Data Mining:

Concepts and Techniques

Mining Frequent Patterns, Association and Correlations: Basic Concepts and Methods

- Basic Concepts
- Frequent Itemset Mining Methods
- Which Patterns Are Interesting?—Pattern Evaluation Methods
- Summary

What Is Frequent Pattern Analysis?

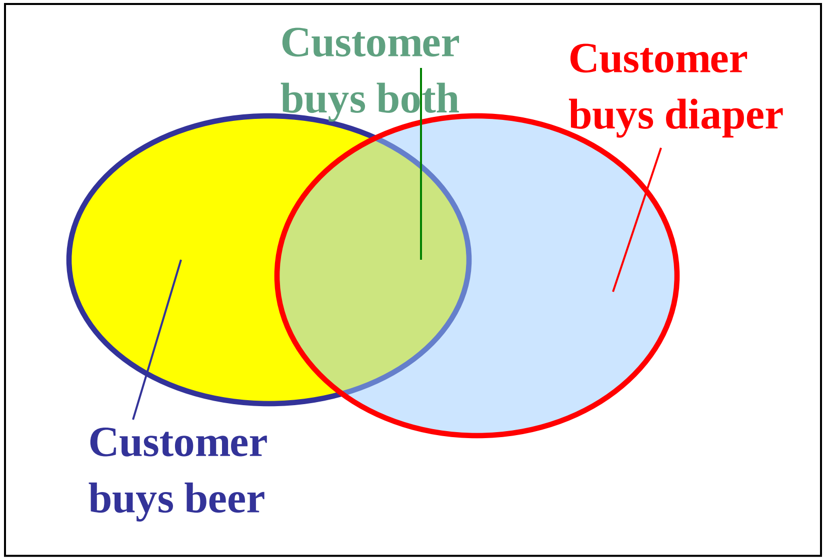
- Frequent pattern: a pattern (a set of items, subsequences, substructures, etc.) that occurs frequently in a data set
- First proposed by Agrawal, Imielinski, and Swami [AIS93] in the context of frequent itemsets and association rule mining
- Motivation: Finding inherent regularities in data
  - What products were often purchased together?— Beer and diapers?!
  - What are the subsequent purchases after buying a PC?
  - What kinds of DNA are sensitive to this new drug?
  - Can we automatically classify web documents?
- Applications
  - Basket data analysis, cross-marketing, catalog design, sale campaign analysis, Web log (click stream) analysis, and DNA sequence analysis.

Why Is Freq. Pattern Mining Important?

- Freq. pattern: An intrinsic and important property of datasets
- Foundation for many essential data mining tasks
  - Association, correlation, and causality analysis
  - Sequential, structural (e.g., sub-graph) patterns
  - Pattern analysis in spatiotemporal, multimedia, time-series, and stream data
  - Classification: discriminative, frequent pattern analysis
  - Cluster analysis: frequent pattern-based clustering
  - Data warehousing: iceberg cube and cube-gradient
  - Semantic data compression: fascicles
  - Broad applications

Basic Concepts: Frequent Patterns

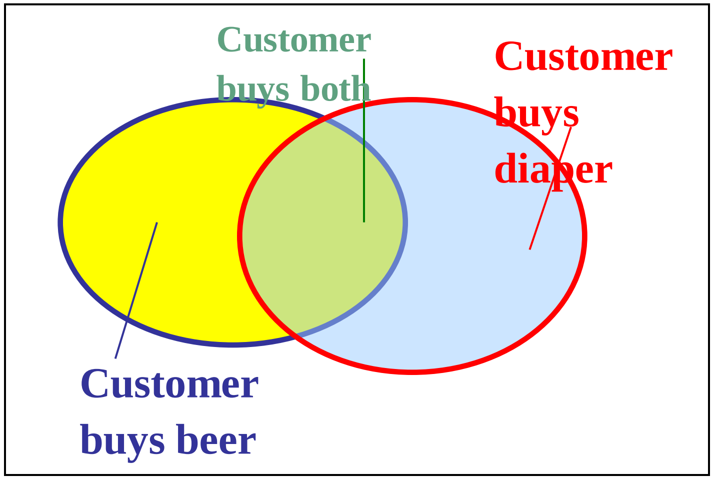
Tid	Items bought
10	Beer, Nuts, Diaper
20	Beer, Coffee, Diaper
30	Beer, Diaper, Eggs
40	Nuts, Eggs, Milk
50	Nuts, Coffee, Diaper, Eggs, Milk



- itemset: A set of one or more items
- k-itemset $X = \{x_1, \ldots, x_k\}$
- *(absolute) support*, or, *support count* of X: Frequency or occurrence of an itemset X
- *(relative) support*, *s*, is the fraction of transactions that contains X (i.e., the probability that a transaction contains X)
- An itemset X is *frequent* if X's support is no less than a *minsup* threshold

Basic Concepts: Association Rules

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



- Find all the rules $X \rightarrow Y$ with minimum support and confidence
  - support, *s*, probability that a transaction contains $X \cup Y$
  - confidence, *c*, conditional probability that a transaction having X also contains *Y*

Let minsup = 50%, minconf = 50%

Freq. Pat.: Beer:3, Nuts:3, Diaper:4, Eggs:3, {Beer, Diaper}:3

- Association rules: (many more!)
  - *Beer* → *Diaper* (60%, 100%)
  - *Diaper* → *Beer* (60%, 75%)

Closed Patterns and Max-Patterns

- A long pattern contains a combinatorial number of sub-patterns, e.g., $\{a_1, \ldots, a_{100}\}$ contains $({}_{100}{}^{1}) + ({}_{100}{}^{2}) + \ldots + ({}_{1}{}^{1}{}_{0}{}^{0}{}_{0}{}^{0}) = 2^{100} - 1 = 1.27*10^{30}$ sub-patterns!
- Solution: *Mine closed patterns and max-patterns instead*
- An itemset X is closed if X is *frequent* and there exists *no super-pattern* Y ɔ X, *with the same support* as X (proposed by Pasquier, et al. @ ICDT'99)
- An itemset X is a max-pattern if X is frequent and there exists no frequent super-pattern Y ɔ X (proposed by Bayardo @ SIGMOD'98)
- Closed pattern is a lossless compression of freq. patterns
  - Reducing the # of patterns and rules

Closed Patterns and Max-Patterns

- Exercise. DB = $\{<a_1, \ldots, a_{100}>, < a_1, \ldots, a_{50}>\}$
  - Min_sup = 1.
- What is the set of closed itemset?
  - $<a_1, \ldots, a_{100}>$: 1
  - $< a_1, \ldots, a_{50}>$: 2
- What is the set of max-pattern?
  - $<a_1, \ldots, a_{100}>$: 1
- What is the set of all patterns?
  - !!

Computational Complexity of Frequent Itemset Mining

- How many itemsets are potentially to be generated in the worst case?
  - The number of frequent itemsets to be generated is senstive to the minsup threshold
  - When minsup is low, there exist potentially an exponential number of frequent itemsets
  - The worst case: M^N where M: # distinct items, and N: max length of transactions
- The worst case complexty vs. the expected probability
  - Ex. Suppose Walmart has 10^4 kinds of products
    - The chance to pick up one product 10^{-4}
    - The chance to pick up a particular set of 10 products: $\sim 10^{-40}$
    - What is the chance this particular set of 10 products to be frequent 10^3 times in 10^9 transactions?

Mining Frequent Patterns, Association and Correlations: Basic Concepts and Methods

- Basic Concepts
- Frequent Itemset Mining Methods
- Which Patterns Are Interesting?—Pattern Evaluation Methods
- Summary

Scalable Frequent Itemset Mining Methods

- Apriori: A Candidate Generation-and-Test Approach
- Improving the Efficiency of Apriori
- FPGrowth: A Frequent Pattern-Growth Approach
- ECLAT: Frequent Pattern Mining with Vertical Data Format

The Downward Closure Property and Scalable Mining Methods

- The downward closure property of frequent patterns
  - Any subset of a frequent itemset must be frequent
  - If **{beer, diaper, nuts}** is frequent, so is **{beer, diaper}**
  - i.e., every transaction having {beer, diaper, nuts} also contains {beer, diaper}
- Scalable mining methods: Three major approaches
  - Apriori (Agrawal & Srikant@VLDB'94)
  - Freq. pattern growth (FPgrowth—Han, Pei & Yin @SIGMOD'00)
  - Vertical data format approach (Charm—Zaki & Hsiao @SDM'02)

Apriori: A Candidate Generation & Test Approach

- Apriori pruning principle: If there is any itemset which is infrequent, its superset should not be generated/tested! (Agrawal & Srikant @VLDB'94, Mannila, et al. @ KDD' 94)
- Method:
  - Initially, scan DB once to get frequent 1-itemset
  - Generate length (k+1) candidate itemsets from length k frequent itemsets
  - Test the candidates against DB
  - Terminate when no frequent or candidate set can be generated

The Apriori Algorithm—An Example

$Sup_{min} = 2$

Database TDB

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

1st scan

C_1

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

L_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

L_2

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

C_3

Itemset
{B, C, E}

3rd scan

L_3

Itemset	sup
{B, C, E}	2

The Apriori Algorithm (Pseudo-Code)

C_k: Candidate itemset of size k
L_k : frequent itemset of size k

L_1 = {frequent items};
for (k = 1; L_k !=$\varnothing$; k++) **do begin**
    C_{k+1} = candidates generated from L_k;
    for each transaction t in database do
        increment the count of all candidates in C_{k+1} that
        are contained in t
    L_{k+1} = candidates in C_{k+1} with min_support
    end
return $\cup_k L_k$;

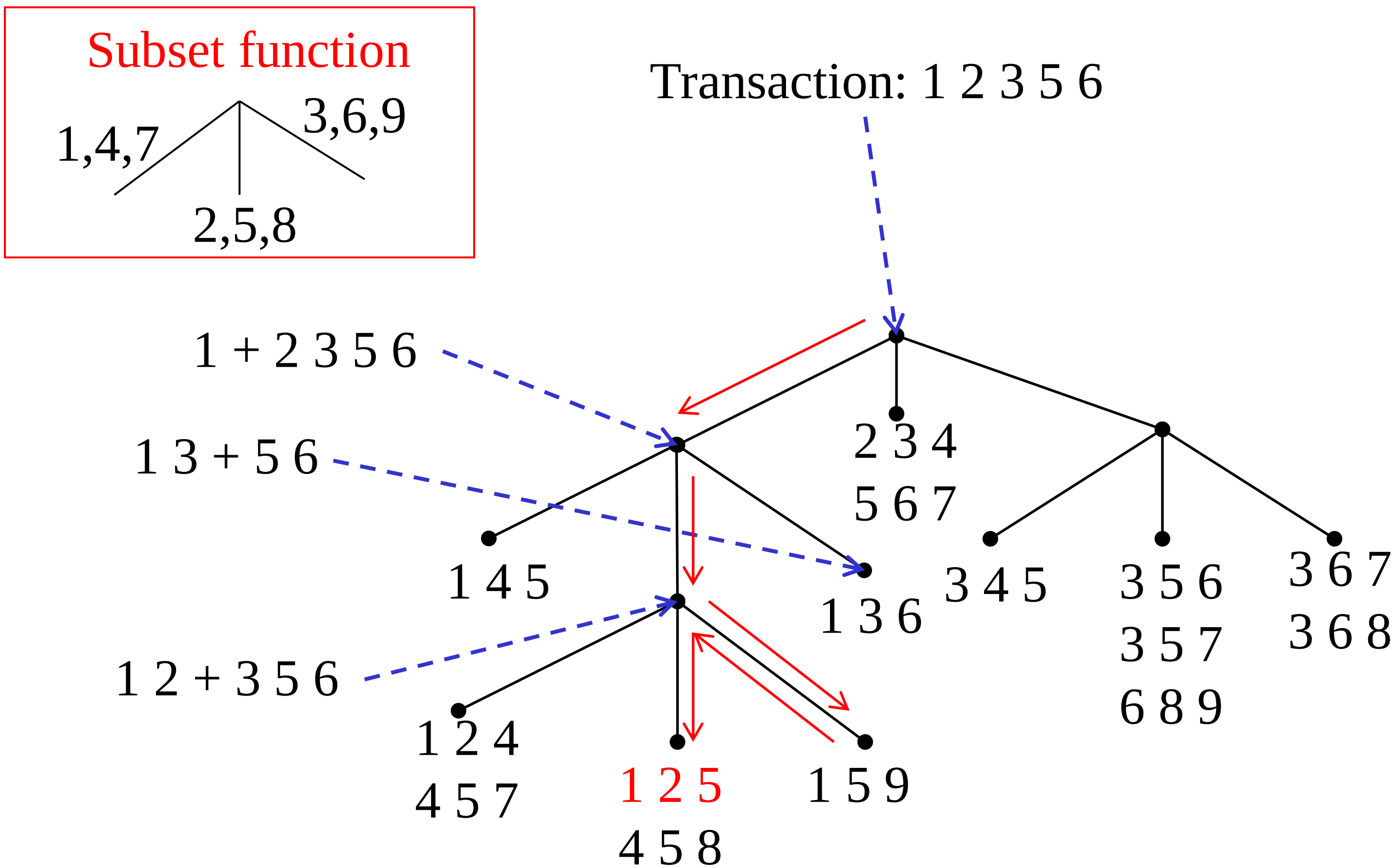
Implementation of Apriori

- How to generate candidates?
  - Step 1: self-joining L_k
  - Step 2: pruning
- Example of Candidate-generation
  - L_3={*abc, abd, acd, ace, bcd*}
  - Self-joining: L_3*L_3
    - *abcd* from *abc* and *abd*
    - *acde* from *acd* and *ace*
  - Pruning:
    - *acde* is removed because *ade* is not in L_3
  - C_4 = {*abcd*}

How to Count Supports of Candidates?

- Why counting supports of candidates a problem?
  - The total number of candidates can be very huge
  - One transaction may contain many candidates
- Method:
  - Candidate itemsets are stored in a *hash-tree*
  - *Leaf* node of hash-tree contains a list of itemsets and counts
  - *Interior* node contains a hash table
  - *Subset function*: finds all the candidates contained in a transaction

Counting Supports of Candidates Using Hash Tree



Candidate Generation: An SQL Implementation

- SQL Implementation of candidate generation
  - Suppose the items in L_{k-1} are listed in an order
  - Step 1: self-joining L_{k-1}

    insert into $\boldsymbol{C_k}$

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

    from ***L_{k-1} p, L_{k-1} 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

    forall ***itemsets c in C_k*** do

    forall ***(k-1)-subsets s of c*** do

    if *(s is not in L_{k-1})* **then delete** *c* **from** C_k
- Use object-relational extensions like UDFs, BLOBs, and Table functions for efficient implementation [See: S. Sarawagi, S. Thomas, and R. Agrawal. Integrating association rule mining with relational database systems: Alternatives and implications. SIGMOD'98]

Scalable Frequent Itemset Mining Methods

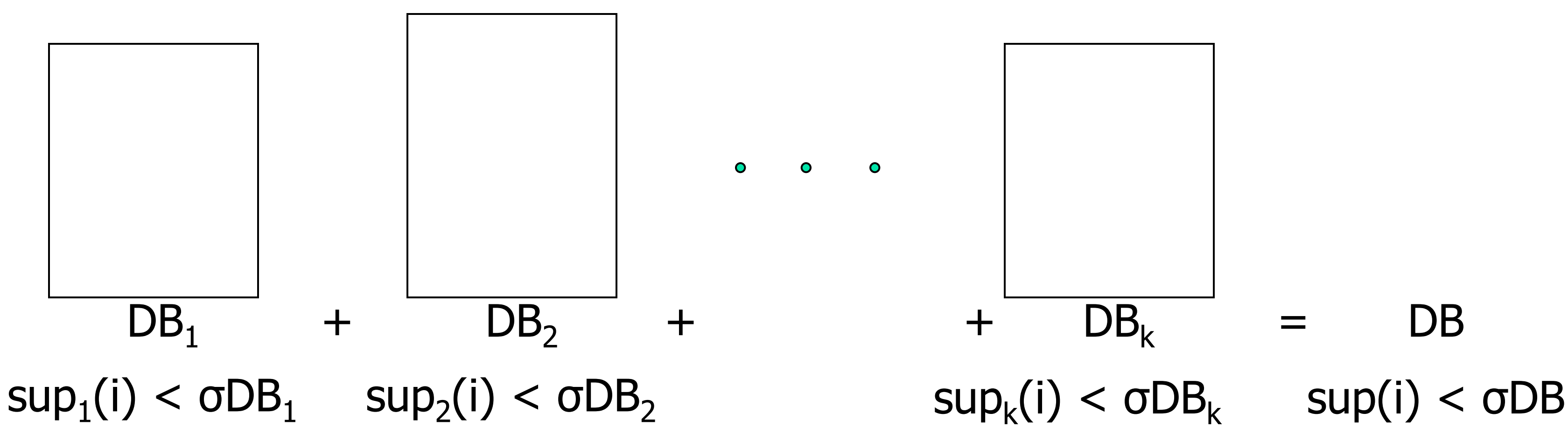
- Apriori: A Candidate Generation-and-Test Approach
- Improving the Efficiency of Apriori
- FPGrowth: A Frequent Pattern-Growth Approach
- ECLAT: Frequent Pattern Mining with Vertical Data Format
- Mining Close Frequent Patterns and Maxpatterns

Further Improvement of the Apriori Method

- Major computational challenges
  - Multiple scans of transaction database
  - Huge number of candidates
  - Tedious workload of support counting for candidates
- Improving Apriori: general ideas
  - Reduce passes of transaction database scans
  - Shrink number of candidates
  - Facilitate support counting of candidates

Partition: Scan Database Only Twice

- Any itemset that is potentially frequent in DB must be frequent in at least one of the partitions of DB
  - Scan 1: partition database and find local frequent patterns
  - Scan 2: consolidate global frequent patterns
- A. Savasere, E. Omiecinski and S. Navathe, *VLDB'95*



DHP: Reduce the Number of Candidates

- A *k*-itemset whose corresponding hashing bucket count is below the threshold cannot be frequent
  - Candidates: a, b, c, d, e
  - Hash entries
    - {ab, ad, ae}
    - {bd, be, de}
    - …
  - Frequent 1-itemset: a, b, d, e
  - ab is not a candidate 2-itemset if the sum of count of {ab, ad, ae} is below support threshold
- J. Park, M. Chen, and P. Yu. An effective hash-based algorithm for mining association rules. *SIGMOD'95*

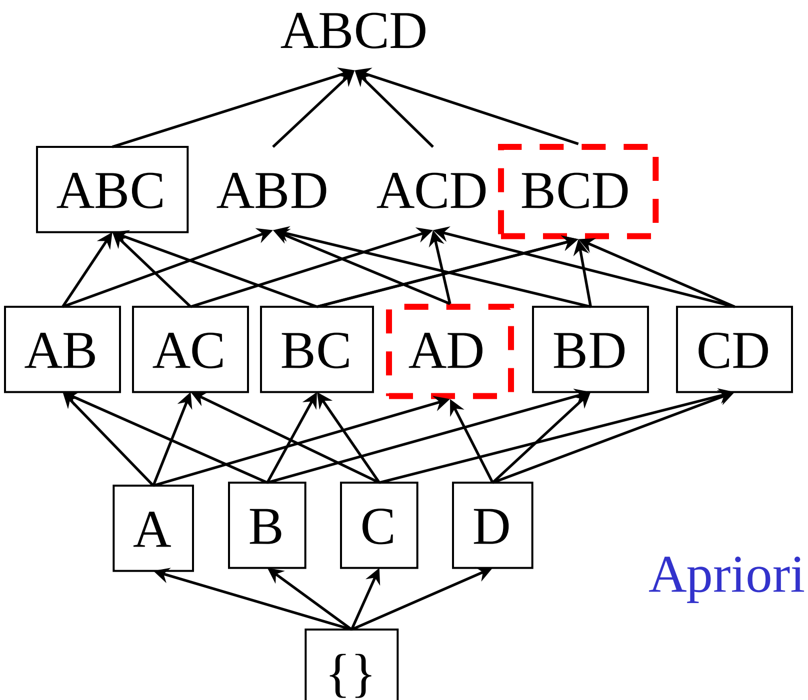
count	itemsets
35	{ab, ad, ae}
88	{bd, be, de}
. . .	. . .
102	{yz, qs, wt}

Hash Table

Sampling for Frequent Patterns

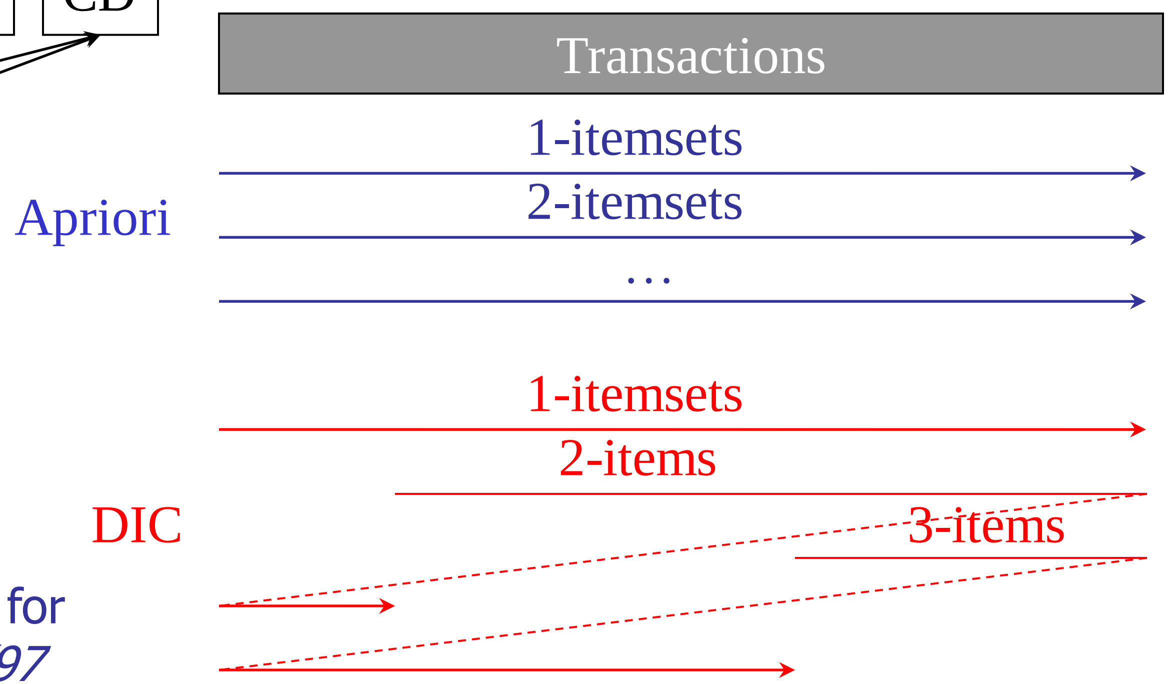
- Select a sample of original database, mine frequent patterns within sample using Apriori
- Scan database once to verify frequent itemsets found in sample, only *borders* of closure of frequent patterns are checked
  - Example: check *abcd* instead of *ab, ac, …, etc.*
- Scan database again to find missed frequent patterns
- H. Toivonen. Sampling large databases for association rules. In *VLDB'96*

DIC: Reduce Number of Scans



Itemset lattice

- Once both A and D are determined frequent, the counting of AD begins
- Once all length-2 subsets of BCD are determined frequent, the counting of BCD begins



S. Brin R. Motwani, J. Ullman, and S. Tsur. Dynamic itemset counting and implication rules for market basket data. *SIGMOD'97*

Scalable Frequent Itemset Mining Methods

- Apriori: A Candidate Generation-and-Test Approach
- Improving the Efficiency of Apriori
- FPGrowth: A Frequent Pattern-Growth Approach
- ECLAT: Frequent Pattern Mining with Vertical Data Format
- Mining Close Frequent Patterns and Maxpatterns

Pattern-Growth Approach: Mining Frequent Patterns Without Candidate Generation

- Bottlenecks of the Apriori approach
  - Breadth-first (i.e., level-wise) search
  - Candidate generation and test
    - Often generates a huge number of candidates
- The FPGrowth Approach (J. Han, J. Pei, and Y. Yin, SIGMOD' 00)
  - Depth-first search
  - Avoid explicit candidate generation
- Major philosophy: Grow long patterns from short ones using local frequent items only
  - "abc" is a frequent pattern
  - Get all transactions having "abc", i.e., project DB on abc: DB|abc
  - "d" is a local frequent item in DB|abc → abcd is a frequent pattern

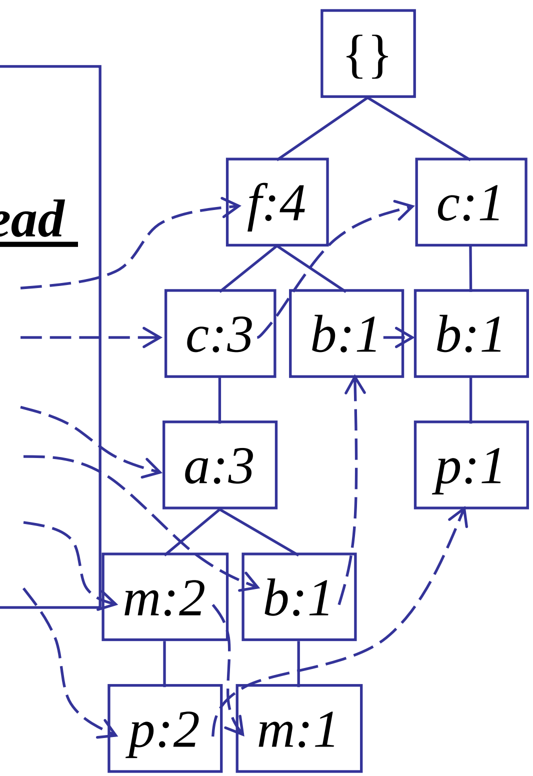
Construct FP-tree from a Transaction Database

TID	*Items bought*	*(ordered) frequent items*
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*}**

***min_support* = 3**

Header Table

Item	*frequency*	*head*
f	4	
c	4	
a	3	
b	3	
m	3	
p	3	



1. Scan DB once, find frequent 1-itemset (single item pattern)
2. Sort frequent items in frequency descending order, f-list
3. Scan DB again, construct FP-tree

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

Partition Patterns and Databases

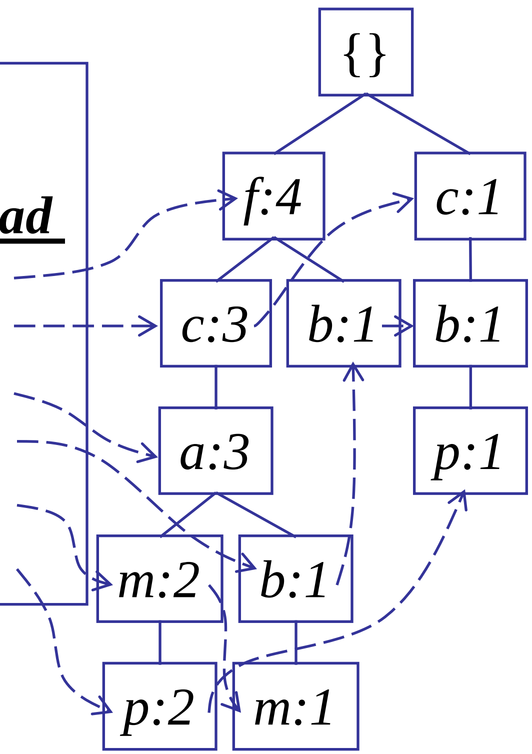
- Frequent patterns can be partitioned into subsets according to f-list
  - F-list = f-c-a-b-m-p
  - Patterns containing p
  - Patterns having m but no p
  - …
  - Patterns having c but no a nor b, m, p
  - Pattern f
- Completeness and non-redundency

Find Patterns Having P From P-conditional Database

- Starting at the frequent item header table in the FP-tree
- Traverse the FP-tree by following the link of each frequent item *p*
- Accumulate all of *transformed prefix paths* of item *p* to form *p*'s conditional pattern base

Header Table

Item	*frequency*	*head*
f	4	
c	4	
a	3	
b	3	
m	3	
p	3	

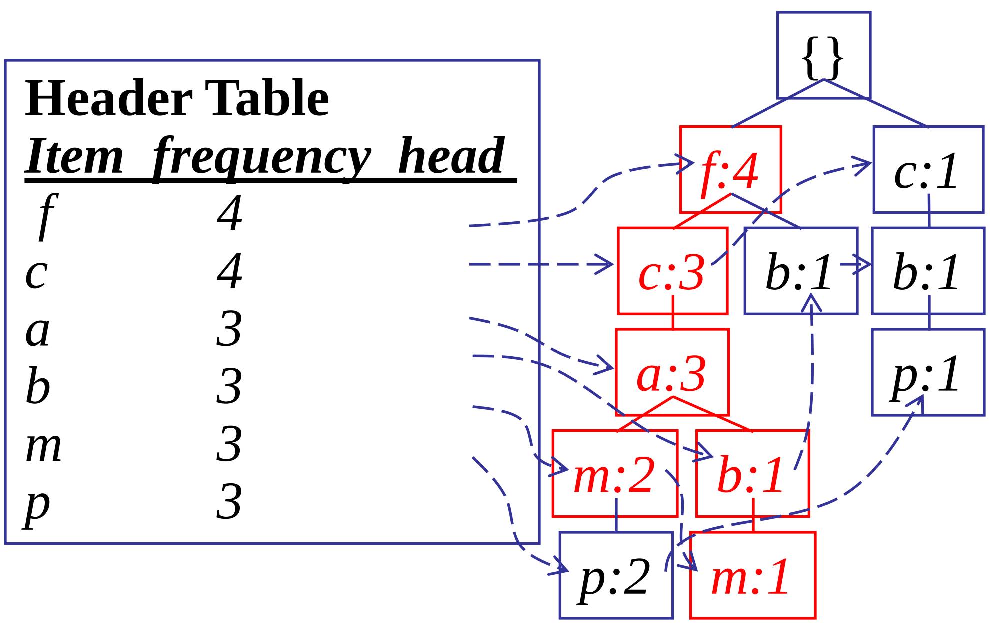


***Conditional* pattern bases**

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

From Conditional Pattern-bases to Conditional FP-trees

- For each pattern-base
  - Accumulate the count for each item in the base
  - Construct the FP-tree for the frequent items of the pattern base



m-conditional **pattern base:**

fca:2, fcab:1

→

{}
|
f:3
|
c:3
|
a:3

m-conditional **FP-tree**

→

All frequent patterns relate to *m*

m,

fm, cm, am,

fcm, fam, cam,

fcam

Recursion: Mining Each Conditional FP-tree

{}
|
f:3
|
c:3
|
a:3

m-conditional **FP-tree**

Cond. pattern base of "am": (fc:3)

{}
|
f:3
|
c:3

am-conditional **FP-tree**

Cond. pattern base of "cm": (f:3)

{}
|
f:3

cm-conditional **FP-tree**

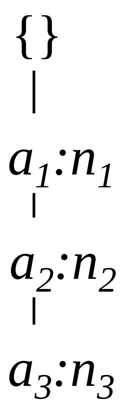
Cond. pattern base of "cam": (f:3)

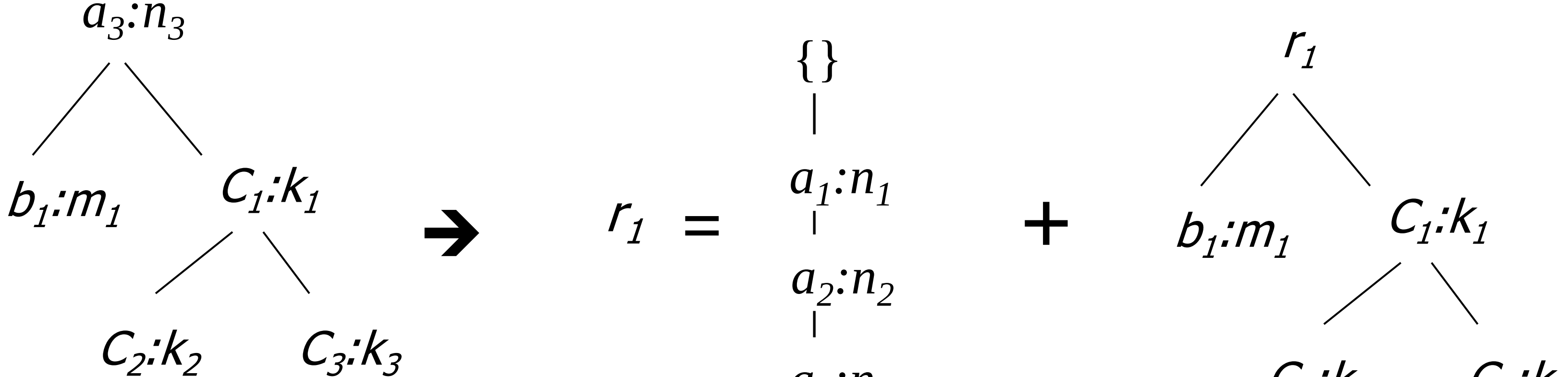
{}
|
f:3

cam-conditional **FP-tree**

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





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)

The Frequent Pattern Growth Mining Method

- Idea: Frequent pattern growth
  - Recursively grow frequent patterns by pattern and database partition
- Method
  - For each frequent item, construct its conditional pattern-base, and then its conditional FP-tree
  - Repeat the process on each newly created conditional FP-tree
  - Until the resulting FP-tree is empty, or it contains only one path—single path will generate all the combinations of its sub-paths, each of which is a frequent pattern