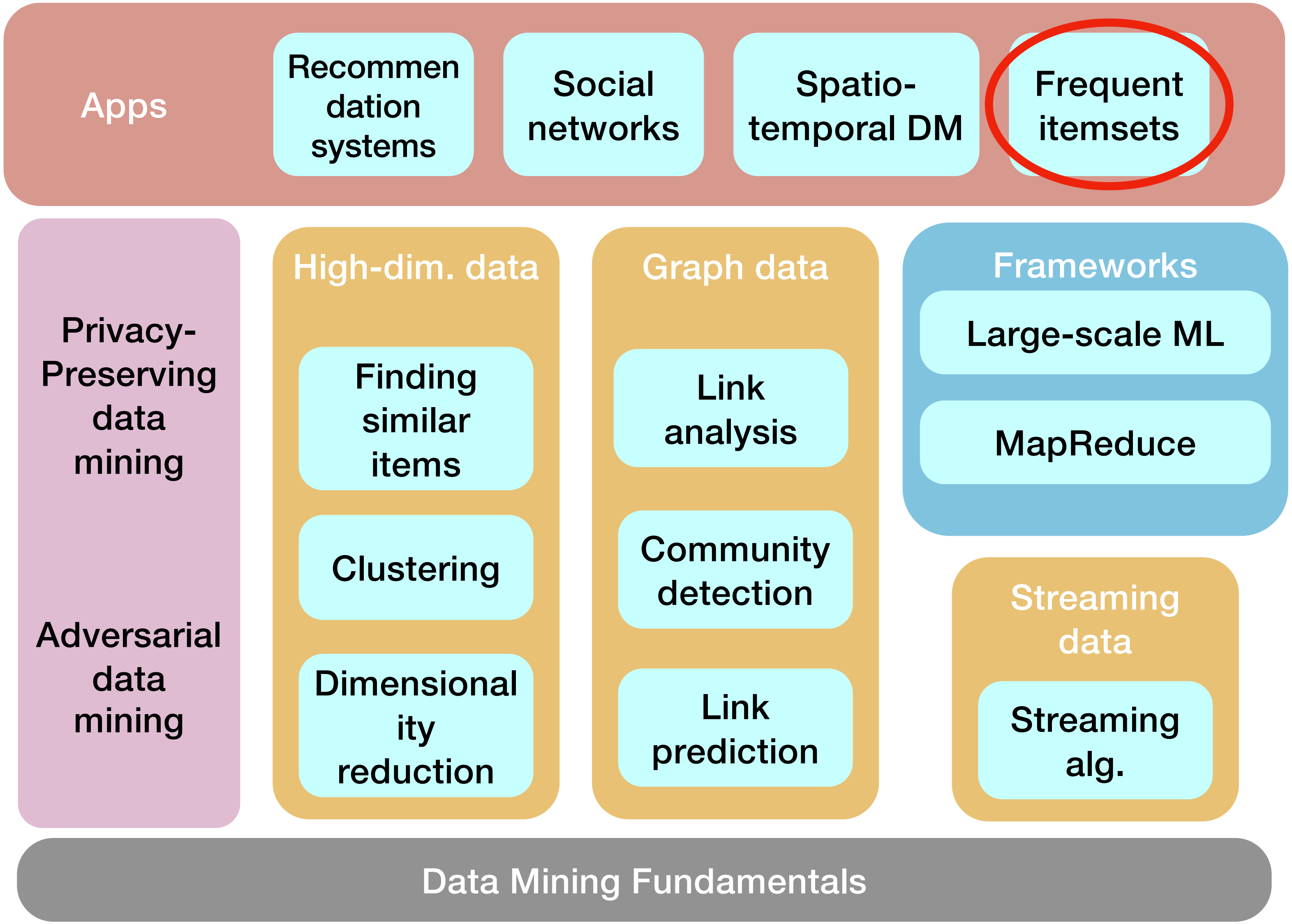
Course Schedule

Sun.	Mon.	Tue.	Wed.	Thur.	Fri.	
		Lecture Apr. 13		Experiment Apr. 15		Week 8
		Lecture Apr. 20		Experiment Apr. 22		Week 9
Lecture Apr. 25		Lecture Apr. 27		Experiment Apr. 29		Week 10
				Lecture May 6		Week 11
		Lecture May 11		Experiment May 13		Week 12

Frequent Itemsets

Liyao Xiang
http://xiangliyao.cn/
Shanghai Jiao Tong University

Course Landscape



Outline

- Basic Concepts in Frequent Pattern Mining
- Frequent Itemset Mining Methods
- Pattern Evaluation Methods

Outline

- Basic Concepts in Frequent Pattern Mining
- Frequent Itemset Mining Methods
- Pattern Evaluation Methods

Basic Concepts

- Frequent pattern: a pattern (a set of items, subsequences, substructures …) that appear frequently in a database

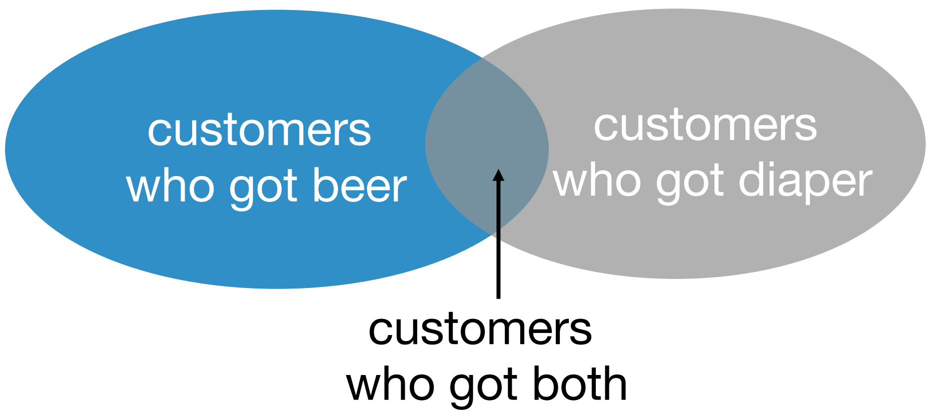
WAL★MART
ALWAYS LOW PRICES
Always
WE SELL FOR LESS
MANAGER DAVID KINMAN
(816) 224 - 4800
ST# 4068 OP# 00002825 TE# 39 TR# 03366
HAMBCOMBO 007874235056K 3.17 X
SLC OF CHEES 007874223757K 0.20 X
** VOIDED ENTRY **
HAMBCOMBO 007874235055K 3.17-X
** VOIDED ENTRY **
SLC OF CHEES 007874223757K 0.20-X
SUBTOTAL 0.00
CHEESEBURGER 007874223303K 2.17 X
MED DRINK 007874223281K 0.98 X
SUBTOTAL 3.15
TAX 1 3.000 % 0.09
TAX 3 3.475 % 0.11
TOTAL 3.35
CASH TEND 20.35
CHANGE DUE 17.00
4-21 ORDER #666
TC# 3680 8362 6229 1132 7028
Ready for the Weekend?
Check Out Lawn and Garden
04/21/05 12:25:25

- Finding frequent patterns is key to mining associations, correlations, clustering, classification and other relationships among data.
- Applications: basket data analysis, cross-marketing, catalog design …

Basic Concepts

- 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: the fraction of transactions that contains X over all transaction
- An itemset X is frequent if X's support is no less than a defined threshold min_sup

TID	Items Purchased
10	Beer, Nuts, Diaper
20	Beer, Coffee, Diaper
30	Beer, Diaper, Eggs
40	Nuts, Eggs, Milk
50	Nuts, Coffee, Diaper, Eggs, Milk



Basic Concepts

- support: probability that a transaction contains X&Y

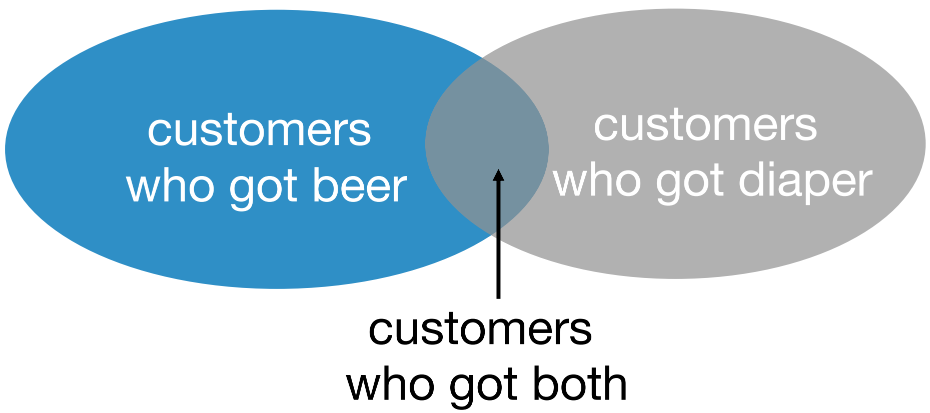
  support(X⇒Y) = P(X&Y)

- confidence: conditional prob. that a transaction having X also contains Y

  confidence(X⇒Y) = P(Y|X)

$$P(Y|X) = \frac{\text{support}(X\&Y)}{\text{support}(X)}$$

TID	Items Purchased
10	Beer, Nuts, Diaper
20	Beer, Coffee, Diaper
30	Beer, Diaper, Eggs
40	Nuts, Eggs, Milk
50	Nuts, Coffee, Diaper, Eggs, Milk



Basic Concepts

- min_sup: minimum support threshold
- min_conf: minimum support confidence threshold
- e.g., find all rules X ⇒ Y with min_sup and min_conf

let min_sup = 50%, min_conf = 50%

frequent pattern: Beer: 3, Nuts: 3, Diaper: 4, Eggs: 3, {Beer, Diaper}: 3

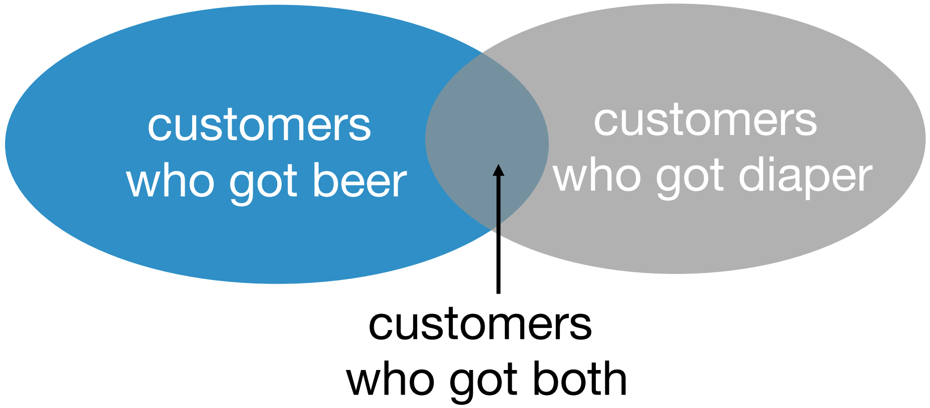
- Association rules:

  Beer⇒Diaper (60%, 100%)

  Diaper⇒Beer (60%, 75%)

$$P(Y|X) = \frac{\text{support}(X\&Y)}{\text{support}(X)}$$

TID	Items Purchased
10	Beer, Nuts, Diaper
20	Beer, Coffee, Diaper
30	Beer, Diaper, Eggs
40	Nuts, Eggs, Milk
50	Nuts, Coffee, Diaper, Eggs, Milk



Basic Concepts

- Association rule mining includes:
    1. Find all frequent itemsets: frequency of itemsets ≥ min_sup
    2. Generate strong association rules from the frequent itemsets
- 1 is the major step, but challenging in that there may be a huge number of itemsets satisfying min_sup
- An itemset is frequent ⇒ each of its subsets is frequent
- Solution: mine closed frequent itemset and maximal frequent itemset
- closed frequent itemset X: X is frequent and there is no super-itemset Y ⊃ X with the same support count as X
    - closed frequent itemset is a lossless compression of frequent itemset
- maximal frequent itemset X: X is frequent and there is no super-itemset Y ⊃ X which is frequent

- e.g., {$<a_1, \ldots, a_{100}>$, $< a_1, \ldots, a_{50}>$}, min_sup = 1
- What is the set of closed frequent itemset?
    - $<a_1, \ldots, a_{100}>$: 1, $< a_1, \ldots, a_{50}>$: 2
- What is the set of maximal frequent itemset?
    - $<a_1, \ldots, a_{100}>$: 1
    - We can assert $<a_2, a_{45}>$ is frequent since $a_2, a_{45} \in < a_1, \ldots, a_{100}>$ but cannot assert their actual support count
- How many itemsets are potentially to be generated in the worst case?
    - When min_sup is low, there exist potentially an exponential number of frequent itemsets
    - Worst case: M^N where M = # distinct items, N = max length of transactions

Summary

- frequent pattern
- k-itemset
- (absolute) support, support count, relative support
- min_sup, confidence
- closed frequent itemset, maximal frequent itemset

Outline

- Basic Concepts in Frequent Pattern Mining
- Frequent Itemset Mining Methods
- Pattern Evaluation Methods

Frequent Itemset Mining Methods

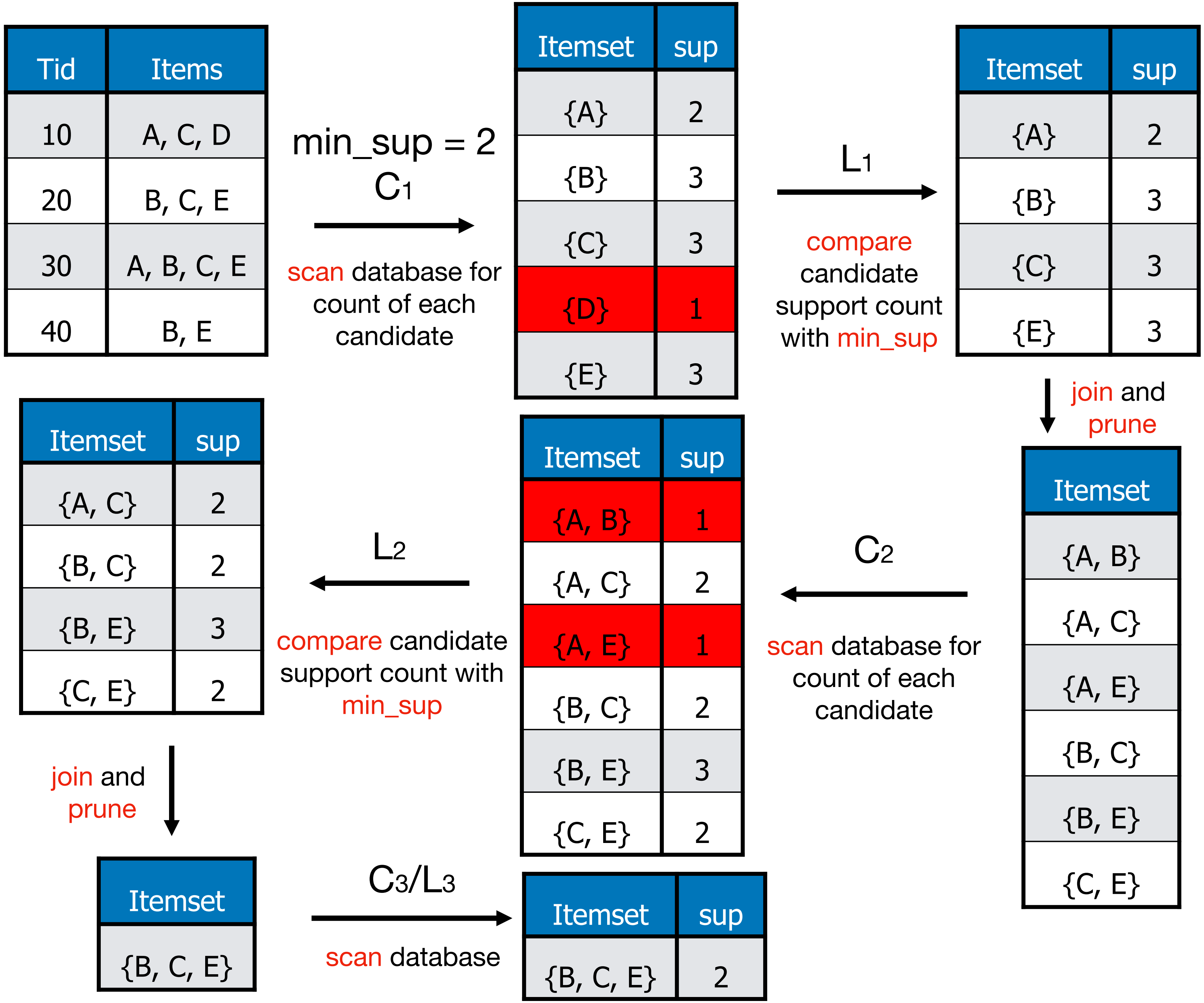
- Apriori: A Candidate Generation-and-Test Approach
- Improving the Efficiency of Apriori
- FP-Growth: A Frequent Pattern-Growth Approach
- ECLAT: Frequent Pattern Mining with Vertical Data Format

Apriori

- Downward Closure Property: any subset of a frequent itemset must be frequent
  - e.g., if {beer, diaper, nuts} is frequent, so is {beer, diaper} since every transaction having {beer, diaper, nuts} also contains {beer, diaper}
- Apriori employs a level-wise search where k-itemsets are used to explore (k + 1)-itemsets. Steps:
  1. Scan database once to get frequent 1-itemsets L_1
  2. Join the k-frequent itemsets L_k to generate length (k+1) candidate itemsets C'_{k+1}
  3. Prune C'_{k+1} against the database to get C_{k+1}
  4. Scan (Test) database for the count of each candidate in C_{k+1}, obtain L_{k+1}
  5. Terminate when no frequent or candidate set can be generated

Apriori

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



Tid
Items
10
A, C, D
20
B, C, E
30
A, B, C, E
40
B, E
min_sup = 2
C1
scan database for count of each candidate
Itemset
sup
{A}
2
{B}
3
{C}
3
{D}
1
{E}
3
L1
compare candidate support count with min_sup
Itemset
sup
{A}
2
{B}
3
{C}
3
{E}
3
join and prune
Itemset
{A, B}
{A, C}
{A, E}
{B, C}
{B, E}
{C, E}
C2
scan database for count of each candidate
Itemset
sup
{A, B}
1
{A, C}
2
{A, E}
1
{B, C}
2
{B, E}
3
{C, E}
2
L2
compare candidate support count with min_sup
Itemset
sup
{A, C}
2
{B, C}
2
{B, E}
3
{C, E}
2
join and prune
Itemset
{B, C, E}
C3/L3
scan database
Itemset
sup
{B, C, E}
2

Apriori

C_k: Candidate itemset of size k

L_k: Frequent itemset of size k

L_1 = {1-frequent items};

for (k = 1; L_k !=Ø; k++) **do begin**

  C_{k+1} = candidates generated from L_k; // join and prune

  for each transaction t in database do

    increment the count of all candidates in C_{k+1} that are

     contained in t

  end

  L_{k+1} = candidates in C_{k+1} with min_sup

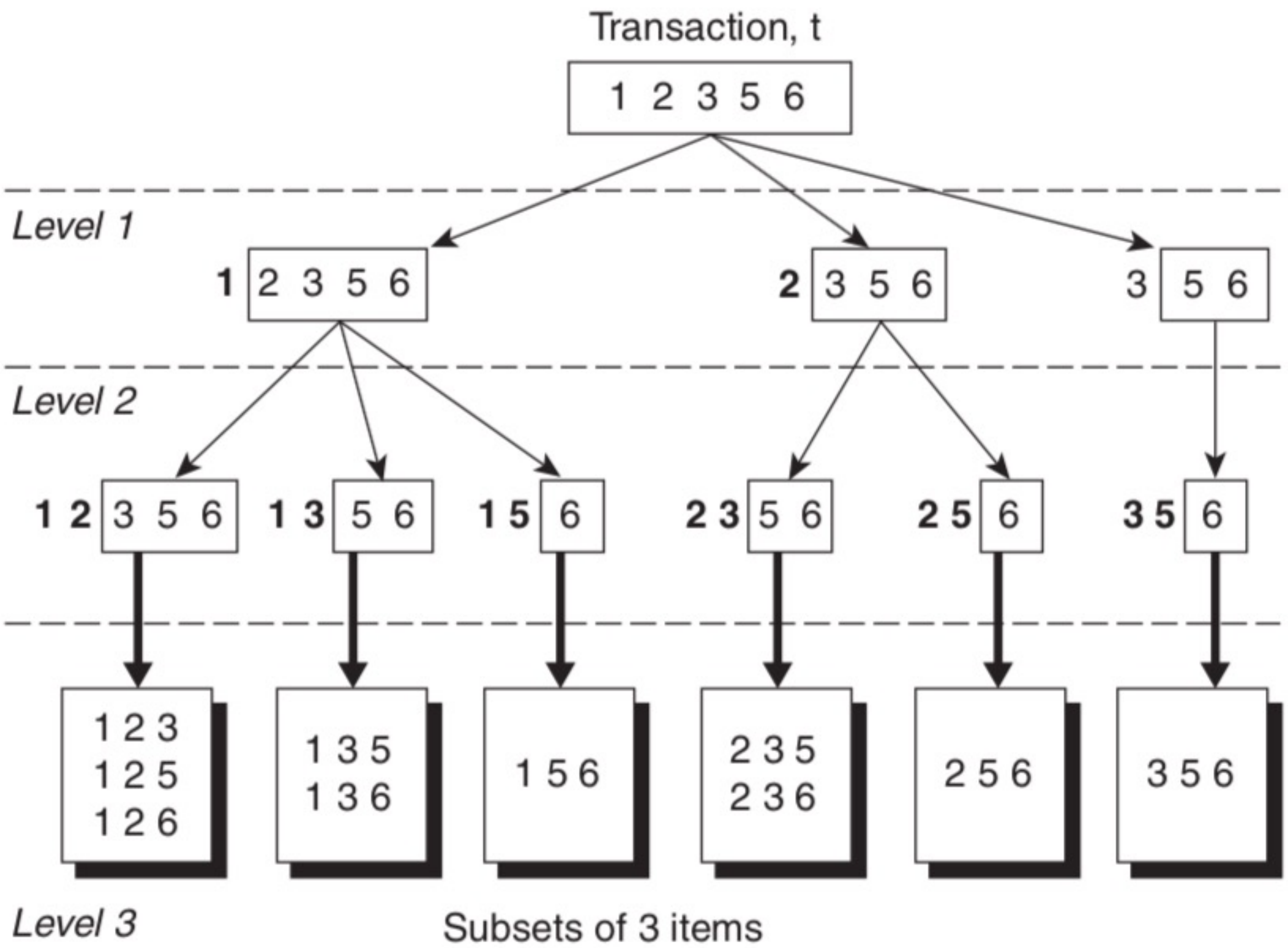
end

return $\cup_k L_k$;

Apriori

- How to count supports of each candidate?
  - The total number of candidates can be huge
  - One transaction may contain many candidates
  - Support Counting Method:
    - store candidate itemsets in a hash-tree
    - leaf node of hash-tree contains a list of itemsets and counts
    - interior node contains a hash table

Apriori



Prefix structure enumerating 3-itemset in Transaction t

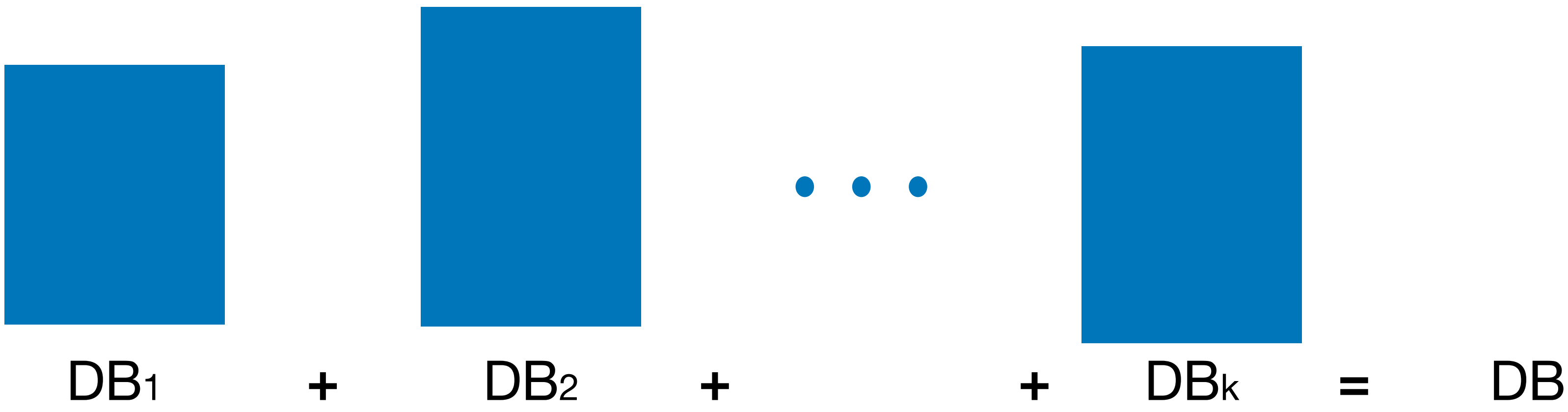
Figures from https://www-users.cs.umn.edu/~kumar001/dmbook/ch6.pdf

Improving the Efficiency of Apriori

- Challenges:
  - Multiple scans of transaction database
  - Huge number of candidates
  - Support counting for candidates
- Improving the Efficiency of Apriori
  - Reduce passes of transaction database scans
  - Shrink number of candidates
  - Facilitate support counting of candidates

Improving the Efficiency of Apriori

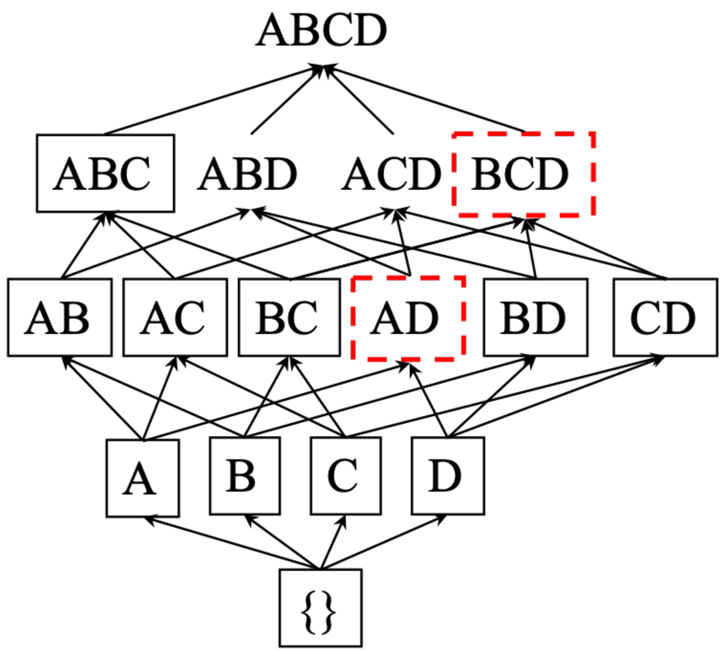
- Partition (reduce scans): partition data to find candidate itemsets
  - Any itemset that is potentially frequent (relative support ≥ min_sup) must be frequent (relative support in the partition ≥ min_sup) in at least one of the partition
    - Scan 1: partition database and find local frequent patterns
    - Scan 2: assess the actual support of each candidate to determine the global frequent itemsets

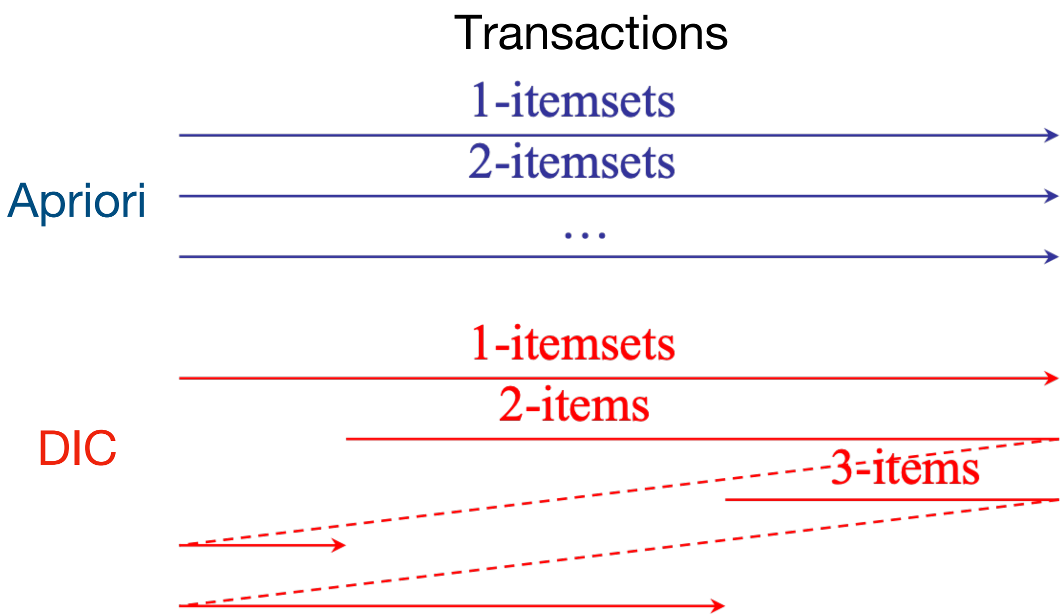


Improving the Efficiency of Apriori

- Dynamic itemset counting (reduce scans): adding candidate itemsets at different points during a scan

- new candidate itemsets can be added at any start point (rather than determined only before scan)





- 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

Improving the Efficiency of Apriori

- Hash-based Technique (shrink number of candidates): hashing itemsets into corresponding buckets
  - A k-itemset whose corresponding hashing bucket count is below min_sup cannot be frequent

min_sup = 3

Create hash table H_2 using hash function $h(x, y) = ((order\ of\ x) \times 10 + (order\ of\ y))\ mod\ 7$

H_2

bucket address	0	1	2	3	4	5	6
bucket count	2	2	4	2	2	4	4
bucket contents	{I1, I4} {I3, I5}	{I1, I5} {I1, I5}	{I2, I3} {I2, I3} {I2, I3} {I2, I3}	{I2, I4} {I2, I4}	{I2, I5} {I2, I5}	{I1, I2} {I1, I2} {I1, I2} {I1, I2}	{I1, I3} {I1, I3} {I1, I3} {I1, I3}

h(1, 4) = 1 * 10 + 4 = 0 mod 7
h(3, 5) = 3 * 10 + 5 = 0 mod 7

Improving the Efficiency of Apriori

- Sampling: mining on a subset of the given data
  - Trade off some degree of accuracy against efficiency
  - Select sample S of original database, mine frequent patterns within S (a lower support threshold) instead of the entire database —> the set of frequent itemsets local to S = L_S
  - Scan the rest of database once to compute the actual frequencies of each itemset in L_S
  - If L_S actually contains all the frequent itemsets, stop; otherwise
  - Scan database again for possible missing frequent itemsets

Frequent Itemset Mining Methods

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

A Frequent-Pattern Growth Approach

- Bottlenecks of Apriori
  - Breadth-first (i.e., level-wise) search
  - Candidate generation and test, often generates a huge number of candidates
- FP-Growth
  - Depth-first search
  - Avoid explicit candidate generation
  - Grow long patterns from short ones using local frequent items
    - “abc” is a frequent pattern
    - Get all transactions having “abc,” i.e., project database D on abc: D | abc
    - “d” is a local frequent item in D | abc —> abcd is a frequent pattern

A Frequent-Pattern Growth Approach

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_sup = 3

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

1. Scan database once, find frequent 1-itemset
2. Sort frequent items in frequency descending order —> F-list

Header Table

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

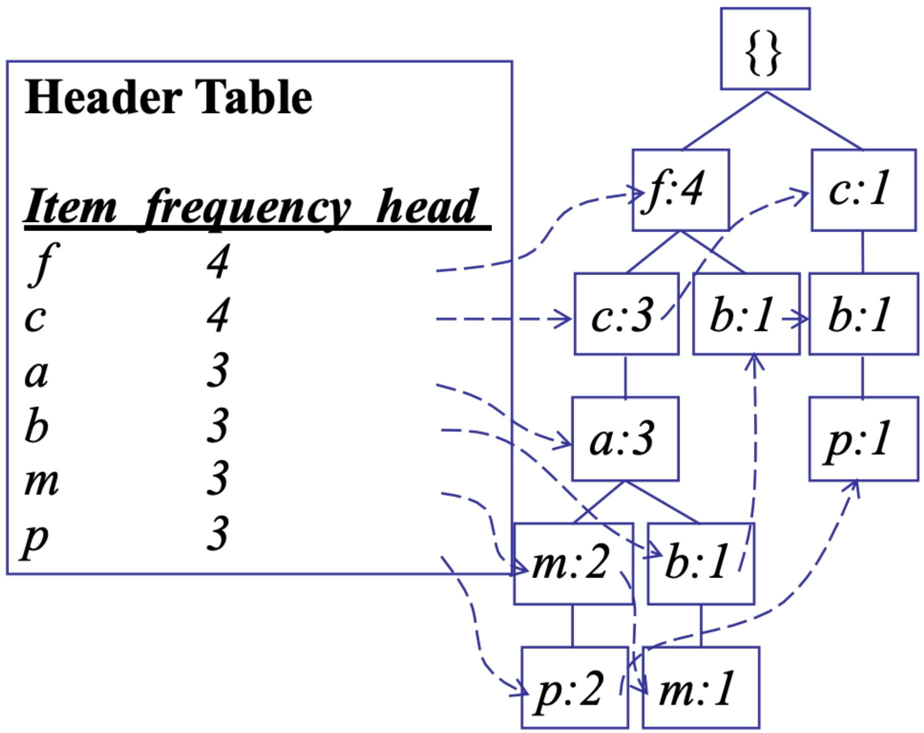
A Frequent-Pattern Growth Approach

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_sup = 3

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

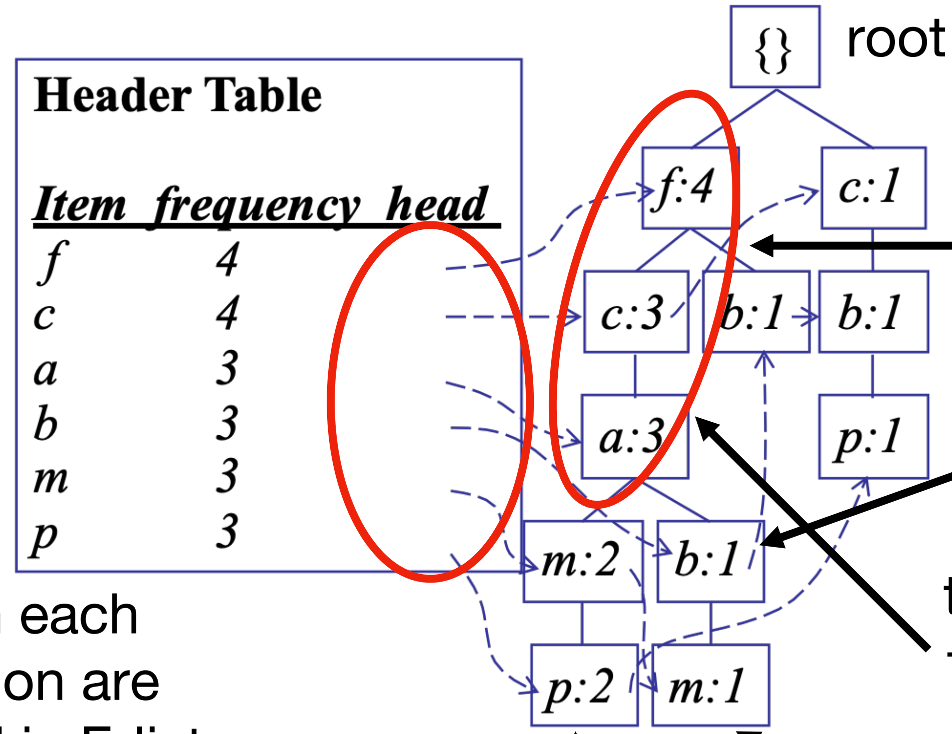
1. Scan database once, find frequent 1-itemset
2. Sort frequent items in frequency descending order —> F-list
3. Scan database again, construct FP-tree
4. Mine FP-tree



How to Construct FP-tree?

FP-tree: a compressed representation of database.
It retains the itemset association information.

To facilitate tree traversal, each item points to its occurrence in the tree via node-link



increment counts of existing nodes

create new nodes

two branches share the common prefix: f,c,a

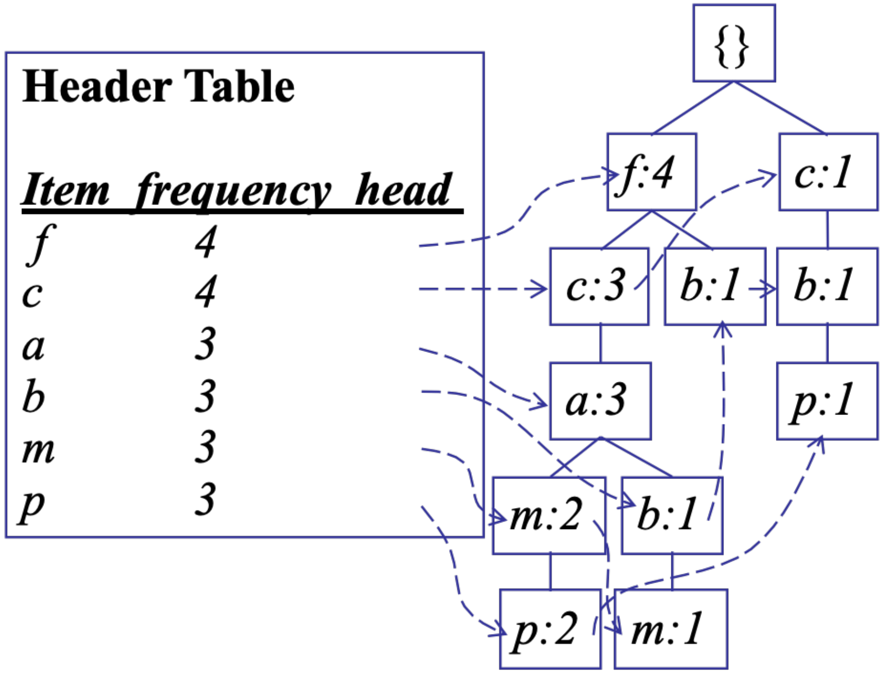
Items in each transaction are processed in F-list order

1st branch is created for transaction: f,c,a,m,p

2nd branch is created for transaction: f,c,a,b,m

How to Mine FP-tree?

1. Start from each frequent length-1 pattern (suffix pattern, usually the last item in F-list) to construct its conditional pattern base (prefix paths co-occurring with the suffix)

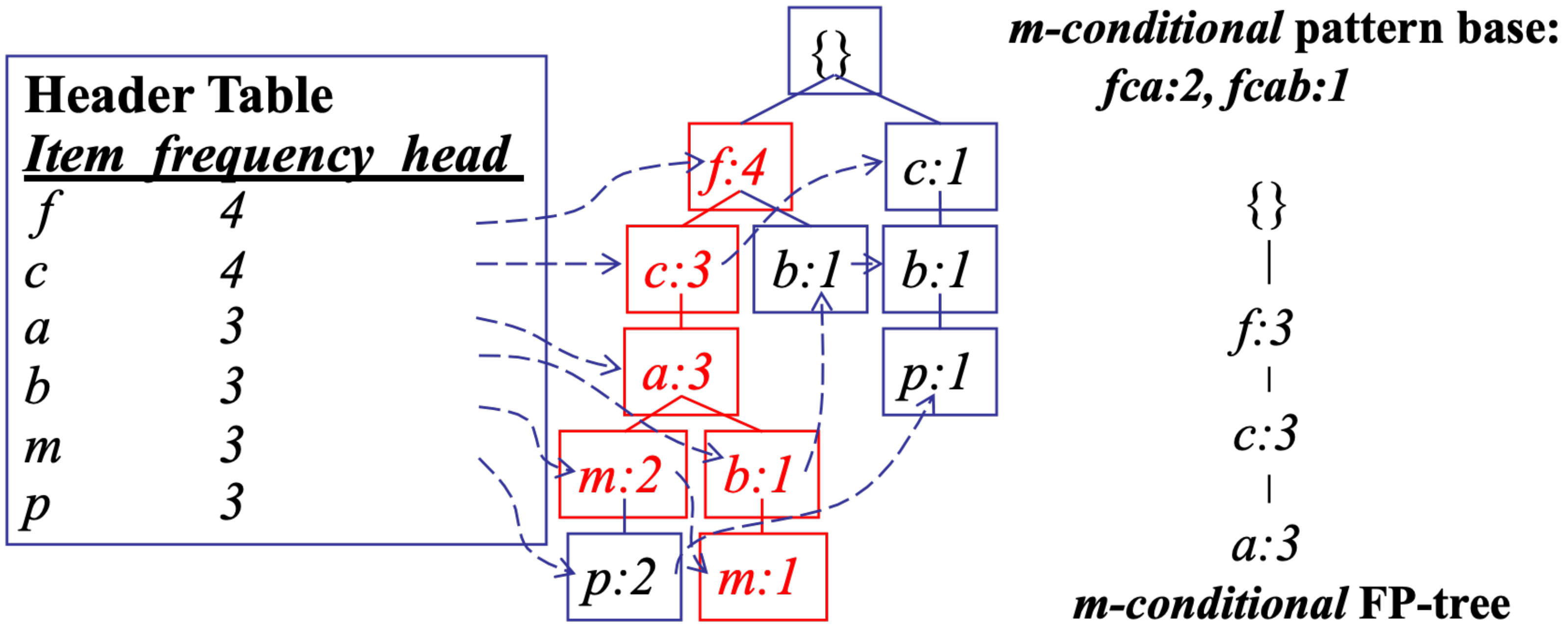


Conditional **pattern bases**

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***

How to Mine FP-tree?

1. Start from each frequent length-1 pattern (suffix pattern, usually the last item in F-list) to construct its conditional pattern base
2. Construct the conditional FP-tree based on the conditional pattern base



How to Mine FP-tree?

1. Start from each frequent length-1 pattern (suffix pattern, usually the last item in F-list) to construct its conditional pattern base
2. Construct the conditional FP-tree based on the conditional pattern base
3. Mining recursively on each conditional FP-tree until the resulting FP-tree is empty, or it contains only a single path — which will generate frequent patterns out of all combinations of its sub-paths

***m-conditional* pattern base:**

fca:2, fcab:1

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

***m-conditional* FP-tree**

All frequent patterns relating to m:
m, fm, cm, am, fcm, fam, cam, fcam

{}
|
f:3
|
c:3

***am-conditional* FP-tree**

fc: 3

{}
|
f:3

***cm-conditional* FP-tree**

f: 3

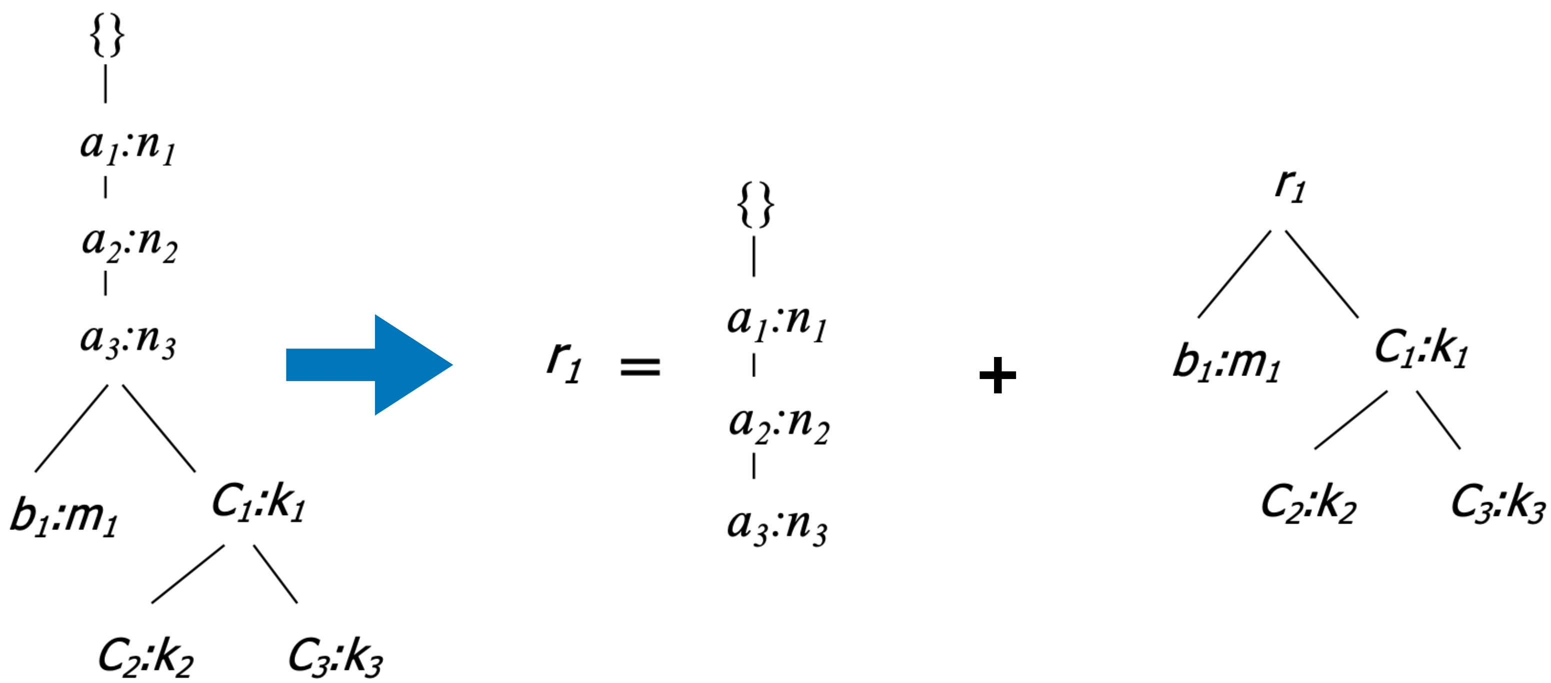
{}
|
f:3

***cam-conditional* FP-tree**

f: 3

Single Prefix Path in FP-tree

- Suppose a (conditional) FP-tree has a shared single prefix-path
- 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

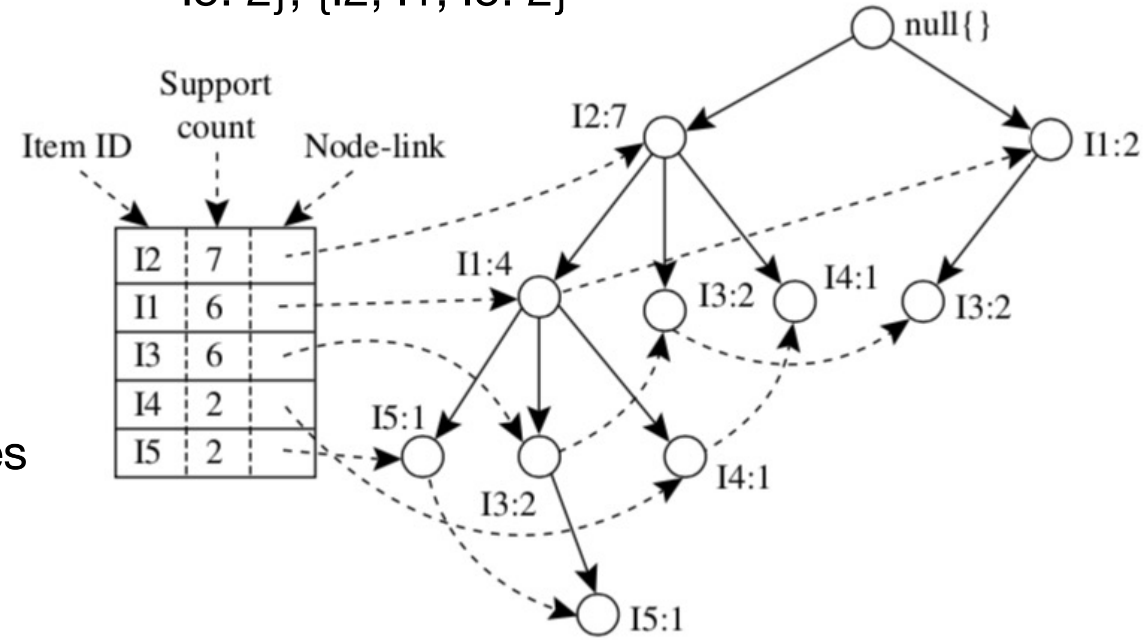


The 2nd Example of FP-Growth

Transactional Data for an *AllElectronics* Branch

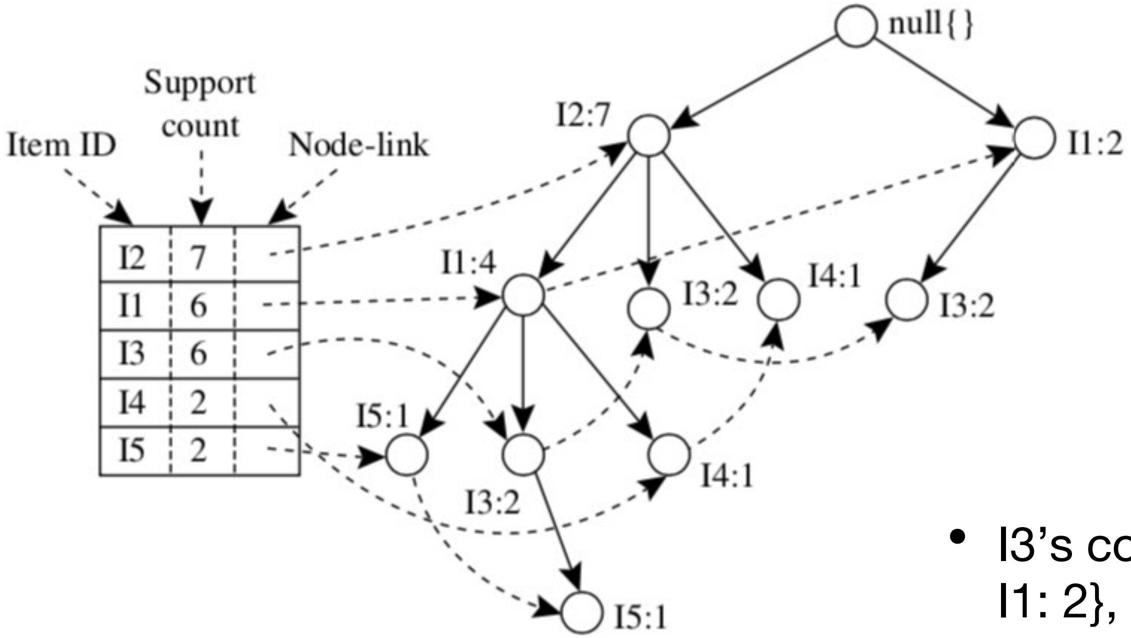
TID	*List of item_IDs*
T100	I1, I2, I5
T200	I2, I4
T300	I2, I3
T400	I1, I2, I4
T500	I1, I3
T600	I2, I3
T700	I1, I3
T800	I1, I2, I3, I5
T900	I1, I2, I3

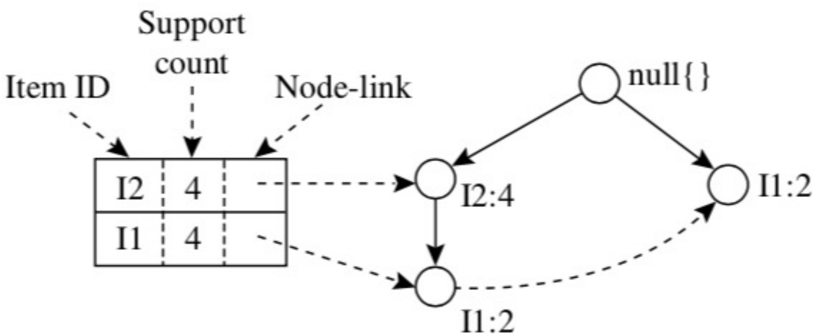
- first we consider I5, the paths formed are <I2, I1, I5: 1> and <I2, I1, I3, I5: 1>
- I5-conditional FP-tree contains only <I2, I1:2>
- I3 is not included since its support count of 1 < min_sup
- The single path generates {I2, I5: 2}, {I1, I5: 2}, {I2, I1, I5: 2}



- Then we consider I4, generates a single-node conditional FP-tree, <I2: 2>, and derives one frequent pattern, {I2, I4: 2}.

The 2nd Example of FP-Growth





- I3’s conditional pattern base: {{I2, I1: 2}, {I2: 2}, {I1: 2}}
- Conditional FP-tree has two branches: <I2: 4, I1: 2> and <I1: 2>
- Frequent patterns {{I2, I3: 4}, {I1, I3: 4}, {I2, I1, I3: 2}}

Scaling FP-Growth

- What if FP-tree cannot fit into memory?
  - Database projection: partition a database into a set of projected databases, then construct and mine FP-tree for each projected database
  - Parallel projection:
    - project the database in parallel for each frequent item
    - all partitions are processed in parallel
    - space costly
  - Partition projection:
    - project a transaction to a frequent item x if there is no any other item after x in the list of frequent items appearing in the transaction
    - a transaction is projected to only one projected database

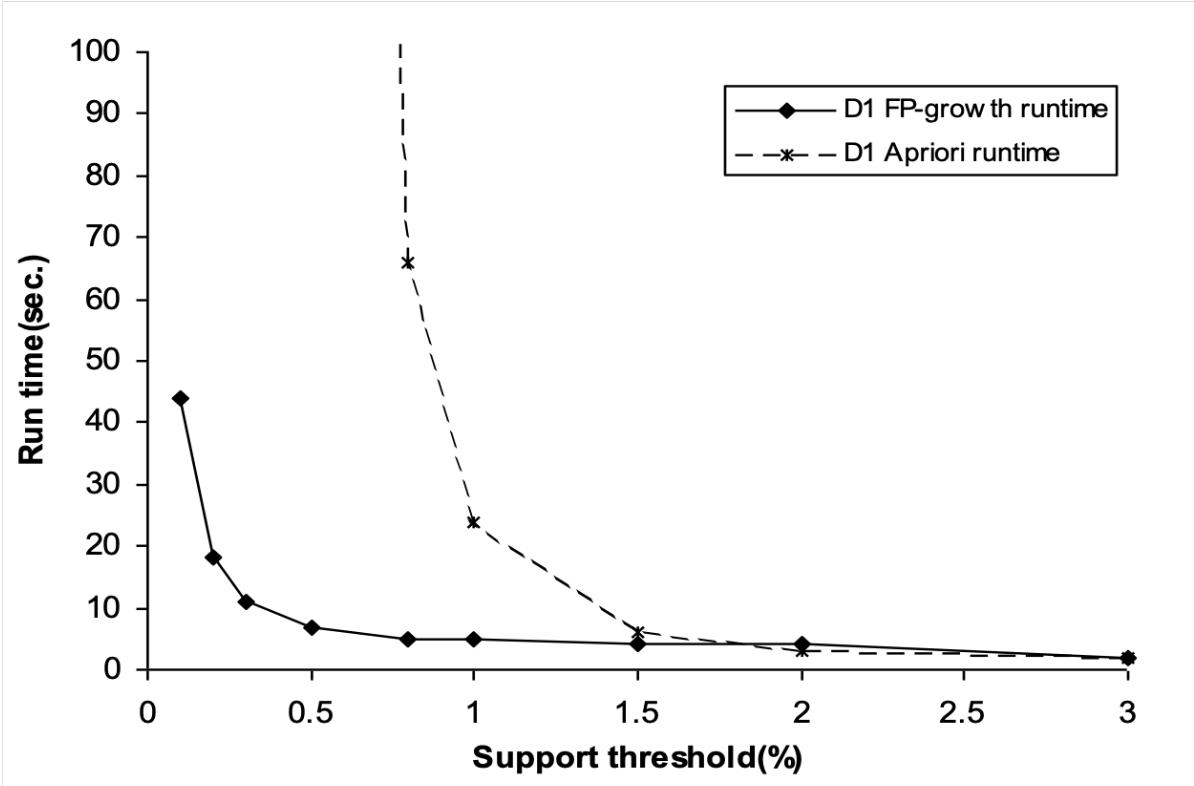
Benefits of FP-tree

- 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: occurs more frequently, the more likely to be shared
  - Never be larger than the original database (not including node-links and the count fields)

Benefits of FP-Growth

- Divide-and-conquer:
  - Decompose both the mining task and database according to the frequent patterns obtained so far
  - Lead to focused search of smaller databases
- Other factors:
  - No candidate generation, no candidate test
  - Compressed database: FP-tree
  - No repeated scan of the entire database
  - Basic operations: count local frequent items and build sub FP-tree, no pattern search and matching

Performance of FP-Growth in Large Datasets



FP-Growth vs. Apriori

Frequent Itemset Mining Methods

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

ECLAT: Frequent Pattern Mining with Vertical Data Format

- Vertical data format: itemset — transID_set
  - transID_set: a set of transaction IDs containing the itemset
- Derive frequent patterns based on the intersections of transID_set

itemset	*TID_set*
I1	{T100, T400, T500, T700, T800, T900}
I2	{T100, T200, T300, T400, T600, T800, T900}
I3	{T300, T500, T600, T700, T800, T900}
I4	{T200, T400}
I5	{T100, T800}

itemset	*TID_set*
{I1, I2}	{T100, T400, T800, T900}
{I1, I3}	{T500, T700, T800, T900}
{I1, I4}	{T400}
{I1, I5}	{T100, T800}
{I2, I3}	{T300, T600, T800, T900}
{I2, I4}	{T200, T400}
{I2, I5}	{T100, T800}
{I3, I5}	{T800}

itemset	*TID_set*
{I1, I2, I3}	{T800, T900}
{I1, I2, I5}	{T100, T800}

ECLAT: Frequent Pattern Mining with Vertical Data Format

- Vertical data format: itemset — transID_set
  - transID_set: a set of transaction IDs containing the itemset
- Derive frequent patterns based on the intersections of transID_set
- Use diffset to reduce the cost of storing long transID_set
  - {I1} = {T100, T400, T500, T700, T800, T900}
  - {I1, I2} = {T100, T400, T800, T900}
  - diffset({I1}, {I1, I2}) = {T500, T700}

Summary

- Frequent itemset mining methods:
  - Apriori: candidate generation-and-test
  - Improving efficiency of Apriori: partition, dynamic item counting, hash-based technique, sampling
  - FP-Growth: depth-first search
  - Scaling of FP-Growth: database projection
  - Frequent pattern mining with vertical data format

Outline

- Basic Concepts in Frequent Pattern Mining
- Frequent Itemset Mining Methods
- Pattern Evaluation Methods

Pattern Evaluation Methods: Correlations

- play basketball ⇒ eat cereal [40%, 66.7%] is misleading
  - the overall % of students eating cereal is 75% > 66.7%
- play basketball ⇒ not eat cereal [20%, 33.3%] is more accurate
- Lift: a measure of dependent/correlated event

$$\text{lift} = \frac{P(A,B)}{P(A)P(B)} = \frac{P(B|A)}{P(B)}$$

	Basketball	Not basketball	Sum (row)
Cereal	2000	1750	3750
Not cereal	1000	250	1250
Sum(col.)	3000	2000	5000

$$lift(Basketball, Cereal) = \frac{2000/5000}{(3000/5000) \times (3750/5000)} = 0.89$$

< 1, negatively correlated

$$lift(Basketball, Notcereal) = \frac{1000/5000}{(3000/5000) \times (1250/5000)} = 1.33$$

Other Pattern Evaluation Methods

- χ^2 measure, all_confidence measure, max_confidence measure, Kulczynski measure, …

Reading

- “Data Mining: Concepts and Techniques, 3rd Edition,” Chapter 6.