Pattern Discovery: Basic Concepts

Pattern Discovery: Basic Concepts

- What Is Pattern Discovery?   Why Is It Important?
- Basic Concepts: Frequent Patterns and Association Rules
- Compressed Representation: Closed Patterns and Max-Patterns

What Is Pattern Discovery?
Why Is It Important?

What Is Pattern Discovery?

- 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
- Pattern discovery: Uncovering patterns from massive data sets
- Motivation examples:
  - What products were often purchased together?
  - What are the subsequent purchases after buying an iPad?
  - What code segments likely contain copy-and-paste bugs?
  - What word sequences likely form phrases in this corpus?

Pattern Discovery: Why Is It Important?

- Finding inherent regularities in a data set
- Foundation for many essential data mining tasks
  - Association, correlation, and causality analysis
  - Mining sequential, structural (e.g., sub-graph) patterns
  - Pattern analysis in spatiotemporal, multimedia, time-series, and stream data
  - 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

Basic Concepts: Frequent Patterns and Association Rules

Basic Concepts: k-Itemsets and Their Supports

Tid	Items bought
10	Beer, Nuts, Diaper
20	Beer, Coffee, Diaper
30	Beer, Diaper, Eggs
40	Nuts, Eggs, Milk
50	Nuts, Coffee, Diaper, Eggs, Milk

- Itemset: A set of one or more items
- k-itemset: $X = \{x_1, \ldots, x_k\}$
  - Ex. {Beer, Nuts, Diaper} is a 3-itemset
- *(absolute) support (count)* of X, sup{X}: Frequency or the number of occurrences of an itemset X
  - Ex. sup{Beer} = 3
  - Ex. sup{Diaper} = 4
  - Ex. sup{Beer, Diaper} = 3
  - Ex. sup{Beer, Eggs} = 1
- *(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

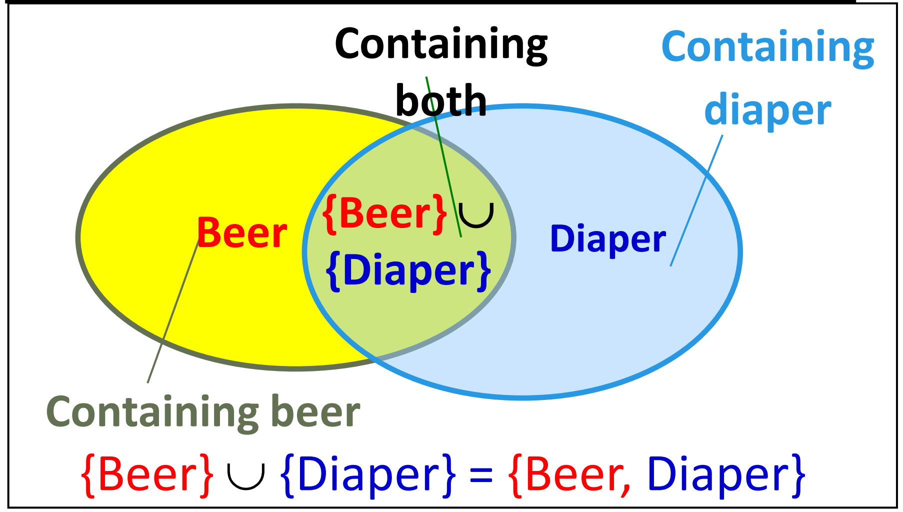
Tid	Items bought
10	Beer, Nuts, Diaper
20	Beer, Coffee, Diaper
30	Beer, Diaper, Eggs
40	Nuts, Eggs, Milk
50	Nuts, Coffee, Diaper, Eggs, Milk

- 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

- Comparing with itemsets, rules can be more telling
  - Ex. *Diaper* → *Beer*
    - *Buying diapers may likely lead to buying beers*
- How strong is this rule? (support, confidence)
  - Measuring association rules: $X \rightarrow Y$ (s, c)
    - Both *X* and *Y* are itemsets
  - Support, *s*: The probability that a transaction contains $X \cup Y$
    - Ex. s{Diaper, Beer} = 3/5 = 0.6 (i.e., 60%)
  - Confidence, *c: The conditional probability* that a transaction containing X also contains *Y*
    - Calculation: $c = sup(X \cup Y) / sup(X)$
    - Ex. *c* = sup{Diaper, Beer}/sup{Diaper} = ¾ = 0.75

Tid	Items bought
10	Beer, Nuts, Diaper
20	Beer, Coffee, Diaper
30	Beer, Diaper, Eggs
40	Nuts, Eggs, Milk
50	Nuts, Coffee, Diaper, Eggs, Milk



Note: $X \cup Y$: the union of two itemsets

- The set contains both 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* ≥ *minconf*

Tid	Items bought
10	Beer, Nuts, Diaper
20	Beer, Coffee, Diaper
30	Beer, Diaper, Eggs
40	Nuts, Eggs, Milk
50	Nuts, Coffee, Diaper, Eggs, Milk

- 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?)

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

Compressed Representation: Closed Patterns and Max-Patterns

Challenge: There Are Too Many Frequent Patterns!

- A long pattern contains a combinatorial number of sub-patterns
- How many frequent itemsets does the following TDB_1 contain?
  - $TDB_{1:}$ T_1: $\{a_1, \ldots, a_{50}\}$; T_2: $\{a_1, \ldots, a_{100}\}$
  - Assuming (absolute) *minsup* = 1
  - Let's have a try

  1-itemsets: $\{a_1\}$: 2, $\{a_2\}$: 2, …, $\{a_{50}\}$: 2, $\{a_{51}\}$: 1, …, $\{a_{100}\}$: 1,

  2-itemsets: $\{a_1, a_2\}$: 2, …, $\{a_1, a_{50}\}$: 2, $\{a_1, a_{51}\}$: 1 …, …, $\{a_{99}, a_{100}\}$: 1,

  …, …, …, …

  99-itemsets: $\{a_1, a_2, \ldots, a_{99}\}$: 1, …, $\{a_2, a_3, \ldots, a_{100}\}$: 1

  100-itemset: $\{a_1, a_2, \ldots, a_{100}\}$: 1

A too huge set for any one to compute or store!

- The total number of frequent itemsets:

$$\binom{100}{1} + \binom{100}{2} + \binom{100}{3} + \cdots + \binom{100}{100} = 2^{100} - 1$$

Expressing Patterns in Compressed Form: Closed Patterns

- How to handle such a challenge?
- Solution 1: **Closed patterns**: A pattern (itemset) X is closed if X is *frequent*, and there exists *no super-pattern* Y ↄ X, *with the same support* as X
  - Let Transaction DB TDB_1: T_1: $\{a_1, \ldots, a_{50}\}$; T_2: $\{a_1, \ldots, a_{100}\}$
  - Suppose *minsup* = 1. How many closed patterns does TDB_1 contain?
    - Two: P_1: "$\{a_1, \ldots, a_{50}\}$: 2"; P_2: "$\{a_1, \ldots, a_{100}\}$: 1"
- Closed pattern is a lossless compression of frequent patterns
  - Reduces the # of patterns but does not lose the support information!
  - You will still be able to say: "$\{a_2, \ldots, a_{40}\}$: 2", "$\{a_5, a_{51}\}$: 1"

Expressing Patterns in Compressed Form: Max-Patterns

- Solution 2: **Max-patterns**: A pattern X is a max-pattern if X is frequent and there exists no frequent super-pattern $Y \supset X$
- Difference from close-patterns?
  - Do not care the real support of the sub-patterns of a max-pattern
  - Let Transaction DB TDB_1: T_1: $\{a_1, \ldots, a_{50}\}$; T_2: $\{a_1, \ldots, a_{100}\}$
  - Suppose *minsup* = 1. How many max-patterns does TDB_1 contain?
    - One: P: "$\{a_1, \ldots, a_{100}\}$: 1"
- Max-pattern is a lossy compression!
  - We only know $\{a_1, \ldots, a_{40}\}$ is frequent
  - But we do not know the real support of $\{a_1, \ldots, a_{40}\}$, …, any more!
- Thus in many applications, mining close-patterns is more desirable than mining max-patterns

Summary

Summary: Pattern Discovery: Basic Concepts

- What Is Pattern Discovery?   Why Is It Important?
- Basic Concepts: Frequent Patterns and Association Rules
- Compressed Representation: Closed Patterns and Max-Patterns

Recommended Readings

- 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

Efficient Pattern Mining Methods

Efficient Pattern Mining Methods

- The Downward Closure Property of Frequent Patterns
- The Apriori Algorithm
- Extensions or Improvements of Apriori
- Mining Frequent Patterns by Exploring Vertical Data Format
- FPGrowth: A Frequent Pattern-Growth Approach
- Mining Closed Patterns

The Downward Closure Property of Frequent Patterns

The Downward Closure Property of Frequent Patterns

- Observation: From TDB$_1$: T_1: {a_1, …, a_{50}}; T_2: {a_1, …, a_{100}}
  - We get a frequent itemset: {a_1, …, a_{50}}
  - Also, its subsets are all frequent: {a_1}, {a_2}, …, {a_{50}}, {a_1, a_2}, …, {a_1, …, a_{49}}, …
  - There must be some hidden relationships among frequent patterns!
- The downward closure (also called "Apriori") property of frequent patterns
  - If **{beer, diaper, nuts}** is frequent, so is **{beer, diaper}**
  - Every transaction containing {beer, diaper, nuts} also contains {beer, diaper}
  - Apriori: Any subset of a frequent itemset must be frequent
- Efficient mining methodology
  - If any subset of an itemset S is infrequent, then there is no chance for S to be frequent—why do we even have to consider S!?

A sharp knife for pruning!

Apriori Pruning and Scalable Mining Methods

- <u>Apriori pruning principle</u>: 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)

The Apriori Algorithm

Apriori: A Candidate Generation & Test Approach

- Outline of Apriori (level-wise, candidate generation and test)
  - Initially, 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 $\cup_k F_k$ // return F_k generated at each level

The Apriori Algorithm—An Example

minsup = 2

Database TDB

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

1^{st} scan

C_1

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}

2^{nd} 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}

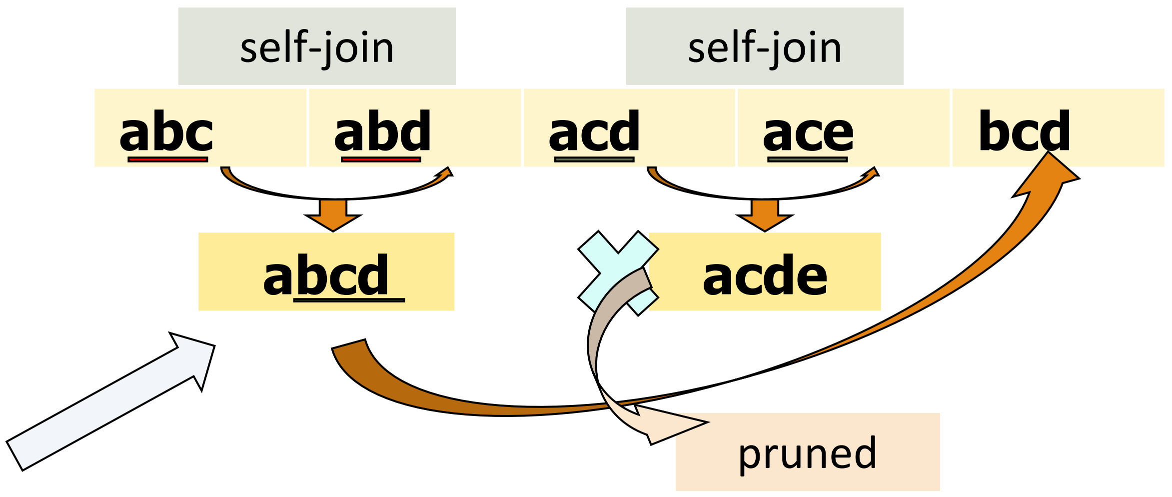
3^{rd} scan

F_3

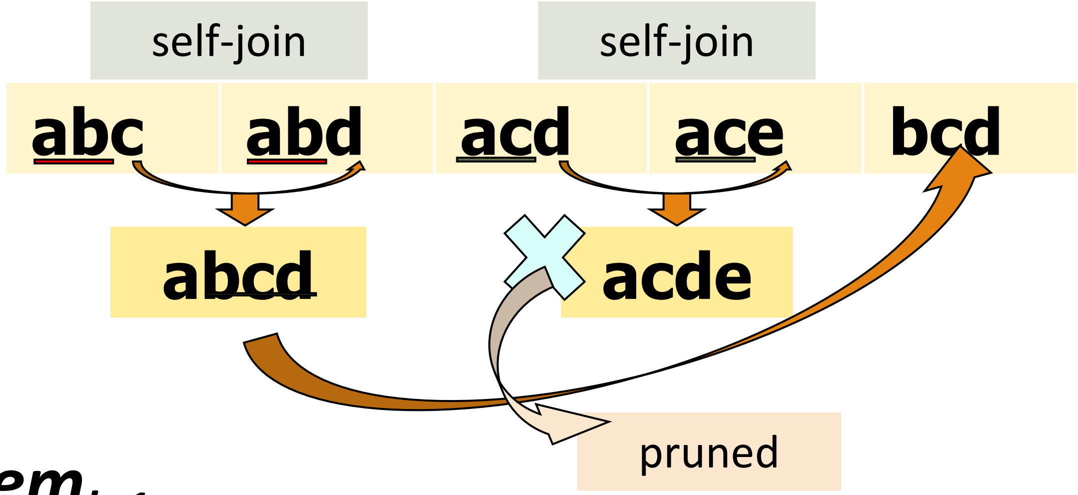
Itemset	sup
{B, C, E}	2

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: An SQL Implementation



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

  insert into $\boldsymbol{C_k}$

  select ***p.item***$_1$, ***p.item***$_2$, …, ***p.item***$_{k-1}$, ***q.item***$_{k-1}$

  from $\boldsymbol{F_{k-1}}$ ***as p***, $\boldsymbol{F_{k-1}}$ ***as q***

  where ***p.item***$_1$ = ***q.item***$_1$, …, ***p.item***$_{k-2}$ = ***q.item***$_{k-2}$, ***p.item***$_{k-1}$ < ***q.item***$_{k-1}$
- Step 2: pruning

  for all ***itemsets c in*** $\boldsymbol{C_k}$ do

  for all ***(k-1)-subsets s of c*** do

  if *(s is not in* F_{k-1}*)* **then delete** *c* **from** C_k

Extensions or Improvements of Apriori

Apriori: Improvements and Alternatives

- Reduce passes of transaction database scans
  - Partitioning (e.g., Savasere, et al., 1995)
  - Dynamic itemset counting (Brin, et al., 1997)

To be discussed in subsequent slides

- Shrink the number of candidates
  - Hashing (e.g., DHP: Park, et al., 1995)
  - Pruning by support lower bounding (e.g., Bayardo 1998)
  - Sampling (e.g., Toivonen, 1996)

To be discussed in subsequent slides

- Exploring special data structures
  - Tree projection (Agarwal, et al., 2001)
  - H-miner (Pei, et al., 2001)
  - Hypecube decomposition (e.g., LCM: Uno, et al., 2004)

Partitioning: Scan Database Only Twice

- Theorem: *Any itemset that is potentially frequent in TDB must be frequent in at least one of the partitions of TDB*

Here is the proof!

$$TDB_1 \quad + \quad TDB_2 \quad + \quad \cdots \quad + \quad TDB_k \quad = \quad TDB$$

$$sup_1(X) < \sigma|TDB_1| \qquad sup_2(X) < \sigma|TDB_2| \qquad \ldots \qquad sup_k(X) < \sigma|TDB_k| \qquad sup(X) < \sigma|TDB|$$

- Method: Scan DB twice (A. Savasere, E. Omiecinski and S. Navathe, *VLDB'95*)
  - Scan 1: Partition database so that each partition can fit in main memory (why?)
    - Mine local frequent patterns in this partition
  - Scan 2: Consolidate global frequent patterns
    - Find global frequent itemset candidates (those frequent in at least one partition)
    - Find the true frequency of those candidates, by scanning TDB_i one more time

Direct Hashing and Pruning (DHP)

- DHP (Direct Hashing and Pruning): (J. Park, M. Chen, and P. Yu, SIGMOD'95)
- Hashing: Different itemsets may have the same hash value: v = *hash*(itemset)
- 1st scan: When counting the 1-itemset, hash 2-itemset to calculate the bucket count
- Observation: A *k*-itemset cannot be frequent if its corresponding hashing bucket count is below the *minsup* threshold
- Example: At the 1st scan of TDB, count 1-itemset, and
  - Hash 2-itemsets in the transaction to its bucket
    - {ab, ad, ce}
    - {bd, be, de}
    - …
  - At the end of the first scan,
    - *if minsup = 80, remove ab, ad, ce, since count*{*ab, ad, ce*} < 80

Itemsets	Count
{*ab, ad, ce*}	35
{*bd, be, de*}	298
……	…
{*yz, qs, wt*}	58

Hash Table

Mining Frequent Patterns by Exploring Vertical Data Format

Exploring Vertical Data Format: ECLAT

- ECLAT (Equivalence Class Transformation): A depth-first search algorithm using set intersection [Zaki et al. @KDD'97]
- Tid-List: List of transaction-ids containing an itemset
- Vertical format: $t(e) = \{T_{10}, T_{20}, T_{30}\}$; $t(a) = \{T_{10}, T_{20}\}$; $t(ae) = \{T_{10}, T_{20}\}$
- Properties of Tid-Lists
  - $t(X) = t(Y)$: X and Y always happen together (e.g., $t(ac) = t(d)$)
  - $t(X) \subset t(Y)$: transaction having X always has Y (e.g., $t(ac) \subset t(ce)$)
- Deriving frequent patterns based on vertical intersections
- Using diffset to accelerate mining
  - Only keep track of differences of tids
  - $t(e) = \{T_{10}, T_{20}, T_{30}\}$, $t(ce) = \{T_{10}, T_{30}\}$ → 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

FPGrowth: A Pattern Growth Approach

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
- 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 containing 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)

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*

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

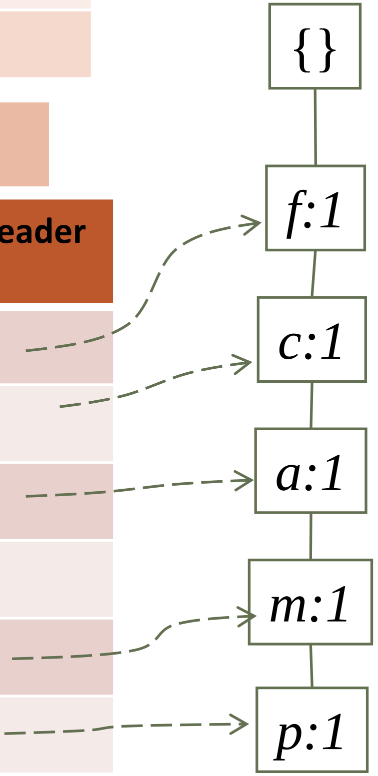
   F-list = f-c-a-b-m-p

3. Scan DB again, construct FP-tree
   - The frequent itemlist of each transaction is inserted as a branch, with shared sub-branches merged, counts accumulated

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*

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

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

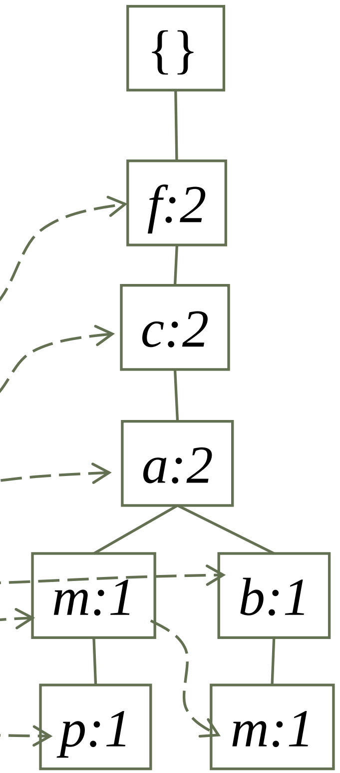
2. Sort frequent items in frequency descending order, f-list

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

3. Scan DB again, construct FP-tree
   - The frequent itemlist of each transaction is inserted as a branch, with shared sub-branches merged, counts accumulated

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*

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

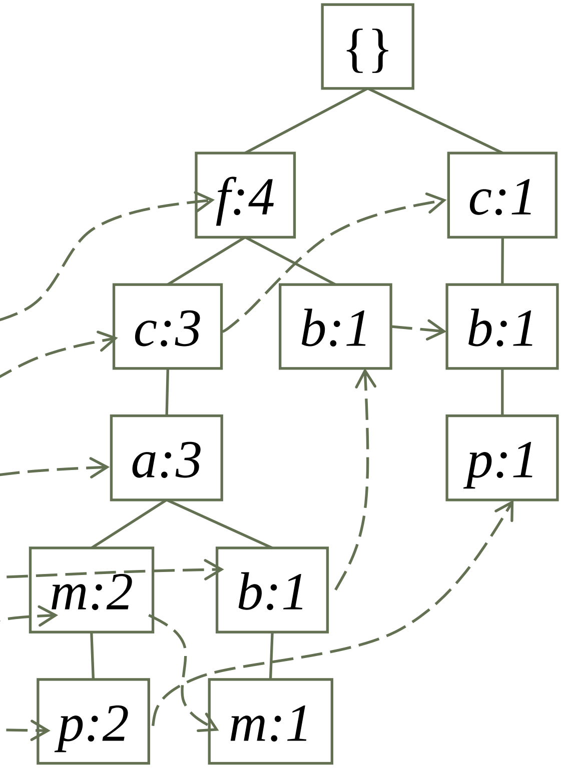
   F-list = f-c-a-b-m-p

3. Scan DB again, construct FP-tree
   - The frequent itemlist of each transaction is inserted as a branch, with shared sub-branches merged, counts accumulated

Header Table

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

After inserting all the frequent itemlists

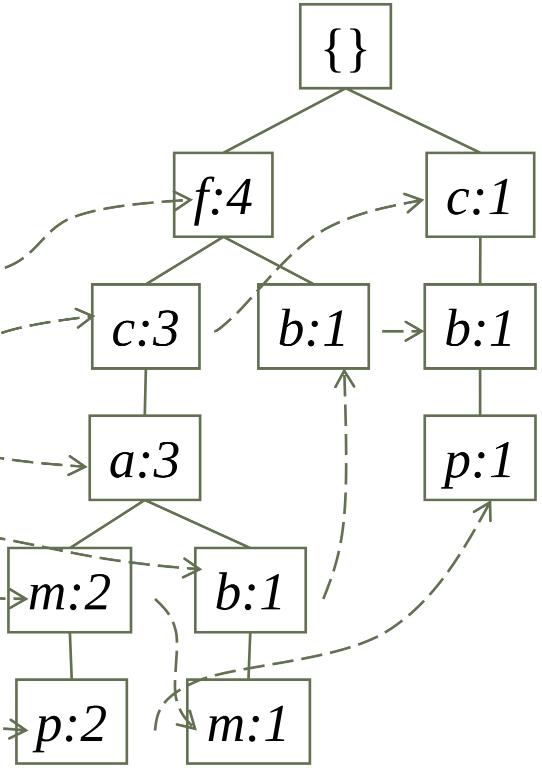


Mining FP-Tree: Divide and Conquer Based on Patterns and Data

- Pattern mining can be partitioned according to current patterns
  - Patterns containing *p*: *p*'s conditional database: *fcam:2, cb:1*
    - *p*'s conditional database (i.e., the database under the condition that *p* exists):
      - *transformed prefix paths* of item *p*
  - Patterns having m but no p: m's conditional database: *fca:2, fcab: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*
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

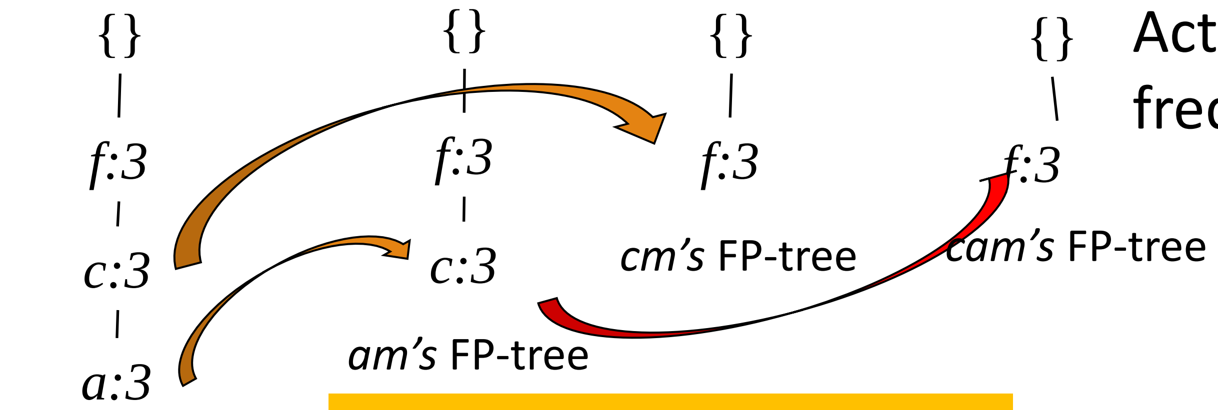
item	*cond. data base*
c	*f:3*
a	*fc:3*
b	*fca:1, f:1, c:1*
m	*fca:2, fcab:1*
p	*fcam:2, cb:1*

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

p's conditional DB: ***fcam:2, cb:1 → c: 3***

m's conditional DB: ***fca:2, fcab:1 → fca: 3***

b's conditional DB: ***fca:1, f:1, c:1 → ϕ***



m's FP-tree

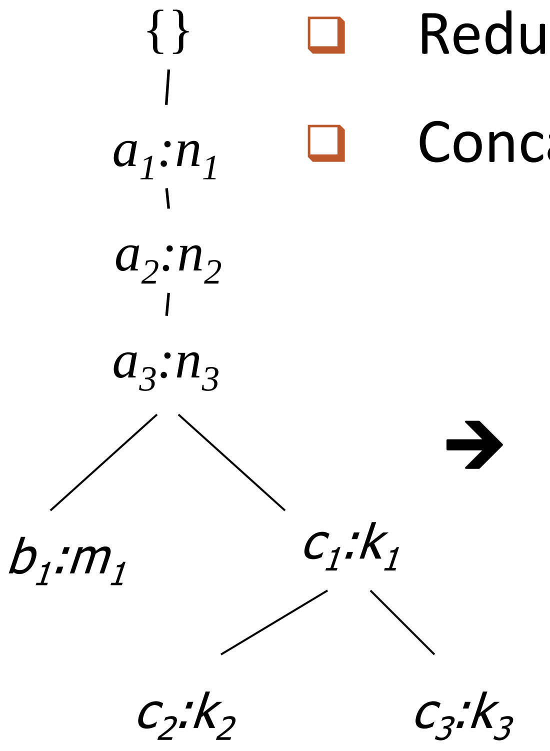
Then, mining m's FP-tree: fca:3

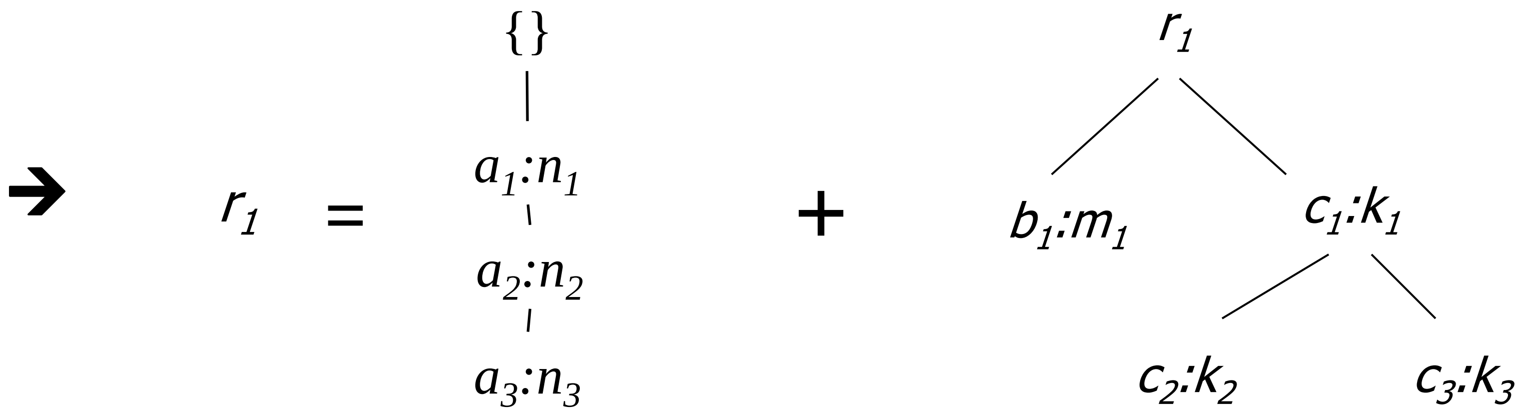
Actually, for single branch FP-tree, all the frequent patterns can be generated in one shot

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

A Special Case: Single Prefix Path in FP-tree

- Suppose a (conditional) FP-tree T has a shared single prefix-path P
- Mining can be decomposed into two parts
  - Reduction of the single prefix path into one node
  - Concatenation of the mining results of the two parts





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
- 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

Scaling FP-growth by Item-Based Data Projection

- What if FP-tree cannot fit in memory?—Do not construct FP-tree
  - "Project" the database based on frequent single items
  - Construct & mine FP-tree for each projected DB
- Parallel projection vs. partition projection
  - Parallel projection: Project the DB on each frequent item
    - Space costly, all partitions can be processed in parallel
  - Partition projection: Partition the DB in order
    - Passing the unprocessed parts to subsequent partitions

Trans. DB

f_2 f_3 f_4 g h
f_3 f_4 i j
f_2 f_4 k
f_1 f_3 h
…

Assume only f's are frequent & the frequent item ordering is: f_1-f_2-f_3-f_4

Parallel projection

f_4-proj. DB	f_3-proj. DB
f_2 f_3	f_2
f_3	f_1
f_2	…
…	

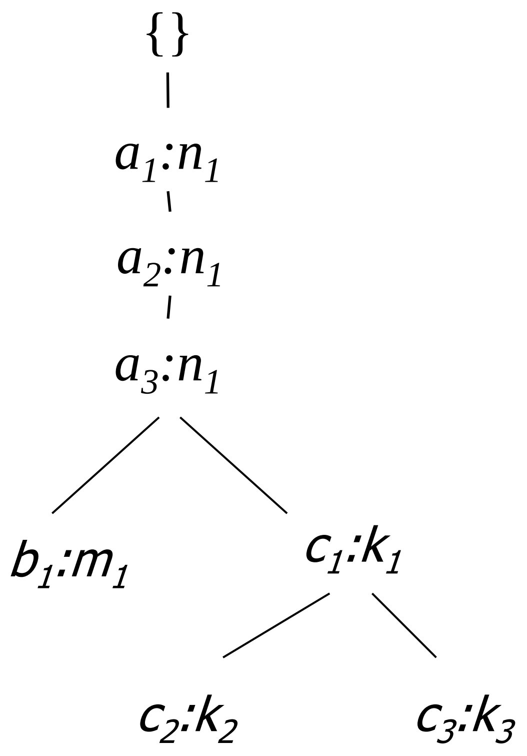
Partition projection

f_4-proj. DB	f_3-proj. DB
f_2 f_3	f_1
f_3	…
f_2	
…	

f_2 will be projected to f_3-proj. DB only when processing f_4-proj. DB

Mining Closed Patterns

CLOSET+: Mining Closed Itemsets by Pattern-Growth



- Efficient, *direct* mining of closed itemsets
- Intuition:
  - If an FP-tree contains a single branch as shown left
  - "a_1,a_2, a_3" should be merged
- Itemset merging: If Y appears in every occurrence of X, then Y is merged with X
  - *d*-proj. db: {acef, acf} → *acfd*-proj. db: {e}
  - Final closed itemset: acfd:2
- There are many other tricks developed
  - For details, see J. Wang, et al,, "CLOSET+: Searching for the Best Strategies for Mining Frequent Closed Itemsets", KDD'03

TID	Items
1	acdef
2	abe
3	cefg
4	acdf

Let minsupport = 2

a:3, c:3, d:2, e:3, f:3

F-List: a-c-e-f-d

Summary

Summary: Efficient Pattern Mining Methods

- The Downward Closure Property of Frequent Patterns
- The Apriori Algorithm
- Extensions or Improvements of Apriori
- Mining Frequent Patterns by Exploring Vertical Data Format
- FPGrowth: A Frequent Pattern-Growth Approach
- Mining Closed Patterns

Recommended Readings

- 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