CS 512 Data Mining Principles

Mining Frequent Patterns, Association and Correlations

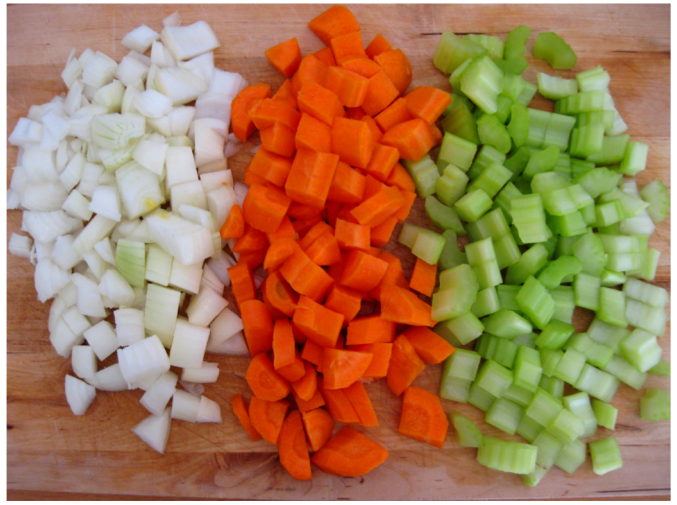
Hanghang Tong, Computer Science, Univ. Illinois at Urbana-Champaign, 2021

Mining Frequent Patterns, Association and Correlations

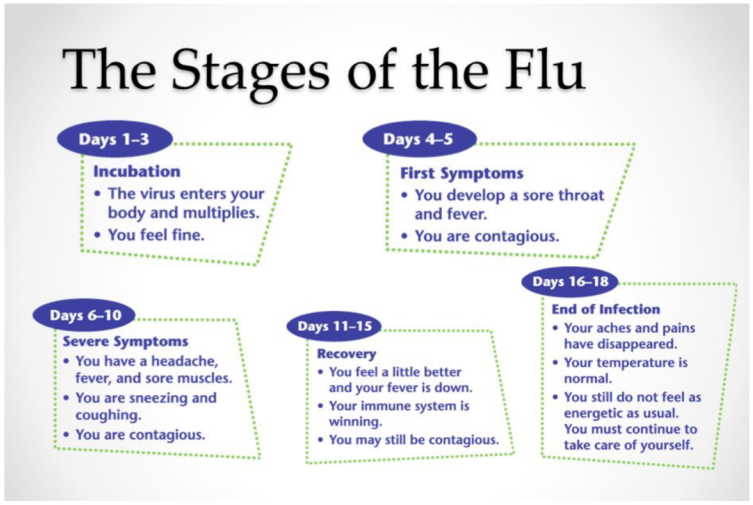
- Basic Techniques 
- Sequential Pattern Mining
- Graph Pattern Mining
- Summary

What are Patterns?

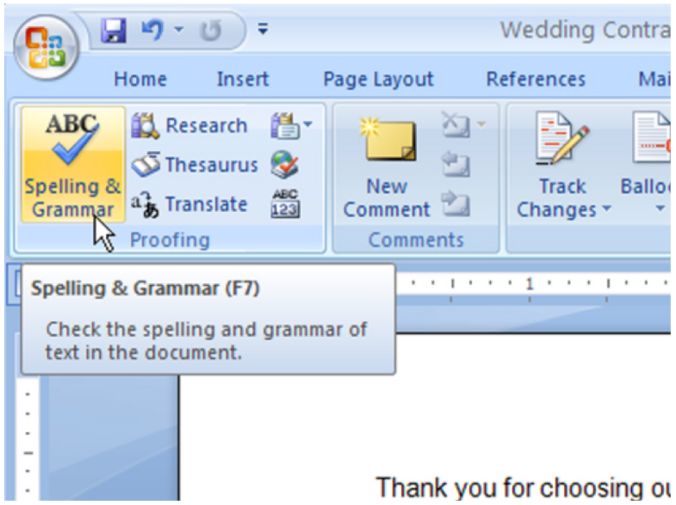
- What are patterns?
  - Patterns: A set of items, subsequences, or substructures that occur frequently together (or strongly correlated) in a data set
  - Patterns represent intrinsic and important properties of datasets



Frequent item set



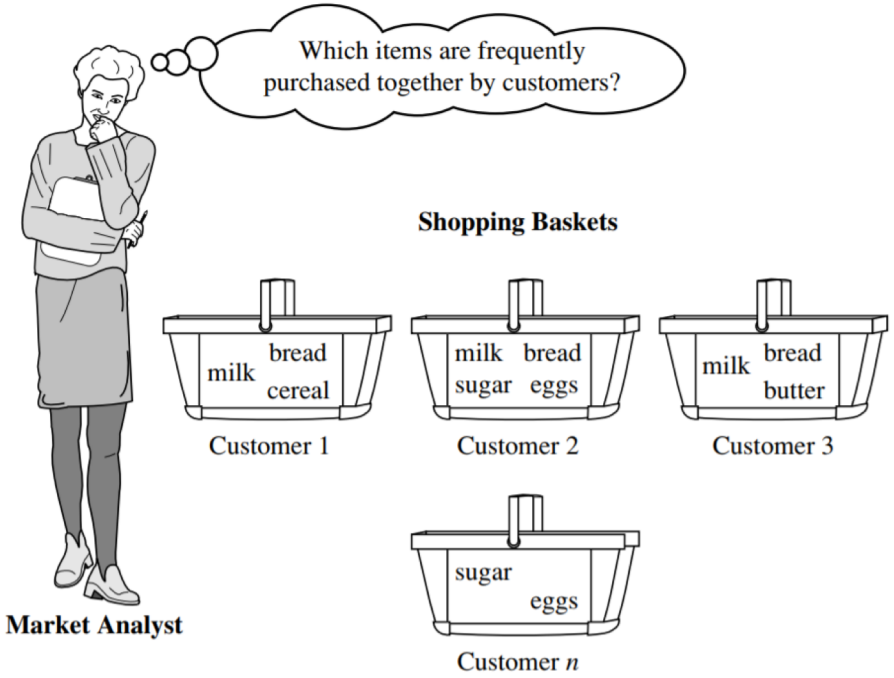
Frequent sequences



Frequent structures

What Is Pattern Discovery?

- Pattern discovery: Uncovering patterns from massive data sets
- It can answer questions such as:
  - What products were often purchased together?
  - What are the subsequent purchases after buying an iPad?



Pattern Discovery: Why Is It Important?

- Foundation for many essential data mining tasks
  - Association, correlation, and causality analysis
  - Mining **sequential**, structural (e.g., sub-graph) patterns
  - **Classification**: Discriminative pattern-based analysis
  - **Cluster** analysis: Pattern-based subspace clustering
- Broad applications
  - Market basket analysis, cross-marketing, catalog design, sale campaign analysis, Web log analysis, biological sequence analysis
  - Many types of data: spatiotemporal, multimedia, time-series, and stream data

Basic Concepts: Transactional Database

- Transactional Database (TDB)
  - Each transaction is associated with an identifier, called a TID.
  - May also have counts associated with each item sold

Tid	Items bought
1	Beer, Nuts, Diaper
2	Beer, Coffee, Diaper
3	Beer, Diaper, Eggs
4	Nuts, Eggs, Milk
5	Nuts, Coffee, Diaper, Eggs, Milk

Basic Concepts: k-Itemsets and Their Supports

- Itemset: A set of one or more items

$$I = \{I_1, I_2, \cdots, I_m\}$$

- k-itemset: An itemset containing k items:

  $X = \{x_1, \ldots, x_k\}$

  - Ex. {Beer, Nuts, Diaper} is a 3-itemset

- Absolute support (count)
  - sup{X} = occurrences of an itemset X
    - Ex. sup{Beer} = 3
    - Ex. sup{Diaper} = 4
    - Ex. sup{Beer, Diaper} = 3
    - Ex. sup{Beer, Eggs} = 1

Tid	Items bought
1	Beer, Nuts, Diaper
2	Beer, Coffee, Diaper
3	Beer, Diaper, Eggs
4	Nuts, Eggs, Milk
5	Nuts, Coffee, Diaper, Eggs, Milk

- Relative support
  - *s*{*X*} = The fraction of transactions that contains X (i.e., the probability that a transaction contains X)
  - Ex. s{Beer} = 3/5 = 60%
  - Ex. s{Diaper} = 4/5 = 80%
  - Ex. s{Beer, Eggs} = 1/5 = 20%

Basic Concepts: Frequent Itemsets (Patterns)

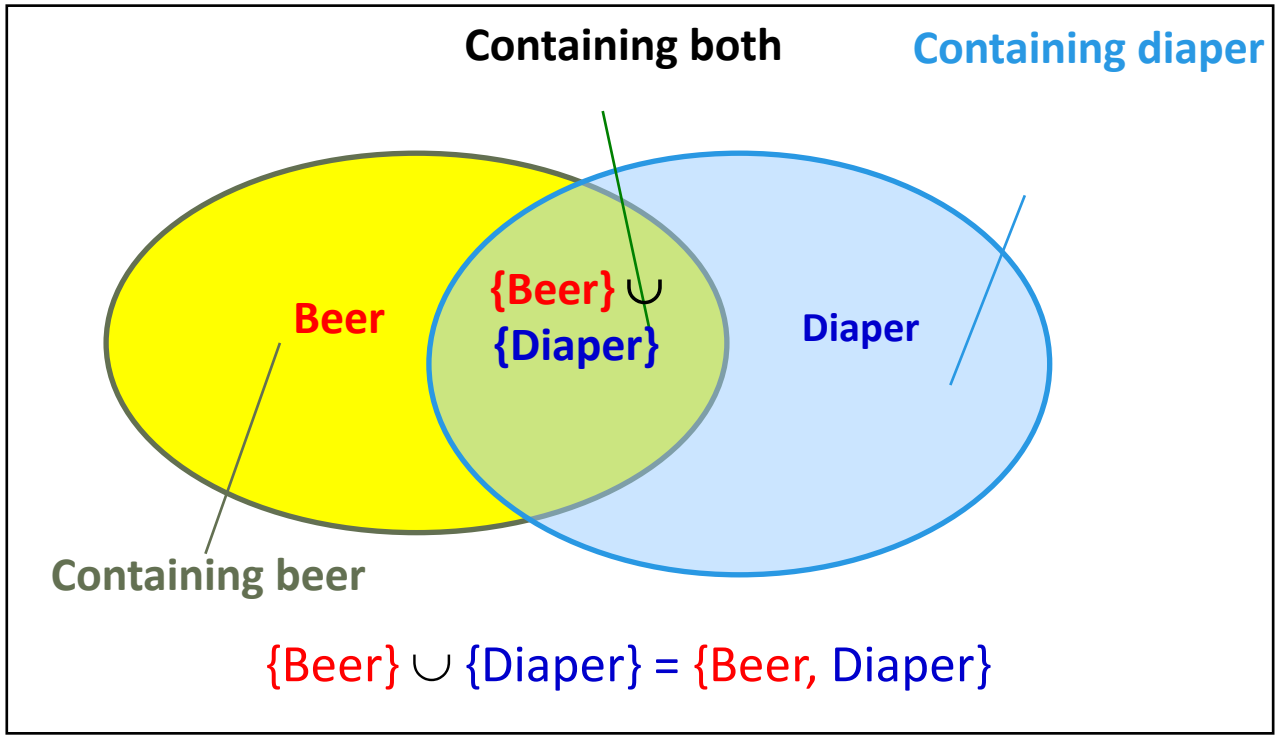
- An itemset (or a pattern) X is *frequent* if the support of X is no less than a *minsup* threshold σ
- Let σ = *50%* (σ: *minsup* threshold) for the given 5-transaction dataset
  - All the frequent 1-itemsets:
    - Beer: 3/5 (60%); Nuts: 3/5 (60%); Diaper: 4/5 (80%); Eggs: 3/5 (60%)
  - All the frequent 2-itemsets:
    - {Beer, Diaper}: 3/5 (60%)
  - All the frequent 3-itemsets?
    - None

Tid	Items bought
1	Beer, Nuts, Diaper
2	Beer, Coffee, Diaper
3	Beer, Diaper, Eggs
4	Nuts, Eggs, Milk
5	Nuts, Coffee, Diaper, Eggs, Milk

- Why do these itemsets (shown on the left) form the complete set of frequent *k*-itemsets (patterns) for any *k*?
- **Observation**: We may need an efficient method to mine a complete set of frequent patterns

From Frequent Itemsets to Association Rules

- Compared with itemsets, association rules can be more telling
  - Ex. *Diaper → Beer*
    - *Buying diapers may likely lead to buying beers*



Note: X ∪ Y: the union of two itemsets

- The set contains both X and Y

Association Rules

- How do we compute the strength of an association rule $X \rightarrow Y$ (Both X and Y are itemsets)?
- We first compute the following two metrics, s and c.
  - Support of $X \cup Y$
    - Ex. s{Diaper, Beer} = 3/5 = 0.6 (i.e., 60%)
  - Confidence of $X \rightarrow Y$
    - The *conditional probability* that a transaction containing X also contains *Y:*

      $c = \sup(X, Y) / \sup(X)$
    - Ex. c = sup{Diaper, Beer}/sup{Diaper} = ¾ = 0.75

Tid	Items bought
1	Beer, Nuts, Diaper
2	Beer, Coffee, Diaper
3	Beer, Diaper, Eggs
4	Nuts, Eggs, Milk
5	Nuts, Coffee, Diaper, Eggs, Milk

- In pattern analysis, we are often interested in those rules that dominate the database, and these two metrics ensure the popularity and correlation of X and Y.

Mining Frequent Itemsets and Association Rules

- **Association rule mining**
  - Given two thresholds: *minsup, minconf*
  - Find all of the rules, $X \rightarrow Y$ (s, c) such that s ≥ *minsup* and $c \geq minconf$
- Let *minsup = 50%*
  - Freq. 1-itemsets: Beer: 3, Nuts: 3, Diaper: 4, Eggs: 3
  - Freq. 2-itemsets: {Beer, Diaper}: 3
- Let *minconf = 50%*
  - *Beer* → *Diaper* (60%, 100%)
  - *Diaper* → *Beer* (60%, 75%)

(Q: Are these all rules?)

Tid	Items bought
1	Beer, Nuts, Diaper
2	Beer, Coffee, Diaper
3	Beer, Diaper, Eggs
4	Nuts, Eggs, Milk
5	Nuts, Coffee, Diaper, Eggs, Milk

- **Observations:**
  - Mining association rules and mining frequent patterns are very close problems
  - Scalable methods are needed for mining large datasets

Efficient Pattern Mining Methods

- The Apriori Algorithm
- Mining Frequent Patterns by Exploring Vertical Data Format
- FPGrowth: A Frequent Pattern-Growth Approach

The Downward Closure Property of Frequent Patterns

- **Frequent** itemset: $\{a_1, \ldots, a_{50}\}$
  - Subsets are all **frequent**: $\{a_1\}, \{a_2\}, \ldots, \{a_{50}\}, \{a_1, a_2\}, \ldots, \{a_1, \ldots, a_{49}\}, \ldots$
- Downward closure (Apriori): Any subset of a frequent itemset must be frequent
  - If **{beer, diaper, nuts}** is frequent, so is **{beer, diaper}**
  - If ***ANY*** subset of an itemset S is **infrequent**, then there is no chance for S to be frequent.

A sharp knife for pruning!

Apriori Pruning and Scalable Mining Methods

- Apriori pruning principle: If there is any itemset which is infrequent, its superset should not even be generated! (Agrawal & Srikant @VLDB'94, Mannila, et al. @ KDD' 94)
- Scalable mining Methods: Three major approaches
  - Level-wise, join-based approach: Apriori (Agrawal & Srikant@VLDB'94)
  - Vertical data format approach: Eclat (Zaki, Parthasarathy, Ogihara, Li @KDD'97)
  - Frequent pattern projection and growth: FPgrowth (Han, Pei, Yin @SIGMOD'00)

Apriori: A Candidate Generation & Test Approach

- Outline of Apriori (level-wise, candidate generation and test)
  - Scan DB once to get frequent 1-itemset
  - Repeat
    - Generate length-(k+1) candidate itemsets from length-k frequent itemsets
    - Test the candidates against DB to find frequent (k+1)-itemsets
    - Set k := k +1
  - Until no frequent or candidate set can be generated
  - Return all the frequent itemsets derived

The Apriori Algorithm (Pseudo-Code)

C_k: Candidate itemset of size k

F_k : Frequent itemset of size k

```
K := 1;
F_k := {frequent items};   // frequent 1-itemset
While (F_k != ∅) do {      // when F_k is non-empty
   C_{k+1} := candidates generated from F_k;  // candidate generation
   Derive F_{k+1} by counting candidates in C_{k+1} with respect to TDB at minsup;
   k := k + 1
   }
return ∪_k F_k             // return F_k generated at each level
```

The Apriori Algorithm—An Example

Database TDB

Tid	Items
10	A, C, D
20	B, C, E
30	A, B, C, E
40	B, E

minsup = 2

C_1

1st scan

Itemset	sup
{A}	2
{B}	3
{C}	3
{D}	1
{E}	3

F_1

Itemset	sup
{A}	2
{B}	3
{C}	3
{E}	3

C_2

Itemset
{A, B}
{A, C}
{A, E}
{B, C}
{B, E}
{C, E}

2nd scan

C_2

Itemset	sup
{A, B}	1
{A, C}	2
{A, E}	1
{B, C}	2
{B, E}	3
{C, E}	2

F_2

Itemset	sup
{A, C}	2
{B, C}	2
{B, E}	3
{C, E}	2

C_3

Itemset
{B, C, E}

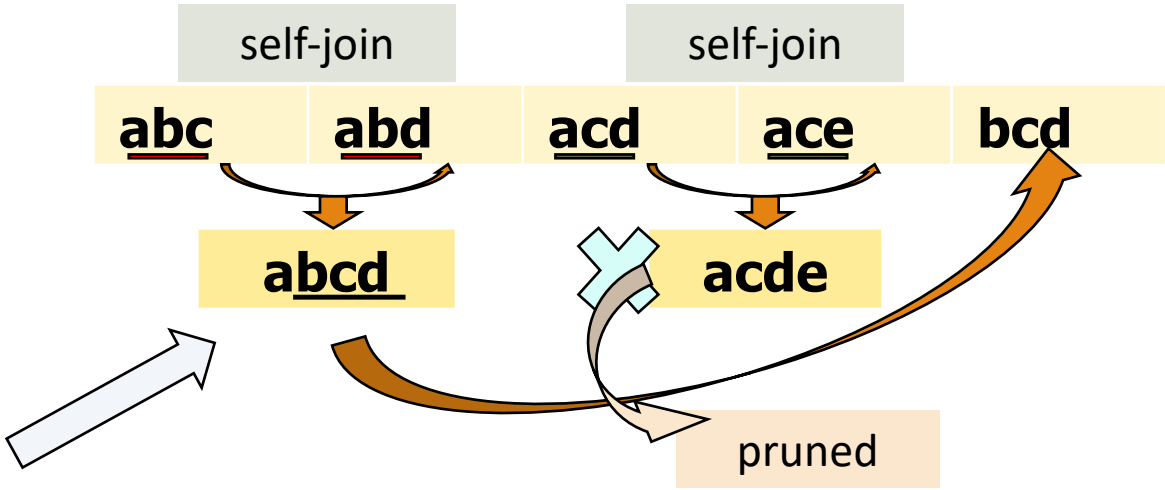
3rd scan

F_3

Itemset	sup
{B, C, E}	2

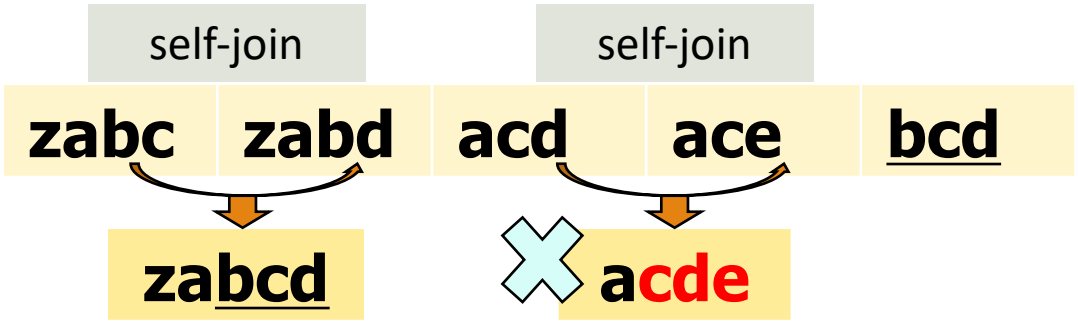
Apriori: Implementation Tricks

- How to generate candidates?
  - Step 1: self-joining F_k
  - Step 2: pruning
- Example of candidate-generation
  - F_3 = {*abc, abd, acd, ace, bcd*}
  - Self-joining: F_3*F_3
    - *abcd* from *abc* and *abd*
    - *acde* from *acd* and *ace*
  - Pruning:
    - *acde* is removed because *ade* is not in F_3
  - C_4 = {*abcd*}



Candidate Generation (Pseudo-Code)

- Suppose the items in F_{k-1} are listed in an order
- // Step 1: Joining



for each p **in** F_{k-1}

for each q **in** F_{k-1}

if $p.item_1 = q.item_1, \ldots, p.item_{k-2} = q.item_{k-2}, p.item_{k-1} < q.item_{k-1}$ **{**

c = **join**(p, q)

- // Step 2: pruning

if has_infrequent_subset(c, F_{k-1})

continue // prune

else add c **to** C_k

}

Exploring Vertical Data Format: ECLAT

- ECLAT (Equivalence Class Transformation): A **depth-first search** algorithm using set intersection [Zaki et al. @KDD'97]
- Vertical format
- Properties of Tid-Lists
  - t(X) = t(Y): X and Y always happen together (e.g., t(ac} = t(d})
  - t(X) ⊂ t(Y): transaction having X always has Y (e.g., t(ac) ⊂ t(ce))
- Frequent patterns: vertical intersections
- Using diffset to accelerate mining
  - Only keep track of differences of tid-Lists
  - t(e) = {T_{10}, T_{20}, T_{30}}, t(ce) = {T_{10}, T_{30}} → Diffset (ce, e) = {T_{20}}

A transaction DB in Horizontal Data Format

Tid	Itemset
10	a, c, d, e
20	a, b, e
30	b, c, e

The transaction DB in Vertical Data Format

Item	TidList
a	10, 20
b	20, 30
c	10, 30
d	10
e	10, 20, 30

t(e) = {T_{10}, T_{20}, T_{30}};
t(a) = {$T_{10,}$ T_{20}};
t(ae) = {$T_{10,}$ T_{20}}

Why Mining Frequent Patterns by Pattern Growth?

- Apriori: A ***breadth-first search*** mining algorithm
  - First find the complete set of frequent k-itemsets
  - Then derive frequent (k+1)-itemset candidates
  - Scan DB again to find true frequent (k+1)-itemsets

Why Mining Frequent Patterns by Pattern Growth?

- Motivation for a different mining methodology
  - Can we develop a *depth-first search* mining algorithm?
  - For a frequent itemset ρ, can subsequent search be confined to only those transactions that contain ρ?
- Such thinking leads to a frequent pattern growth approach:
  - **FPGrowth** (J. Han, J. Pei, Y. Yin, "Mining Frequent Patterns without Candidate Generation," SIGMOD 2000)

Prerequisite: Find frequent 1-itemset

TID	Items in the Transaction
100	{f, a, c, d, g, i, m, p}
200	{a, b, c, f, l, m, o}
300	{b, f, h, j, o, w}
400	{b, c, k, s, p}
500	{a, f, c, e, l, p, m, n}

1. Scan DB once, find single item frequent pattern:

   Let min_support = 3

   f:4, a:3, c:4, b:3, m:3, p:3

2. Sort frequent items in frequency descending order, f-list

   F-list = f-c-a-b-m-p

Example: Construct FP-tree from a Transaction DB

F-list = f-c-a-b-m-p

TID	Items in the Transaction	Ordered, frequent itemlist
100	{f, a, c, d, g, i, m, p}	f, c, a, m, p
200	{a, b, c, f, l, m, o}	f, c, a, b, m
300	{b, f, h, j, o, w}	f, b
400	{b, c, k, s, p}	c, b, p
500	{a, f, c, e, l, p, m, n}	f, c, a, m, p

3. Scan DB again, find the ordered frequent itemlist for each transaction

Example: Construct FP-tree from a Transaction DB

TID	Items in the Transaction	Ordered, frequent itemlist
100	{f, a, c, d, g, i, m, p}	f, c, a, m, p
200	{a, b, c, f, l, m, o}	f, c, a, b, m
300	{b, f, h, j, o, w}	f, b
400	{b, c, k, s, p}	c, b, p
500	{a, f, c, e, l, p, m, n}	f, c, a, m, p

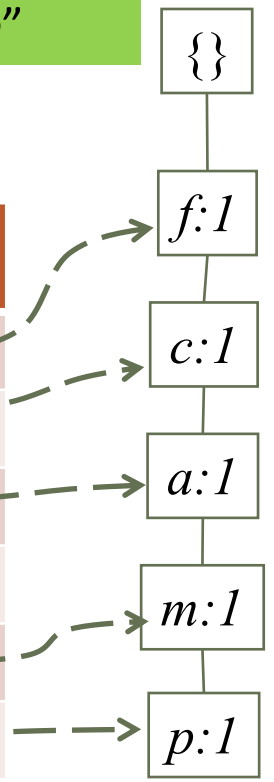
4. For each transaction, insert the ordered frequent itemlist into an FP-tree, with shared sub-branches merged, counts accumulated

F-list = f-c-a-b-m-p

After inserting the 1st frequent Itemlist: "*f, c, a, m, p*"

Header Table

Item	Frequency	header
f	4	
c	4	
a	3	
b	3	
m	3	
p	3	



Example: Construct FP-tree from a Transaction DB

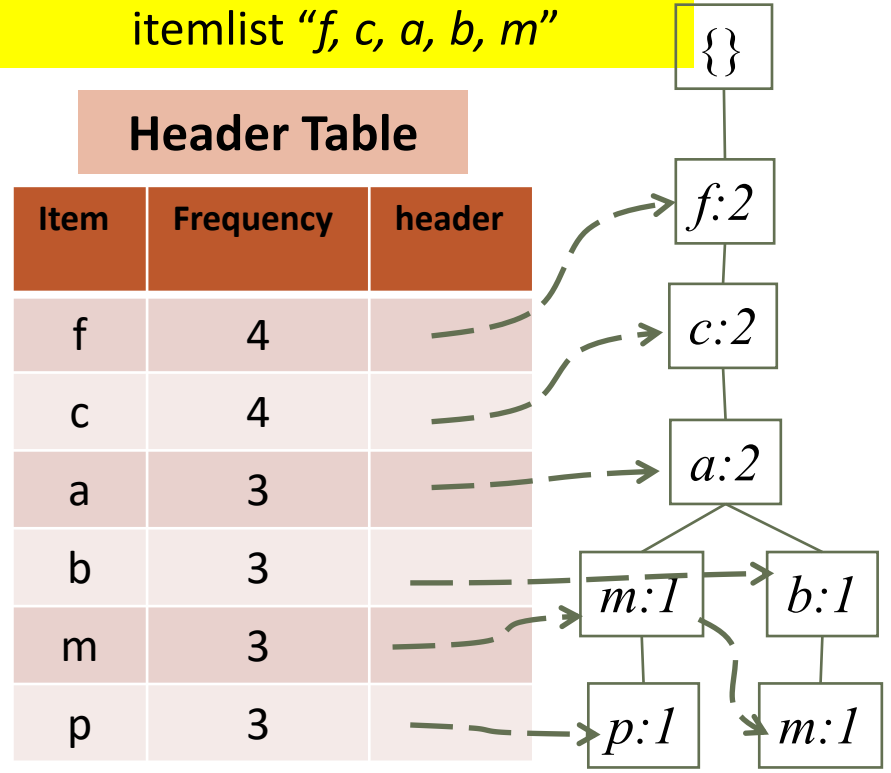
TID	Items in the Transaction	Ordered, frequent itemlist
100	{f, a, c, d, g, i, m, p}	f, c, a, m, p
200	{a, b, c, f, l, m, o}	f, c, a, b, m
300	{b, f, h, j, o, w}	f, b
400	{b, c, k, s, p}	c, b, p
500	{a, f, c, e, l, p, m, n}	f, c, a, m, p

4. For each transaction, insert the ordered frequent itemlist into an FP-tree, with shared sub-branches merged, counts accumulated

After inserting the 2nd frequent itemlist "*f, c, a, b, m*"

Header Table

Item	Frequency	header
f	4	
c	4	
a	3	
b	3	
m	3	
p	3	



Example: Construct FP-tree from a Transaction DB

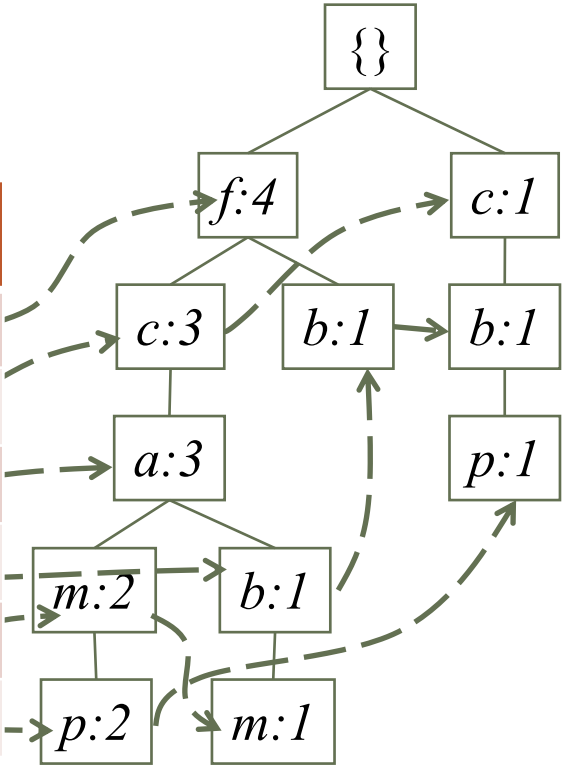
TID	Items in the Transaction	Ordered, frequent itemlist
100	{f, a, c, d, g, i, m, p}	f, c, a, m, p
200	{a, b, c, f, l, m, o}	f, c, a, b, m
300	{b, f, h, j, o, w}	f, b
400	{b, c, k, s, p}	c, b, p
500	{a, f, c, e, l, p, m, n}	f, c, a, m, p

4. For each transaction, insert the ordered frequent itemlist into an FP-tree, with shared sub-branches merged, counts accumulated

After inserting all the frequent itemlists

Header Table

Item	Frequency	header
f	4	
c	4	
a	3	
b	3	
m	3	
p	3	

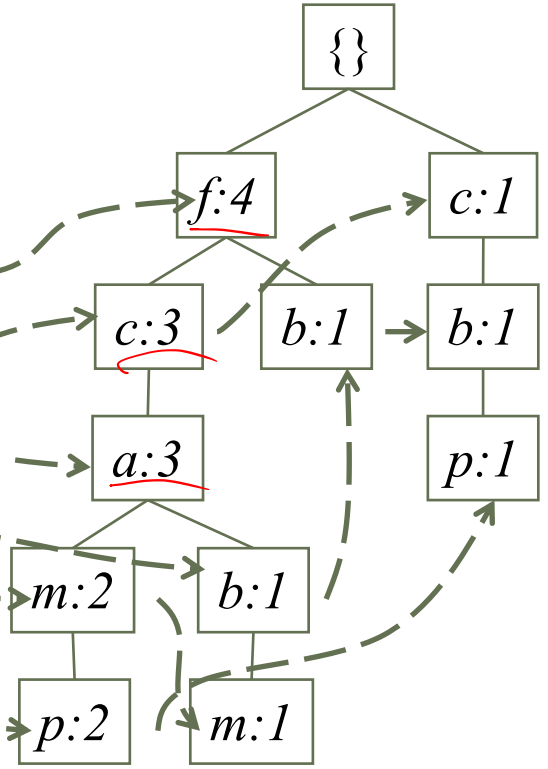


Mining FP-Tree: Divide and Conquer Based on Patterns and Data

- Pattern mining can be partitioned according to current patterns
  - We start to calculate the conditional database from bottom to top (from the least frequent item)
  - Conditional database: the database under the condition that *p* exists
    - *p*'s conditional database (Patterns containing *p*): *fcam:2, cb:1*

min_support = 3

Item	Frequency	Header
f	4	
c	4	
a	3	
b	3	
m	3	
p	3	



Conditional database of each pattern

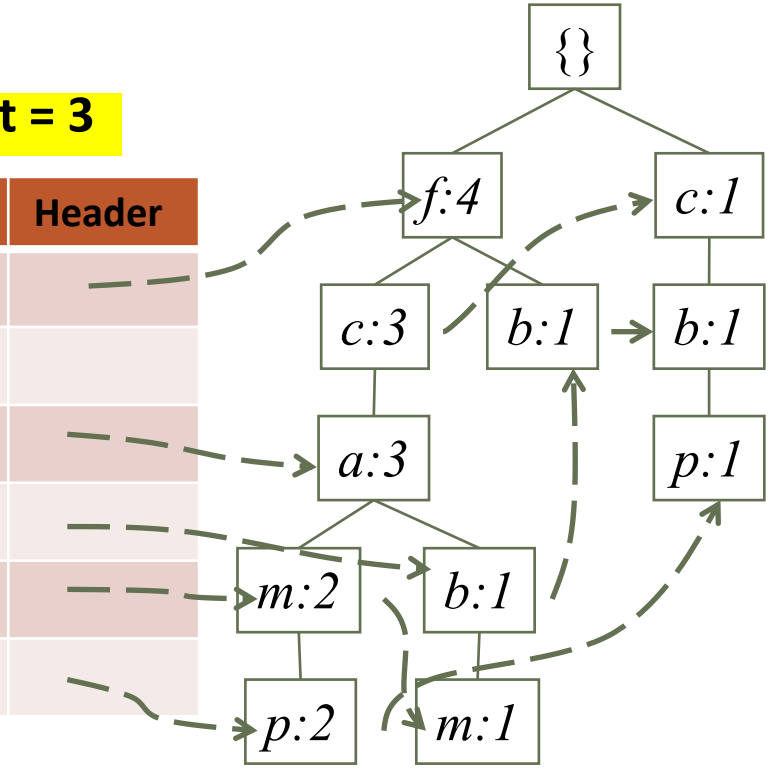
Item	***Conditional database***
p	***fcam:2, cb:1***

Mining FP-Tree: Divide and Conquer Based on Patterns and Data

- *p*'s conditional database (Patterns containing *p*): *fcam:2, cb:1*
- *After calculating p's conditional database, we calculate m's conditional database*

min_support = 3

Item	Frequency	Header
f	4	
c	4	
a	3	
b	3	
m	3	
p	3	



Conditional database of each pattern

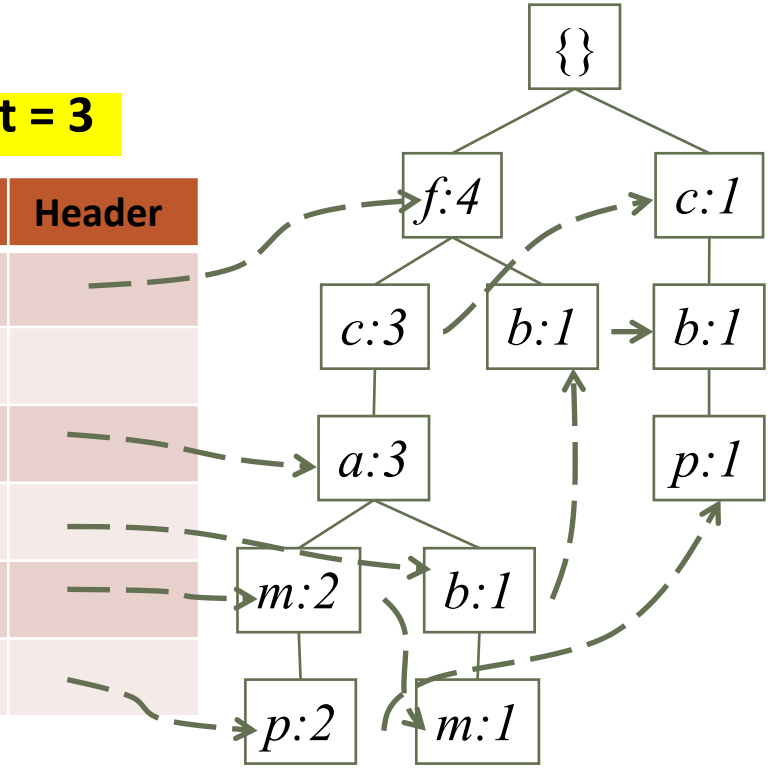
Item	***Conditional database***
m	***fca:2, fcab:1***
p	***fcam:2, cb:1***

Mining FP-Tree: Divide and Conquer Based on Patterns and Data

- *Repeat and calculate the conditional database of b, a, and c*
- *Since f is the most frequent item, we don't need to compute its conditional dataset*

min_support = 3

Item	Frequency	Header
f	4	
c	4	
a	3	
b	3	
m	3	
p	3	



Conditional database of each pattern

Item	***Conditional database***
c	***f:3***
a	***fc:3***
b	***fca:1, f:1, c:1***
m	***fca:2, fcab:1***
p	***fcam:2, cb:1***

Mine Each Conditional Database Recursively

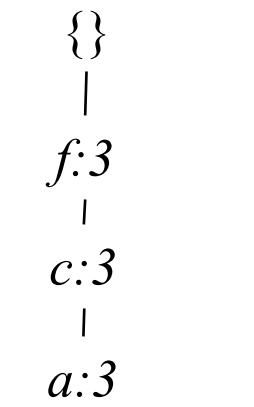
min_support = 3

Conditional Data Bases

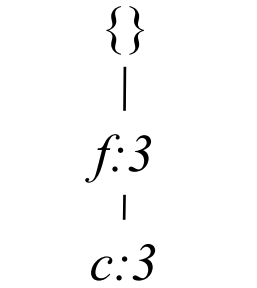
item	***cond. data base***
c	***f:3***
a	***fc:3***
b	***fca:1, f:1, c:1***
m	***fca:2, fcab:1***
p	***fcam:2, cb:1***

- For each conditional database
  - Mine single-item patterns
  - Construct its FP-tree & mine it

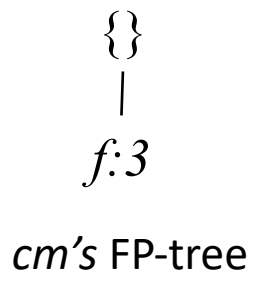
e.g., mining m's FP-tree



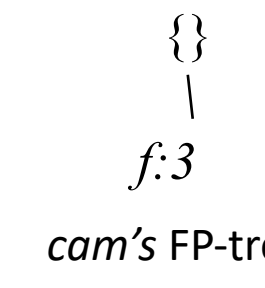
m's FP-tree



am's FP-tree



cm's FP-tree



cam's FP-tree

Actually, for single branch FP-tree, all the frequent patterns can be generated in one shot

m: 3
fm: 3, cm: 3, am: 3
fcm: 3, fam:3, cam: 3
fcam: 3

FPGrowth: Mining Frequent Patterns by Pattern Growth

- Essence of frequent pattern growth (FPGrowth) methodology
  - Find frequent single items and partition the database based on each such single item pattern
  - Recursively grow frequent patterns by doing the above for each *partitioned database* (also called the pattern's *conditional database*)
  - To facilitate efficient processing, an efficient data structure, FP-tree, can be constructed

FPGrowth: Mining Frequent Patterns by Pattern Growth

- Mining becomes
  - Recursively construct and mine (conditional) FP-trees
  - Until the resulting FP-tree is empty, or until it contains only one path—single path will generate all the combinations of its sub-paths, each of which is a frequent pattern

Pattern Evaluation

- Limitation of the Support-Confidence Framework
- Interestingness Measures: Lift and χ^2
- Null-Invariant Measures

How to Judge if a Rule/Pattern Is Interesting?

- Pattern-mining will generate a large set of patterns/rules
  - Not all the generated patterns/rules are interesting
- Interestingness measures: Objective vs. subjective
  - Objective interestingness measures
    - Support, confidence, correlation, …
  - Subjective interestingness measures:
      - Different users may judge interestingness differently
    - Let a user specify
      - Query-based: Relevant to a user's particular request
    - Judge against one's knowledge-base
      - unexpected, freshness, timeliness

Limitation of the Support-Confidence Framework

- Are *s* and *c* interesting in association rules: "A ⇒ B" [*s*, *c*]? Be careful!
- Example: Suppose one school may have the following statistics on # of students who may play basketball and/or eat cereal:

	play-basketball	not play-basketball	sum (row)
eat-cereal	400	350	750
not eat-cereal	200	50	250
sum(col.)	600	400	1000

2-way contingency table

- Association rule mining may generate the following:
  - *play-basketball* ⇒ *eat-cereal* [40%, 66.7%] (higher s & c)
- But this strong association rule is misleading: The overall % of students eating cereal is 75% > 66.7%, a more telling rule:
  - ¬ *play-basketball* ⇒ *eat-cereal* [35%, 87.5%] (high s & c)

Interestingness Measure: Lift

- Measure of dependent/correlated events: **lift**

$$lift(B,C) = \frac{c(B \rightarrow C)}{s(C)} = \frac{s(B \cup C)}{s(B) \times s(C)}$$

Lift is more telling than s & c

	B	¬B	$\sum_{row}$
C	400	350	750
¬C	200	50	250
$\sum_{col.}$	600	400	1000

- Lift(B, C) may tell how B and C are correlated
  - Lift(B, C) = 1: B and C are independent
  - > 1: positively correlated
  - < 1: negatively correlated
- For our example,

$$lift(B,C) = \frac{400/1000}{600/1000 \times 750/1000} = 0.89$$

$$lift(B,\neg C) = \frac{200/1000}{600/1000 \times 250/1000} = 1.33$$

- Thus, B and C are negatively correlated since lift(B, C) < 1;
  - B and ¬C are positively correlated since lift(B, ¬C) > 1

Interestingness Measure: χ^2

- Another measure to test correlated events: **χ^2**

$$\chi^2 = \sum \frac{(Observed - Expected)^2}{Expected}$$

- For the table on the right,

$$\chi^2 = \frac{(400-450)^2}{450} + \frac{(350-300)^2}{300} + \frac{(200-150)^2}{150} + \frac{(50-100)^2}{100} = 55.56$$

	B	¬B	Σ_{row}
C	400 (450)	350 (300)	750
¬C	200 (150)	50 (100)	250
Σ_{col}	600	400	1000

Expected value

Observed value

- Lookup **χ^2** distribution table => B, C are correlated
- **χ^2**-test shows B and C are negatively correlated since the expected value is 450 but the observed is only 400
- Thus, **χ^2** is also more telling than the support-confidence framework

Lift and χ^2: Are They Always Good Measures?

- Null transactions: Transactions that contain neither B nor C
- Let's examine the new dataset D
  - BC (100) is much rarer than B¬C (1000) and ¬BC (1000), but there are many ¬B¬C (100000)
  - Unlikely B & C will happen together!
- But, Lift(B, C) = 8.44 >> 1 (Lift shows B and C are strongly positively correlated!)
- χ^2 = 670: Observed(BC) >> expected value (11.85)
- *Too many null transactions may "spoil the soup"!*

	B	¬B	$\sum_{row}$
C	100	1000	1100
¬C	1000	100000	101000
$\sum_{col.}$	1100	101000	102100

null transactions

Contingency table with expected values added

	B	¬B	$\sum_{row}$
C	100 (11.85)	1000	1100
¬C	1000 (988.15)	100000	101000
$\sum_{col.}$	1100	101000	102100

Interestingness Measures & Null-Invariance

- *Null invariance* means: The number of null transactions does not matter. Does not change the measure value.
- A few interestingness measures: Some are null invariant

Measure	Definition	Range	Null-Invariant?
$\chi^2(A,B)$	$\sum_{i,j} \frac{(e(a_i,b_j)-o(a_i,b_j))^2}{e(a_i,b_j)}$	$[0, \infty]$	No
$Lift(A,B)$	$\frac{s(A \cup B)}{s(A) \times s(B)}$	$[0, \infty]$	No
$Allconf(A,B)$	$\frac{s(A \cup B)}{max\{s(A), s(B)\}}$	$[0, 1]$	Yes
$Jaccard(A,B)$	$\frac{s(A \cup B)}{s(A)+s(B)-s(A \cup B)}$	$[0, 1]$	Yes
$Cosine(A,B)$	$\frac{s(A \cup B)}{\sqrt{s(A) \times s(B)}}$	$[0, 1]$	Yes
$Kulczynski(A,B)$	$\frac{1}{2}(\frac{s(A \cup B)}{s(A)} + \frac{s(A \cup B)}{s(B)})$	$[0, 1]$	Yes
$MaxConf(A,B)$	$max\{\frac{s(A \cup B)}{s(A)}, \frac{s(A \cup B)}{s(B)}\}$	$[0, 1]$	Yes

Let

$$p = \frac{s(A \cup B)}{s(A)} = P(B|A)$$

$$q = \frac{s(A \cup B)}{s(B)} = P(A|B)$$

p, q are null invariant

Essentially min, max, mean variants of p, q

Null Invariance: An Important Property

- Why is null invariance crucial for the analysis of massive transaction data?
  - Many transactions may contain neither milk nor coffee!

milk vs. coffee contingency table

	$milk$	$\neg milk$	Σ_{row}
$coffee$	mc	$\neg mc$	c
$\neg coffee$	$m\neg c$	$\neg m\neg c$	$\neg c$
Σ_{col}	m	$\neg m$	Σ

- Lift and χ^2 are not null-invariant: not good to evaluate data that contain too many or too few null transactions!
- Many measures are not null-invariant!

Data set	mc	$\neg mc$	$m\neg c$	$\neg m\neg c$	χ^2	$Lift$
D_1	10,000	1,000	1,000	100,000	90557	9.26
D_2	10,000	1,000	1,000	100	0	1
D_3	100	1,000	1,000	100,000	670	8.44
D_4	1,000	1,000	1,000	100,000	24740	25.75
D_5	1,000	100	10,000	100,000	8173	9.18
D_6	1,000	10	100,000	100,000	965	1.97

Comparison of Null-Invariant Measures

- Not all null-invariant measures are created equal
- Which one is better?
  - D_4—D_6 differentiate the null-invariant measures
  - Kulc (Kulczynski 1927) holds firm and is in balance of both directional implications

2-variable contingency table

	$milk$	$\neg milk$	Σ_{row}
$coffee$	mc	$\neg mc$	c
$\neg coffee$	$m\neg c$	$\neg m\neg c$	$\neg c$
Σ_{col}	m	$\neg m$	Σ

All 5 are null-invariant

Data set	mc	$\neg mc$	$m\neg c$	$\neg m\neg c$	$AllConf$	Jaccard	$Cosine$	$Kulc$	$MaxConf$
D_1	10,000	1,000	1,000	100,000	0.91	0.83	0.91	0.91	0.91
D_2	10,000	1,000	1,000	100	0.91	0.83	0.91	0.91	0.91
D_3	100	1,000	1,000	100,000	0.09	0.05	0.09	0.09	0.09
D_4	1,000	1,000	1,000	100,000	0.5	0.33	0.5	0.5	0.5
D_5	1,000	100	10,000	100,000	0.09	0.09	0.29	0.5	0.91
D_6	1,000	10	100,000	100,000	0.01	0.01	0.10	0.5	0.99

Imbalance Ratio with Kulczynski Measure

- IR (Imbalance Ratio): measure the imbalance of two itemsets A and B in rule implications:

$$IR(A,B) = \frac{|s(A) - s(B)|}{s(A) + s(B) - s(A \cup B)}$$

- Kulczynski and Imbalance Ratio (IR) together present a clear picture for all the three datasets D_4 through D_6
  - D_4 is neutral & balanced; D_5 is neutral but imbalanced
  - D_6 is neutral but very imbalanced

Data set	mc	$\neg mc$	$m\neg c$	$\neg m\neg c$	Jaccard	$Cosine$	$Kulc$	IR
D_1	10,000	1,000	1,000	100,000	0.83	0.91	0.91	0
D_2	10,000	1,000	1,000	100	0.83	0.91	0.91	0
D_3	100	1,000	1,000	100,000	0.05	0.09	0.09	0
D_4	1,000	1,000	1,000	100,000	0.33	0.5	0.5	0
D_5	1,000	100	10,000	100,000	0.09	0.29	0.5	0.89
D_6	1,000	10	100,000	100,000	0.01	0.10	0.5	0.99

What Measures to Choose for Effective Pattern Evaluation?

- Null value cases are predominant in many large datasets
  - Neither milk nor coffee is in most of the baskets; neither Mike nor Jim is an author in most of the papers; ……
- *Null-invariance* is an important property
- Lift and χ^2 are good measures if null transactions are not predominant
  - Otherwise, *Kulczynski* + *Imbalance Ratio* should be used to judge the interestingness of a pattern

Mining Frequent Patterns, Association and Correlations

- Basic Techniques
- Sequential Pattern Mining 
- Graph Pattern Mining
- Summary

Sequential Pattern Mining

- Sequential Pattern and Sequential Pattern Mining
- GSP: Apriori-Based Sequential Pattern Mining
- SPADE: Sequential Pattern Mining in Vertical Data Format
- PrefixSpan: Sequential Pattern Mining by Pattern-Growth
- CloSpan: Mining Closed Sequential Patterns
- Constraint-Based Sequential-Pattern Mining

Sequential Pattern Mining

- What kind of patterns are sequential?
- Sequential – The order really matters. You can not swap two items in a sequence and have the same sequence.
- Example: The English language is sequential : Subject -> Verb -> Object.

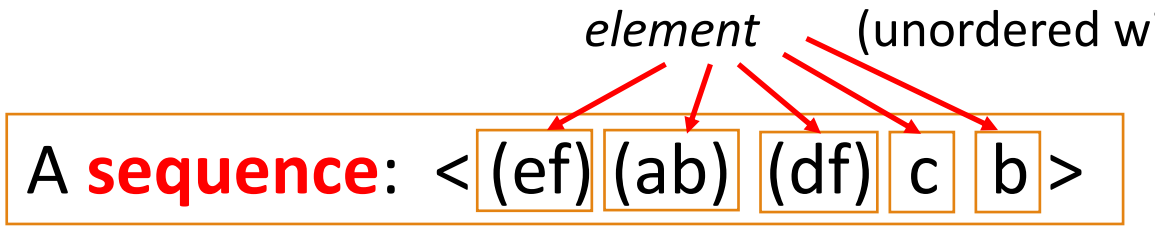
- Other points:
  - For Sequential Pattern Mining, the time which the items occur is **not** considered.
  - Time Series Analysis does take into account the time in which an item occurred.

Sequential Pattern Examples

- Application of Sequential pattern Mining
  - **Customer shopping** → Purchase a laptop first, then a digital camera, and then a smartphone.
  - **Medical treatments** → Go to the doctor, get drugs, doctor monitors progress, doctor reacts accordingly -> more/less drugs
  - **Natural disasters** -> Before the disaster, during the disaster, after the disaster.
  - **Scientific Experiments** → Step 1, Step 2, Step 3.
  - **Stocks Markets** → Stocks go up and down together.
  - **Biological sequences, DNA /Protein**→ If you change the order of proteins, it is a different gene.

Sequential Pattern and Sequential Pattern Mining

- **Sequential pattern mining**: Given a set of sequences, find the complete set of *frequent* subsequences (i.e., satisfying the min_sup threshold)



- An *element* may contain a set of *items* (also called *events*)

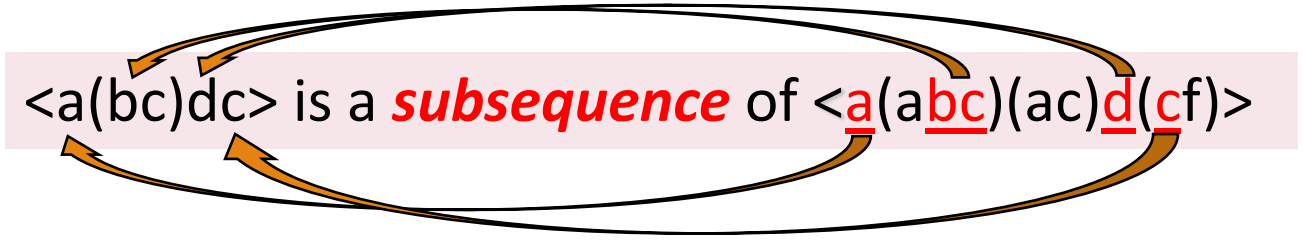
* Items within an element are **unordered** and we list them alphabetically

A **sequence database**

SID	Sequence
10	<a(abc)(ac)d(cf)>
20	<(ad)c(bc)(ae)>
30	<(ef)(ab)(df)cb>
40	<eg(af)cbc>

Sequential Pattern and Sequential Pattern Mining

- **Sequential pattern mining**: Given a set of sequences, find the complete set of *frequent* subsequences (i.e., satisfying the min_sup threshold)



- Given support threshold *min_sup* = 2, <(ab)c> *is* a *sequential pattern*

A **sequence database**

SID	Sequence
10	<a(abc)(ac)d(cf)>
20	<(ad)c(bc)(ae)>
30	<(ef)(ab)(df)cb>
40	<eg(af)cbc>

Sequential Pattern Mining Algorithms

- Algorithm requirement: Efficient, scalable, finding complete set, incorporating various kinds of user-specific constraints
- The Apriori property still holds: If a subsequence s_1 is infrequent, none of s_1's super-sequences can be frequent
- Representative algorithms
  - GSP (Generalized Sequential Patterns): Srikant & Agrawal @ EDBT'96)
  - Vertical format-based mining: SPADE (Zaki@Machine Leanining'00)
  - Pattern-growth methods: PrefixSpan (Pei, et al. @TKDE'04)

GSP: Apriori-Based Sequential Pattern Mining

- Initial candidates: All 8-singleton sequences
  - <a>, <b>, <c>, <d>, <e>, <f>, <g>, <h>
- Scan DB once, count support for each candidate

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

min_sup = 2

Cand.	sup
<a>	3
<b>	5
<c>	4
<d>	3
<e>	3
<f>	2
~~<g>~~	1
~~<h>~~	1

GSP: Apriori-Based Sequential Pattern Mining

- Example: Generate length-2 candidate sequences

min_sup = 2

Cand.	sup
<a>	3
<b>	5
<c>	4
<d>	3
<e>	3
<f>	2
~~<g>~~	1
~~<h>~~	1

singleton * singleton – *Total:* (6 * 6)

	<a>	<b>	<c>	<d>	<e>	<f>
<a>	<aa>	<ab>	<ac>	<ad>	<ae>	<af>
<b>	<ba>	<bb>	<bc>	<bd>	<be>	<bf>
<c>	<ca>	<cb>	<cc>	<cd>	<ce>	<cf>
<d>	<da>	<db>	<dc>	<dd>	<de>	<df>
<e>	<ea>	<eb>	<ec>	<ed>	<ee>	<ef>
<f>	<fa>	<fb>	<fc>	<fd>	<fe>	<ff>

Sets (unordered) – *Total*: (6*5) / 2

	<a>	<b>	<c>	<d>	<e>	<f>
<a>		<(ab)>	<(ac)>	<(ad)>	<(ae)>	<(af)>
<b>			<(bc)>	<(bd)>	<(be)>	<(bf)>
<c>				<(cd)>	<(ce)>	<(cf)>
<d>					<(de)>	<(df)>
<e>						<(ef)>
<f>						

Apriori Pruning

- w/o pruning *(includes g and h):*

  8*8 + 8*7/2 = 92

  length-2 candidates

- w/ pruning:

  6*6 + 6*5/2 = 51

  length-2 candidates

GSP Mining and Pruning

		length
5th scan: 1 cand. 1 length-5 seq. pat.	<(bd)cba>	5
4th scan: 8 cand. 7 length-4 seq. pat.	<abba> <(bd)bc> …	4
3rd scan: 46 cand. 20 length-3 seq. pat. 20 cand. not in DB at all	<abb> <aab> <aba> <baa> <bab> …	3
2nd scan: 51 cand. 19 length-2 seq. pat. 10 cand. not in DB at all	<aa> <ab> … <af> <ba> <bb> … <ff> <(ab)> … <(ef)>	2
1st scan: 8 cand. 6 length-1 seq. pat.	<a> <b> <c> <d> <e> <f> ~~<g> <h>~~	1

6*6 + 6*5/2 = 51

- Remove
  - Candidates not in DB
  - Candidates < min_sup

min_sup = 2

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

GSP Mining and Pruning

- Repeat, starting at k=1 until k<=length
  - Scan DB to find “length-k” frequent sequences
  - Generate “length-(k+1)” candidate sequences from “length-k” frequent sequences using **Apriori**
  - set k = k+1
- Until no frequent sequence or no candidate can be found

GSP (Generalized Sequential Patterns): Srikant & Agrawal @ EDBT’96)
-NOTE: Same team which developed Apriori

Sequential Pattern Mining in Vertical Data Format: The SPADE Algorithm

- A sequence database is mapped to: <SID, EID>
- Grow the subsequences (patterns) one item at a time by Apriori candidate generation

SID	Sequence
1	<a(abc)(ac)d(cf)>
2	<(ad)c(bc)(ae)>
3	<(ef)(ab)(df)cb>
4	<eg(af)cbc>

min_sup = 2

Ref: SPADE (Sequential PAttern Discovery using Equivalent Class) [M. Zaki 2001]

SID	EID	Items
1	1	a
1	2	abc
1	3	ac
1	4	d
1	5	cf
2	1	ad
2	2	c
2	3	bc
2	4	ae
3	1	ef
3	2	ab
3	3	df
3	4	c
3	5	b
4	1	e
4	2	g
4	3	af
4	4	c
4	5	b
4	6	c

a

SID	EID
1	1
1	2
1	3
2	1
2	4
3	2
4	3

b

SID	EID
1	2
2	3
3	2
3	5
4	5

EID (b) < EID (a): Corresponds to: <a(abc)(ac)d(cf)>

ab

SID	EID (a)	EID(b)
1	1	2
2	1	3
3	2	5
4	3	5

ba

SID	EID (b)	EID(a)
1	2	3
2	3	4

aba

SID	EID (a)	EID(b)	EID(a)
1	1	2	3
2	1	3	4

PrefixSpan: A Pattern-Growth Approach

SID	Sequence
10	<a(abc)(ac)d(cf)>
20	<(ad)c(bc)(ae)>
30	<(ef)(ab)(df)cb>
40	<eg(af)cbc>

min_sup = 2

Prefix	Suffix (Projection)
<a>	<(abc)(ac)d(cf)>
<aa>	<(_bc)(ac)d(cf)>
<ab>	<(_c)(ac)d(cf)>

"_" is placeholder for prefix

- PrefixSpan Mining: Prefix vs. Suffix
  - Given two sequences: $\alpha = \langle e_1 e_2 \cdots e_n \rangle$ and $\beta = \langle e'_1 e'_2 \cdots e'_m \rangle \ (m \leq n)$
  - β is the prefix of α, if
    - (1) $e'_i = e_i$ (i<m),
    - (2) $e'_m \subseteq e_m$
    - (3) all the frequent items in $(e_m - e'_m)$ are alphabetically after those in e'_m
  - $\gamma = \langle e''_m e_{m+1} \cdots e_n \rangle$ is the suffix of α wrt β
    - where $e''_m = (e_m - e'_m)$

PrefixSpan: A Pattern-Growth Approach

SID	Sequence
10	<a(abc)(ac)d(cf)>
20	<(ad)c(bc)(ae)>
30	<(ef)(ab)(df)cb>
40	<eg(af)cbc>

min_sup = 2

Prefix	Suffix (Projection)
<a>	<(abc)(ac)d(cf)>
<aa>	<(_bc)(ac)d(cf)>
<ab>	<(_c)(ac)d(cf)>

- Prefix and suffix
  - Given <a(abc)(ac)d(cf)>
  - Prefixes: <a>, <aa>, <a(ab)>, <a(abc)>, …
  - Suffix: Prefixes-based projection

"_" is placeholder for prefix

- PrefixSpan Mining: Prefix Projections
  - Step 1: Find length-1 sequential patterns
    - <a>, <b>, <c>, <d>, <e>, <f>
  - Step 2: Divide search space and mine each projected DB
    - <a>-projected DB,
    - <b>-projected DB,
    - …
    - <f>-projected DB, …

PrefixSpan (Prefix-projected Sequential pattern mining) Pei, et al. @TKDE'04

PrefixSpan: Mining Prefix-Projected DBs

SID	Sequence
10	<a(abc)(ac)d(cf)>
20	<(ad)c(bc)(ae)>
30	<(ef)(ab)(df)cb>
40	<eg(af)cbc>

min_sup = 2

Length-1 sequential patterns
<a>, <b>, <c>, <d>, <e>, <f>

prefix <a>

<a>-projected DB

<(abc)(ac)d(cf)>
<(_d)c(bc)(ae)>
<(_b)(df)cb>
<(_f)cbc>

prefix <b>

<b>-projected DB

<(_c)(ac)d(cf)>
<(_c)(ae)>
<(df)cb>
<c>

prefix <c>, …, <f>

…

… …

prefix <aa>

<aa>-projected DB

..

prefix <af>

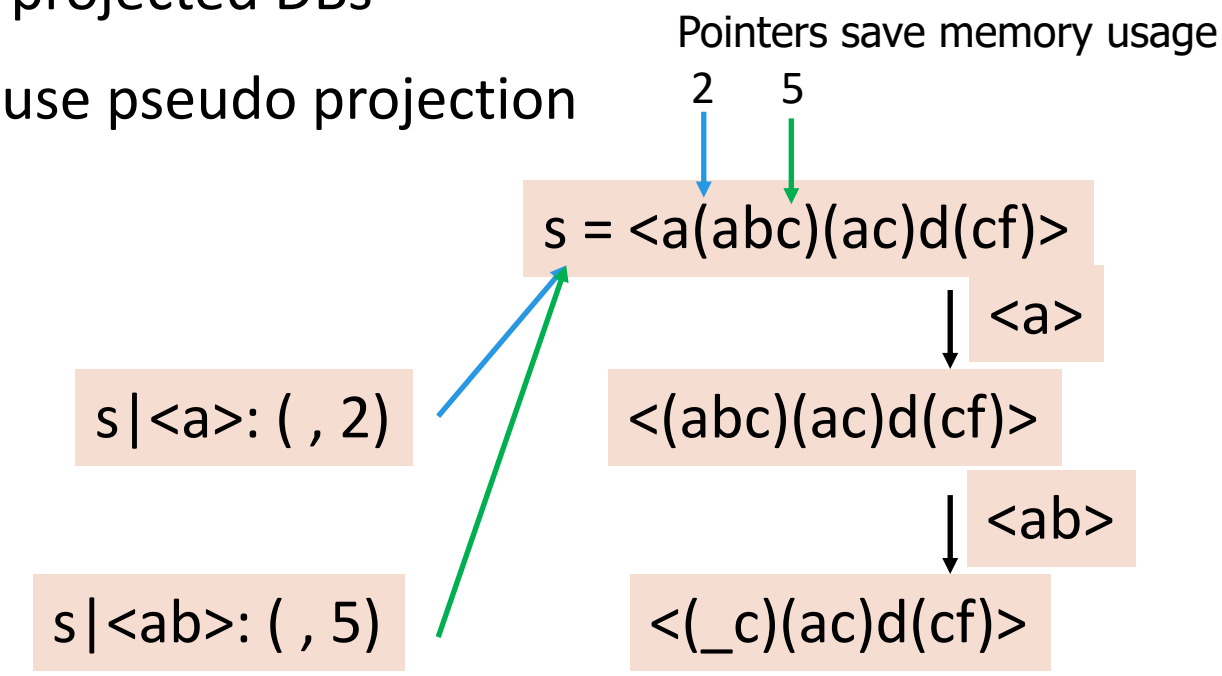
<af>-projected DB

Major strength of PrefixSpan:

- No candidate subseqs. to be generated
- Projected DBs keep shrinking

Implementation Consideration: Pseudo-Projection vs. Physical Projection

- Major cost of PrefixSpan: Constructing projected DBs
  - Suffixes largely repeating in recursive projected DBs
- When DB can be held in main memory, use pseudo projection
  - No physically copying suffixes
  - Pointer to the sequence
  - Offset of the suffix
- But if it does not fit in memory
  - Physical projection
- Suggested approach:
  - Integration of physical and pseudo-projection
  - Swapping to pseudo-projection when the data fits in memory

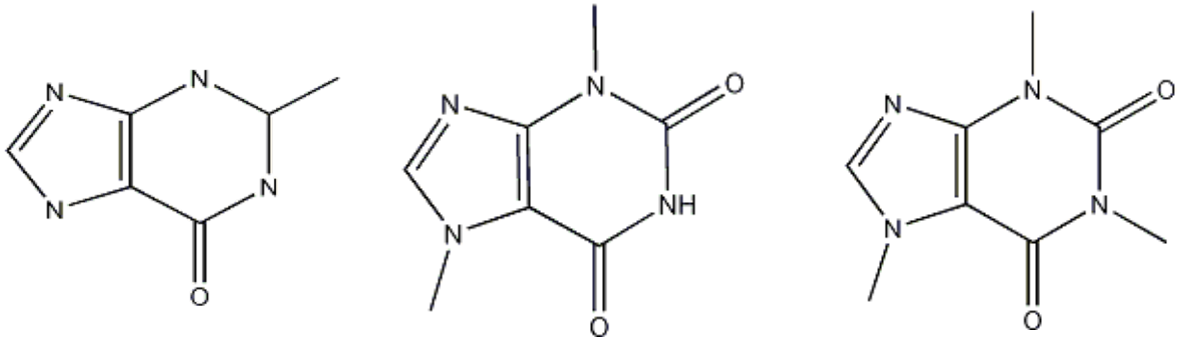


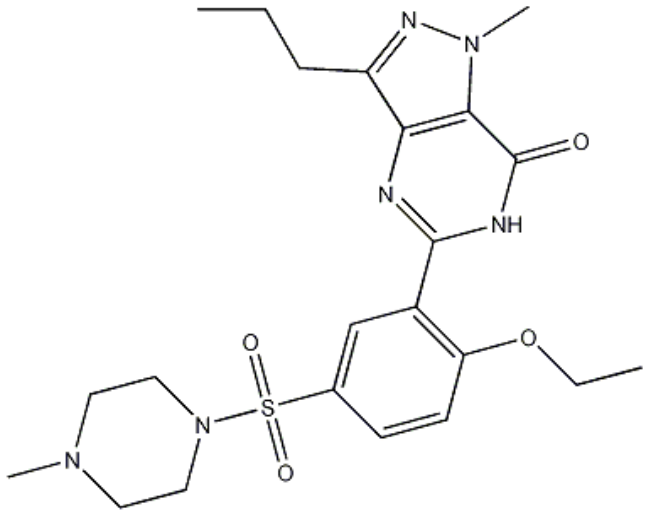
Mining Frequent Patterns, Association and Correlations

- Basic Techniques
- Sequential Pattern Mining
- Graph Pattern Mining 
- Summary

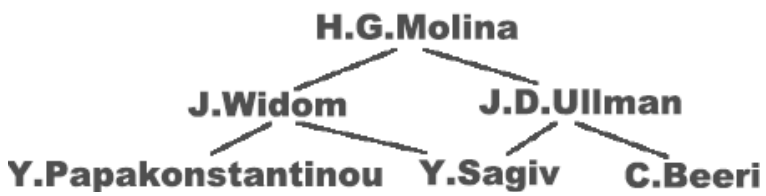
What Is Graph Pattern Mining?

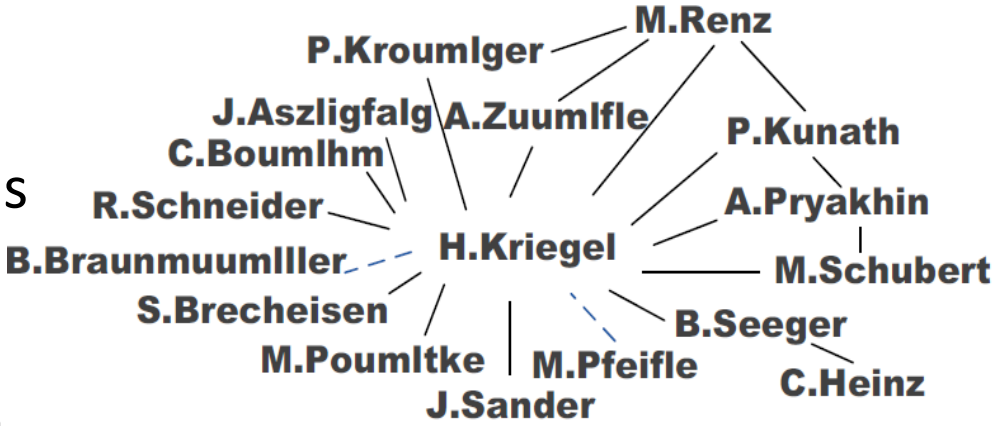
- Chem-informatics:
  - Mining frequent chemical compound structures





- Social networks, web communities, tweets, …
  - Finding frequent research collaboration subgraphs

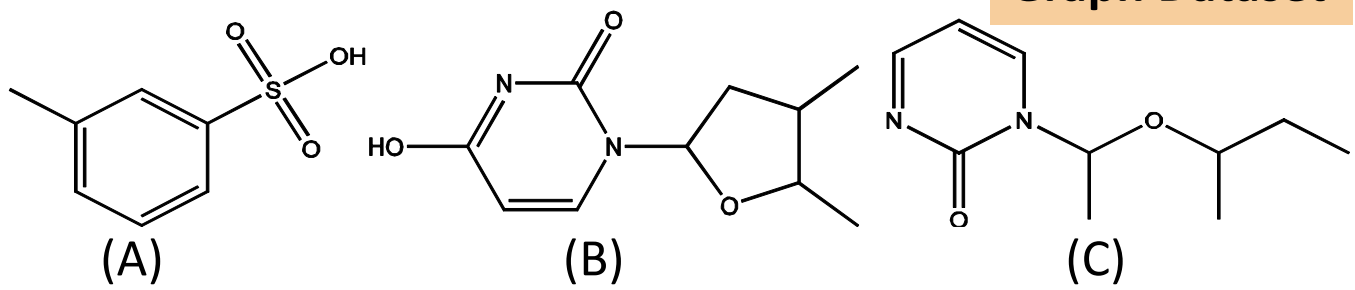




Frequent (Sub)Graph Patterns

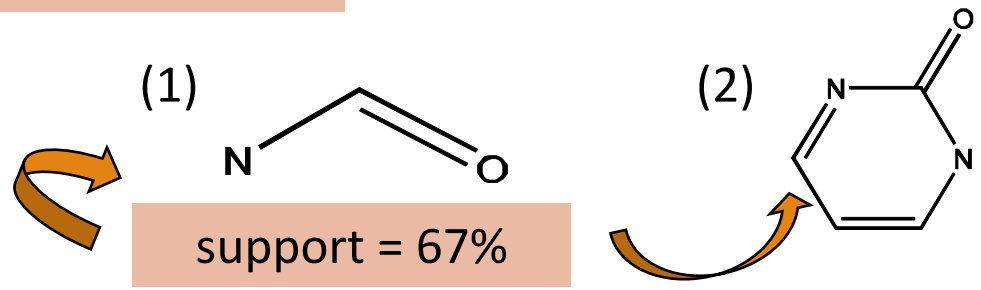
- Given a labeled graph dataset D = {G_1, G_2, …, G_n), the supporting graph set of a subgraph *g* is D_g = {G_i | $g \subseteq G_i$, $G_i \in D$}
  - support(*g*) = $|D_g|/|D|$
- A (sub)graph *g* is ***frequent*** if *support*(*g*) ≥ *min_sup*
- Ex.: Chemical structures
- Alternative:
  - Mining frequent subgraph patterns from a single large graph or network

Graph Dataset



min_sup = 2

Frequent Graph Patterns



Applications of Graph Pattern Mining

- Bioinformatics
  - Gene networks, protein interactions, metabolic pathways
- Chem-informatics: Mining chemical compound structures
- Social networks, web communities, tweets, …
- Cell phone networks, computer networks, …
- Web graphs, XML structures, Semantic Web, information networks
- Software engineering: Program execution flow analysis
- Building blocks for graph classification, clustering, compression, comparison, and correlation analysis
- Graph indexing and graph similarity search

Graph Pattern Mining Algorithms: Different Methodologies

- Generation of candidate subgraphs
  - Apriori vs. pattern growth (e.g., FSG vs. gSpan)
- Search order
  - Breadth vs. depth
- Elimination of duplicate subgraphs
  - Passive vs. active (e.g., gSpan [Yan & Han, 2002])
- Support calculation
  - Store embeddings (e.g., GASTON [Nijssen & Kok, 2004], FFSM [Huan, Wang, & Prins, 2003], MoFa [Borgelt & Berthold, ICDM'02])
- Order of pattern discovery
  - Path → tree → graph (e.g., GASTON [Nijssen & Kok, 2004])

Special Type of Graph - Tree

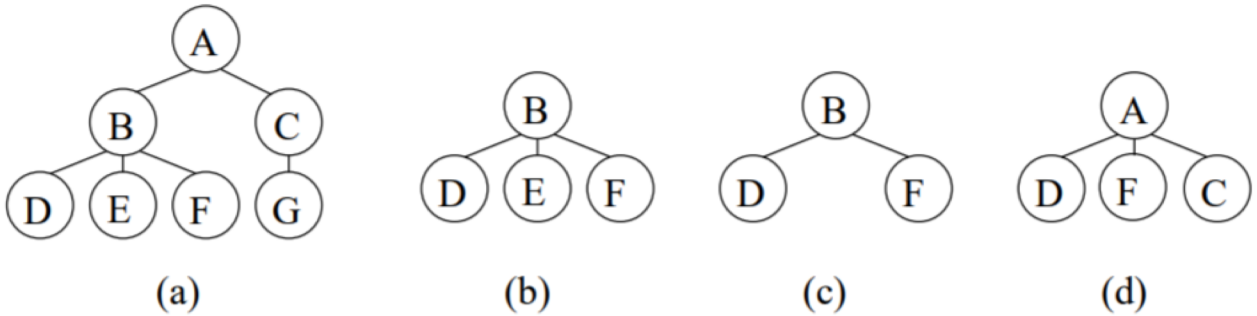
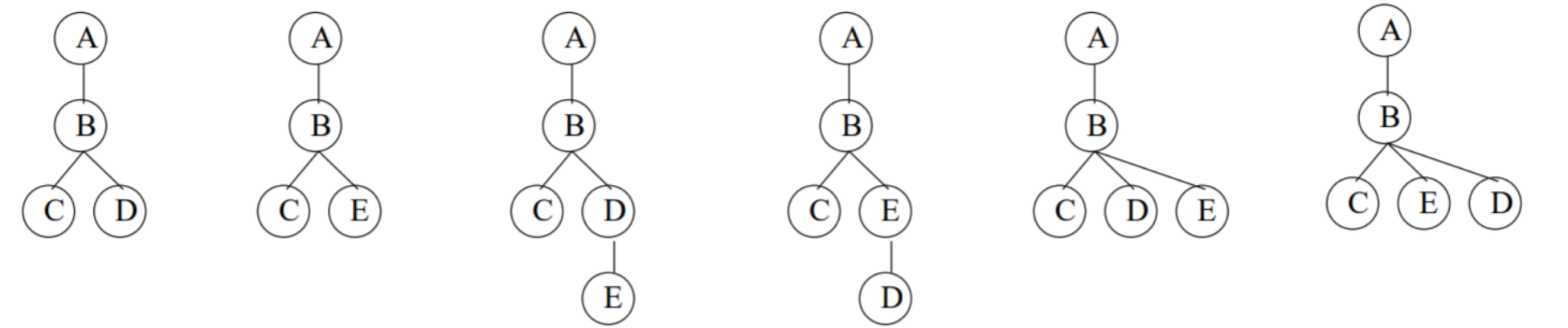


Figure 1. Different Types of Subtrees

- (a): original tree
- (b): bottom-up subtree
- (c): induced subtree
- (d): embedded subtree

Key Ideas

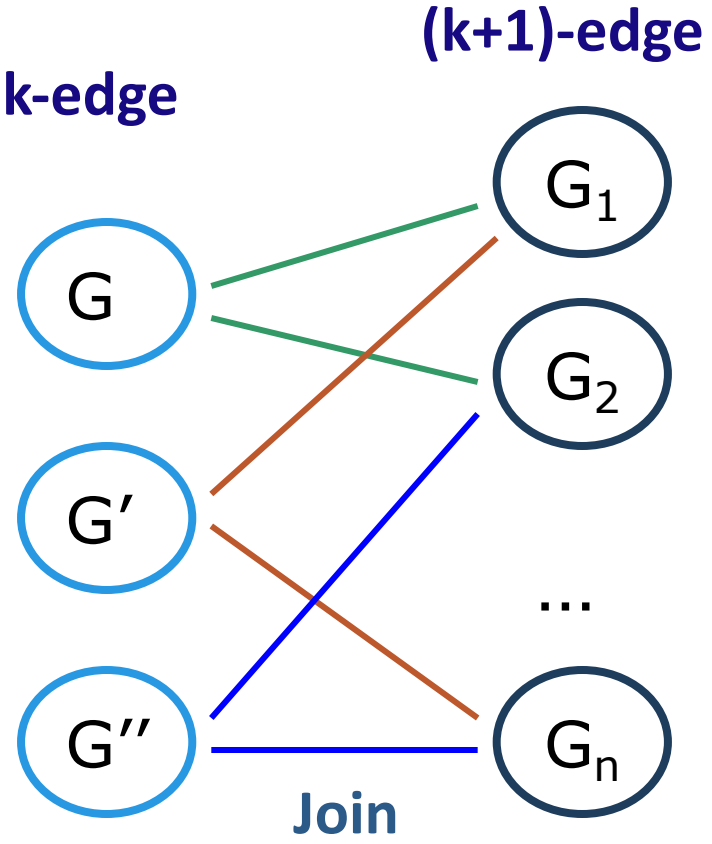
- String encoding (TreeMiner [Zaki, 2002])



- Frequent sequence mining

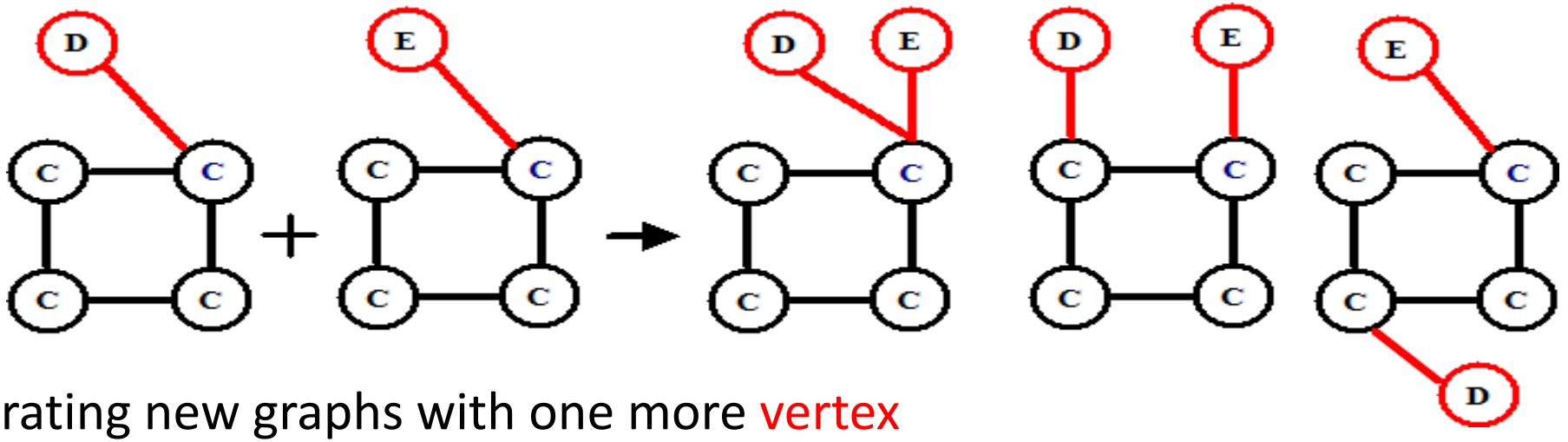
Apriori-Based Approach

- The Apriori property (anti-monotonicity): A size-*k* subgraph is **frequent** if and only if all of its **subgraphs are frequent**
- A candidate size-(*k*+1) edge/vertex subgraph is generated if its corresponding two *k*-edge/vertex subgraphs are frequent
- Iterative mining process:
  - Candidate-generation → candidate pruning → support counting → candidate elimination



Candidate Generation: Vertex Growing vs. Edge Growing

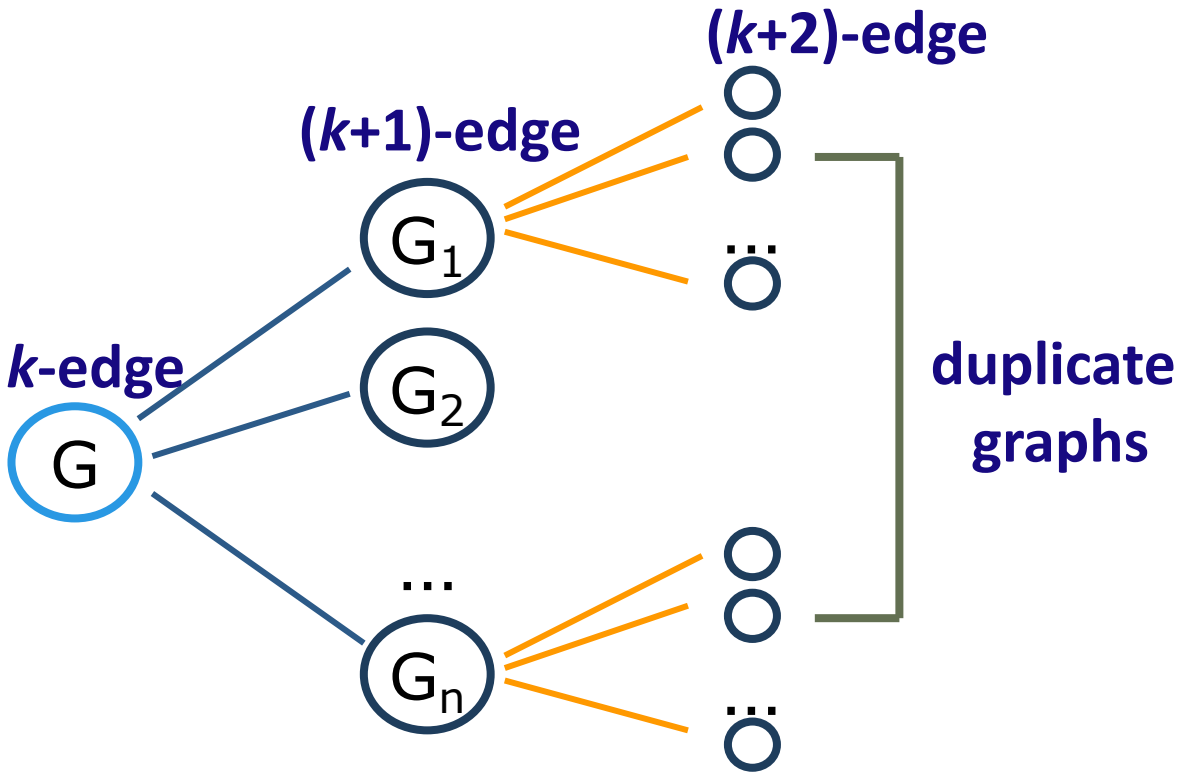
- Methodology: Breadth-search, Apriori joining two size-*k* graphs
  - Many possibilities at generating size-(*k*+1) candidate graphs



- Generating new graphs with one more vertex
  - AGM (Inokuchi, Washio, & Motoda, PKDD'00)
- Generating new graphs with one more edge
  - FSG (Kuramochi & Karypis, ICDM'01)
- Performance shows *via edge growing* is more efficient

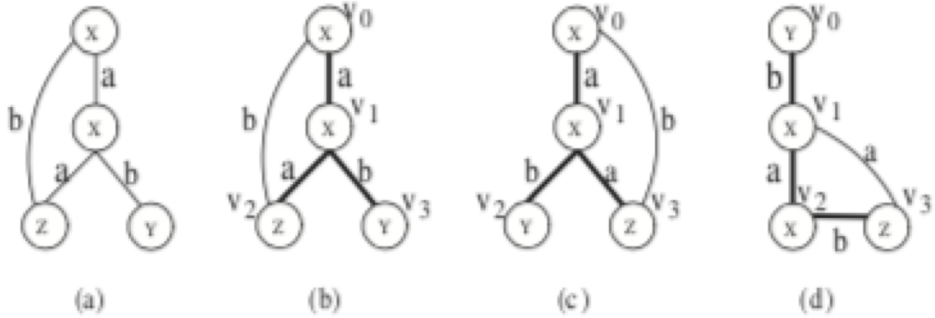
Pattern-Growth Approach

- Depth-first growth of subgraphs from *k*-edge to (*k*+1)-edge, then (*k*+2)-edge subgraphs
- Major challenge
  - Generating many duplicate subgraphs
- Major idea to solve the problem
  - Define an order to generate subgraphs
  - DFS spanning tree: Flatten a graph into a sequence using depth-first search
  - gSpan (Yan & Han, ICDM'02)



gSPAN: Graph Pattern Growth in Order

- **Right-most path extension** in subgraph pattern growth
  - Right-most path: The path from root to the right-most leaf
  - Reduce generation of duplicate subgraphs
- **Completeness:** The enumeration of graphs using right-most path extension is complete
- DFS code: Flatten a graph into a sequence using depth-first search



Mining Frequent Patterns, Association and Correlations

- Basic Techniques
- Sequential Pattern Mining
- Graph Pattern Mining
- Summary 

Summary

- Basic Concepts
  - What Is Pattern Discovery? Why Is It Important?
  - Basic Concepts: Frequent Patterns and Association Rules
- Efficient Pattern Mining Methods
  - The Downward Closure Property of Frequent Patterns
  - The Apriori Algorithm
  - Mining Frequent Patterns by Exploring Vertical Data Format
  - FPGrowth: A Frequent Pattern-Growth Approach
- Pattern Evaluation
  - Interestingness Measures in Pattern Mining
  - Interestingness Measures: Lift and χ^2
  - Null-Invariant Measures
- Sequential Pattern Mining
  - Sequential Pattern and Sequential Pattern Mining
  - GSP: Apriori-Based Sequential Pattern Mining
  - SPADE: Sequential Pattern Mining in Vertical Data Format
  - PrefixSpan: Sequential Pattern Mining by Pattern-Growth
- Graph Pattern Mining
  - Graph Pattern and Graph Pattern Mining
  - Apriori-Based Graph Pattern Mining Methods
  - gSpan: A Pattern-Growth-Based Method

Recommended Readings (Basic Concepts)

- R. Agrawal, T. Imielinski, and A. Swami, “Mining association rules between sets of items in large databases”, in Proc. of SIGMOD'93
- R. J. Bayardo, “Efficiently mining long patterns from databases”, in Proc. of SIGMOD'98
- N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal, “Discovering frequent closed itemsets for association rules”, in Proc. of ICDT'99
- J. Han, H. Cheng, D. Xin, and X. Yan, “Frequent Pattern Mining: Current Status and Future Directions”, Data Mining and Knowledge Discovery, 15(1): 55-86, 2007

Recommended Readings (Efficient Pattern Mining Methods)

- R. Agrawal and R. Srikant, “Fast algorithms for mining association rules”, VLDB'94
- A. Savasere, E. Omiecinski, and S. Navathe, “An efficient algorithm for mining association rules in large databases”, VLDB'95
- J. S. Park, M. S. Chen, and P. S. Yu, “An effective hash-based algorithm for mining association rules”, SIGMOD'95
- S. Sarawagi, S. Thomas, and R. Agrawal, “Integrating association rule mining with relational database systems: Alternatives and implications”, SIGMOD'98
- M. J. Zaki, S. Parthasarathy, M. Ogihara, and W. Li, “Parallel algorithm for discovery of association rules”, Data Mining and Knowledge Discovery, 1997
- J. Han, J. Pei, and Y. Yin, “Mining frequent patterns without candidate generation”, SIGMOD’00
- M. J. Zaki and Hsiao, “CHARM: An Efficient Algorithm for Closed Itemset Mining”, SDM'02
- J. Wang, J. Han, and J. Pei, “CLOSET+: Searching for the Best Strategies for Mining Frequent Closed Itemsets”, KDD'03
- C. C. Aggarwal, M.A., Bhuiyan, M. A. Hasan, “Frequent Pattern Mining Algorithms: A Survey”, in Aggarwal and Han (eds.): Frequent Pattern Mining, Springer, 2014

Recommended Readings (Pattern Evaluation)

- C. C. Aggarwal and P. S. Yu. A New Framework for Itemset Generation. PODS’98
- S. Brin, R. Motwani, and C. Silverstein. Beyond market basket: Generalizing association rules to correlations. SIGMOD'97
- M. Klemettinen, H. Mannila, P. Ronkainen, H. Toivonen, and A. I. Verkamo. Finding interesting rules from large sets of discovered association rules. CIKM'94
- E. Omiecinski. Alternative Interest Measures for Mining Associations. TKDE’03
- P.-N. Tan, V. Kumar, and J. Srivastava. Selecting the Right Interestingness Measure for Association Patterns. KDD'02
- T. Wu, Y. Chen and J. Han, Re-Examination of Interestingness Measures in Pattern Mining: A Unified Framework, Data Mining and Knowledge Discovery, 21(3):371-397, 2010

References: Sequential Pattern Mining

- R. Srikant and R. Agrawal, “Mining sequential patterns: Generalizations and performance improvements”, EDBT’96
- M. Zaki, “SPADE: An Efficient Algorithm for Mining Frequent Sequences”, Machine Learning, 2001
- J. Pei, J. Han, B. Mortazavi-Asl, J. Wang, H. Pinto, Q. Chen, U. Dayal, and M.-C. Hsu, "Mining Sequential Patterns by Pattern-Growth: The PrefixSpan Approach", IEEE TKDE, 16(10), 2004
- X. Yan, J. Han, and R. Afshar, “CloSpan: Mining Closed Sequential Patterns in Large Datasets”, SDM'03
- J. Pei, J. Han, and W. Wang, "Constraint-based sequential pattern mining: the pattern-growth methods", J. Int. Inf. Sys., 28(2), 2007
- M. N. Garofalakis, R. Rastogi, K. Shim: Mining Sequential Patterns with Regular Expression Constraints. IEEE Trans. Knowl. Data Eng. 14(3), 2002
- H. Mannila, H. Toivonen, and A. I. Verkamo, “Discovery of frequent episodes in event sequences”, Data Mining and Knowledge Discovery, 1997

References: Graph Pattern Mining

- C. Borgelt and M. R. Berthold, Mining molecular fragments: Finding relevant substructures of molecules, ICDM'02
- J. Huan, W. Wang, and J. Prins. Efficient mining of frequent subgraph in the presence of isomorphism, ICDM'03
- A. Inokuchi, T. Washio, and H. Motoda. An apriori-based algorithm for mining frequent substructures from graph data, PKDD'00
- M. Kuramochi and G. Karypis. Frequent subgraph discovery, ICDM'01
- S. Nijssen and J. Kok. A Quickstart in Frequent Structure Mining can Make a Difference. KDD'04
- N. Vanetik, E. Gudes, and S. E. Shimony. Computing frequent graph patterns from semistructured data, ICDM'02
- X. Yan and J. Han, gSpan: Graph-Based Substructure Pattern Mining, ICDM'02
- X. Yan and J. Han, CloseGraph: Mining Closed Frequent Graph Patterns, KDD'03
- X. Yan, P. S. Yu, J. Han, Graph Indexing: A Frequent Structure-based Approach, SIGMOD'04
- X. Yan, P. S. Yu, and J. Han, Substructure Similarity Search in Graph Databases, SIGMOD'05