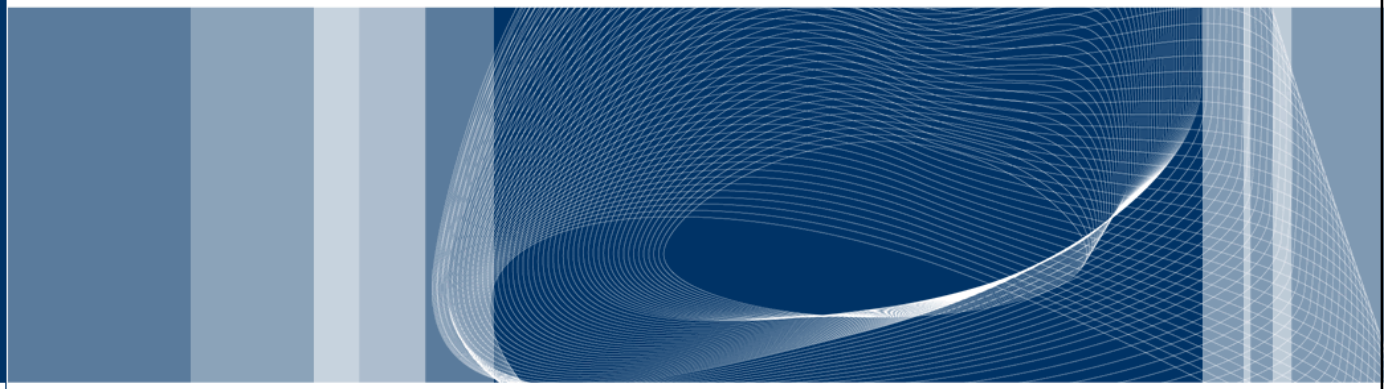
POLITECNICO DI MILANO



Association Rules

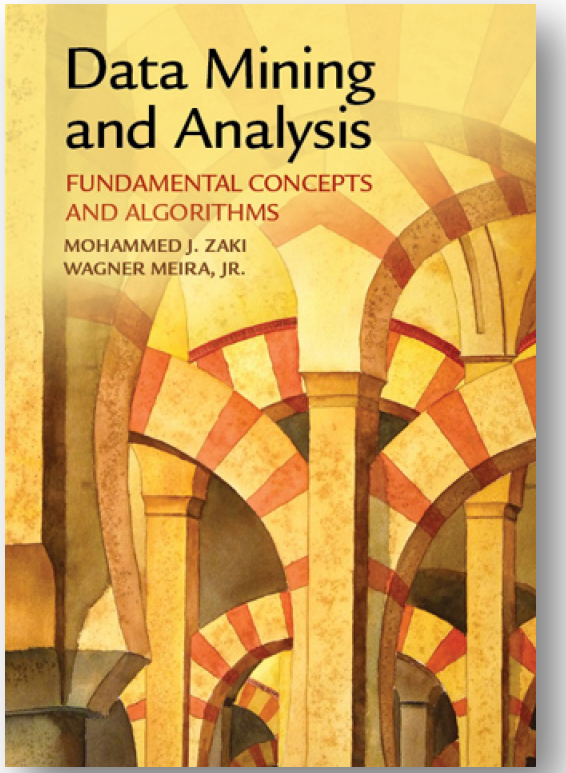
Data Mining and Text Mining

Prof. Pierluca Lanzi

Study Material

- "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
Crème
Fra Boiatéra


RICARD
PASTIS
MARSEILLE
FRANCE
ANGOSTURA
aromatic bitters
Since 1824
ANGOSTURA
aromatic bitters
TOSCHI
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

What is Association Rule Mining?

- Finding frequent patterns and associations among sets of items in transaction databases, relational databases, or 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 a transaction

- Consider the following data set with 8 transactions

- Examples of itemsets
  - {Bread, Milk}
  - {Soda, Chips}

- Examples of association rules
  - {Bread} ⇒ {Milk}
  - {Soda} ⇒ {Chips}
  - {Bread} ⇒ {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

"Itemsets" are the fundamental pattern in association rule mining

What is An Association Rule?

- Implication of the form $X \Rightarrow Y$, where X and Y are itemsets
- Example, {Bread} $\Rightarrow$ {Milk}
- Two rule evaluation metrics
  - Support
  - Confidence

evaluation metrics for association rules

{Bread} ⇒ {Milk}

Support

Fraction of transactions that contain both X and Y

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

Confidence

Measures how often items in Y appear in transactions that contain X

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

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

- They are all binary partitions of the itemset {Milk, Bread, Jam}

- Thus, rules originating from the same itemset have,
  - Same support but
  - Can have different confidence

We can decouple the support and confidence requirements!

First, find the itemsets with support ≥ minsup

Next, generate the rules with confidence ≥ minconf

What are Frequent Itemsets?

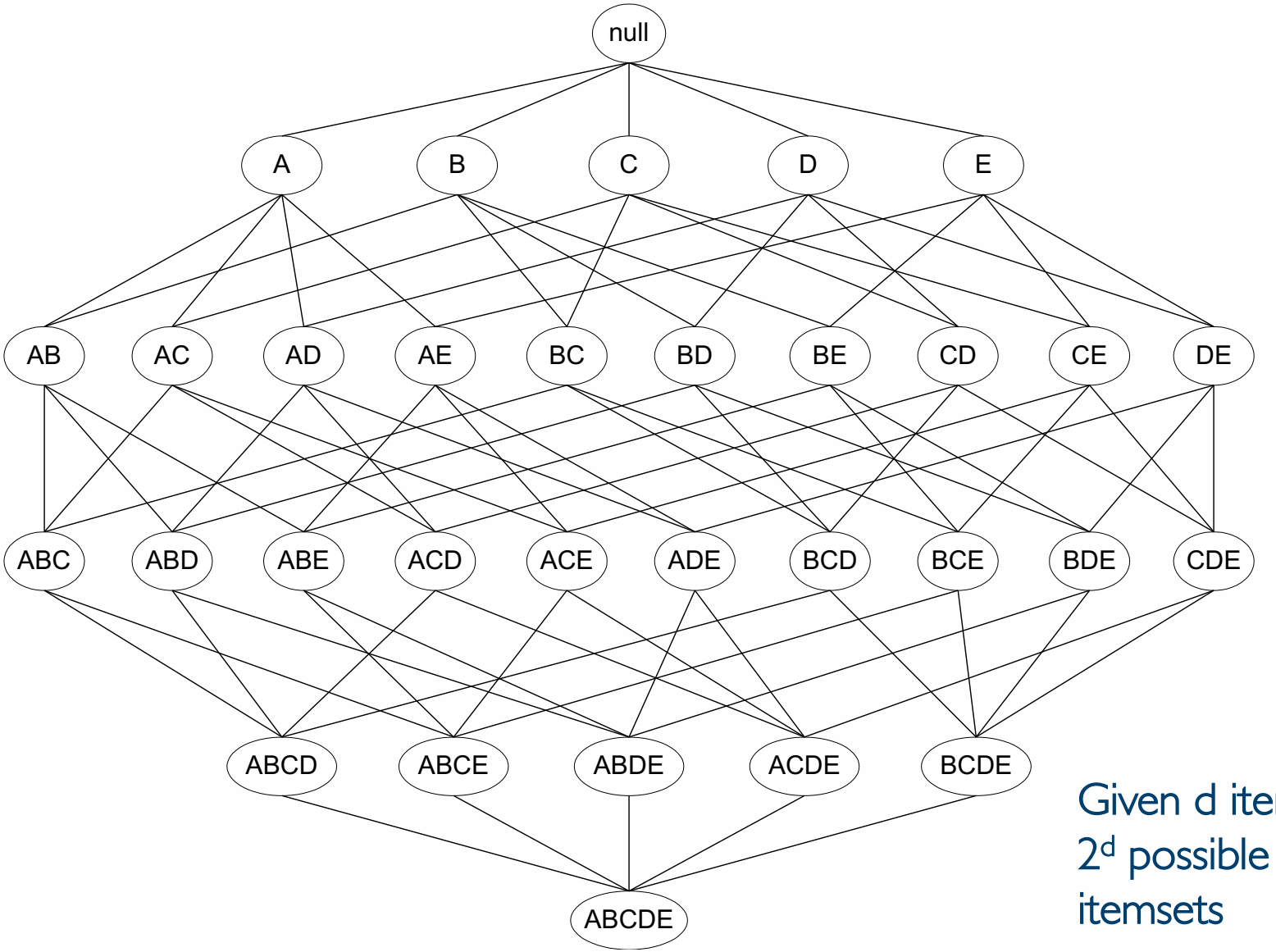
- Frequent Itemset
  - An itemset whose support is greater than or equal to the minsup threshold
- Support
  - Fraction of transactions that contain an itemset
  - s({Milk, Bread}) = 3/8
    s({Soda, Chips}) = 4/8
- Support count (σ)
  - Raw count of occurrence of an itemset
  - For example, σ({Milk, Bread}) = 3 σ({Soda, Chips}) = 4

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

Mining Association Rules: a Two Step Approach

- Frequent Itemset Generation
  - Generate all itemsets whose support ≥ 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*):

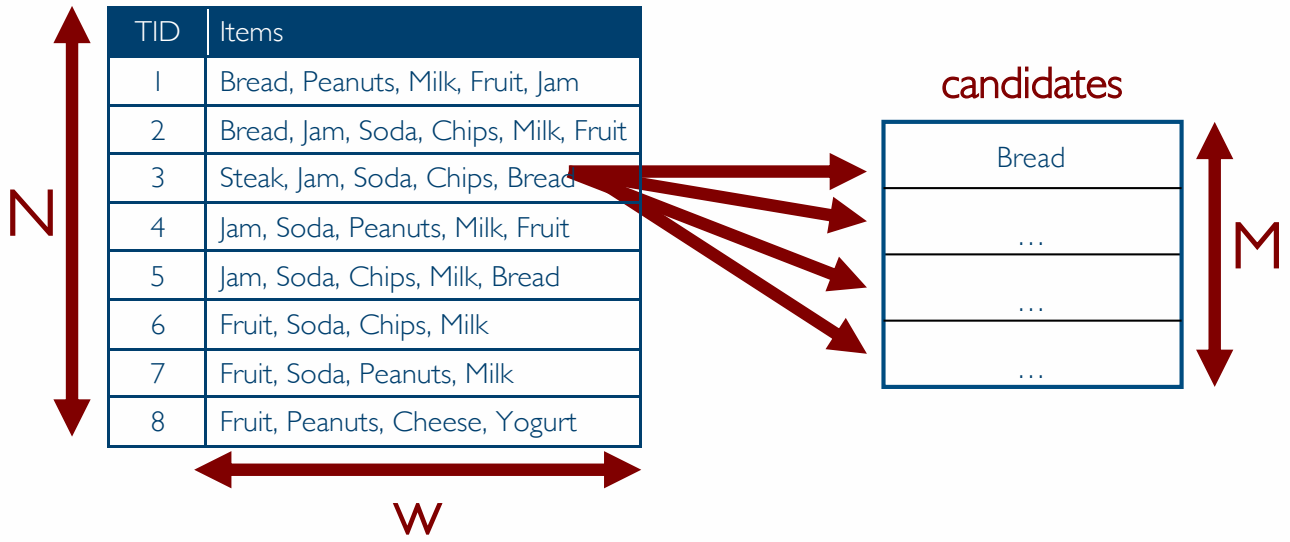
```
1 F ← ∅ // set of frequent itemsets
2 foreach X ⊆ I do
3     sup(X) ← COMPUTESUPPORT (X, D)
4     if sup(X) ≥ minsup then
5         F ← F ∪ {(X, sup(X))}
6 return F
```

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)
```

Brute-Force Generation of Frequent Itemsets

- 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

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: The Apriori Principle

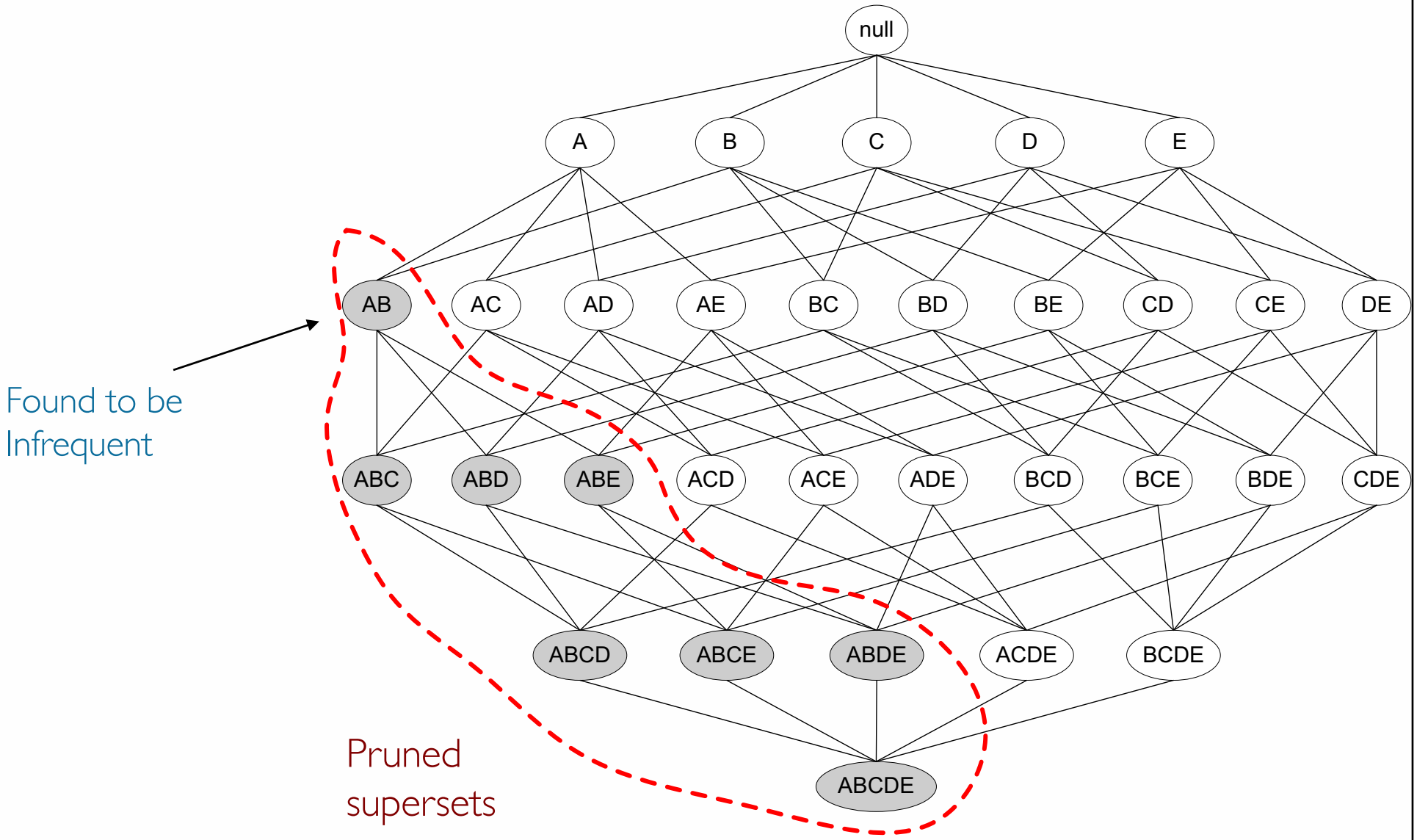
- Apriori principle
  - If an itemset is frequent, then all of its subsets must also be frequent

- 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?

Minimum Support = 4/8

Item	Support
Bread	4/8
Peanuts	4/8
Milk	6/8
Fruit	6/8
Jam	5/8
Soda	6/8
Chips	4/8
Steak	1/8
Cheese	1/8
Yogurt	1/8

1-itemsets

2-Itemset	Support
Bread, Jam	4/8
Peanuts, Fruit	4/8
Milk, Fruit	5/8
Milk, Jam	4/8
Milk, Soda	5/8
Fruit, Soda	4/8
Jam, Soda	4/8
Soda, Chips	4/8

2-itemsets

3-Itemset	Support
Milk, Fruit, Soda	4/8

3-itemsets

Example from the Textbook

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

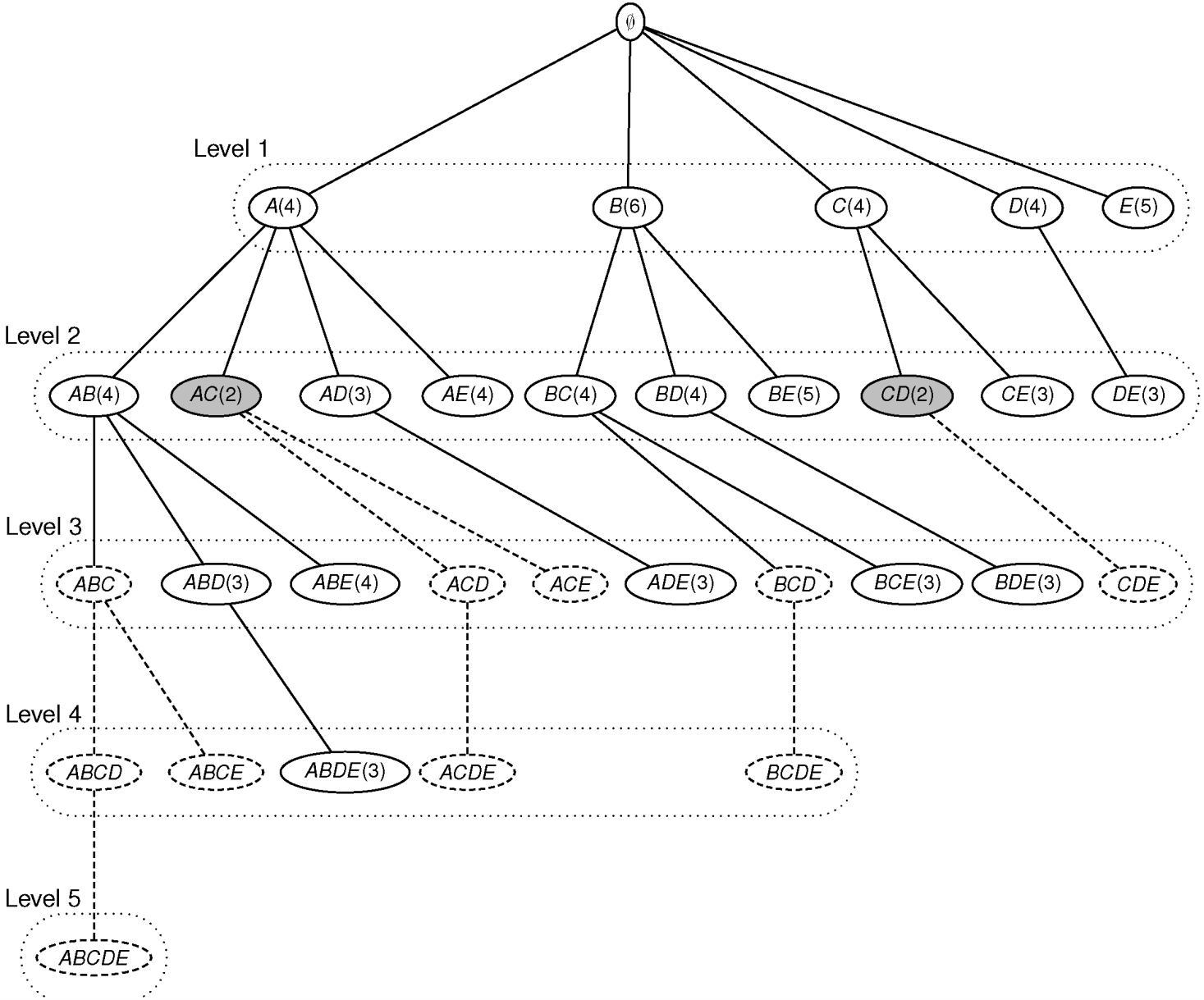
D	*A*	*B*	*C*	*D*	*E*
1	1	1	0	1	1
2	0	1	1	0	1
3	1	1	0	1	1
4	1	1	1	0	1
5	1	1	1	1	1
6	0	1	1	1	0

Binary Database

t	$\mathbf{i}(t)$
1	*ABDE*
2	*BCE*
3	*ABDE*
4	*ABCE*
5	*ABCDE*
6	*BCD*

Transaction Database

Example from the Textbook



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 database
  - 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** $sup(X) \leftarrow sup(X) + 1$

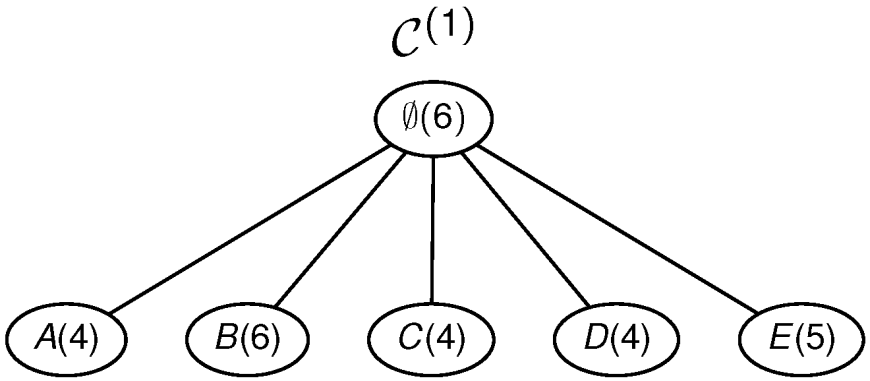
EXTENDPREFIXTREE ($\mathcal{C}^{(k)}$):

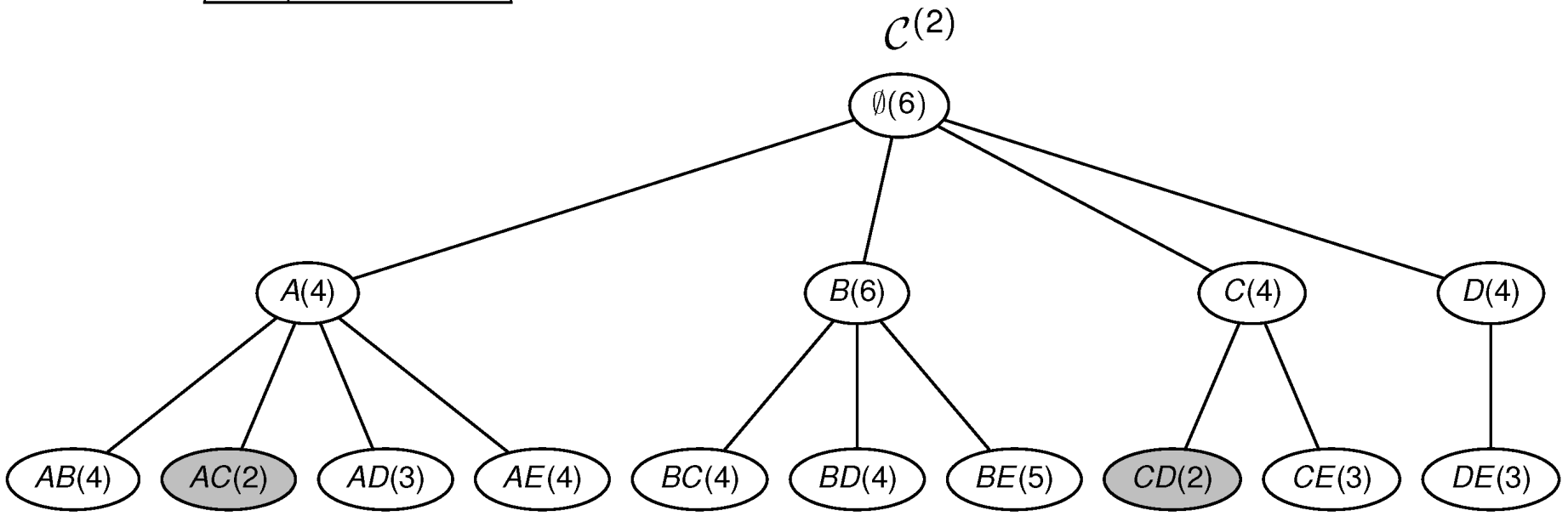
foreach *leaf* $X_a \in \mathcal{C}^{(k)}$ **do**
  foreach *leaf* $X_b \in \text{SIBLING}(X_a)$, *such that* $b > a$ **do**
    $X_{ab} \leftarrow X_a \cup X_b$
    `// prune candidate if there are any infrequent subsets`
    if $X_j \in \mathcal{C}^{(k)}$, ***for all*** $X_j \subset X_{ab}$, *such 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 (min sup=3): First two levels of the search

D

t	$\mathbf{i}(t)$
1	*ABDE*
2	*BCE*
3	*ABDE*
4	*ABCE*
5	*ABCDE*
6	*BCD*





Eclat

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

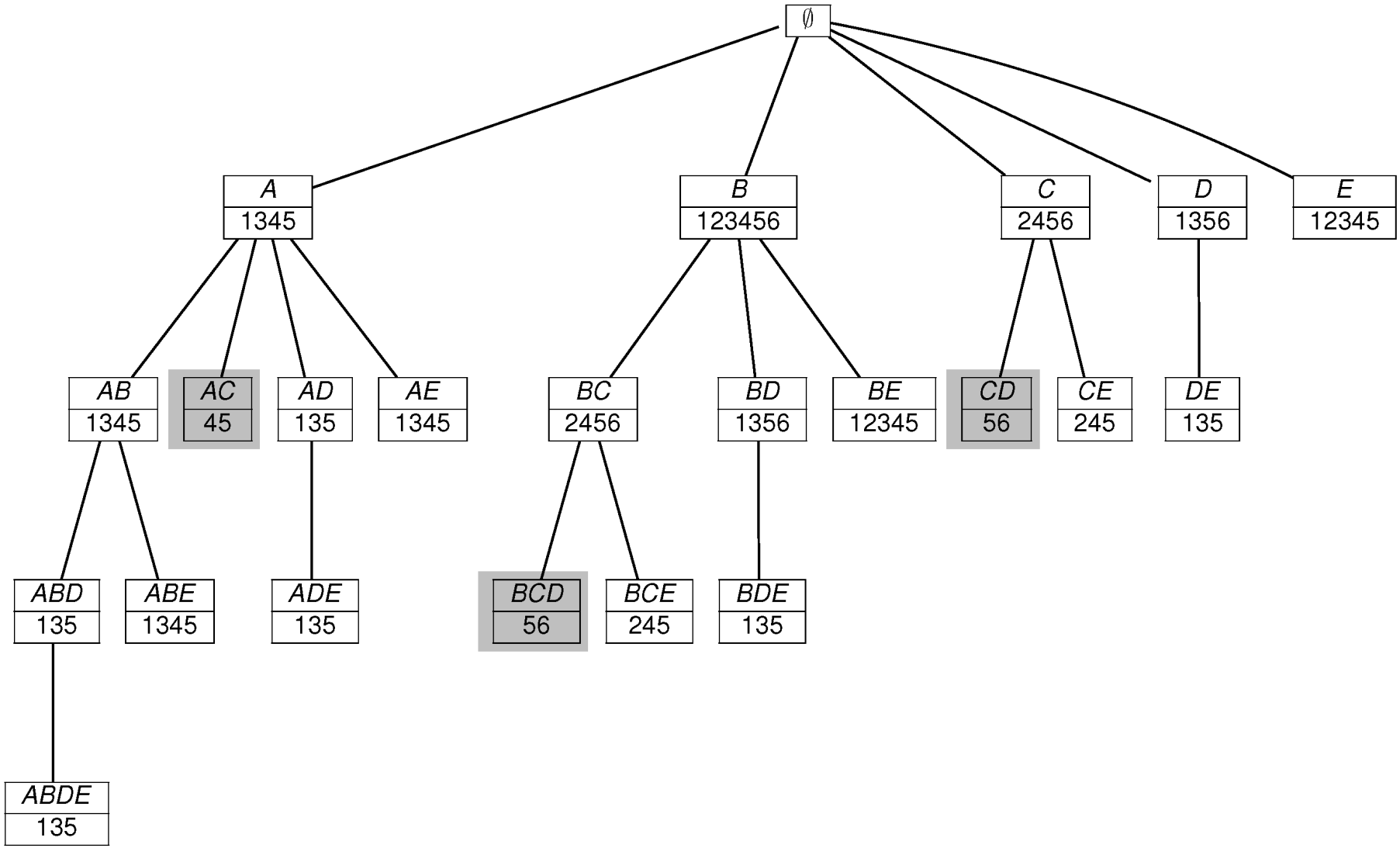
t	**i**(*t*)
1	*ABDE*
2	*BCE*
3	*ABDE*
4	*ABCE*
5	*ABCDE*
6	*BCD*

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			

Vertical 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
- It may need to generate 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

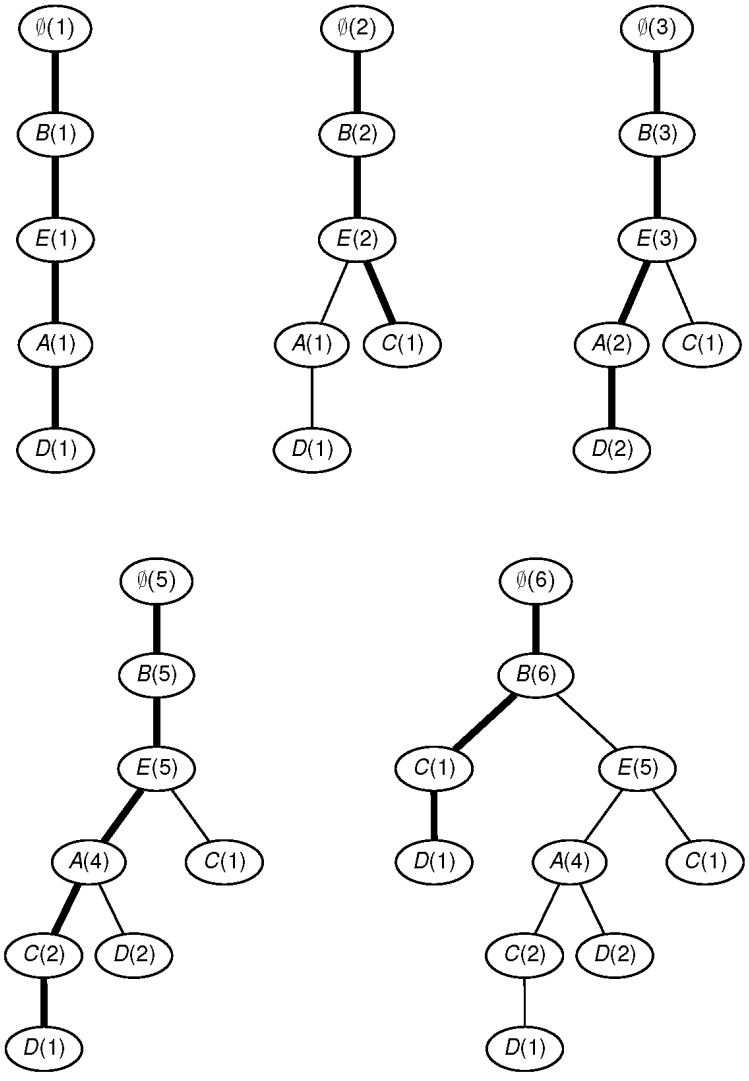
The FP-Growth Algorithm

- 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 the frequent pattern tree
  - For each frequent item i compute the projected FP-tree
  - 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

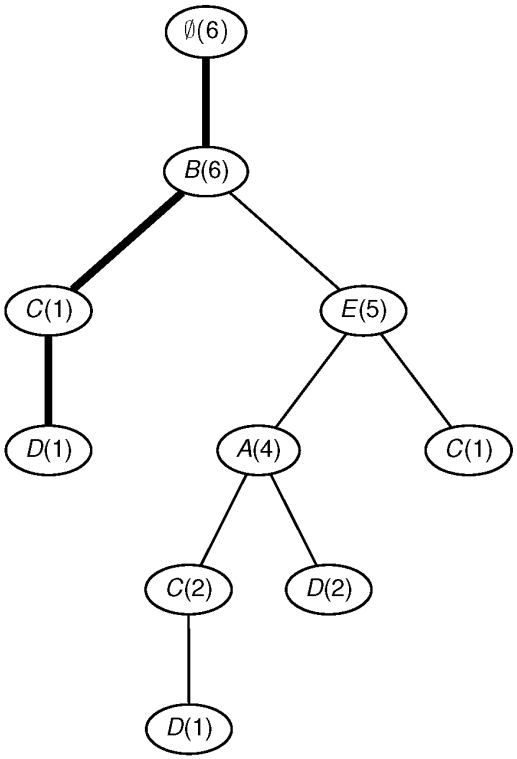
Frequent Pattern Tree

Transactions
BEAD
BEC
BEAD
BEAC
BEACD
BCD

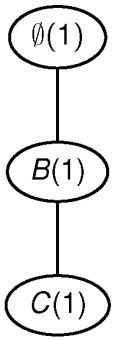


Conditional FP-tree for D

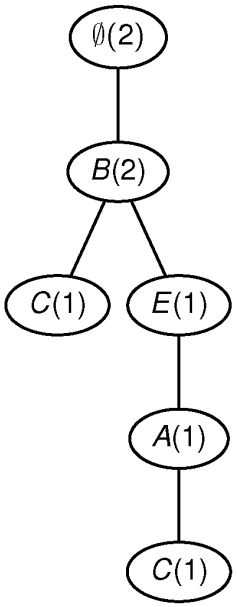
FP-Tree



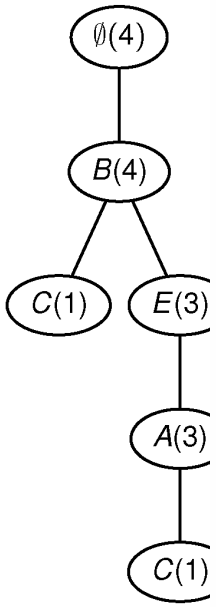
Add BC, $cnt = 1$



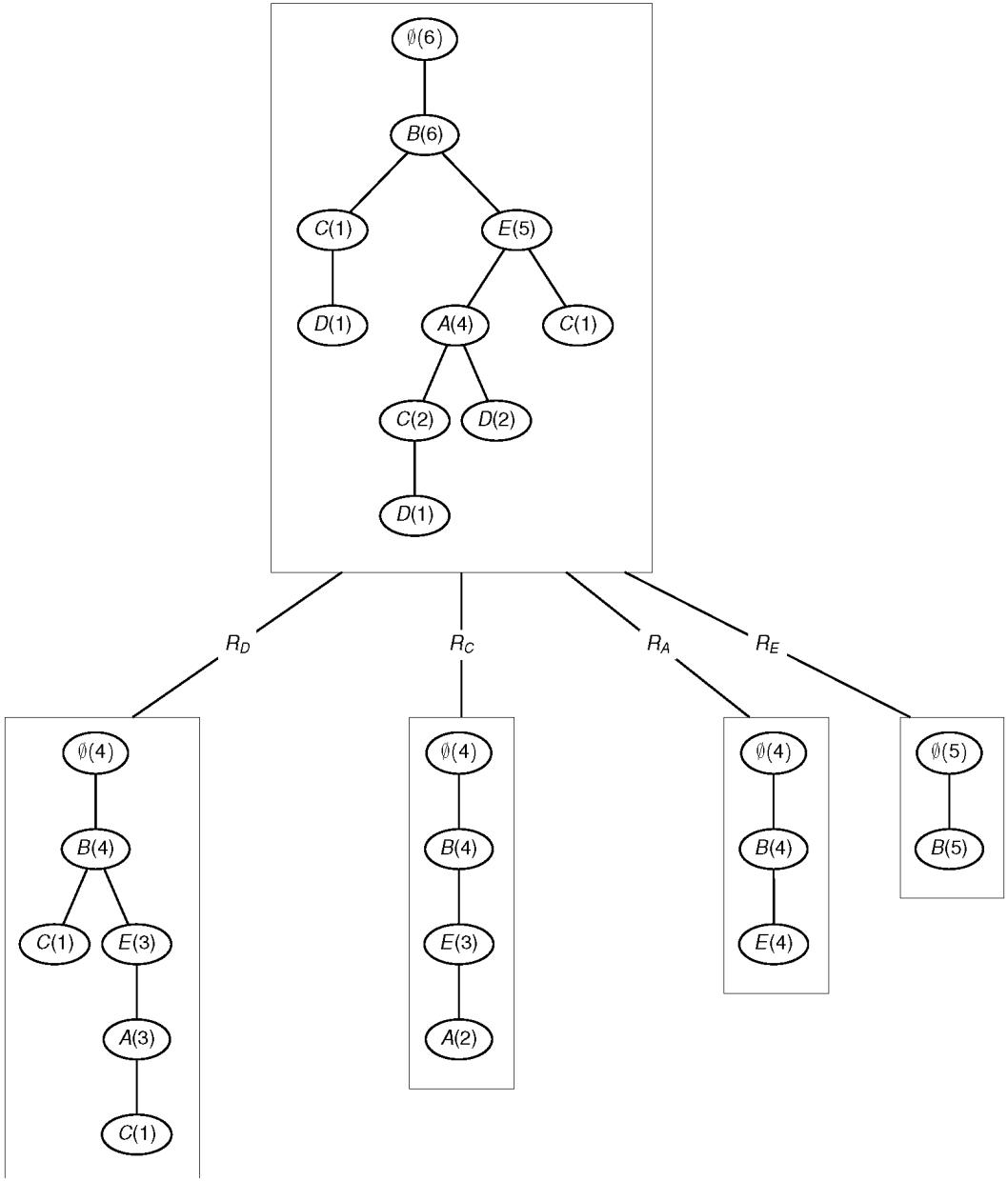
Add $BEAC$, $cnt = 1$



Add BEA, $cnt = 2$



All Conditional FP-Trees



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)

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

Rule Generation

Rule Generation

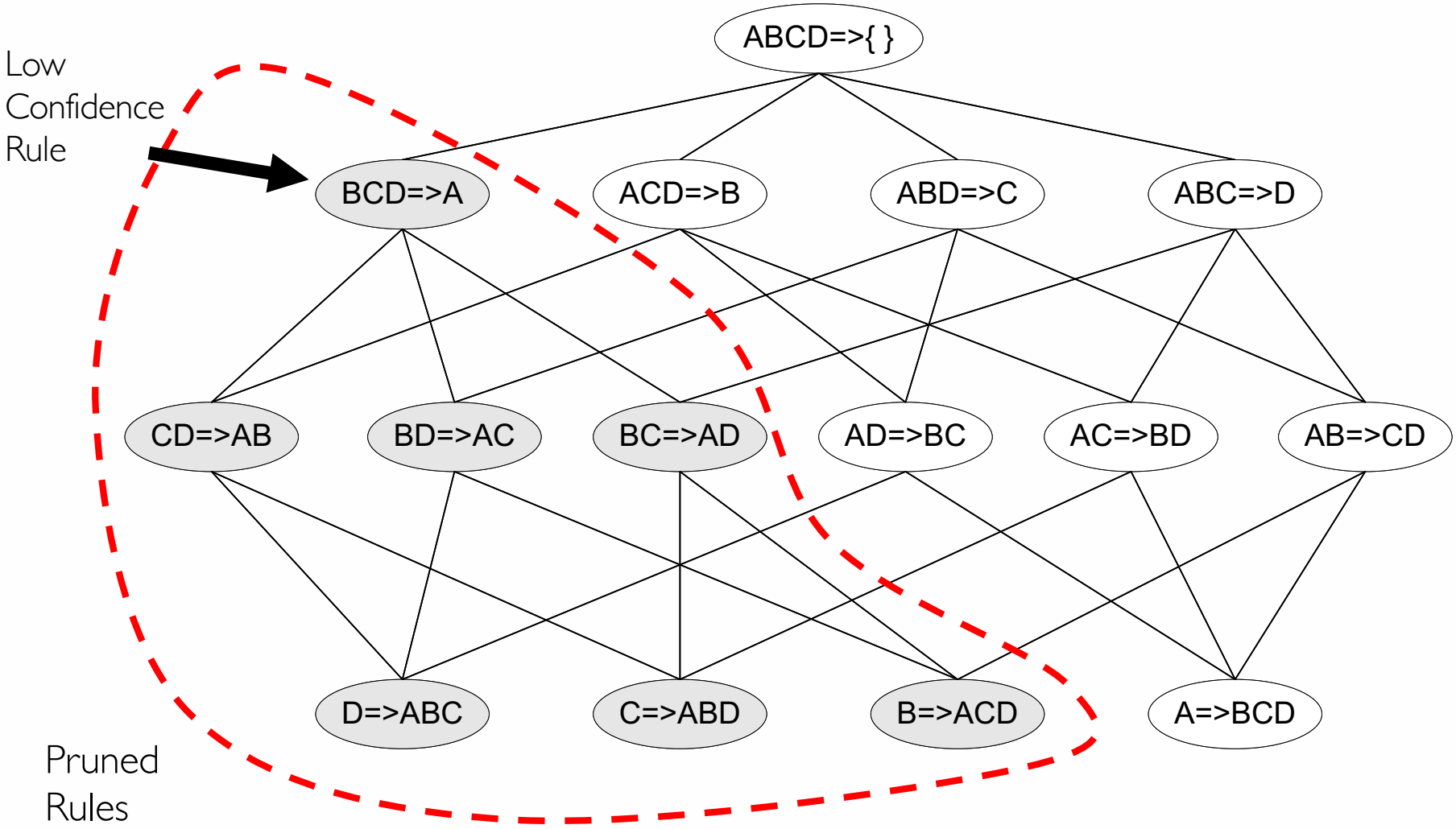
- Given a frequent itemset L, find all non-empty subsets $f \subset L$ such that $f \Rightarrow L - f$ satisfies the minimum confidence requirement
- If {A,B,C,D} is a frequent itemset, candidate rules:
  $ABC \Rightarrow D$, $ABD \Rightarrow C$, $ACD \Rightarrow B$, $BCD \Rightarrow A$, $A \Rightarrow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

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

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)
- However, 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



Association Rules Generation

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

foreach $Z \in \mathcal{F}$, *such that* $|Z| \geq 2$ **do**
  $\mathcal{A} \leftarrow \{X \mid X \subset Z, X \neq \emptyset\}$
  while $\mathcal{A} \neq \emptyset$ **do**
    $X \leftarrow$ maximal element in $\mathcal{A}$
    $\mathcal{A} \leftarrow \mathcal{A} \setminus X$ // remove X from $\mathcal{A}$
    $c \leftarrow sup(Z)/sup(X)$
    if $c \geq minconf$ **then**
      print $X \longrightarrow Y$, $sup(Z)$, c
    else
      $\mathcal{A} \leftarrow \mathcal{A} \setminus \{W \mid W \subset X\}$
      // remove all subsets of X from $\mathcal{A}$

Example of Rule Generation (1)

- Suppose that we computed the following frequent itemsets and we have a minconf of 0.9

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

- We start from the largeset itemset (ABDE) and consider all the possible subsets $\mathcal{A}$ = {ABDE(3), ABD(3), ABE(4), ADE(3), BDE(3), AB(4), AD(3), AE(4), BD(4), BE(5), DE(3), A(4), B(6), D(4), E(5)}

Example of Rule Generation (2)

- The first subset is X = ABD, and the confidence of ABD ⇒ E is 3/3 = 1.0, so we output the rule.
- The next subset is X = ABE, but the corresponding rule ABE ⇒ D is not strong since c(ABE ⇒ D) = 3/4 = 0.75
- We can thus remove from $\mathcal{A}$ all subsets of ABE
- The updated set of antecedents is therefore $\mathcal{A}$ = {ADE(3), BDE(3), AD(3), BD(4), DE(3), D(4)}
- When the algorithm ends, it will output the rules, ABD ⇒ E conf=1.0; ADE ⇒ B conf=1.0; BDE ⇒ A, conf=1.0; AD ⇒ BE conf=1.0; DE ⇒ AB conf=1.0

Rule Assessment Measures

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

Lift

- Lift is the ratio of the observed joint probability of X and Y to the expected joint probability if they were statistically independent,

  $lift(X \Rightarrow Y) = sup(XUY)/sup(X)sup(Y) = conf(X \Rightarrow Y)/sup(Y)$

- Lift is a measure of the deviation from stochastic independence (if it is 1 then X and Y are independent)

- Lift also measure the surprise of the rule. A lift close to 1 means that the support of a rule is expected considering the supports of its components.

- We typically look for values of lift that are much larger (i.e., above expectation) or much smaller (i.e., below expectation) than one.

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 Z cannot be frequent
- However, M alone cannot be used to determine sup(X), we can only use to have a lower-bound, that is, $sup(X) >= sup(Z)$ 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,

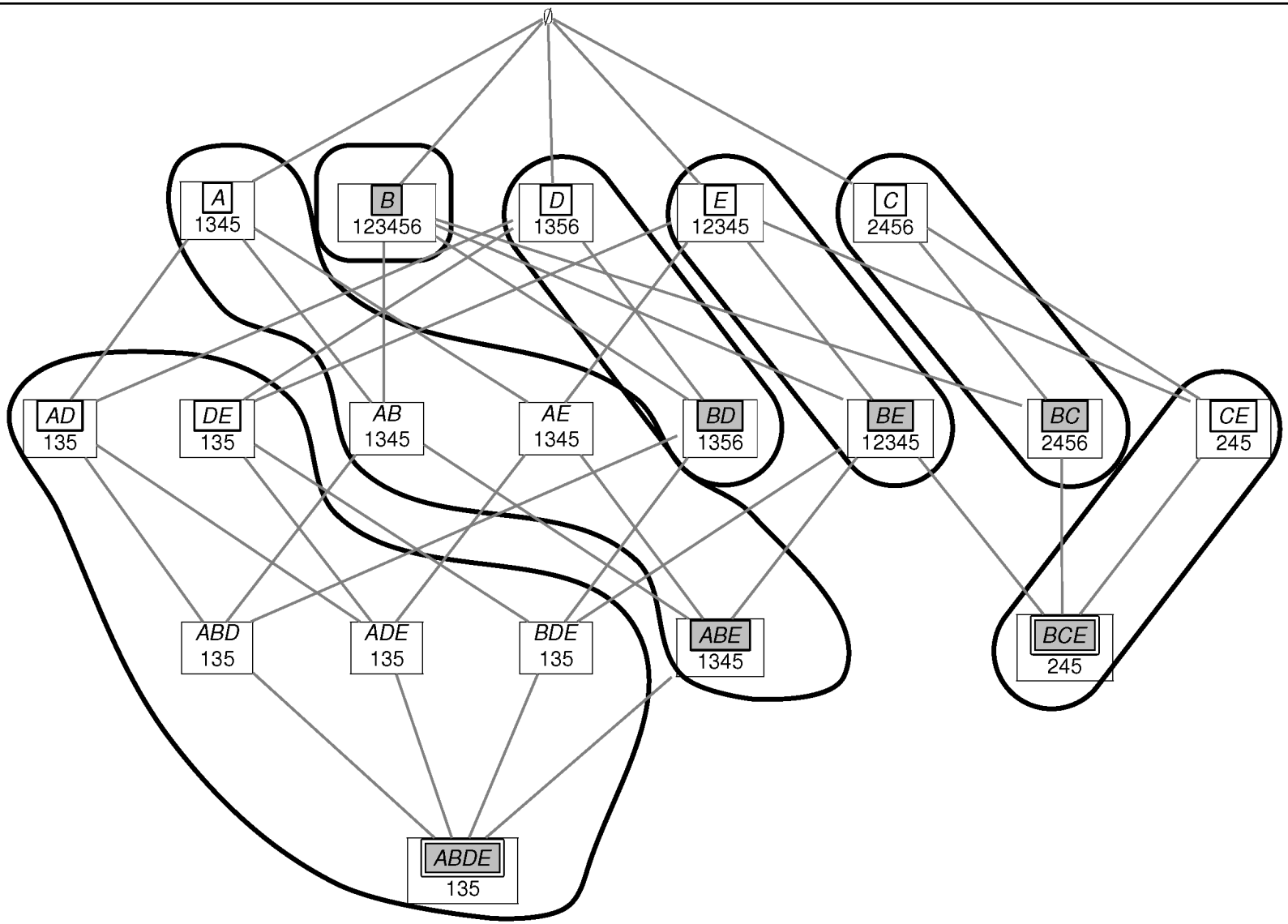
$$sup(X) < sup(Y)$$

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

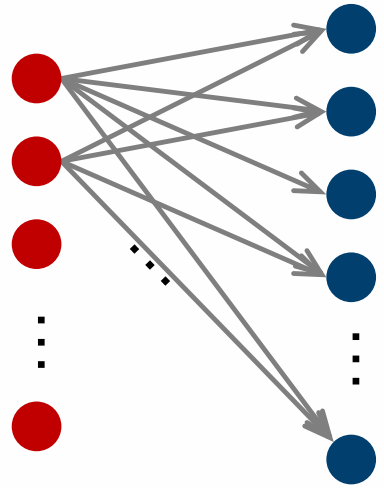
Simple boxes are generators

Shaded boxes with double lines are maximal itemsets

Searching for Small Communities (Trawling)

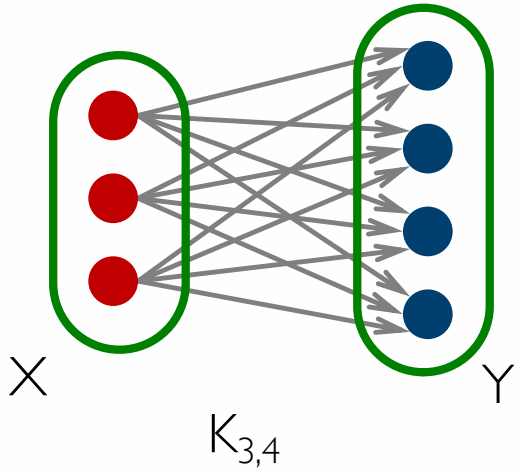
Searching for small communities in the Web graph (Trawling)

- Trawling
  - What is the signature of a community in a Web graph?
  - The underlying intuition, that small communities involve many people talking about the same things
  - Use this to define "topics": what the same people on the left talk about on the right?



Dense 2-layer

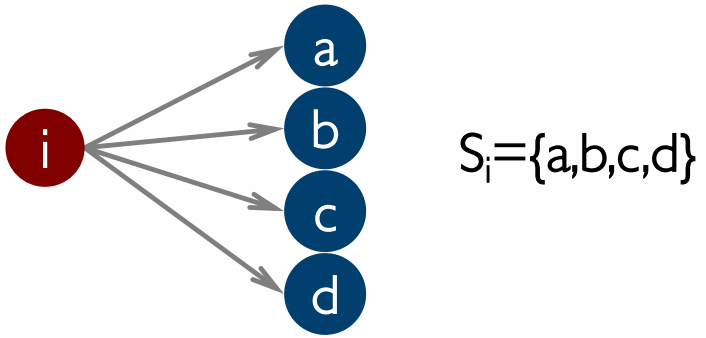
- More formally
  - Enumerate complete bipartite subgraphs $K_{s,t}$
  - $K_{s,t}$ has s nodes on the "left" and t nodes on the "right"
  - The left nodes link to the same node of on the right, forming a fully connected bipartite graph

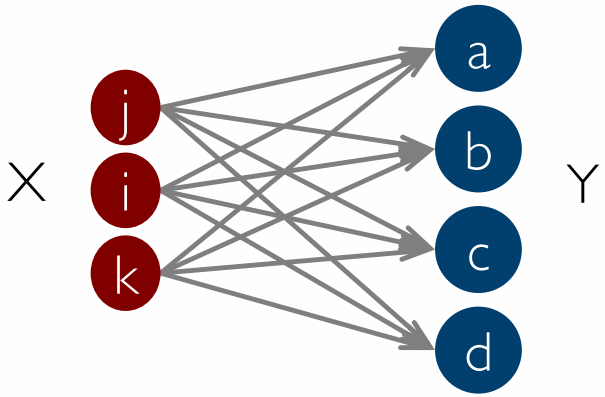


$K_{3,4}$

Mining Bipartite $K_{s,t}$ using Frequent Itemsets

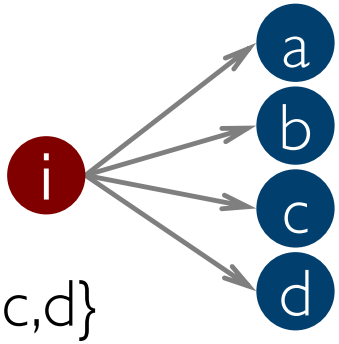
- Searching for such complete bipartite graphs can be viewed as a frequent itemset mining problem
- View each node i as a set S_i of nodes i points to
- $K_{s,t}$ = a set Y of size t that occurs in s sets S_i
- Looking for $K_{s,t}$ is equivalent to setting the frequency threshold to s and look at layer t (i.e., all frequent sets of size t)





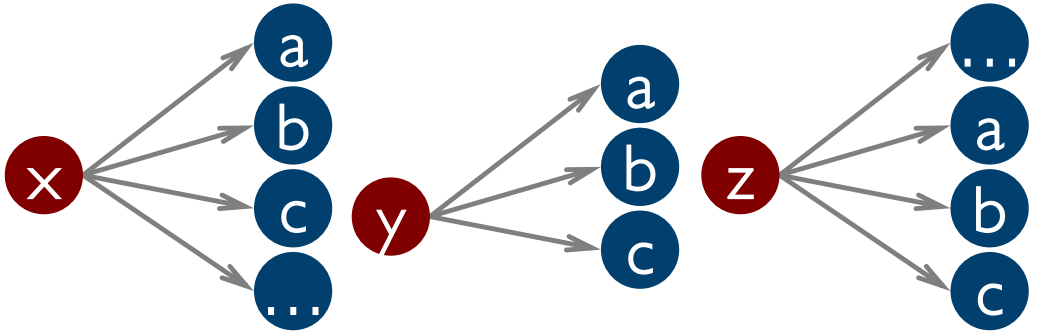
s = minimum support (|X|=s)
t = itemset size (|Y|=t)

View each node **i** as a set S_i of nodes **i** points to



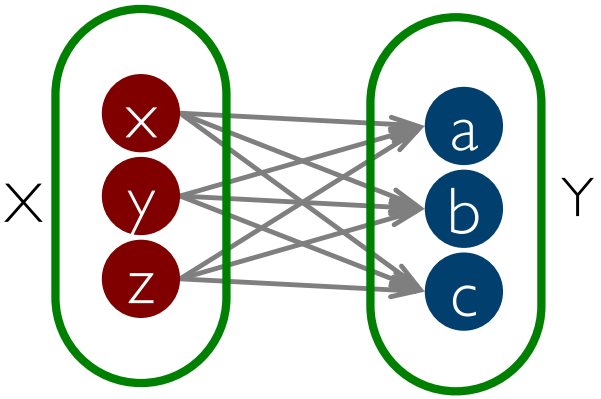
S_i={a,b,c,d}

Suppose we find a frequent itemset Y={a,b,c} of supp s; then, there are s nodes that link to all of {a,b,c}:

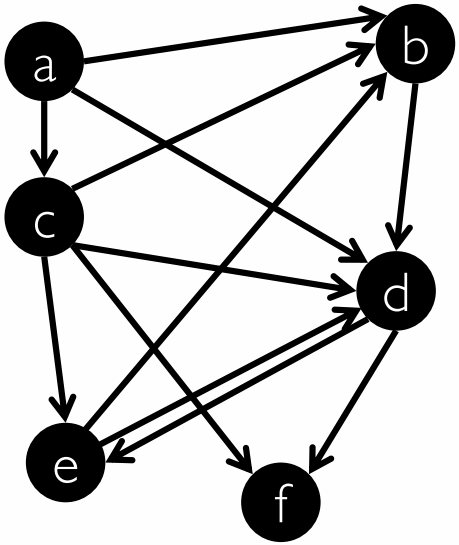


Find frequent itemsets with support s and itemset size t is equivalent to find $K_{s,t}$

$K_{s,t}$ = a set Y of size t that occurs in s sets S_i

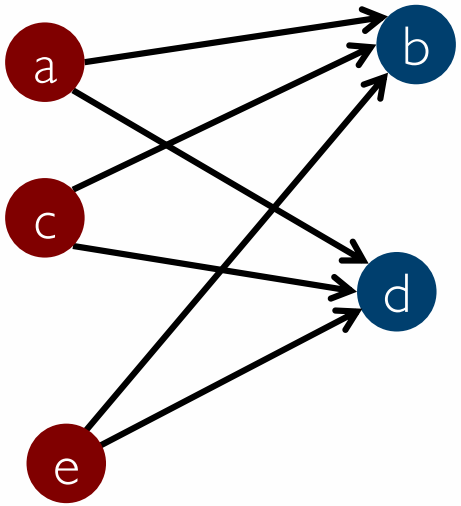


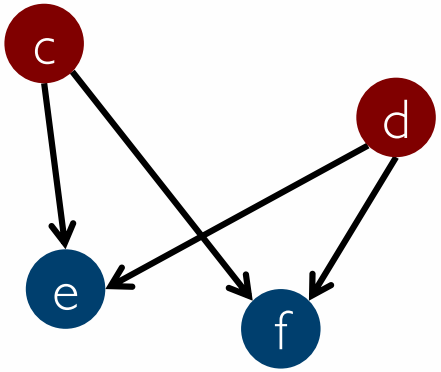
Example



- Support threshold s=2
  - {b,d}: support 3
  - {e,f}: support 2
- And we just found 2 bipartite subgraphs:

- Itemsets
  - a = {b,c,d}
  - b = {d}
  - c = {b,d,e,f}
  - d = {e,f}
  - e = {b,d}
  - f = {}





Mining Frequent Sequences

Sequence Mining: Terminology

- A sequence is an ordered list of symbols $\mathbf{s} = s_1, s_2, \ldots s_k$ where s_i or $\mathbf{s}[i]$ denote the symbol at position i
- Notation $\mathbf{s}[i:j]$ denotes $s_{i+1}, s_{i+2}, \ldots s_j$
- Let $\mathbf{s} = s_1, s_2, \ldots, s_n$ and $\mathbf{r} = r_1, r_2, \ldots, r_m$ we say that $\mathbf{r}$ is a subsequence of s denoted $r \subseteq s$ if there is a function ϕ from [1,m] to [1,n] such that $r[i] = s[\phi(i)]$ and for any position i, j in r,

$$i<j \Rightarrow \phi(i)<\phi(j)$$

- $\mathbf{r}$ is a consecutive subsequence when

$$r_1, r_2, \ldots, r_m = s_j, s_{j+1}, \ldots, s_{j+m-1}$$

Example

- Given **s** = ACTGAACG
- r_1 = CGAAG is a subsequence of **s**
- r_2 = CTGA is a consecutive subsequence of **s**
- r_3 = ACT is a prefix of **s**
- r_4 = GAACG is a suffix of **s**

Frequent Sequences

- Given a database D containing N sequences, the support of a sequence **r** in D is defined as the total number of sequences in D that contains **r**

$$sup(\mathbf{r}) = |\{\mathbf{s}_i \in \mathbf{D} | \mathbf{r} \subseteq s_i\}|$$

- The relative support of r is the percentage of sequences that contain r,

$$rsup(\mathbf{r}) = sup(\mathbf{r})/N$$

- Given a minsup threshold we that r is frequent if sup(**r**)>=minsup

- Note that, in sequence mining, we need to consider all possible permutations of the items not just the combinations

Example

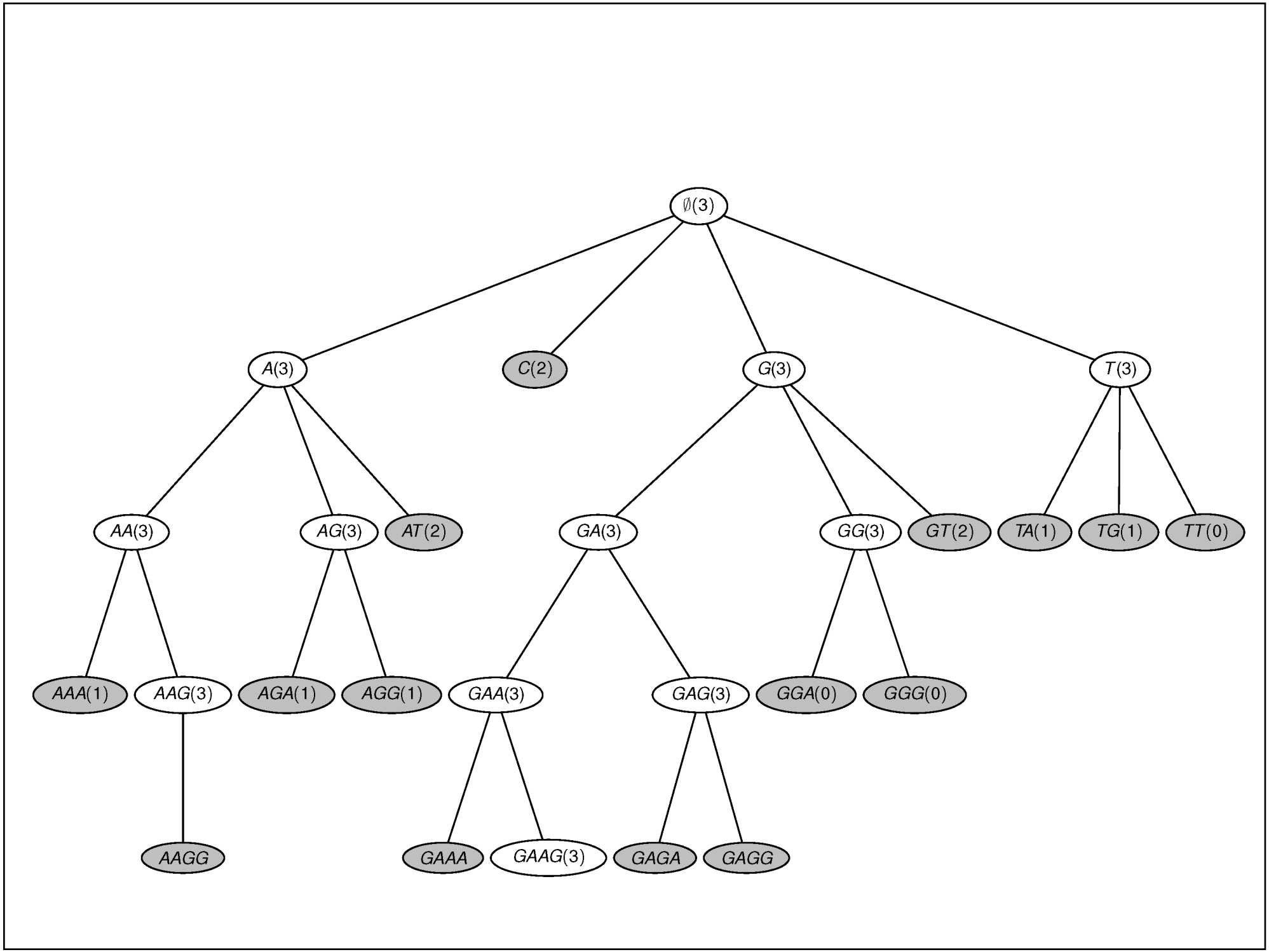
- Given the following sequence database

Id	Sequence
s_1	*CAGAAGT*
s_2	*TGACAG*
s_3	*GAAGT*

- With a minsup of 3, the set of frequent subsequences is
  - A(3), G(3), T(3)
  - AA(3), AG(3), GA(3), GG(3)
  - AAG(3), GAA(3), GAG(3)
  - GAAG(3)

Level-Wise Sequence Mining: GSP Algorithm

- Searches the sequence prefix tree using a level-wise (breath-first search).
- Given the set of frequent sequences at level k, the algorithm generates the candidate for level k+1 and compute the support of each candidate and prune the not frequent ones.
- For each sequence s_i in D, we check if a candidate **r** is a subsequence of **s**, if it is the support of r is incremented. Once the frequent sequences at level k are computed the k+1 candidates are generated.
- For each leaf r_a, the sequence is extended with the last symbol of any other leaf r_b that shares the same prefix (it has the same parent) so $r_{ab} = r_a + r_b[k]$. If r_{ab} is infrequent, we prune it.


∅(3)
A(3)
C(2)
G(3)
T(3)
AA(3)
AG(3)
AT(2)
GA(3)
GG(3)
GT(2)
TA(1)
TG(1)
TT(0)
AAA(1)
AAG(3)
AGA(1)
AGG(1)
GAA(3)
GAG(3)
GGA(0)
GGG(0)
AAGG
GAAA
GAAG(3)
GAGA
GAGG

The GSP Algorithm

GSP (D, Σ, *minsup*):

1 $\mathcal{F} \leftarrow \emptyset$
2 $\mathcal{C}^{(1)} \leftarrow \{\emptyset\}$ // Initial prefix tree with single symbols
3 **foreach** $s \in \Sigma$ **do** Add s as child of $\emptyset$ in $\mathcal{C}^{(1)}$ with $sup(s) \leftarrow 0$
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* **s** $\in \mathcal{C}^{(k)}$ **do**
8 **if** $sup(\mathbf{r}) \geq minsup$ **then** $\mathcal{F} \leftarrow \mathcal{F} \cup \{(\mathbf{r}, sup(\mathbf{r}))\}$
9 **else** remove **s** from $\mathcal{C}^{(k)}$
10 $\mathcal{C}^{(k+1)} \leftarrow$ EXTENDPREFIXTREE ($\mathcal{C}^{(k)}$)
11 $k \leftarrow k + 1$
12 **return** $\mathcal{F}^{(k)}$

The GSP Algorithm

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

foreach $\mathbf{s}_i \in \mathbf{D}$ **do**
  foreach $\mathbf{r} \in \mathcal{C}^{(k)}$ **do**
    if $\mathbf{r} \subseteq \mathbf{s}_i$ **then** $sup(\mathbf{r}) \leftarrow sup(\mathbf{r}) + 1$

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

foreach *leaf* $\mathbf{r}_a \in \mathcal{C}^{(k)}$ **do**
  foreach *leaf* $\mathbf{r}_b \in$ CHILDREN(PARENT($\mathbf{r}_a$)) **do**
    $\mathbf{r}_{ab} \leftarrow \mathbf{r}_a + \mathbf{r}_b[k]$ `// extend` $\mathbf{r}_a$ `with last item of` $\mathbf{r}_b$
    `// prune if there are any infrequent subsequences`
    if $\mathbf{r}_c \in \mathcal{C}^{(k)}$, ***for all*** $\mathbf{r}_c \subset \mathbf{r}_{ab}$, *such that* $|\mathbf{r}_c| = |\mathbf{r}_{ab}| - 1$ **then**
      Add $\mathbf{r}_{ab}$ as child of $\mathbf{r}_a$ with $sup(\mathbf{r}_{ab}) \leftarrow 0$
  if *no extensions from* $\mathbf{r}_a$ **then**
    remove $\mathbf{r}_a$, and all ancestors of $\mathbf{r}_a$ with no extensions, from $\mathcal{C}^{(k)}$
return $\mathcal{C}^{(k)}$

Sequential Pattern Mining

Sequential Pattern Mining

- Association rules do not consider the order of transactions, however, in many applications ordering is significant
- In market basket analysis, it is interesting to know whether people buy some items in sequence
- Example
  - Buying bed first and then bed sheets later
  - Navigational patterns of users in a Web site from sequences of page visits of users

Examples of Sequence

- Web sequence
  - < {Homepage} {Electronics} {Digital Cameras} {Canon Digital Camera} {Shopping Cart} {Order Confirmation} {Return to Shopping} >
- Sequence of initiating events causing the accident at 3-mile Island:
  - <{clogged resin} {outlet valve closure} {loss of feedwater} {condenser polisher outlet valve shut} {booster pumps trip} {main waterpump trips} {main turbine trips} {reactor pressure increases}>
- Sequence of books checked out at a library:
  - <{Fellowship of the Ring} {The Two Towers} {Return of the King}>

- Let $I = \{i_1, i_2, \ldots, i_m\}$ be a set of items.
- A sequence is an ordered list of itemsets.
- We denote a sequence s by $\langle a_1, a_2, \ldots, a_r \rangle$, where a_i is an itemset, also called an element of s.
- An element (or an itemset) of a sequence is denoted by $\{x_1, x_2, \ldots, x_k\}$, where $x_j \in I$ is an item.
- We assume without loss of generality that items in an element of a sequence are in lexicographic order

Sequential Pattern Mining

- Sequence databases consist of sequences of ordered elements or events (with or without time)
- Sequential Pattern Mining is the mining of frequently occurring ordered events or subsequences as patterns
- Sequences consists of nominal (categorical) data. When sequence are based on numerical data, then we apply time series analysis

- The size of a sequence is the number of elements (or itemsets) in the sequence
- The length of a sequence is the number of items in the sequence (a sequence of length k is called k-sequence)
- Given two sequences, $s_1 = \langle a_1, a_2, \ldots, a_r \rangle$ and $s_2 = \langle b_1, b2, \ldots, b_v \rangle$
- s_1 is a subsequence of s_2 if there exist integers

$$1 \leq j_1 < j_2 < \cdots < j_r \leq v$$

such that

$$a_1 \subseteq b_{j_1} \wedge a_2 \subseteq b_{j_2} \wedge \cdots \wedge a_r \subseteq b_{j_v}$$

Example

- Let I = {1, 2, 3, 4, 5, 6, 7, 8, 9}.
- The sequence ⟨{3}{4, 5}{8}⟩ is contained in (or is a subsequence of) ⟨{6} {3, 7}{9}{4, 5, 8}{3, 8}⟩
- In fact, {3} ⊆ {3, 7}, {4, 5} ⊆ {4, 5, 8}, and {8} ⊆ {3, 8}
- However, ⟨{3}{8}⟩ is not contained in ⟨{3, 8}⟩ or vice versa
- The size of the sequence ⟨{3}{4, 5}{8}⟩ is 3, and the length of the sequence is 4.

Goal of Sequential Pattern Mining

- The input is a set S of input data sequences (or sequence database)
- The problem of mining sequential patterns is to find all the sequences that have a user-specified minimum support
- Each such sequence is called a frequent sequence, or a sequential pattern
- The support for a sequence is the fraction of total data sequences in S that contains this sequence

Terminology

- Itemset
  - Non-empty set of items
  - Each itemset is mapped to an integer
- Sequence
  - Ordered list of itemsets
- Support for a Sequence
  - Fraction of total customers that support a sequence.
- Maximal Sequence
  - A sequence that is not contained in any other sequence
- Large Sequence
  - Sequence that meets minisup

Frequent Sequences

- Given a database D $\{s_1, \ldots, s_N\}$ of N sequences and a sequence r, we define the support count of r as

$$sup(r) = |\{s_i \mid r \text{ is a subsequence of } s_i\}|$$

- And the support (or relative support),

$$rsup(r) = sup(r)/N$$

Example

Customer ID	Transaction Time	Items Bought
1	June 25 '93	30
1	June 30 '93	90
2	June 10 '93	10,20
2	June 15 '93	30
2	June 20 '93	40,60,70
3	June 25 '93	30,50,70
4	June 25 '93	30
4	June 30 '93	40,70
4	July 25 '93	90
5	June 12 '93	90

Customer ID	Customer Sequence
1	< (30) (90) >
2	< (10 20) (30) (40 60 70) >
3	< (30) (50) (70) >
4	< (30) (40 70) (90) >
5	< (90) >

Maximal seq with support > 40%
< (30) (90) >
< (30) (40 70) >

Note: Use Minisup of 40%, no less than two customers must support the sequence
< (10 20) (30) > Does not have enough support (Only by Customer #2)
< (30) >, < (70) >, < (30) (40) > … are not maximal.

Example

- Given the following sequence database and a minsup of 2

SID	Sequence
10	<(a)(abc)(ac)(d)(cf)>
20	<(ad)(c)(bc)(ae)>
30	<(ef)(ab)(df)(c)(b)>
40	<(e)(g)(af)(c)(b)(c)>

- <(a)(bc)(d)(c)> is a subsequence of <(a)(abc)(ac)(d)(cf)>

- <(ab)(c)> is a frequent sequence or a sequential pattern

Example

- Given the sequence database

SID	Sequence
10	<(bd)(c)(b)(ac)>
20	<(bf)(ce)(b)(fg)>
30	<(ah)(bf)(a)(b)(f)>
40	<(be)(ce)(d)>
50	<(a)(bd)(b)(c)(b)(ade)>

- Given a minsup of 2, <(bd)cb> is a sequential pattern (or frequent sequence)

Apriori principle still applies

If a sequence S is not frequent,
then every super-sequence of S is not frequent

In the previous example,

Mining Sequential Patterns (Outline of the basic principle)

- Level-by-level do
  - Generate candidate sequences
  - Scan database to collect support counts
  - Use Apriori property to prune candidates
- Only generate candidates satisfying Apriori property

- Limitations
  - It can generate a massive amount of candidates (1000 frequent 1-sequences can generate around 1.5 million candidates)
  - It requires many scans of the database

- Accordingly, as for association rules, there are several prefix-based extensions (e.g. PrefixSpan) that deal with all these limitations

Example

Seq. ID	Sequence
10	<(bd)(c)(b)(ac)>
20	<(bf)(ce)(b)(fg)>
30	<(ah)(bf)(a)(b)(f)>
40	<(be)(ce)(d)>
50	<(a)(bd)(b)(c)(b)(ade)>

1st scan - 8 candidates
6 sequential patterns (length 1)

<a> <b> <c> <d> <e> <f> <g> <h>

2nd scan - 51 candidates
19 sequential patterns (length 2)
10 candidates were not in the database

<aa> <ab> … <af> <ba> <bb> … <ff>
<(ab)> … <(ef)>

3rd scan - 46 candidates
19 sequential patterns (length 3)
20 candidates were not in the database

<abb> <aab> <aba> <baa> <bab> …

Association Rules for Classification

Mining Association Rules for Classification (the CBA algorithm)

- Association rule mining assumes that the data consist of a set of transactions. Thus, the typical tabular representation of data used in classification must be mapped into such a format.
- Association rule mining is then applied to the new dataset and the search is focused on association rules in which the tail identifies a class label

$$X \Rightarrow c_i \text{ (where } c_i \text{ is a class label)}$$

- The association rules are pruned using the pessimistic error-based method used in C4.5
- Finally, rules are sorted to build the final classifier.

The Weather Dataset

Outlook	Temp	Humidity	Windy	Play
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Overcast	Hot	High	False	Yes
Rainy	Mild	High	False	Yes
Rainy	Cool	Normal	False	Yes
Rainy	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Sunny	Mild	High	False	No
Sunny	Cool	Normal	False	Yes
Rainy	Mild	Normal	False	Yes
Sunny	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Rainy	Mild	High	True	No

```
outlook=overcast ==> play=yes
humidity=normal windy=FALSE ==> play=yes
outlook=rainy windy=FALSE ==> play=yes
outlook=sunny humidity=high ==> play=no
outlook=rainy windy=TRUE ==> play=no

(default class is the majority class)
```

Run the notebooks for this lecture

Download and run SPMF library

http://www.philippe-fournier-viger.com/spmf/index.php?link=download.php