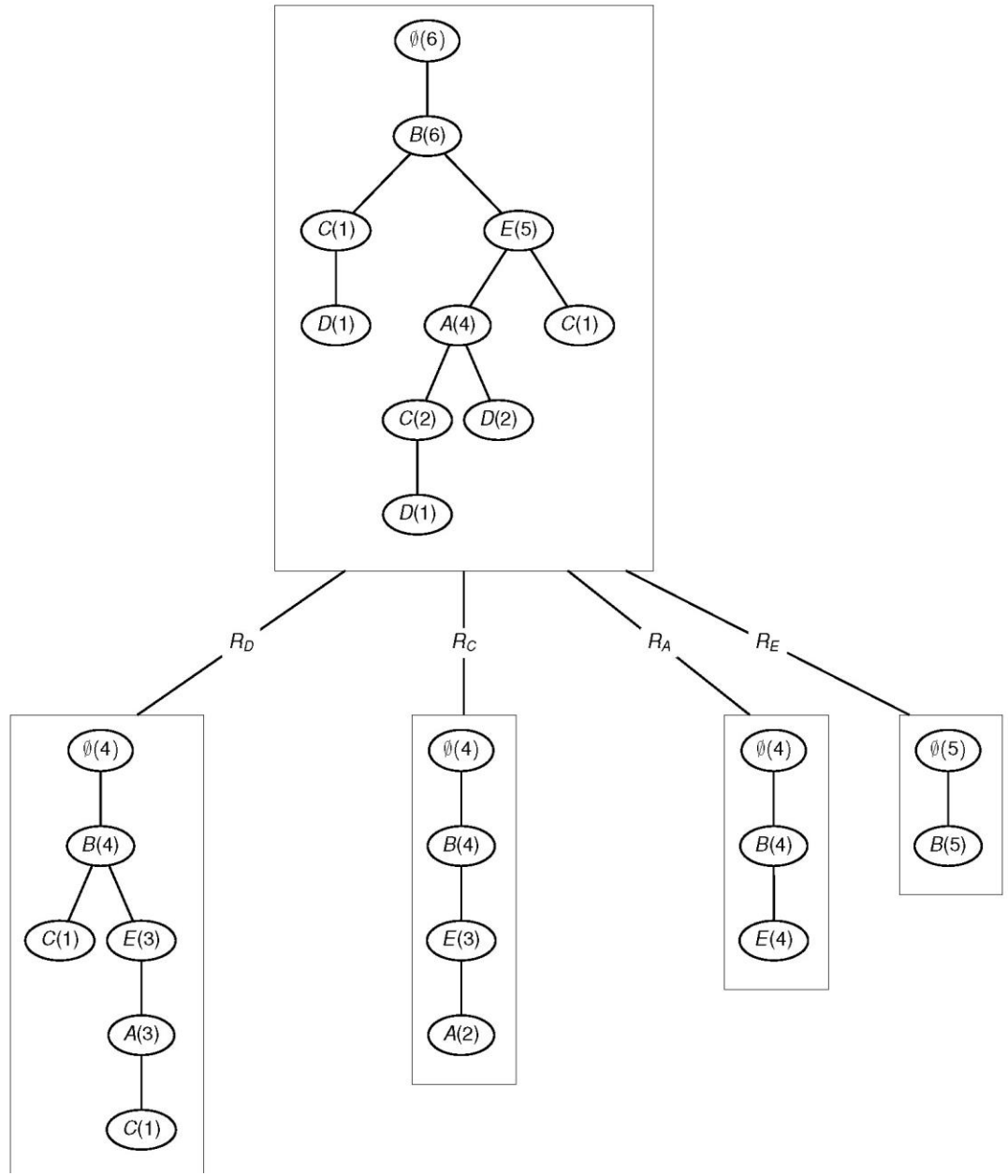


Add $BEAC$, $cnt = 1$



Add BEA, $cnt = 2$




∅(6)
B(6)
C(1)
E(5)
D(1)
A(4)
C(1)
C(2)
D(2)
D(1)
R_D
R_C
R_A
R_E
∅(4)
B(4)
C(1)
E(3)
A(3)
C(1)
∅(4)
B(4)
E(3)
A(2)
∅(4)
B(4)
E(4)
∅(5)
B(5)

Rule Generation

Rule Generation

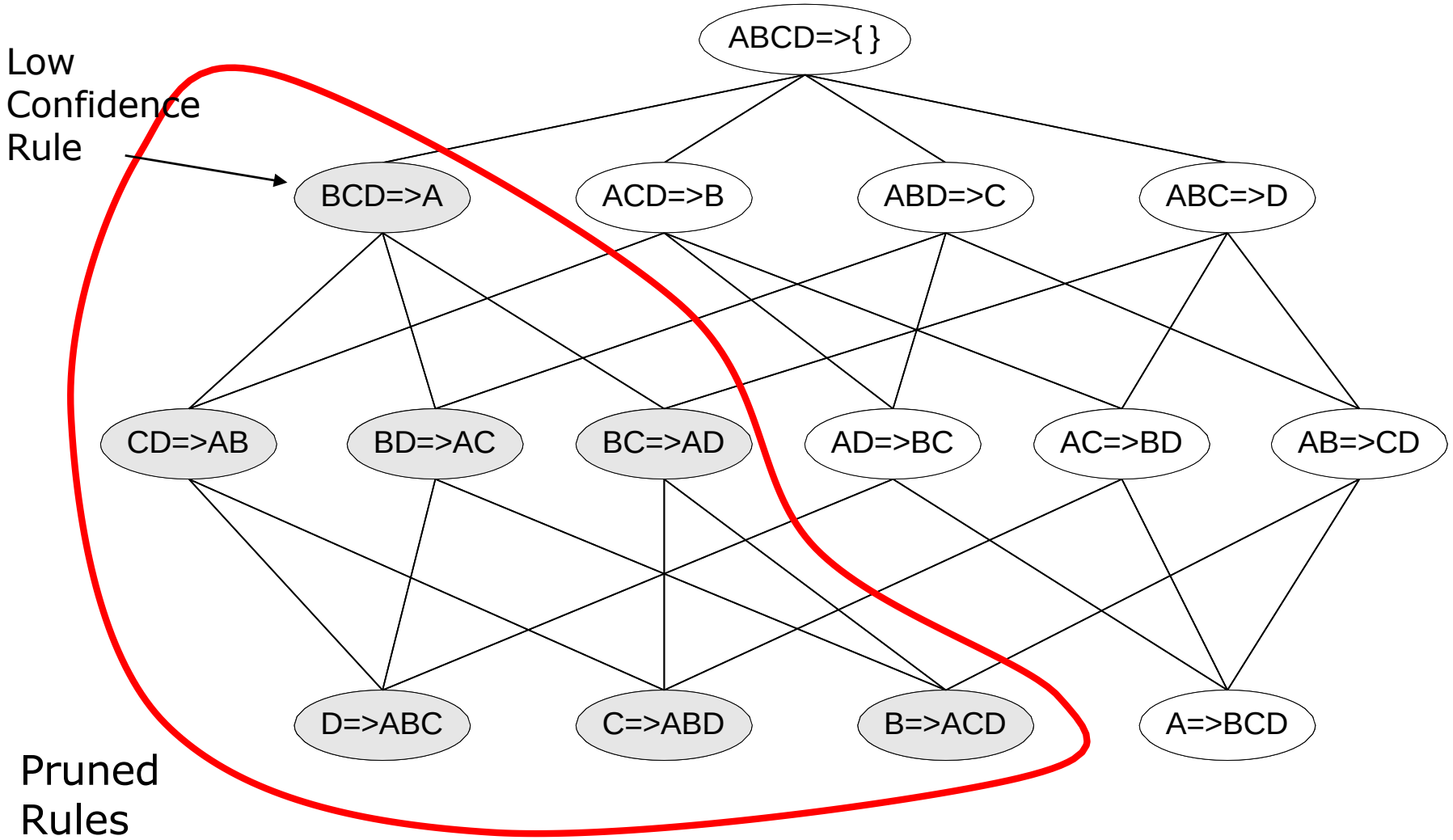
- Given a frequent itemset L, find all non-empty subsets f ⊂ L such that f → L – f satisfies the minimum confidence requirement
- If {A,B,C,D} is a frequent itemset, candidate rules:
  ABC → D, ABD → C, ACD → B, BCD → A, A → BCD,
  B → ACD, C → ABD, D → ABC, AB → CD, AC → BD,
  AD → BC, BC → AD, BD → AC, CD → AB

- If |L| = k, then there are 2k – 2 candidate association rules (ignoring L → ∅ and ∅ → L)

How to efficiently generate rules from frequent itemsets?

- Confidence does not have an anti-monotone property
- 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
- L = {A,B,C,D}: c(ABC → D) ≥ c(AB → CD) ≥ c(A → BCD)
- Confidence is anti-monotone with respect to the number of items on the right hand side of the rule

Rule Generation for Apriori Algorithm

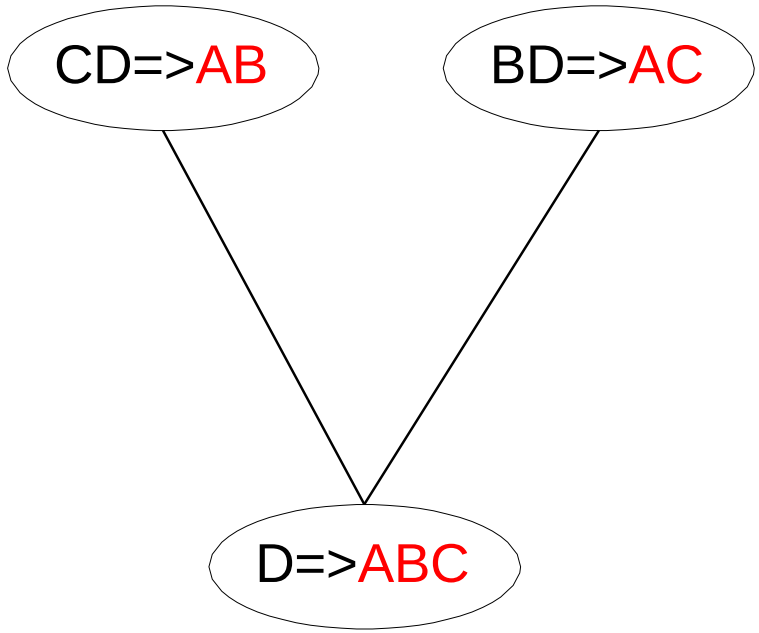


Rule Generation for Apriori Algorithm

- 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



AssociationRules ($\mathcal{F}$, *minconf*):

```
1  foreach Z ∈ F, such that |Z| ≥ 2 do
2      A ← {X | X ⊂ Z, X ≠ ∅}
3      while A ≠ ∅ do
4          X ← maximal element in A
5          A ← A \ X // remove X from A
6          c ← sup(Z)/sup(X)
7          if c ≥ minconf then
8              print X ⟶ Y, sup(Z), c
9          else
10             A ← A \ {W | W ⊂ X}
               // remove all subsets of X from A
```

How to Set an Appropriate Minsup?

- If minsup is set too high, we could miss itemsets involving interesting rare items (e.g., expensive products)
- If minsup is set too low, it is computationally expensive and the number of itemsets is very large
- A single minimum support threshold may not be effective

Benefits of the FP-Tree Structure

- Completeness
  - Preserve complete information for frequent pattern mining
  - Never break a long pattern of any transaction
- Compactness
  - Reduce irrelevant info—infrequent items are gone
  - Items in frequency descending order: the more frequently occurring, the more likely to be shared
  - Never be larger than the original database (not count node-links and the count field)
  - For Connect-4 DB, compression ratio could be over 100

Why is FP-Growth the Winner?

- Divide-and-conquer:
  - decompose both the mining task and DB according to the frequent patterns obtained so far
  - leads to focused search of smaller databases
- Other factors
  - No candidate generation, no candidate test
  - Compressed database: FP-tree structure
  - No repeated scan of entire database
  - Basic ops—counting local freq items and building sub FP-tree, no pattern search and matching

Summarizing Itemsets

Maximal Frequent Itemsets

- A frequent itemset X is called maximal if it has no frequent supersets
- The set of all maximal frequent itemsets, given as

$$M = \{X \mid X \in F \text{ and } !\exists Y \supset X, \text{ such that } Y \in F\}$$

- M is a condensed representation of the set of all frequent itemset F, because we can determine whether any itemset is frequent or not using M
- If there is a maximal itemset Z such that $X \subseteq Z$, then X must be frequent, otherwise X cannot be frequent
- However, M alone cannot be used to determine sup(X), we can only use to have a lower-bound, that is, $sup(X) \geq sup(Z)$ if $X \subseteq Z \in M$.

Closed Frequent Itemsets

- An itemset X is closed if all supersets of X have strictly less support, that is,

$$\sup(X) > \sup(Y), \text{ for all } Y \supset X$$

- The set of all closed frequent itemsets C is a condensed representation, as we can determine whether an itemset X is frequent, as well as the exact support of X using C alone

Minimal Generators

- A frequent itemset X is a minimal generator if it has no subsets with the same support

$$G = \{ X \mid X \in F \text{ and } !\exists Y \subset X, \text{ such that } sup(X) = sup(Y) \}$$

- Thus, all subsets of X have strictly higher support, that is,

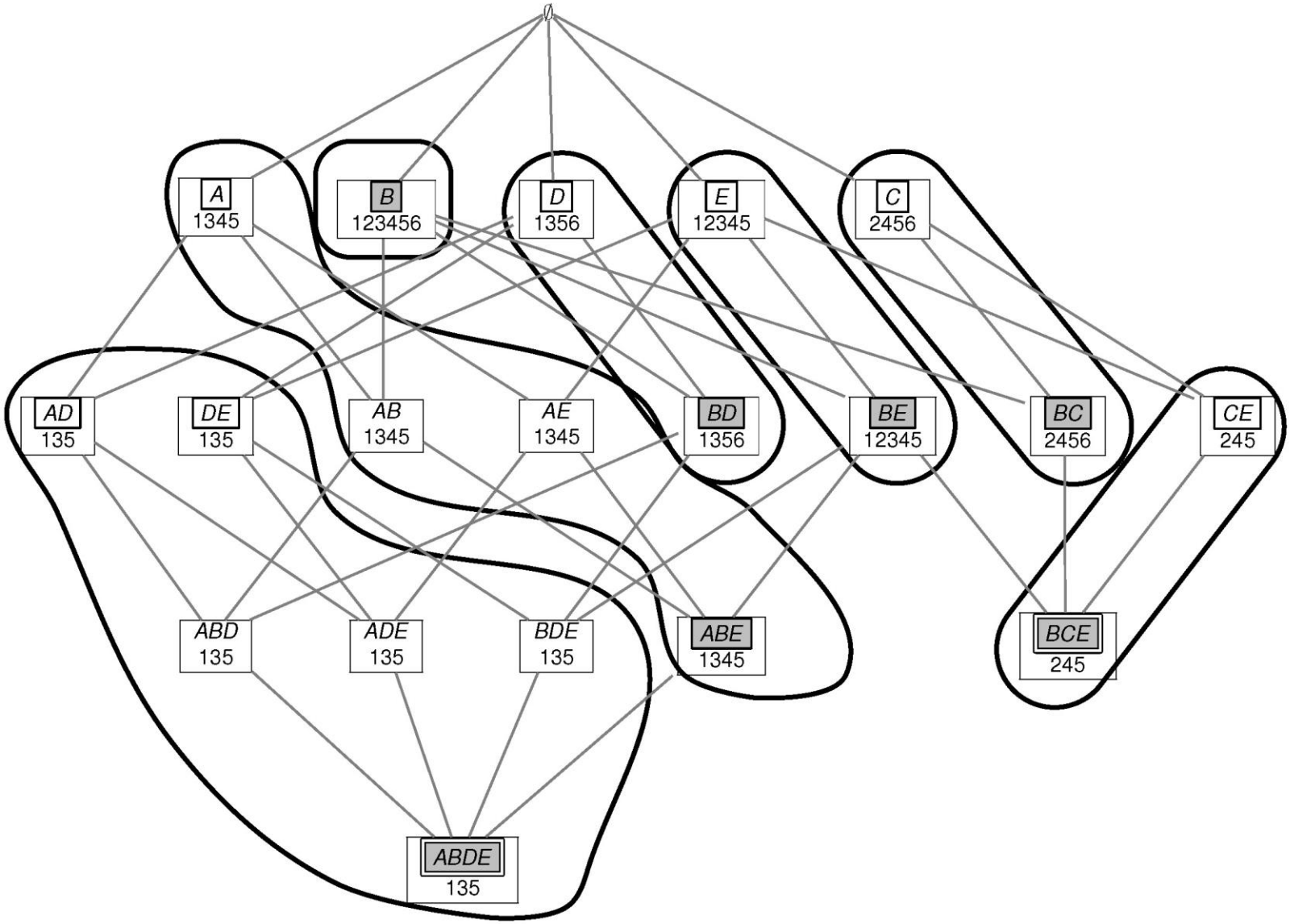
$$sup(X) < sup(Y)$$

Transaction database

Tid	Itemset
1	*ABDE*
2	*BCE*
3	*ABDE*
4	*ABCE*
5	*ABCDE*
6	*BCD*

Frequent itemsets ($minsup = 3$)

sup	Itemsets
6	*B*
5	*E*, *BE*
4	*A*, *C*, *D*, *AB*, *AE*, *BC*, *BD*, *ABE*
3	*AD*, *CE*, *DE*, *ABD*, *ADE*, *BCE*, *BDE*, *ABDE*



Shaded boxes are closed itemsets

Simple boxes are generators

Shaded boxes with double lines are maximal itemsets