Transaction data analysis and association rules
www.mimuw.edu.pl/~son/datamining

Nguyen Hung Son

This presentation was prepared on the basis of the following public materials:

1. Jiawei Han and Micheline Kamber, „Data mining, concept and techniques" http://www.cs.sfu.ca
2. Gregory Piatetsky-Shapiro, „kdnuggest", http://www.kdnuggets.com/data_mining_course/

Lecture plan

- Association rules
- Algorithm Apriori
- Algorithm Apriori-Tid
- FP-tree

What Is Association Mining?

- Association rule mining:
  - Finding frequent patterns, associations, correlations, or causal structures among sets of items or objects in transaction databases, relational databases, and other information repositories.
- Applications:
  - Basket data analysis, cross-marketing, catalog design, loss-leader analysis, clustering, classification, etc.
- Examples.

Rule form: "Body => Head [support, confidence]".

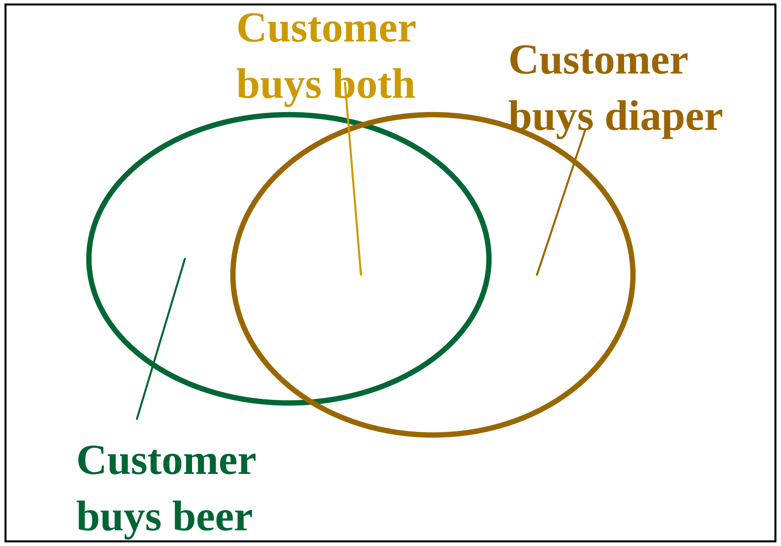
buys(x, "diapers") => buys(x, "beers") [0.5%, 60%]

major(x, "CS") ^ takes(x, "DB") => grade(x, "A") [1%, 75%]

Association Rule: Basic Concepts

- Given: (1) database of transactions, (2) each transaction is a list of items (purchased by a customer in a visit)
- Find: all rules that correlate the presence of one set of items with that of another set of items
  - E.g., *98% of people who purchase tires and auto accessories also get automotive services done*
- Applications
  - $* \Rightarrow$ *Maintenance Agreement* (What the store should do to boost Maintenance Agreement sales)
  - *Home Electronics* $\Rightarrow *$ (What other products should the store stocks up?)
  - Attached mailing in direct marketing
  - Detecting "ping-pong"ing of patients, faulty "collisions"

Rule Measures: Support and Confidence



- Find all the rules $X \& Y \Rightarrow Z$ with minimum confidence and support
  - support, *s*, probability that a transaction contains {X 🗐 Y 🗐 Z}
  - confidence, *c*, conditional probability that a transaction having {X 🗐 Y} also contains *Z*

Transaction ID	Items Bought
2000	A,B,C
1000	A,C
4000	A,D
5000	B,E,F

Let minimum support 50%, and minimum confidence 50%, we have

- $A \Rightarrow C$ (50%, 66.6%)
- $C \Rightarrow A$ (50%, 100%)

Association Rule Mining: A Road Map

- Boolean vs. quantitative associations (Based on the types of values handled)
  - buys(x, “SQLServer”) ^ buys(x, “DMBook”) => buys(x, “DBMiner”) [0.2%, 60%]
  - age(x, “30..39”) ^ income(x, “42..48K”) => buys(x, “PC”) [1%, 75%]
- Single dimension vs. multiple dimensional associations (see ex. above)
- Single level vs. multiple-level analysis
  - What brands of beers are associated with what brands of diapers?
- Various extensions
  - Correlation, causality analysis
    - Association does not necessarily imply correlation or causality
  - Maxpatterns and closed itemsets
  - Constraints enforced
    - E.g., small sales (sum < 100) trigger big buys (sum > 1,000)?

Lecture plan

- Association rules
- Algorithm Apriori
- Algorithm Apriori-Tid
- FP-tree

Mining Association Rules – An Example

Transaction ID	Items Bought
2000	A,B,C
1000	A,C
4000	A,D
5000	B,E,F

Min. support 50%
Min. confidence 50%

Frequent Itemset	Support
{A}	75%
{B}	50%
{C}	50%
{A,C}	50%

For rule $A \Rightarrow C$:

$\text{support} = \text{support}(\{A \cup C\}) = 50\%$

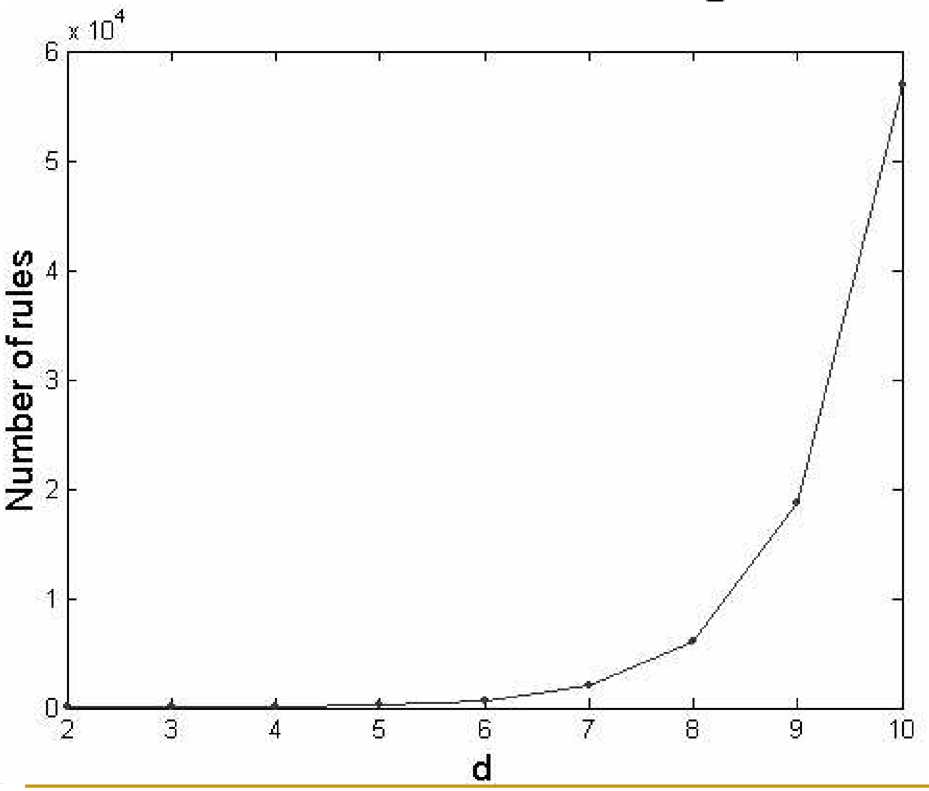
$\text{confidence} = \text{support}(\{A \cup C\})/\text{support}(\{A\}) = 66.6\%$

The Apriori principle:

Any subset of a frequent itemset must be frequent

Possible number of rules

- Given d unique items
- Total number of itemsets = 2^d
- Total number of possible association rules:



$$R = \sum_{k=1}^{d-1}\left[\binom{d}{k} \times \sum_{j=1}^{d-k}\binom{d-k}{j}\right]$$
$$= 3^d - 2^{d+1} + 1$$

If d=6, R = 602 rules

How to Mine Association Rules?

- Two step approach:
  1. Generate all frequent itemsets (sets of items whose support > *minsup*)
  2. Generate high confidence association rules from each frequent itemset
     - Each rule is a binary partition of a frequent itemset
- Frequent itemset generation is more expensive operation.
  (There are 2^d possible itemsets)

Mining Frequent Itemsets: the Key Step

- Find the *frequent itemsets*: the sets of items that have minimum support
  - A subset of a frequent itemset must also be a frequent itemset
    - i.e., if {*AB*} is a frequent itemset, both {*A*} and {*B*} should be a frequent itemset
  - Iteratively find frequent itemsets with cardinality from 1 to *k* (*k*-itemset)
- Use the frequent itemsets to generate association rules.

Reducing Number of Candidates

- Apriori principle:

– If an itemset is frequent, then all of its subsets must also be frequent

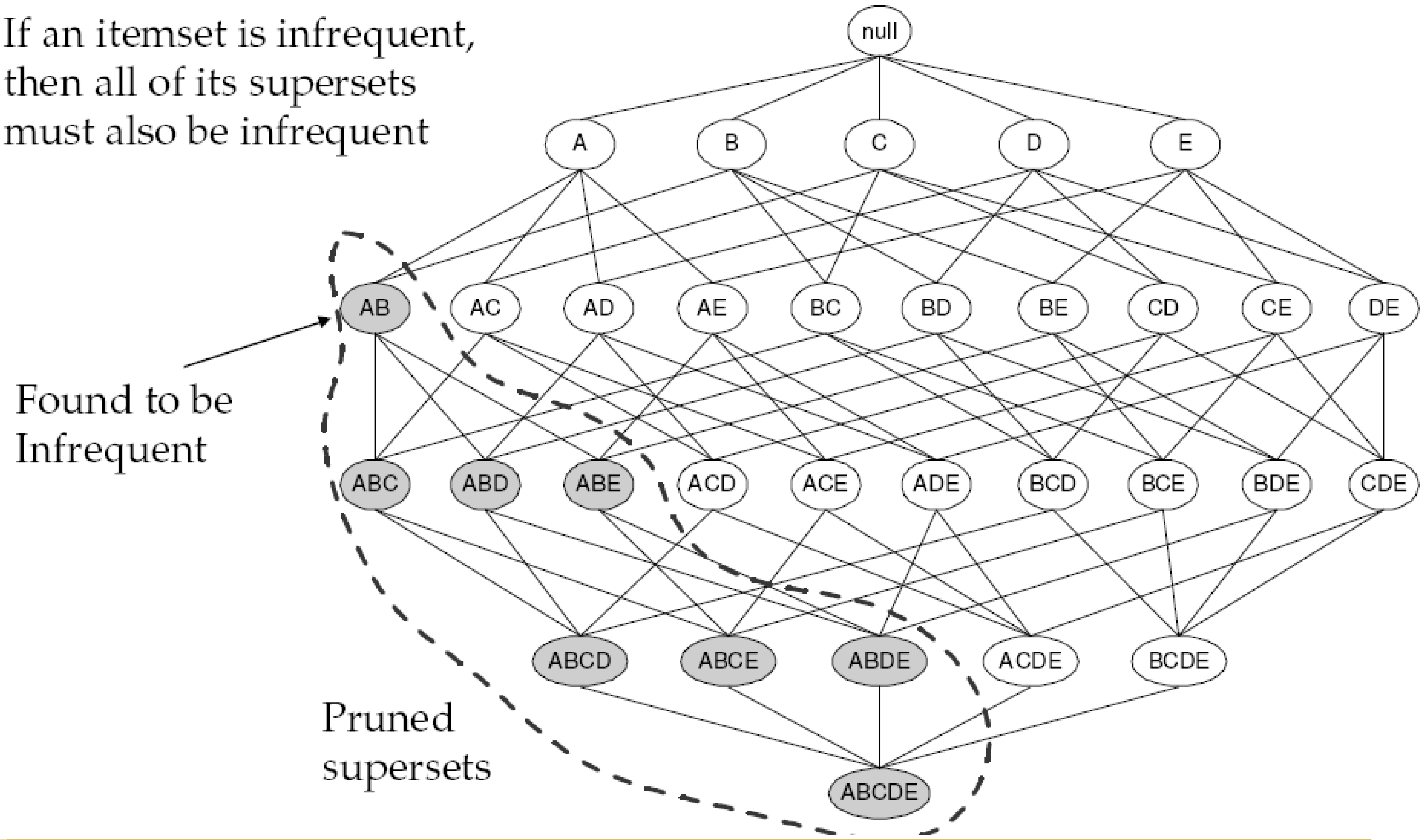
- Apriori principle holds due to the following property of the support measure:

$$\forall X, Y : (X \subseteq Y) \Rightarrow s(X) \geq s(Y)$$

  - Support of an itemset never exceeds the support of any of its subsets
  - This is known as the anti-monotone property of support

Key observation

If an itemset is infrequent, then all of its supersets must also be infrequent



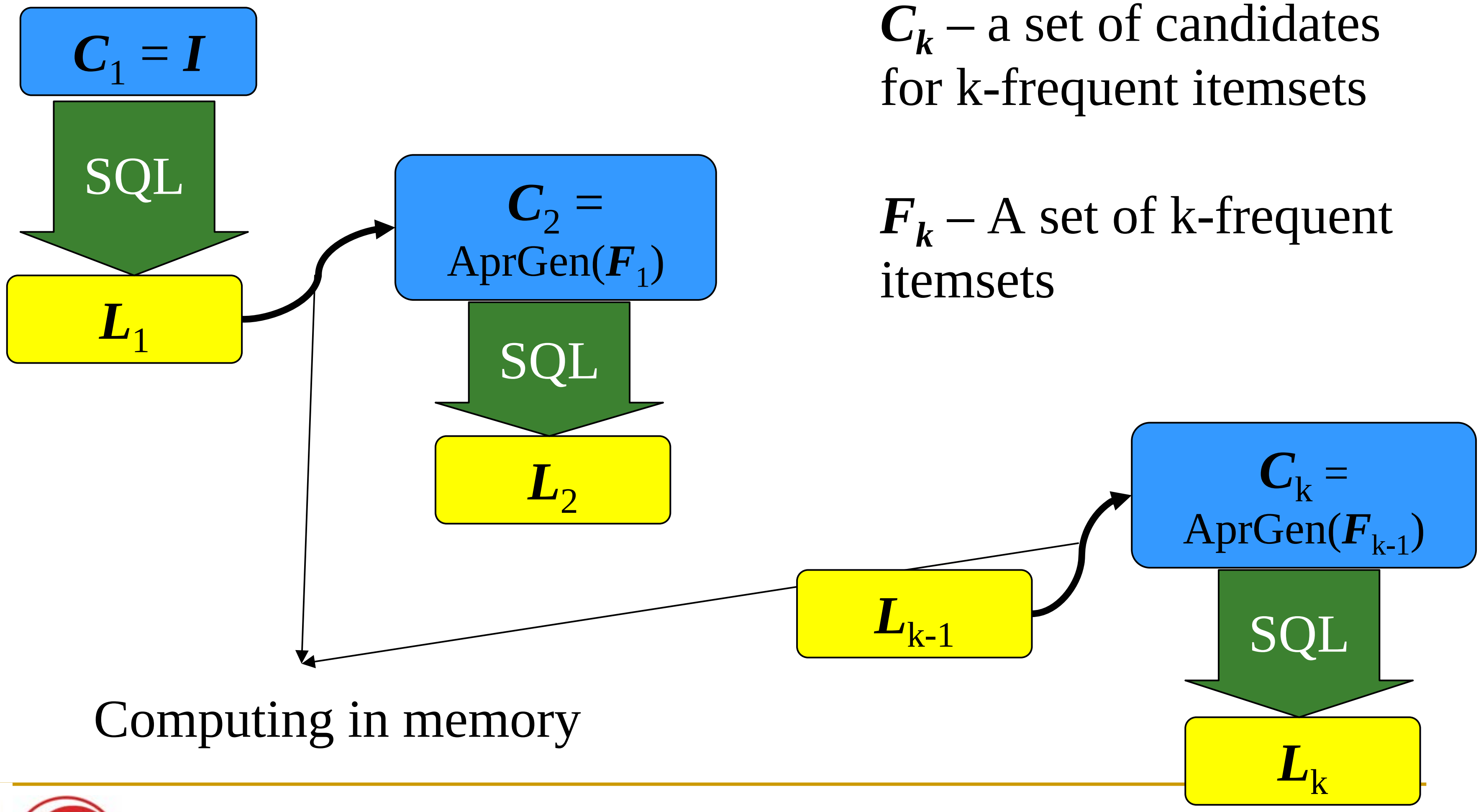
The Apriori Algorithm

- Join Step: C_k is generated by joining L_{k-1} with itself
- Prune Step: Any (k-1)-itemset that is not frequent cannot be a subset of a frequent k-itemset
- Pseudo-code:

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

$L_1$ = {frequent items};
for ($k$ = 1; $L_k$ !=∅; $k$++) 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 $\cup_k L_k$;
```

An idea of Apriori algorithm



C_k – a set of candidates for k-frequent itemsets

F_k – A set of k-frequent itemsets

Apriori Algorithm — Example

Database D

TID	Items
100	1 3 4
200	2 3 5
300	1 2 3 5
400	2 5

Scan D $\longrightarrow$

C_1

itemset	sup.
{1}	2
{2}	3
{3}	3
{4}	1
{5}	3

$\longrightarrow$

L_1

itemset	sup.
{1}	2
{2}	3
{3}	3
{5}	3

C_2

itemset
{1 2}
{1 3}
{1 5}
{2 3}
{2 5}
{3 5}

Scan D $\longleftarrow$

C_2

itemset	sup
{1 2}	1
{1 3}	2
{1 5}	1
{2 3}	2
{2 5}	3
{3 5}	2

$\longleftarrow$

L_2

itemset	sup
{1 3}	2
{2 3}	2
{2 5}	3
{3 5}	2

C_3

itemset
{2 3 5}

Scan D $\longrightarrow$

L_3

itemset	sup
{2 3 5}	2

How to Generate Candidates?

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

```
insert into $C_k$
select $p.item_1, p.item_2, \ldots, p.item_{k-1}, q.item_{k-1}$
from $L_{k-1}\ p, L_{k-1}\ q$
where $p.item_1=q.item_1, \ldots, p.item_{k-2}=q.item_{k-2}, p.item_{k-1} < q.item_{k-1}$
```

- 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 $C_k$
```

Example of Generating Candidates

- 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*}

- L_3={*abc, abd, abe acd, ace, bcd*}
- Self-joining: $L_3 * L_3$
  - *abcd* from *abc* and *abd*
  - *abce*
  - *abde*
  - *acde* from *acd* and *ace*

Illustration of candidate generation

Item	Count
Bread	4
Coke	2
Milk	4
Beer	3
Diaper	4
Eggs	1

Items (1-itemsets)

Itemset	Count
{Bread,Milk}	3
{Bread,Beer}	2
{Bread,Diaper}	3
{Milk,Beer}	2
{Milk,Diaper}	3
{Beer,Diaper}	3

Pairs (2-itemsets)

Minimum Support = 3

If every subset is considered,

$^6C_1 + {^6C_2} + {^6C_3} = 41$

With support-based pruning,

$6 + 6 + 2 = 14$

Triplets (3-itemsets

Itemset	Count
{Bread,Milk,Diaper}	3
{Milk,Diaper,Beer}	2

Rule generation

- Given a frequent itemset L, find all non-empty subsets f $\subseteq$ L such that f => L – f satisfies the minimum confidence requirement
- If {A,B,C,D} is a frequent itemset, candidate rules:

  ABC =>D, ABD =>C, ACD =>B, BCD =>A,

  A =>BCD, B =>ACD, C =>ABD, D =>ABC

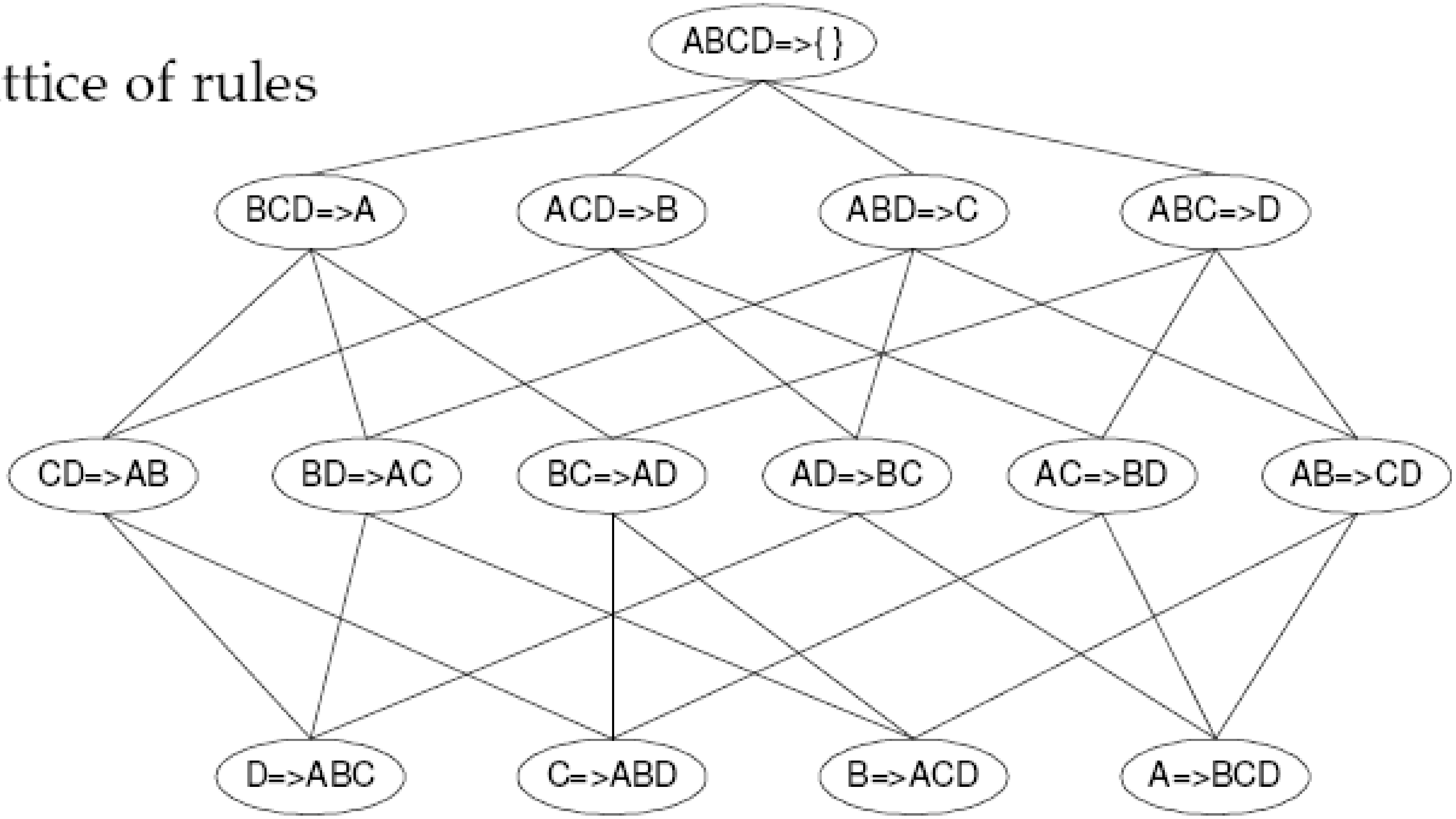
  AB =>CD, AC =>BD, AD =>BC, BC =>AD,

  BD =>AC, CD =>AB,
- If |L| = k, then there are $2^k – 2$ candidate association rules (ignoring L => $\emptyset$ and $\emptyset$ => L)

Rule generation

- How to efficiently generate rules from frequent itemsets?
  - In general, confidence does not have an antimonotone property
  - But confidence of rules generated from the same itemset has an anti-monotone property
  - L = {A,B,C,D}:
    c(ABC => D) ≥ c(AB => CD) ≥ c(A=>BCD)
- Confidence is non-increasing as number of items in rule consequent increases

Lattice of rules



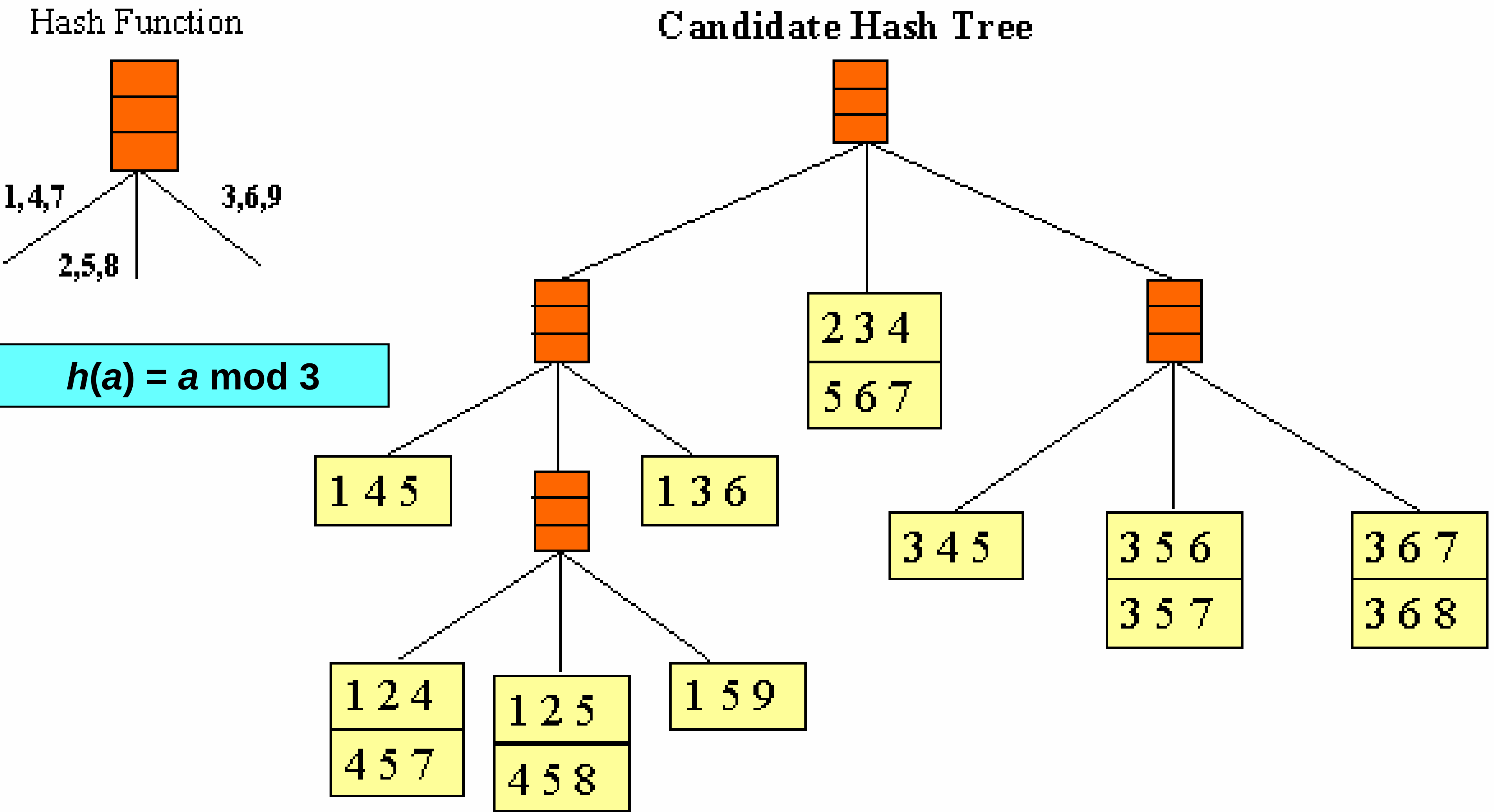
Apriori for rule generation

- Candidate rule is generated by merging two rules that share the same prefix in the rule consequent
    - join(CD=>AB, BD=>AC) would produce the candidate rule D => ABC
    - Prune rule D=>ABC if its subset AD=>BC does not have high confidence

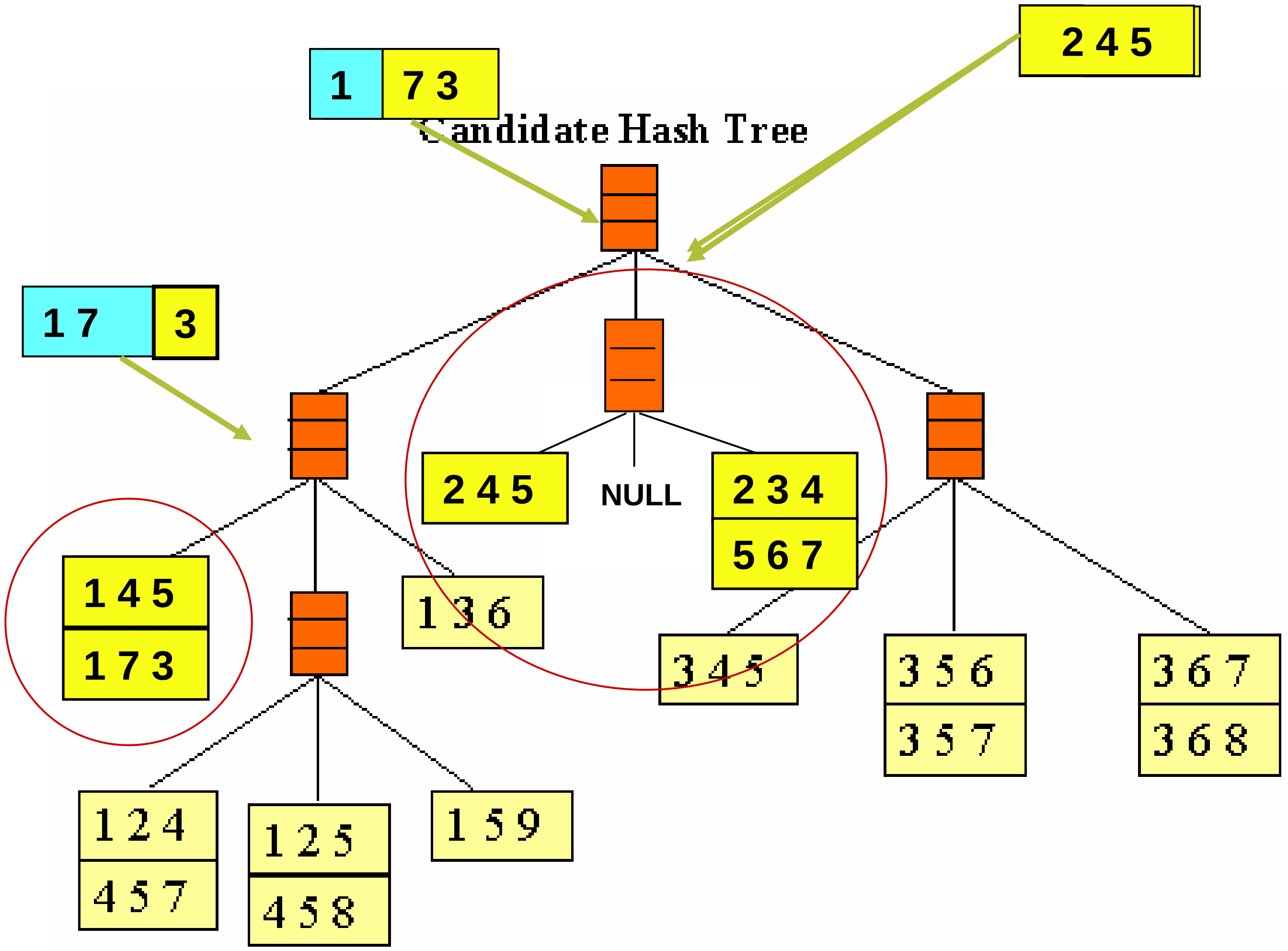
How to Count Supports of Candidates?

- 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

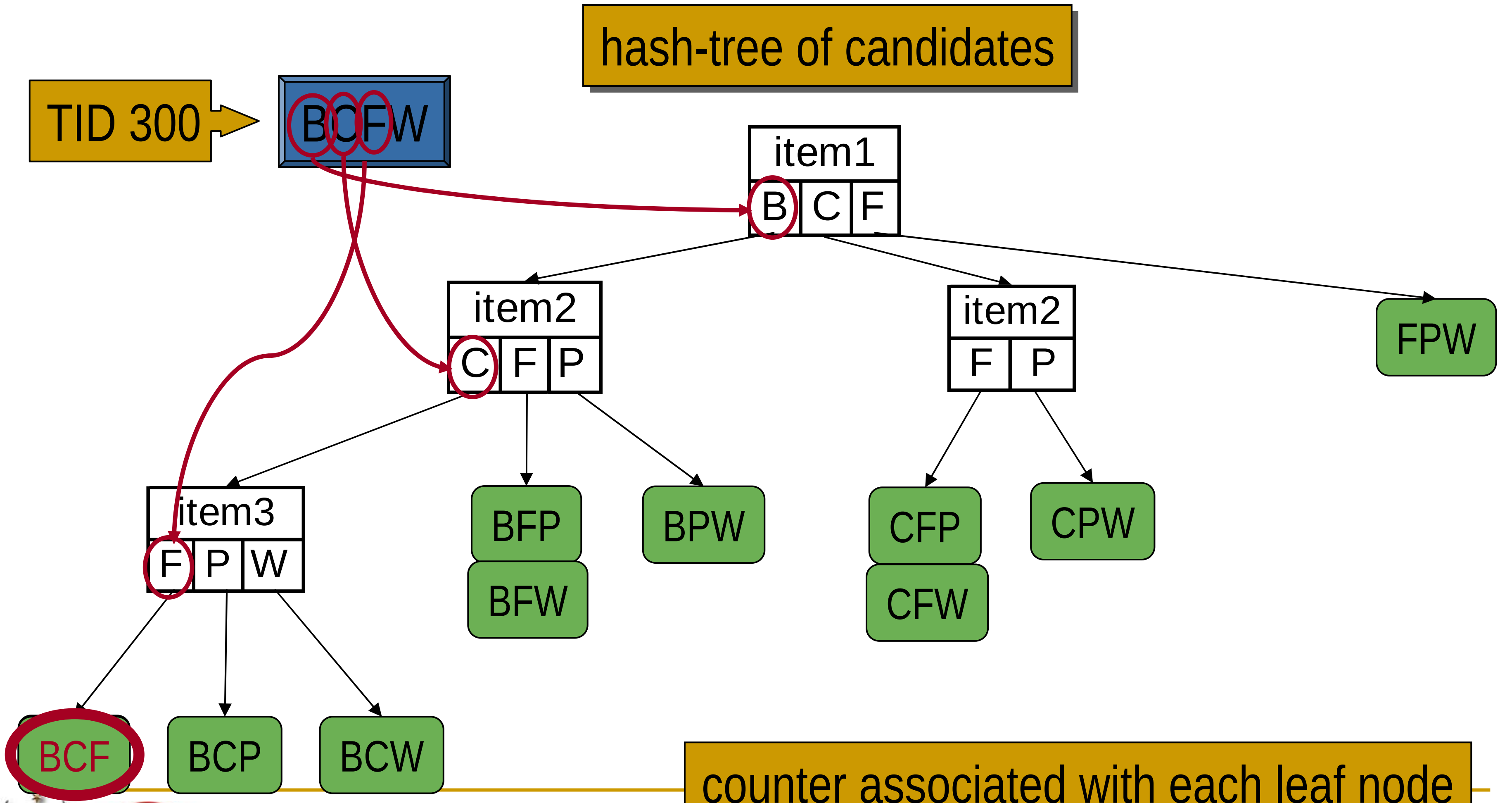
Hash tree


Hash Function
1,4,7
2,5,8
3,6,9
h(a) = a mod 3
Candidate Hash Tree
2 3 4
5 6 7
1 4 5
1 3 6
3 4 5
3 5 6
3 5 7
3 6 7
3 6 8
1 2 4
4 5 7
1 2 5
4 5 8
1 5 9

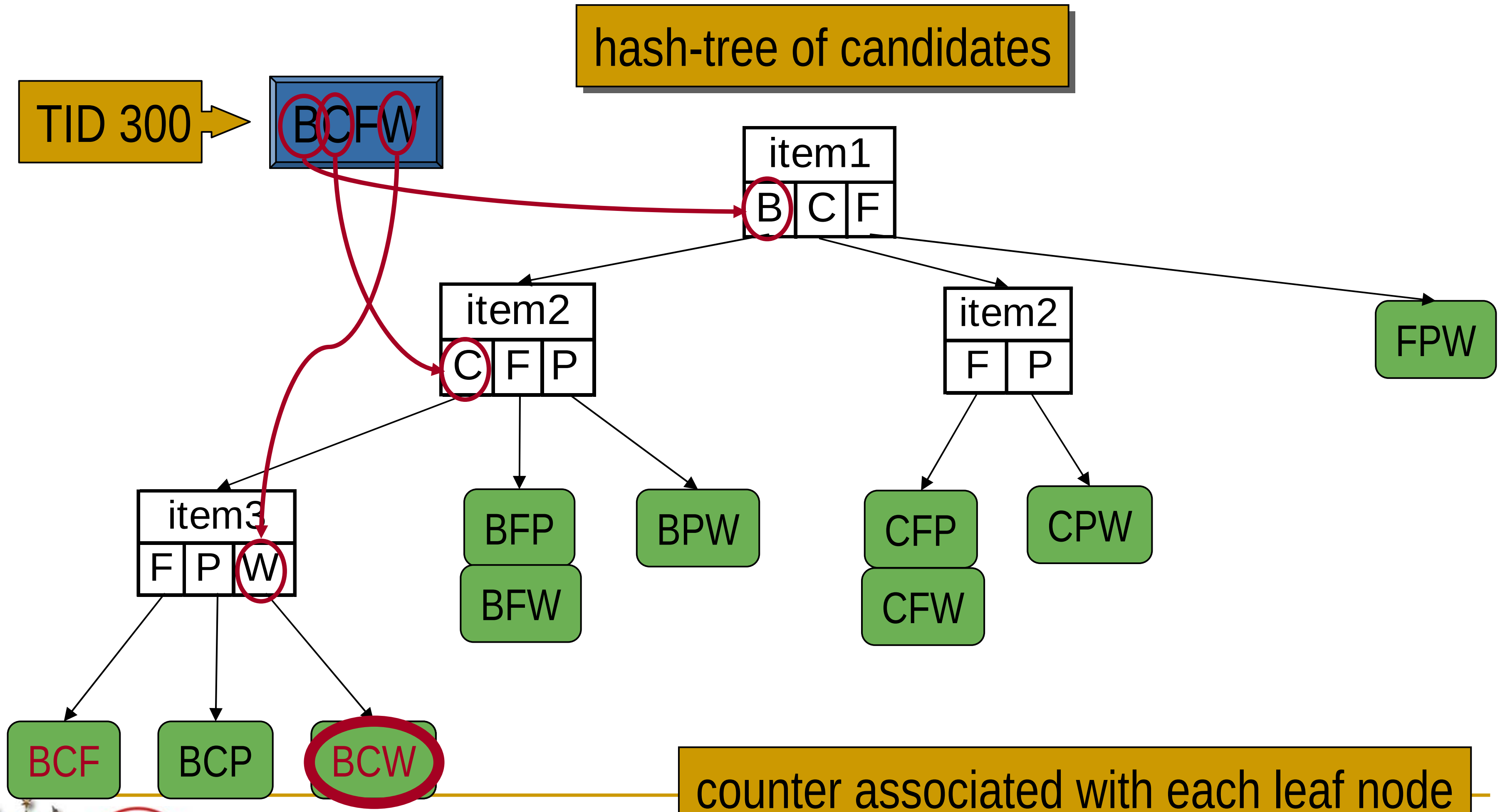
Insert a candidate to hash-tree



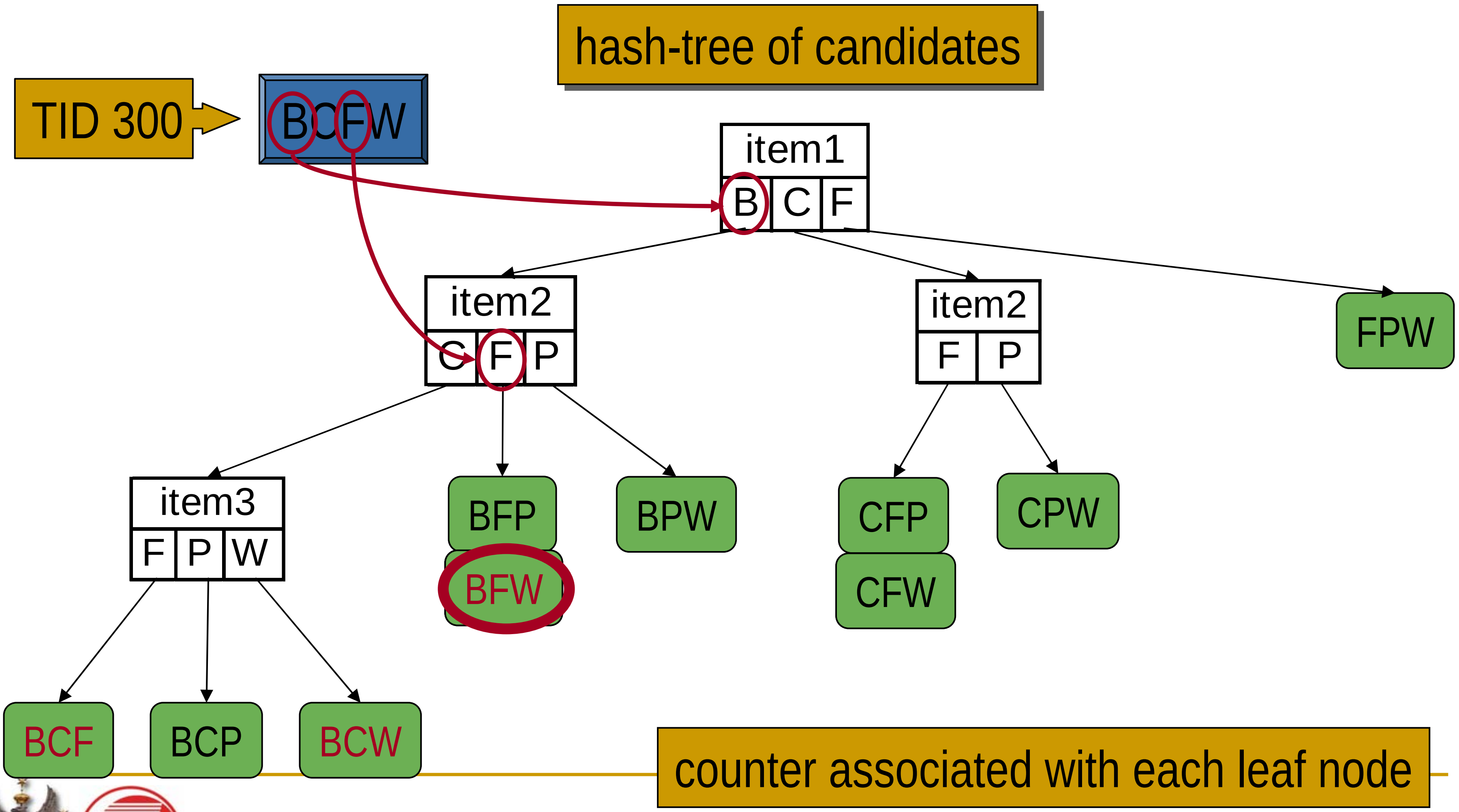
Apriori Candidate evaluation: Finding candidates contained in transaction



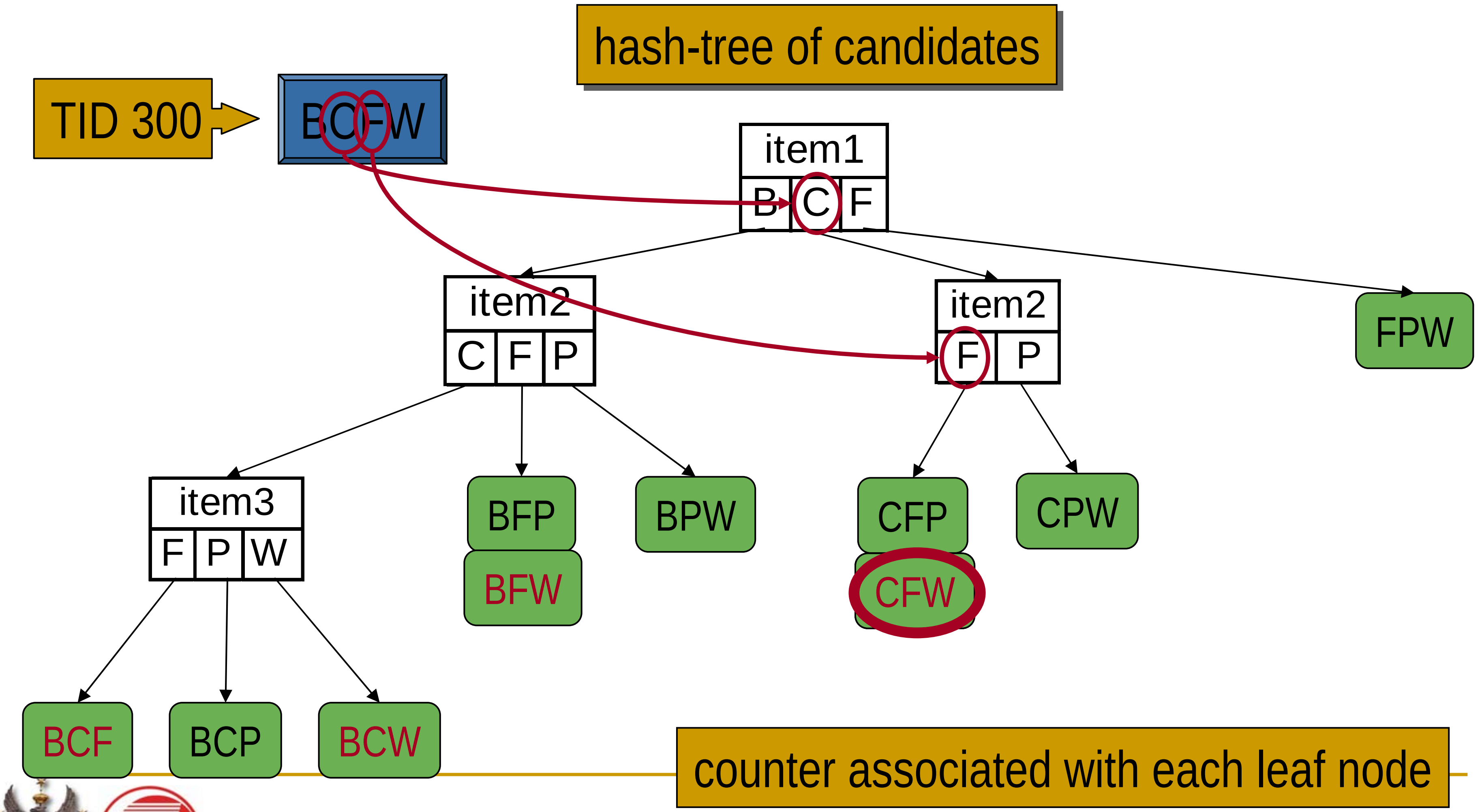
Apriori Candidate evaluation: Finding candidates contained in transaction



Apriori Candidate evaluation
Finding candidates contained in transaction



Apriori Candidate evaluation
Finding candidates contained in transaction



Lecture plan

- Association rules
- Algorithm Apriori
- Algorithm Apriori-Tid
- FP-tree

Observations

- Apriori algorithm scans the whole database to determine supports of candidates
- **Improvement:**
  - Using new data structure called *counting_base to* store only those transactions which can support the actual list of candidates
  - Algorithm AprioriTid

AprioriTid

Input: transaction data set **D**, *min_sup* – minimal support
Output: the set of all frequent itemset **F**
Variables: *CB_k- counting_base* at kth iteration of the algorithm

```
1: $F_1$ = {frequent 1-itemsets}
2: $k = 2$;
3: while ($F_{k-1}$ is not empty) do {
4:         $C_k$ = Apriori_generate ($F_{k-1}$);
           $CB_k$ = Counting_base_generate ($C_k$, $CB_{k-1}$)
           Support_count ($C_k$, $CB_k$);
5:         $F_k = \{c \in C_k \mid support(c) \geq min\_support\}$;
   }
6: F = sum of all $F_k$;
```

AprioriTid: *Counting_base_generate*

Step 1:

counting_base = {(r_i, S_i): r_i is the ID and S_i is the itemset of the ith transaction}

Step i:

counting_base = {(r, S_i): S_i is created as a joint of S_{i-1} with S_{i-1} as follows:

```
        IF $\{u_1\, u_2 \ldots u_{i-2}\, a\}$ and $\{u_1\, u_2 \ldots u_{i-2}\, b\} \in S_{i-1}$ THEN
                $\{u_1\, u_2 \ldots u_{i-2}\, a\, b\} \in S_i$
}
```

AprioriTid: Example

D = {(1,acd), (2, bce), (3,abce), (4,be)}.
min_sup = 0.5

$$\begin{tikzcd}[column sep=small]
\textbf{Step 1} & & \\
\textit{counting\_base} = \{(1,\{a,c,d\}), (2,\{b,c,e\}), (3,\{a,b,c,e\}), (4,\{b, e\})\} \arrow[dd] \arrow[ddr] & & F_1 = \{a, b, c, e\} \arrow[d] \arrow[ddll] \\
& & C_2 = \{ab, ac, ae, bc, be, ce\} \arrow[d] \\
\textbf{Step 2} \quad \textit{counting\_base} = \{(1,\{ac\}), (2,\{bc,be,ce\}), (3,\{ab,ac,ae,bc,be,ce\}), (4,\{be\})\} \arrow[dd] \arrow[ddr] & & F_2 = \{ac, bc, be, ce\} \arrow[d] \arrow[ddll] \\
& & C_3 = \{bce\} \arrow[d] \\
\textbf{Step 3} \quad \textit{counting\_base} = \{(2,\{bce\}),(3,\{bce\})\} & & F_3 = \{bce\}
\end{tikzcd}$$

Is Apriori Fast Enough? — Performance Bottlenecks

- The core of the Apriori algorithm:
  - Use frequent (k – 1)-itemsets to generate candidate frequent k-itemsets
  - Use database scan and pattern matching to collect counts for the candidate itemsets
- The bottleneck of *Apriori*: candidate generation
  - Huge candidate sets:
    - 10^4 frequent 1-itemset will generate 10^7 candidate 2-itemsets
    - To discover a frequent pattern of size 100, e.g., $\{a_1, a_2, \ldots, a_{100}\}$, one needs to generate $2^{100} \approx 10^{30}$ candidates.
  - Multiple scans of database:
    - Needs (n + 1) scans, n is the length of the longest pattern

Algorithm AprioriHybrid

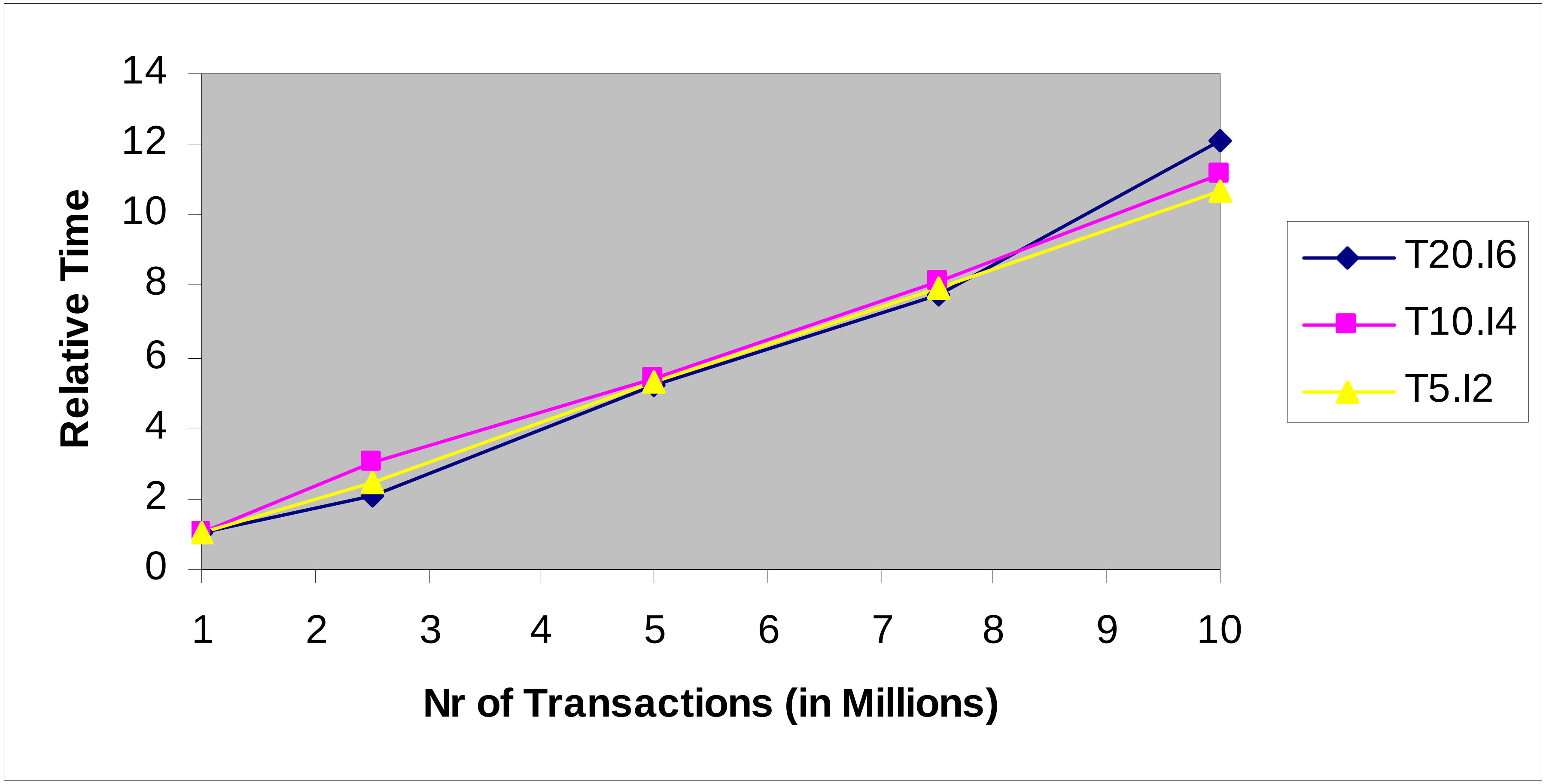
- AprioriTid replaces pass over data by pass over TC_k
  - effective when TC_k becomes small compared to size of database
- AprioriTid beats Apriori
  - when TC_k sets fit in memory
  - distribution of large itemsets has long tail
- Hybrid algorithm AprioriHybrid
  - use Apriori in initial passes
  - switch to AprioriTid when TC_k expected to fit in memory

Algorithm AprioriHybrid

- Heuristic used for switching
  - estimate size of TC_k from C_k
    - size(TC_k) = Σ $_{\text{candidates } c \in C_k}$ support(c) + number of transactions
  - if TC_k fits in memory and nr of candidates decreasing then switch to AprioriTid
- AprioriHybrid outperforms Apriori and AprioriTid in almost all cases
  - little worse if switch pass is last one
    - cost of switching without benefits
  - AprioriHybrid up to 30% better than Apriori, up to 60% better than AprioriTid

AprioriHybrid Scale-up Experiment

name	|MB|
T5.I2.D10M	239
T10.I4.D10M	439
T20.I6.D10M	838



Lecture plan

- Association rules
- Algorithm Apriori
- Algorithm Apriori-Tid
- FP-tree

Mining Frequent Patterns Without Candidate Generation

- Compress a large database into a compact, Frequent-Pattern tree (FP-tree) structure
  - highly condensed, but complete for frequent pattern mining
  - avoid costly database scans
- Develop an efficient, FP-tree-based frequent pattern mining method
  - A divide-and-conquer methodology: decompose mining tasks into smaller ones
  - Avoid candidate generation: sub-database test only!

Construct FP-tree from a Transaction DB

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}	{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}

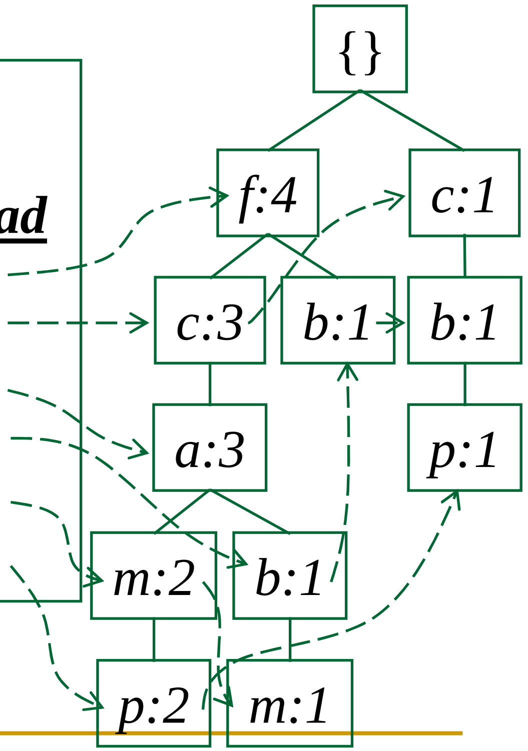
min_support = 0.5

Steps:

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

Header Table

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



Benefits of the FP-tree Structure

- Completeness:
  - never breaks a long pattern of any transaction
  - preserves complete information for frequent pattern mining
- Compactness
  - reduce irrelevant information—infrequent items are gone
  - frequency descending ordering: more frequent items are more likely to be shared
  - never be larger than the original database (if not count node-links and counts)
  - Example: For Connect-4 DB, compression ratio could be over 100

Mining Frequent Patterns Using FP-tree

- General idea (divide-and-conquer)
  - Recursively grow frequent pattern path using the FP-tree
- Method
  - For each 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)

Major Steps to Mine FP-tree

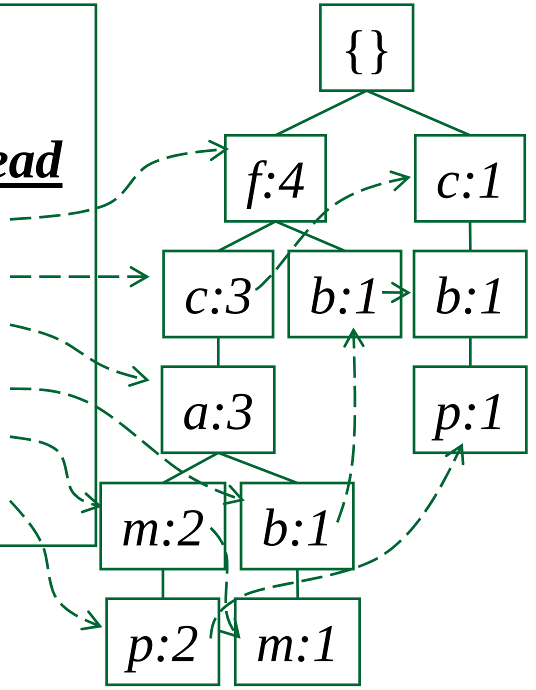
1) Construct conditional pattern base for each node in the FP-tree
2) Construct conditional FP-tree from each conditional pattern-base
3) Recursively mine conditional FP-trees and grow frequent patterns obtained so far
    - If the conditional FP-tree contains a single path, simply enumerate all the patterns

Step 1: From FP-tree to Conditional Pattern Base

- Starting at the frequent header table in the FP-tree
- Traverse the FP-tree by following the link of each frequent item
- Accumulate all of transformed prefix paths of that item to form a 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***

Properties of FP-tree for Conditional Pattern Base Construction

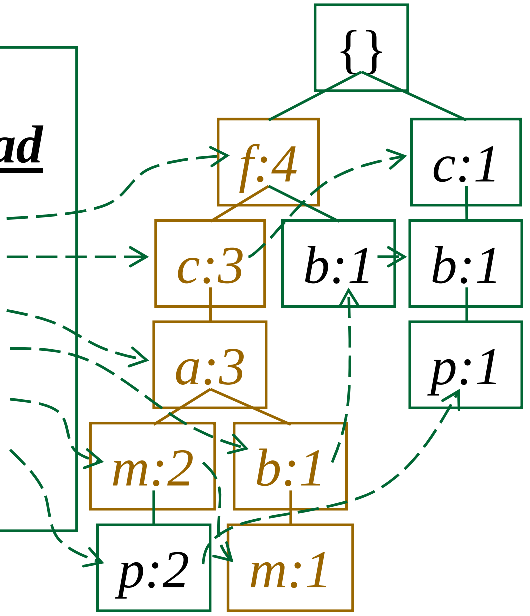
- Node-link property
  - For any frequent item a_i, all the possible frequent patterns that contain a_i can be obtained by following a_i's node-links, starting from a_i's head in the FP-tree header
- Prefix path property
  - To calculate the frequent patterns for a node a_i in a path P, only the prefix sub-path of a_i in P need to be accumulated, and its frequency count should carry the same count as node a_i.

Step 2: Construct Conditional FP-tree

- 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

Header Table

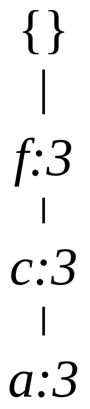
Item	***frequency***	***head***
f	*4*	
c	*4*	
a	*3*	
b	*3*	
m	*3*	
p	*3*	



m-conditional **pattern base:**

fca:2, fcab:1

➔



m-conditional **FP-tree**

➔

All frequent patterns concerning *m*

m,

fm, cm, am,

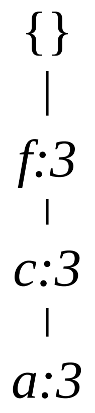
fcm, fam, cam,

fcam

Mining Frequent Patterns by Creating Conditional Pattern-Bases

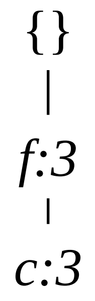
Item	Conditional pattern-base	Conditional FP-tree
p	{(fcam:2), (cb:1)}	{(c:3)}|p
m	{(fca:2), (fcab:1)}	{(f:3, c:3, a:3)}|m
b	{(fca:1), (f:1), (c:1)}	Empty
a	{(fc:3)}	{(f:3, c:3)}|a
c	{(f:3)}	{(f:3)}|c
f	Empty	Empty

Step 3: Recursively mine the conditional FP-tree



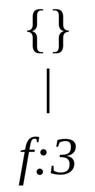
m-conditional **FP-tree**

Cond. pattern base of “am”: (fc:3)



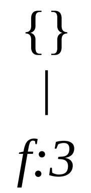
am-conditional **FP-tree**

Cond. pattern base of “cm”: (f:3)



cm-conditional **FP-tree**

Cond. pattern base of “cam”: (f:3)



cam-conditional **FP-tree**

Single FP-tree Path Generation

- Suppose an FP-tree T has a single path P
- The complete set of frequent pattern of T can be generated by enumeration of all the combinations of the sub-paths of P

$$\begin{tikzcd}
\{\} \arrow[d, dash] \\
f{:}3 \arrow[d, dash] \\
c{:}3 \arrow[d, dash] \\
a{:}3
\end{tikzcd}$$

$\rightarrow$

All frequent patterns concerning *m*

***m*,**

***fm*, *cm*, *am*,**

***fcm*, *fam*, *cam*,**

fcam

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

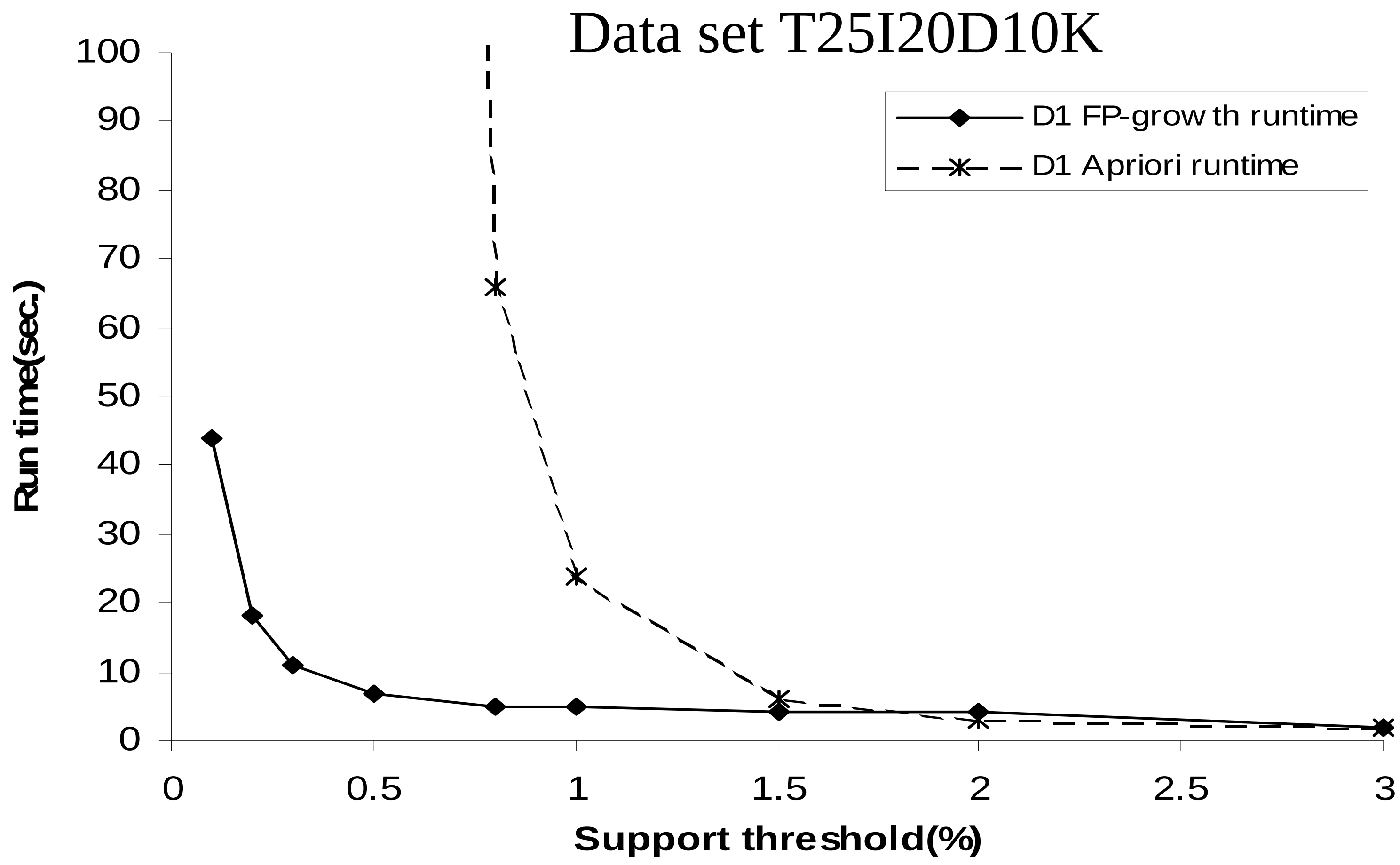
Principles of Frequent Pattern Growth

- Pattern growth property
  - Let α be a frequent itemset in DB, B be α's conditional pattern base, and β be an itemset in B. Then $\alpha \cup \beta$ is a frequent itemset in DB iff β is frequent in B.
- "*abcdef*" is a frequent pattern, if and only if
  - "*abcde*" is a frequent pattern, and
  - "*f*" is frequent in the set of transactions containing "*abcde*"

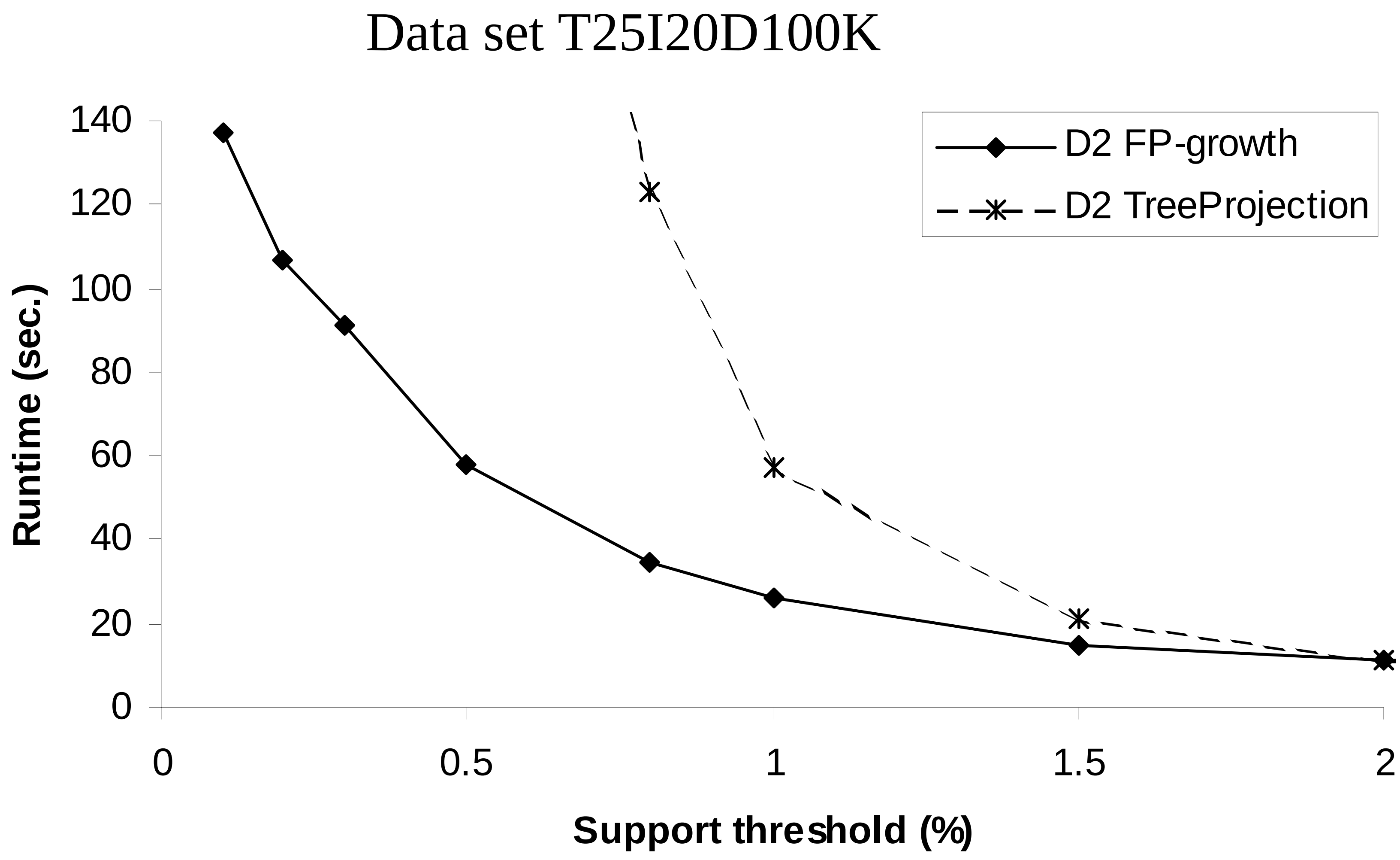
Why Is Frequent Pattern Growth Fast?

- Our performance study shows
  - FP-growth is an order of magnitude faster than Apriori, and is also faster than tree-projection
- Reasoning
  - No candidate generation, no candidate test
  - Use compact data structure
  - Eliminate repeated database scan
  - Basic operation is counting and FP-tree building

FP-growth vs. Apriori: Scalability With the Support Threshold



FP-growth vs. Tree-Projection: Scalability with Support Threshold



Some issues on association mining

- Interestingness measures
- Pattern visualization
- Multi-level association rules
- Discretization
- Mining association rules with constrains

Interestingness Measurements

- Objective measures

  Two popular measurements:

  ☆ *support;* and

  ◷ *confidence*

- Subjective measures (Silberschatz & Tuzhilin, KDD95)

  A rule (pattern) is interesting if

  ☆ it is *unexpected* (surprising to the user); and/or

  ◷ *actionable* (the user can do something with it)

Criticism to Support and Confidence

- Example 1: (Aggarwal & Yu, PODS98)
  - Among 5000 students
    - 3000 play basketball
    - 3750 eat cereal
    - 2000 both play basket ball and eat cereal
  - *play basketball* $\Rightarrow$ *eat cereal* [40%, 66.7%] is misleading because the overall percentage of students eating cereal is 75% which is higher than 66.7%.
  - *play basketball* $\Rightarrow$ *not eat cereal* [20%, 33.3%] is far more accurate, although with lower support and confidence

	basketball	not basketball	sum(row)
cereal	2000	1750	3750
not cereal	1000	250	1250
sum(col.)	3000	2000	5000

Criticism to Support and Confidence (Cont.)

- Example 2:
  - X and Y: positively correlated,
  - X and Z, negatively related
  - support and confidence of X=>Z dominates

X	1	1	1	1	0	0	0	0
Y	1	1	0	0	0	0	0	0
Z	0	1	1	1	1	1	1	1

- We need a measure of dependent or correlated events

$$corr_{A,B} = \frac{P(A \cup B)}{P(A)P(B)}$$

Rule	Support	Confidence
X=>Y	25%	50%
X=>Z	37.50%	75%

- P(B|A)/P(B) is also called the lift of rule A => B

Other Interestingness Measures: Interest

- Interest (correlation, lift) $$\frac{P(A \wedge B)}{P(A)P(B)}$$
  - taking both P(A) and P(B) in consideration
  - P(A^B)=P(B)*P(A), if A and B are independent events
  - A and B negatively correlated, if the value is less than 1; otherwise A and B positively correlated

X	1	1	1	1	0	0	0	0
Y	1	1	0	0	0	0	0	0
Z	0	1	1	1	1	1	1	1

Itemset	Support	Interest
X,Y	25%	2
X,Z	37.50%	0.9
Y,Z	12.50%	0.57

References

- R. Agarwal, C. Aggarwal, and V. V. V. Prasad. A tree projection algorithm for generation of frequent itemsets. In Journal of Parallel and Distributed Computing (Special Issue on High Performance Data Mining), 2000.
- R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. SIGMOD'93, 207-216, Washington, D.C.
- R. Agrawal and R. Srikant. Fast algorithms for mining association rules. VLDB'94 487-499, Santiago, Chile.
- R. Agrawal and R. Srikant. Mining sequential patterns. ICDE'95, 3-14, Taipei, Taiwan.
- R. J. Bayardo. Efficiently mining long patterns from databases. SIGMOD'98, 85-93, Seattle, Washington.
- S. Brin, R. Motwani, and C. Silverstein. Beyond market basket: Generalizing association rules to correlations. SIGMOD'97, 265-276, Tucson, Arizona.
- S. Brin, R. Motwani, J. D. Ullman, and S. Tsur. Dynamic itemset counting and implication rules for market basket analysis. SIGMOD'97, 255-264, Tucson, Arizona, May 1997.
- K. Beyer and R. Ramakrishnan. Bottom-up computation of sparse and iceberg cubes. SIGMOD'99, 359-370, Philadelphia, PA, June 1999.
- D.W. Cheung, J. Han, V. Ng, and C.Y. Wong. Maintenance of discovered association rules in large databases: An incremental updating technique. ICDE'96, 106-114, New Orleans, LA.
- M. Fang, N. Shivakumar, H. Garcia-Molina, R. Motwani, and J. D. Ullman. Computing iceberg queries efficiently. VLDB'98, 299-310, New York, NY, Aug. 1998.

References (2)

- G. Grahne, L. Lakshmanan, and X. Wang. Efficient mining of constrained correlated sets. ICDE'00, 512-521, San Diego, CA, Feb. 2000.
- Y. Fu and J. Han. Meta-rule-guided mining of association rules in relational databases. KDOOD'95, 39-46, Singapore, Dec. 1995.
- T. Fukuda, Y. Morimoto, S. Morishita, and T. Tokuyama. Data mining using two-dimensional optimized association rules: Scheme, algorithms, and visualization. SIGMOD'96, 13-23, Montreal, Canada.
- E.-H. Han, G. Karypis, and V. Kumar. Scalable parallel data mining for association rules. SIGMOD'97, 277-288, Tucson, Arizona.
- J. Han, G. Dong, and Y. Yin. Efficient mining of partial periodic patterns in time series database. ICDE'99, Sydney, Australia.
- J. Han and Y. Fu. Discovery of multiple-level association rules from large databases. VLDB'95, 420-431, Zurich, Switzerland.
- J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. SIGMOD'00, 1-12, Dallas, TX, May 2000.
- T. Imielinski and H. Mannila. A database perspective on knowledge discovery. Communications of ACM, 39:58-64, 1996.
- M. Kamber, J. Han, and J. Y. Chiang. Metarule-guided mining of multi-dimensional association rules using data cubes. KDD'97, 207-210, Newport Beach, California.
- M. Klemettinen, H. Mannila, P. Ronkainen, H. Toivonen, and A.I. Verkamo. Finding interesting rules from large sets of discovered association rules. CIKM'94, 401-408, Gaithersburg, Maryland.

References (3)

- F. Korn, A. Labrinidis, Y. Kotidis, and C. Faloutsos. Ratio rules: A new paradigm for fast, quantifiable data mining. VLDB'98, 582-593, New York, NY.
- B. Lent, A. Swami, and J. Widom. Clustering association rules. ICDE'97, 220-231, Birmingham, England.
- H. Lu, J. Han, and L. Feng. Stock movement and n-dimensional inter-transaction association rules. SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery (DMKD'98), 12:1-12:7, Seattle, Washington.
- H. Mannila, H. Toivonen, and A. I. Verkamo. Efficient algorithms for discovering association rules. KDD'94, 181-192, Seattle, WA, July 1994.
- H. Mannila, H Toivonen, and A. I. Verkamo. Discovery of frequent episodes in event sequences. Data Mining and Knowledge Discovery, 1:259-289, 1997.
- R. Meo, G. Psaila, and S. Ceri. A new SQL-like operator for mining association rules. VLDB'96, 122-133, Bombay, India.
- R.J. Miller and Y. Yang. Association rules over interval data. SIGMOD'97, 452-461, Tucson, Arizona.
- R. Ng, L. V. S. Lakshmanan, J. Han, and A. Pang. Exploratory mining and pruning optimizations of constrained associations rules. SIGMOD'98, 13-24, Seattle, Washington.
- N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Discovering frequent closed itemsets for association rules. ICDT'99, 398-416, Jerusalem, Israel, Jan. 1999.

References (4)

- J.S. Park, M.S. Chen, and P.S. Yu. An effective hash-based algorithm for mining association rules. SIGMOD'95, 175-186, San Jose, CA, May 1995.
- J. Pei, J. Han, and R. Mao. CLOSET: An Efficient Algorithm for Mining Frequent Closed Itemsets. DMKD'00, Dallas, TX, 11-20, May 2000.
- J. Pei and J. Han. Can We Push More Constraints into Frequent Pattern Mining? KDD'00. Boston, MA. Aug. 2000.
- G. Piatetsky-Shapiro. Discovery, analysis, and presentation of strong rules. In G. Piatetsky-Shapiro and W. J. Frawley, editors, Knowledge Discovery in Databases, 229-238. AAAI/MIT Press, 1991.
- B. Ozden, S. Ramaswamy, and A. Silberschatz. Cyclic association rules. ICDE'98, 412-421, Orlando, FL.
- J.S. Park, M.S. Chen, and P.S. Yu. An effective hash-based algorithm for mining association rules. SIGMOD'95, 175-186, San Jose, CA.
- S. Ramaswamy, S. Mahajan, and A. Silberschatz. On the discovery of interesting patterns in association rules. VLDB'98, 368-379, New York, NY..
- S. Sarawagi, S. Thomas, and R. Agrawal. Integrating association rule mining with relational database systems: Alternatives and implications. SIGMOD'98, 343-354, Seattle, WA.
- A. Savasere, E. Omiecinski, and S. Navathe. An efficient algorithm for mining association rules in large databases. VLDB'95, 432-443, Zurich, Switzerland.
- A. Savasere, E. Omiecinski, and S. Navathe. Mining for strong negative associations in a large database of customer transactions. ICDE'98, 494-502, Orlando, FL, Feb. 1998.

References (5)

- C. Silverstein, S. Brin, R. Motwani, and J. Ullman. Scalable techniques for mining causal structures. VLDB'98, 594-605, New York, NY.
- R. Srikant and R. Agrawal. Mining generalized association rules. VLDB'95, 407-419, Zurich, Switzerland, Sept. 1995.
- R. Srikant and R. Agrawal. Mining quantitative association rules in large relational tables. SIGMOD'96, 1-12, Montreal, Canada.
- R. Srikant, Q. Vu, and R. Agrawal. Mining association rules with item constraints. KDD'97, 67-73, Newport Beach, California.
- H. Toivonen. Sampling large databases for association rules. VLDB'96, 134-145, Bombay, India, Sept. 1996.
- D. Tsur, J. D. Ullman, S. Abitboul, C. Clifton, R. Motwani, and S. Nestorov. Query flocks: A generalization of association-rule mining. SIGMOD'98, 1-12, Seattle, Washington.
- K. Yoda, T. Fukuda, Y. Morimoto, S. Morishita, and T. Tokuyama. Computing optimized rectilinear regions for association rules. KDD'97, 96-103, Newport Beach, CA, Aug. 1997.
- M. J. Zaki, S. Parthasarathy, M. Ogihara, and W. Li. Parallel algorithm for discovery of association rules. Data Mining and Knowledge Discovery, 1:343-374, 1997.
- M. Zaki. Generating Non-Redundant Association Rules. KDD'00. Boston, MA. Aug. 2000.
- O. R. Zaiane, J. Han, and H. Zhu. Mining Recurrent Items in Multimedia with Progressive Resolution Refinement. ICDE'00, 461-470, San Diego, CA, Feb. 2000.