EECS-4412: *Data Mining*

Frequent Pattern & Association Rule Mining

Parke Godfrey

(Thanks to *Aijun An* & *Jiawei Han*)

Outline

- Basic concepts of association rule learning
- Apriori algorithm
- FP-Growth Algorithm
- Finding interesting rules.

Why Mining Association Rules?

- Objective:
  - Finding interesting co-occurring items (or objects, events) in a given data set.

- Examples:
  - Given a database of transactions, each transaction is a list of items (purchased by a customer in a visit), you may find:

    computer → financial_management_software [support=2%, confidence=60%]

  - From a student database, you may find
    - major(x, “CS”) ^ gpa(x, “A”) → has_taken(x, “DB”) [1%, 75%]

Why Mining Association Rules? (Cont'd)

- Popular application: *Basket data analysis*

  - place items frequently bought together close to each other to increase sales of these items.

  - *?* → *iPad* (What the store should do to boost sales of the particular product, i.e., *iPad*)

  - *iPad* → *?* (What other products should the store stock up?)

What Kind of Databases?

Transactional database TDB

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

- Itemset: a set of items
- A *transaction* is a tuple (tid, X)
  - Transaction ID tid
  - Itemset X
- A ***transactional database*** is a set of transactions
  - In many cases, a transaction database can be treated as a set of itemsets (ignore TIDs)

- Association rule from TDB (relates two itemsets):
  - {a, c, m} → {l} [support=40%, confidence=66.7%]

What Kind of Databases? (*Contd*)

Relational database (RDB)

Day	Outlook	Temp	Humid	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

- An attribute-value pair
  - E.g., Outlook = sunny
- Record: a set of attribute-value pairs
- Relational DB: a set of records

- Association rule from RDB (relates two sets of attribute-value pairs):
  (Outlook=sunny)$\wedge$(Temp=hot) $\rightarrow$ (PlayTennis=No)
  [support=14%, confidence=100%]

Definition of Association Rule

- An association rule is of the form:

$$\underset{\text{antecedent}}{X} \rightarrow \underset{\text{consequent}}{Y}\ [\text{support, confidence}]$$

where

  - $X \subset I$, $Y \subset I$, $X \cap Y = \emptyset$ and I is a set of items (objects or attribute-value pairs).

  - support: probability that a transaction (or a record) contains X and Y, i.e.,

$$\text{support}\,(X \rightarrow Y) = P(X \cup Y)$$

  - confidence: conditional probability that a transaction (or a record) having X also contains Y, i.e.,

$$\text{confidence}(X \rightarrow Y) = P(Y|X)$$

- A rule associates one set of items (or attribute-value pairs) with another set of items (or attribute-value pairs)

Support and Confidence: Example

$$\text{support}(X \rightarrow Y) = P(X \cup Y)$$

$$\text{confidence}(X \rightarrow Y) = P(Y|X)$$

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

Relative frequency is used to estimate the probability.

- $\{A\} \rightarrow \{C\}$ (50%, 66.7%)
- $\{C\} \rightarrow \{A\}$ (50%, 100%)
- $\{A, C\} \rightarrow \{B\}$ (25%, 50%)
- $\{A, B\} \rightarrow \{E\}$ (0%, 0%)

Mining Association Rules

- Problem statement

  Given a *minimum support* (*min_sup*), also called *support threshold*, and a *minimum confidence* (*min_conf*), also called *confidence threshold*, find all association rules that satisfy both *min_sup* and *min_conf* from a data set *D*.

Basic Concepts

- Strong rules:

  An association rule is *strong* if it satisfies both *min_sup* and *min_conf*.

- *k*-itemset: an itemset that contains *k* items.
  - {computer, financial_management_software} is a 2-itemset.

- Support count of an itemset in data set *D*: number of transactions in *D* that contain the itemset.

- Minimum support count = *min_sup* × total number of transactions in a data set.

- Frequent itemset in a data set *D*: itemset whose support count in *D* is at least the minimum support count.

How to Mine Association Rules

- A two-step process:
  - Find all frequent itemsets ---- the key step
  - Generate strong association rules from frequent itemsets.

- Example: given min_sup=50% and min_conf=50%

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

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

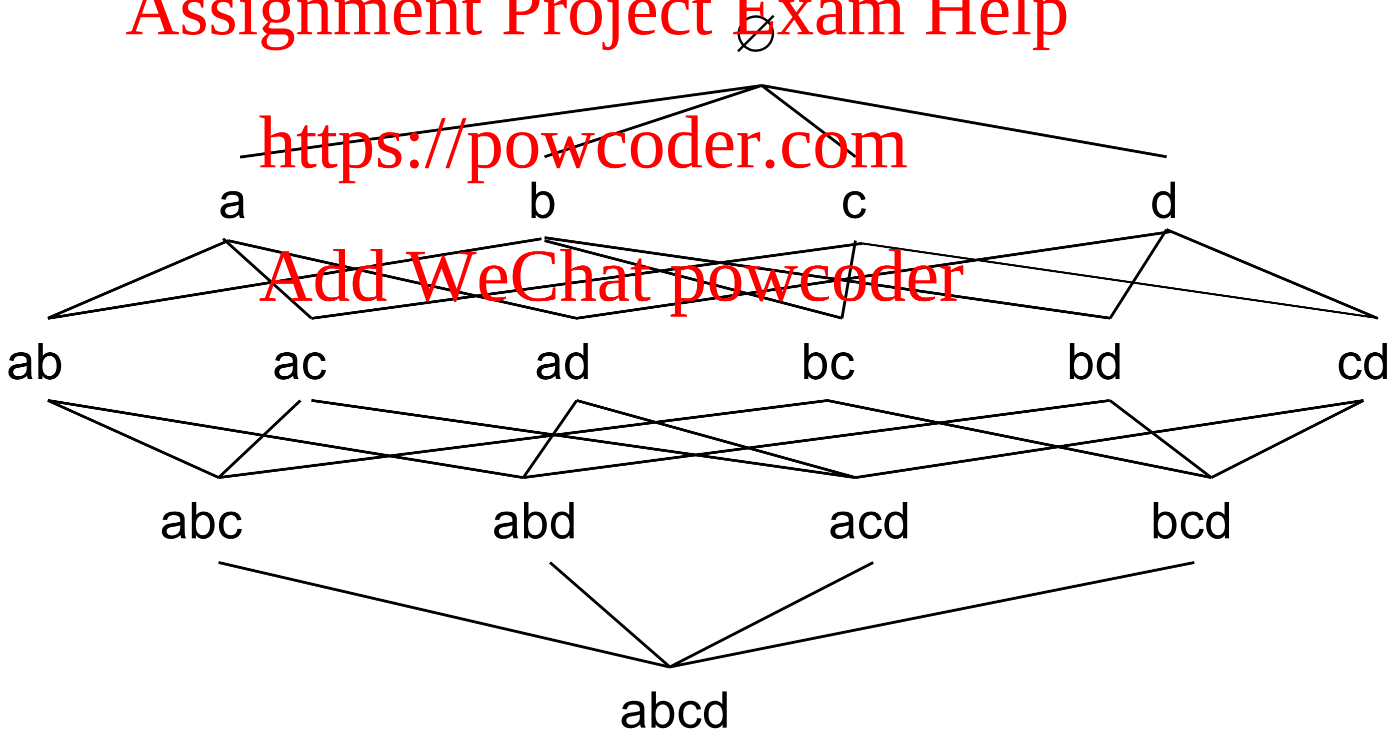
- Generate strong rules:
  - $\{A\} \rightarrow \{C\}$ [support=50%, confidence=66.6%]
  - $\{C\} \rightarrow \{A\}$ [support=50%, confidence=100%]

Finding Frequent Itemsets

- Objective: given a database, find all the itemsets (or sets of attribute-value pairs) that satisfy the minimum support count.
- Algorithms
  - Apriori
  - FP-Growth
  - H-Mine
  - Eclat
  - Partition
  - CLOSET
  - CHARM
  - etc.

Search Space for Finding All Frequent Itemsets

- Search space for DB with 4 items:



Apriori

- The Apriori property (an anti-monotone property), also called *downward closure* property:

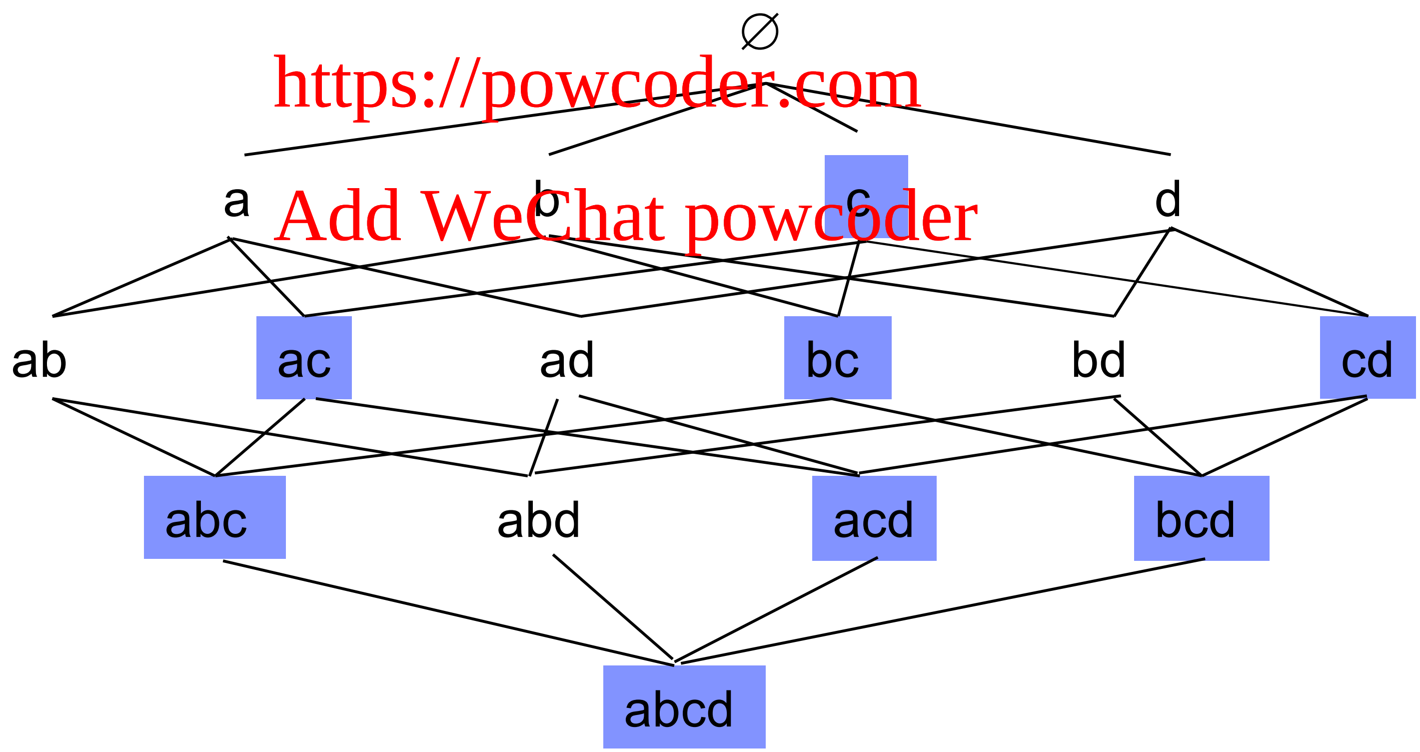
  Any nonempty subset of a frequent itemset must be frequent

  - i.e., if {*A*,*B*,*C*} is a frequent itemset, {A,B}, {A,C}, {B,C}, {*A*}, {*B*} and {C} must also be frequent.
  - This is because a transaction containing {A, B, C} also contains {A,B}, {B,C}, …. Thus, the support count of a subset is not less than that of the superset.

- No superset of any infrequent itemset should be checked.
  - Many item combinations can be pruned from the search space.
- Apriori-based mining (level-wise iterations)
  - Generate length (k+1) candidate itemsets from frequent k-itemsets
  - Test the candidates against DB

Pruning Search Space Using Apriori Property

- If *c* is not frequent, all the sets containing *c* are pruned:



The Apriori Algorithm

- Based on the Apriori property, use *iterative level-wise approach* and *candidate generation-and-test*
- Pseudo-code:

C_k: a set of candidate itemsets of size k
L_k : the set of frequent itemsets of size k

L_1 = {frequent items}; *(Scan database to find all frequent 1-itemsets)*
for (k = 1; L_k $!=\varnothing$; k++) **do begin**

    C_{k+1} = candidates generated from L_k;
    if $C_{k+1} != \varnothing$
        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} satisfying min_support
    end
return $\cup_k L_{k-1}$;

Scan database to calculate support for each itemset in C_{k+1}

Level-wise Generation process: $L_k \rightarrow C_{k+1} \rightarrow L_{k+1}$

The 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

C_1

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

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

C_2

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

L_2

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

C_3

itemset
{2 3 5}

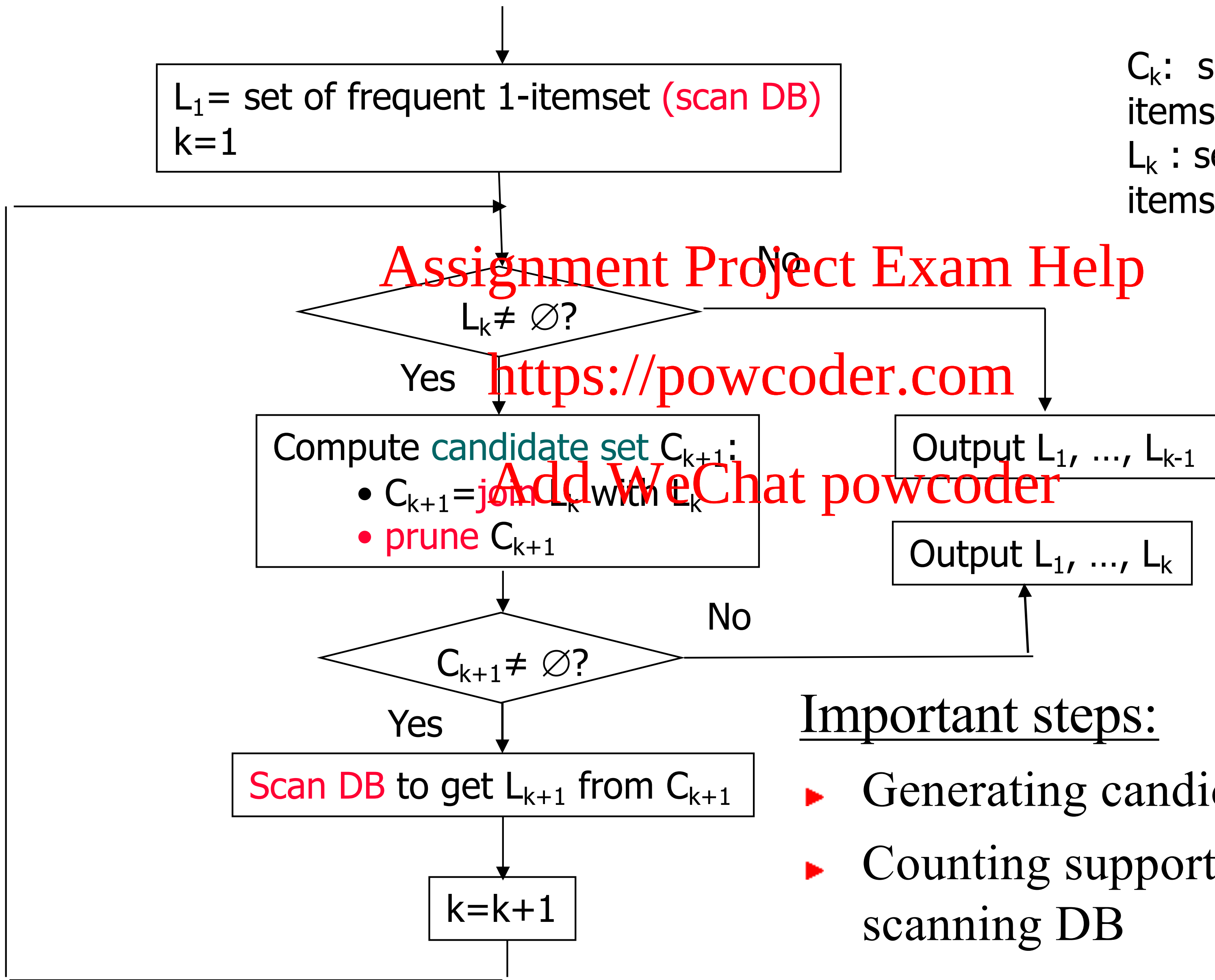
Scan D

L_3

itemset	sup
{2 3 5}	2

Assume:
min_sup_count = 2

Apriori Algorithm (Flow Chart)



C_k: set of candidate k-itemsets
L_k : set of frequent k-itemsets

<u>Important steps:</u>

- Generating candidates
- Counting supports of candidates by scanning DB

How to Generate Candidates?

(i.e., How to Generate C_{k+1} from L_k)

- Given L_k = the set of frequent k-itemsets
- List the items in each itemset of L_k in an order

L_3 =

{1 2 3}
{1 2 4}
{1 3 4}
{1 3 5}
{2 3 4}

- Given L_k , generate C_{k+1} in two steps:
  - Join Step: Join L_k with L_k by joining two k-itemsets in L_k. Two k-itemsets are joinable if their first (k-1) items are the same and the last item in the first itemset is smaller than the last item in the second itemset (the condition for joining two members of L_k).

    Now, C_4={{1 2 3 4}, {1 3 4 5}}
  - Prune Step: Delete all candidates in C_{k+1} that have a non-frequent subset by checking all length-k subsets of a candidate
    - Now, C_4={{1 2 3 4}}

Example of Candidate-generation

- L_4={*abcd, abcg, abdg, abef, abeh, acdg, bcdg*}
- Self-joining: $L_4 * L_4$
  - *abcdg* from *abcd* and *abcg*
  - *abefh* from *abef* and *aceh*
- Pruning:
  - *abefh* is removed because *abfh or aefh or befh* is not in L_4
- C_5={*abcdg*}

The 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

C_1

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

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

C_2

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

L_2

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

C_3

itemset
{2 3 5}

Scan D

L_3

itemset	sup
{2 3 5}	2

Assume:
min_sup_count = 2

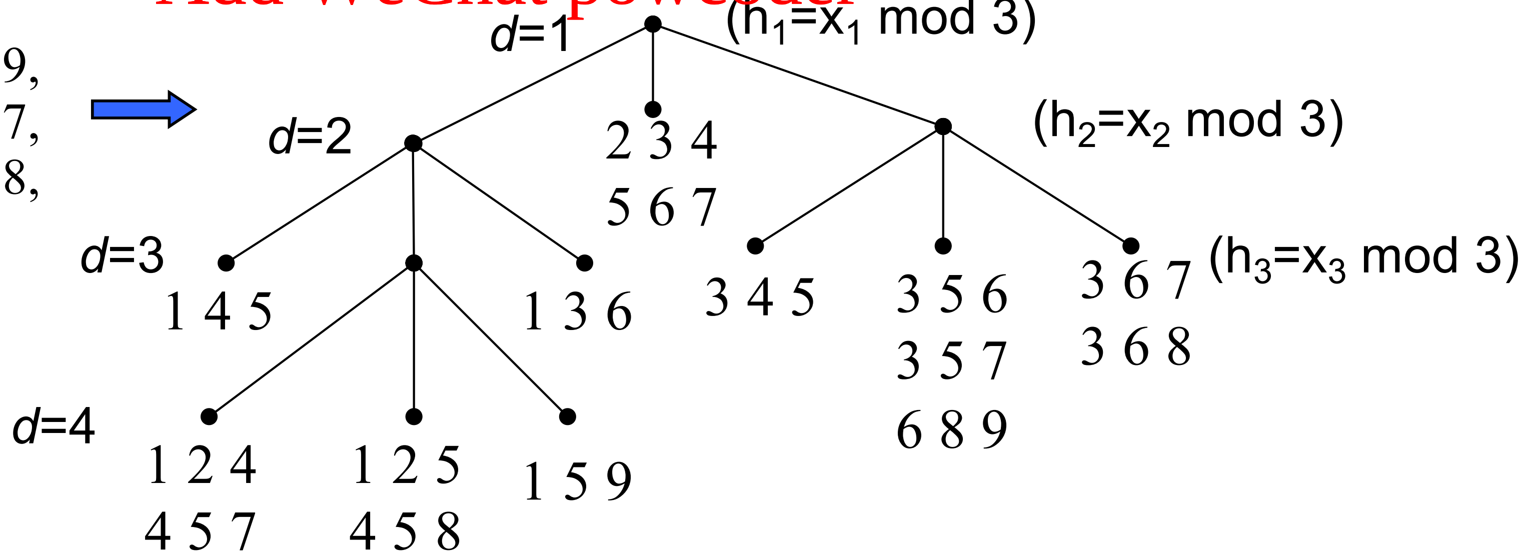
Support Counting of Candidates in DB scan

- Objective of candidate support counting
  - Find frequent itemsets L_k from a set of candidates C_k
- Naïve method: match each candidate with each transaction. Time consuming when
  - the total number of candidates is very large
  - one transaction contains many candidates
- Hash tree method:
  - Store candidate itemsets in C_k in a *hash-tree*
    - Leaf node of hash-tree contains a list of candidates and their counts
    - Interior node contains a hash table
  - Use a *subset function* to find all the candidates contained in a transaction

Building a Hash Tree

- Initially treat the root as a leaf node
- Insert itemsets in C_k into the tree
- When the number of itemsets in a leaf node exceeds a specified threshold (such as 3 in the following example), convert the leaf node into an interior node by
  - hashing the itemsets according to the d-th item of the itemset, where d is the level of the node.

Given C_3 = { 124, 125, 136, 145, 159, 234, 345, 356, 357, 367, 368, 457, 458, 567, 689 }



Subset Function

- Functionality
  - Given C_k (in a hash tree) and a transaction t, find all the candidates in C_k contained in t and increase the count of these candidates:

    Subset (C_k, t): candidate itemsets contained in t

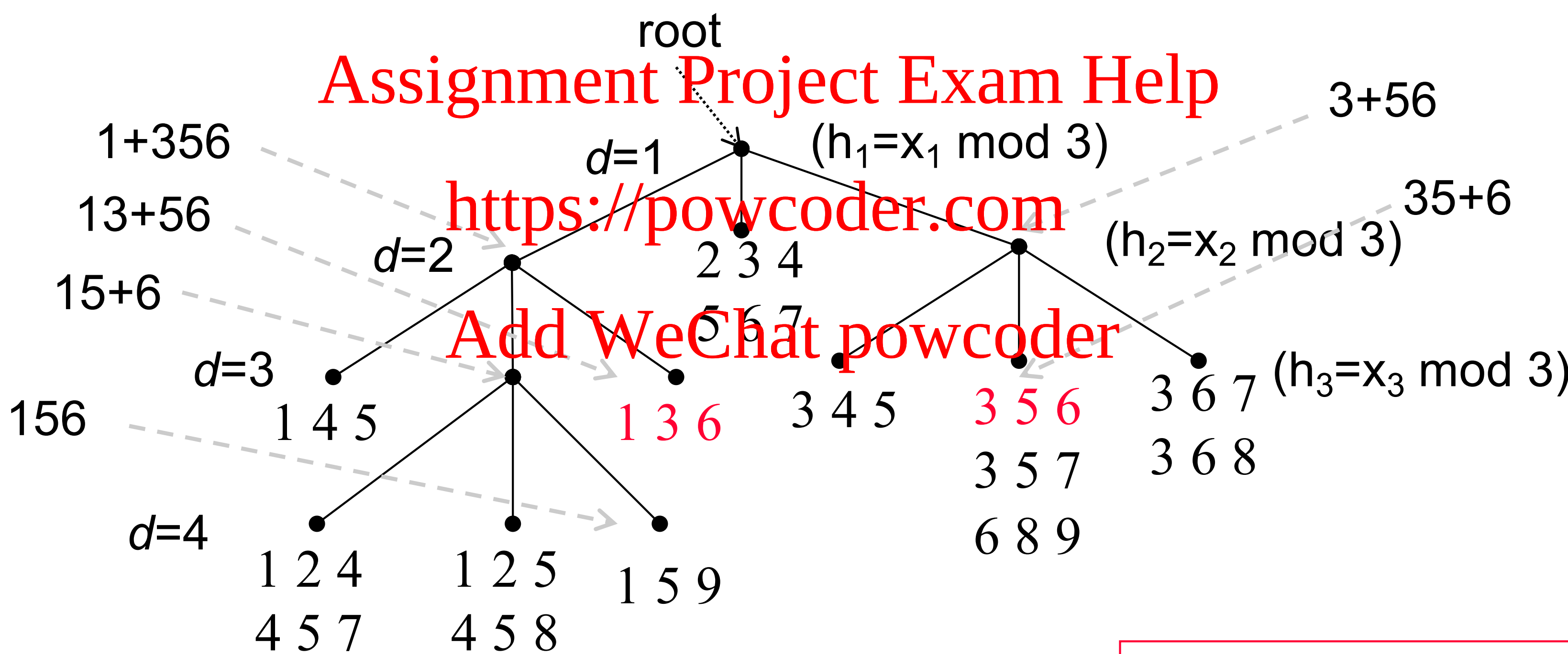
- Method

  - At root, hash on every item in t (until the kth item from the end) .

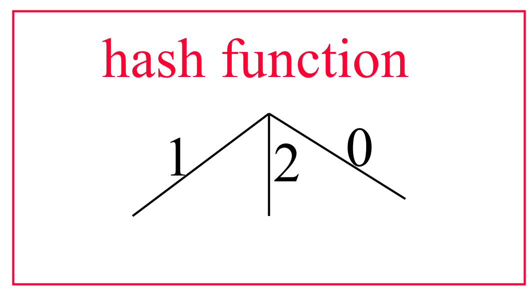
  - At an interior node reached by hashing on item i, hash on each item that comes after i in t (until the $(k\text{-}d+1)th$ item from the end, where d is the level of the node in the tree), recursively apply to the nodes in the corresponding bucket
  - At a leaf, find itemsets contained in t
- Benefit
  - Don't have to match each candidate with each transaction.

Example: finding the candidates contained in a transaction.

- Given a transaction {1 3 5 6} and hash tree:



It only goes to the leaf that may contain subsets of the transaction.



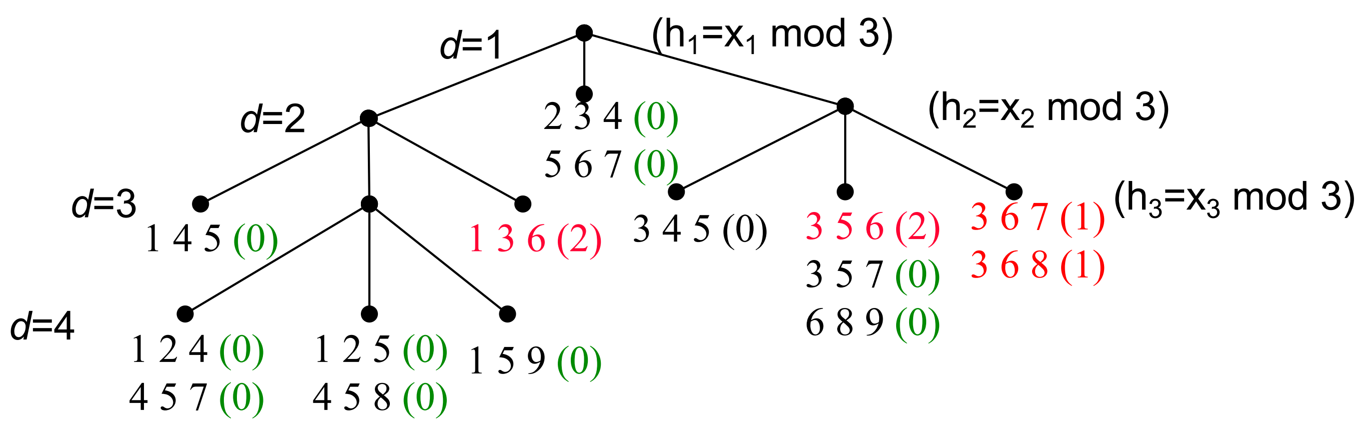
Scan DB to Obtain the Counts of All Length-3 Candidates

- Assume a transaction DB:

  {1,3,5,6}
  {2,3,5}
  {1,3,6,7,8}
  {3,5,6}
  .....

- The counts of all the candidates in the hash tree are initialized to zero.

- Apply the subset function on each transaction and increase the count of a candidate if the transaction contains the candidate



How to Mine Association Rules

- A two-step process:
  - *Find all frequent itemsets* *---- done*
  - Generate strong association rules from frequent itemsets.

- Example: given min_sup=50% and min_conf=50%

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

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

- Generate strong rules:
  - $\{A\} \rightarrow \{C\}$ [support=50%, confidence=66.6%]
  - $\{C\} \rightarrow \{A\}$ [support=50%, confidence=100%]

Generate Association Rules from Frequent Itemsets

- Naïve algorithm:

for each frequent itemset *l* whose length $\geq$ 2 do
    for each **nonempty proper** subset *s* of *l* do
        if (support(*l*) / support(*s*) >= min_conf)
            output the rule $s \rightarrow l - s$, with
                support = support(l) and
                confidence = support(l) / support(s)

Note that we only need to check the confidence.
Do we need to scan the database? **No**. *Why*?

Generate Association Rules from Frequent Itemsets

- Example:
  - Given a frequent itemset: $l = \{A, B, C\}$
  - nonempty proper subsets of l are $\{A,B\}$, $\{A,C\}$, $\{B,C\}$, $\{A\}$, $\{B\}$, $\{C\}$
  - resulting association rules:

$$\{A,B\} \rightarrow \{C\}, \quad confidence = \frac{support(\{A,B,C\})}{support(\{A,B\})} = 50\%$$

$$\{A,C\} \rightarrow \{B\}, \quad confidence = \frac{support(\{A,B,C\})}{support(\{A,C\})} = 100\%$$

$$\{B,C\} \rightarrow \{A\}, \quad confidence = \frac{support(\{A,B,C\})}{support(\{B,C\})} = 100\%$$

$$\{A\} \rightarrow \{B,C\}, \quad confidence = \frac{support(\{A,B,C\})}{support(\{A\})} = 33\%$$

$$\{B\} \rightarrow \{A,C\}, \quad confidence = \frac{support(\{A,B,C\})}{support(\{B\})} = 29\%$$

$$\{C\} \rightarrow \{A,B\}, \quad confidence = \frac{support(\{A,B,C\})}{support(\{C\})} = 100\%$$

These confidence values are make-up numbers. This example is not continued previous examples.

If minimum confidence threshold is 70%, only 3 rules are outputted.

Is Apriori Fast Enough? *Performance Bottlenecks*

- The core of the Apriori algorithm:
  - Use frequent *k* -itemsets to generate candidate (*k+1*)-itemsets
  - Use database scan and pattern matching to collect counts for the candidate itemsets to generate frequent (k+1)-itemsets from k+1 candidate set

- The bottleneck of *Apriori*: candidate generation and testing

  - Huge candidate sets:
    - 10^4 frequent 1-itemsets will generate more than 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 at least $2^{100} \approx 10^{30}$ candidates.
  - Multiple scans of database:
    - Needs *n* or *n+1* scans, *n* is the length of the longest frequent pattern

Improving Apriori: General Ideas

- Shrink the number of candidates
  - Hash-based technique

    (DHP — *direct hashing and pruning* — algorithm)
- Reduce the number of database scans on disk
  - Partitioning data (Partition algorithm)
- Avoid candidate generation
  - FP-growth

Shrink the Number of Candidates (*DHP*)

- *Hash-based technique* can be used to reduce the size of C_k, especially C_2
- Build a hash table when scanning DB to generate L_1.
  - For all 2-itemsets in each transaction, hash into the buckets and increase counts.

Hash function: $h(x, y) = ((id \text{ of } x) \times 10 + (id \text{ of } y)) \bmod 7$

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}

Only this array is stored

Shrink the Number of Candidates (*DHP*) (Cont'd)

- If the count of a bucket is less than minimum support count, all the itemsets in the bucket are removed from C_2

The hashtable in the last slide was generated from this data set →

TID	items
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

Reduce the number of disk scans (*Partition*)

- Partition DB
  - Each partition is held in main memory
- Any itemset that is potentially frequent in DB must be frequent in at least one of the partitions of DB (can be proved)
  - Scan 1: partition database and for each partition find local frequent patterns
  - Scan 2: consolidate global frequent patterns
- A. Savasere, E. Omiecinski, and S. Navathe. An efficient algorithm for mining association in large databases. In *VLDB'95*

Improving Apriori: General Ideas

- *Shrink the number of candidates*
  - *Hash-based technique*
- *Reduce the number of database scans*
  - *Partitioning data*
- Avoid candidate generation
  - FP-growth (next)

FP-Growth

- J. Han, J. Pei, and Y. Yin. *Mining Frequent Patterns without Candidate Generation.*, Proc. 2000 ACM-SIGMOD Int. Conf. on Management of Data (SIGMOD'00), Dallas, TX, May 2000.
- J. Han, J. Pei, Y. Yin and R. Mao, *Mining Frequent Patterns without Candidate Generation: A Frequent-Pattern Tree Approach*, Data Mining and Knowledge Discovery, 8(1):53-87, 2004. (http://www-faculty.cs.uiuc.edu/~hanj/pdf/dami04_fptree.pdf)
- Chapter 6.2.4 (3rd edition) or Chapter 5.2.4 (2nd Edition)

Mining Frequent Patterns Without Candidate Generation (FP-growth)

Two major steps:

- 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
- Mine frequent patterns (itemsets) from an FP-tree
  - A divide-and-conquer methodology: decompose mining tasks into smaller ones
  - Efficient: avoid candidate generation -- generate frequent patterns from the tree directly.

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

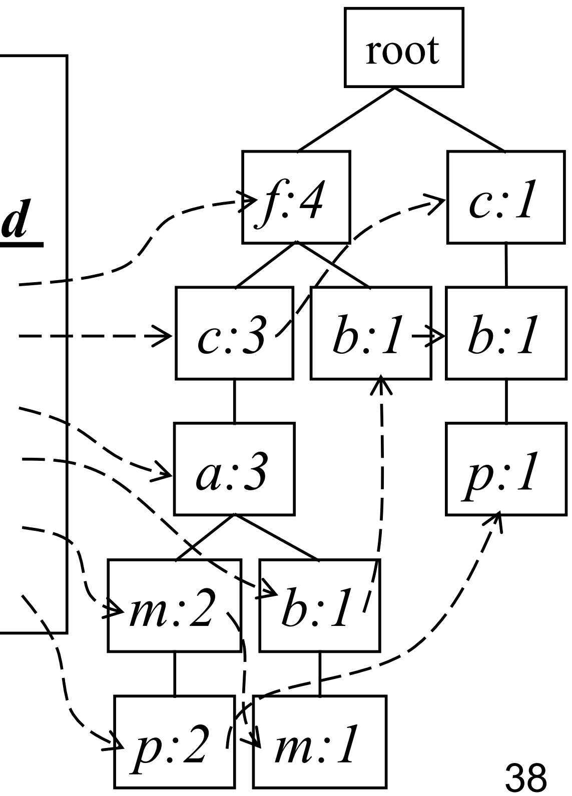
minimum suppot count =3

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

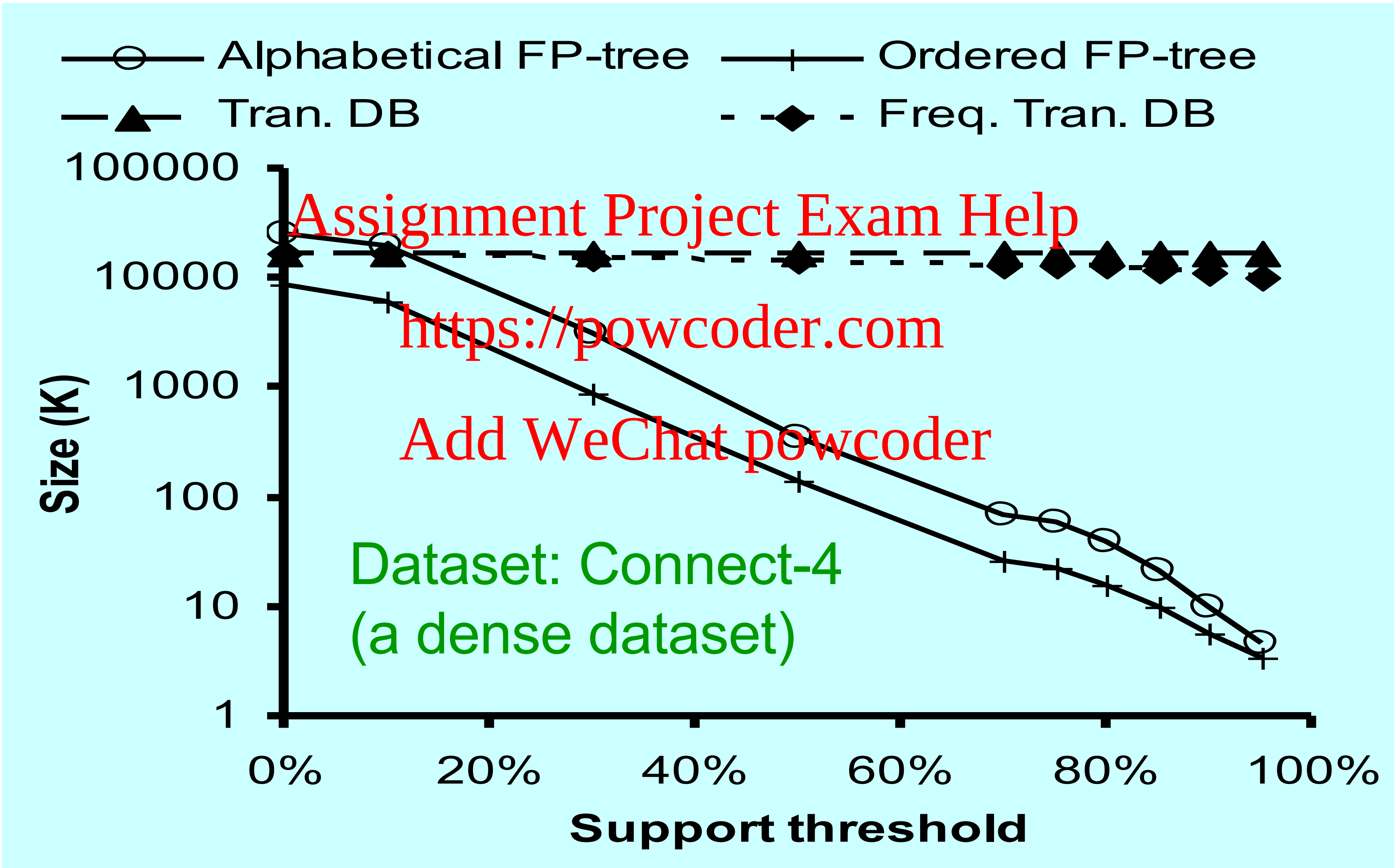
Item	*count*	*head*
f	*4*	
c	*4*	
a	*3*	
b	*3*	
m	*3*	
p	*3*	



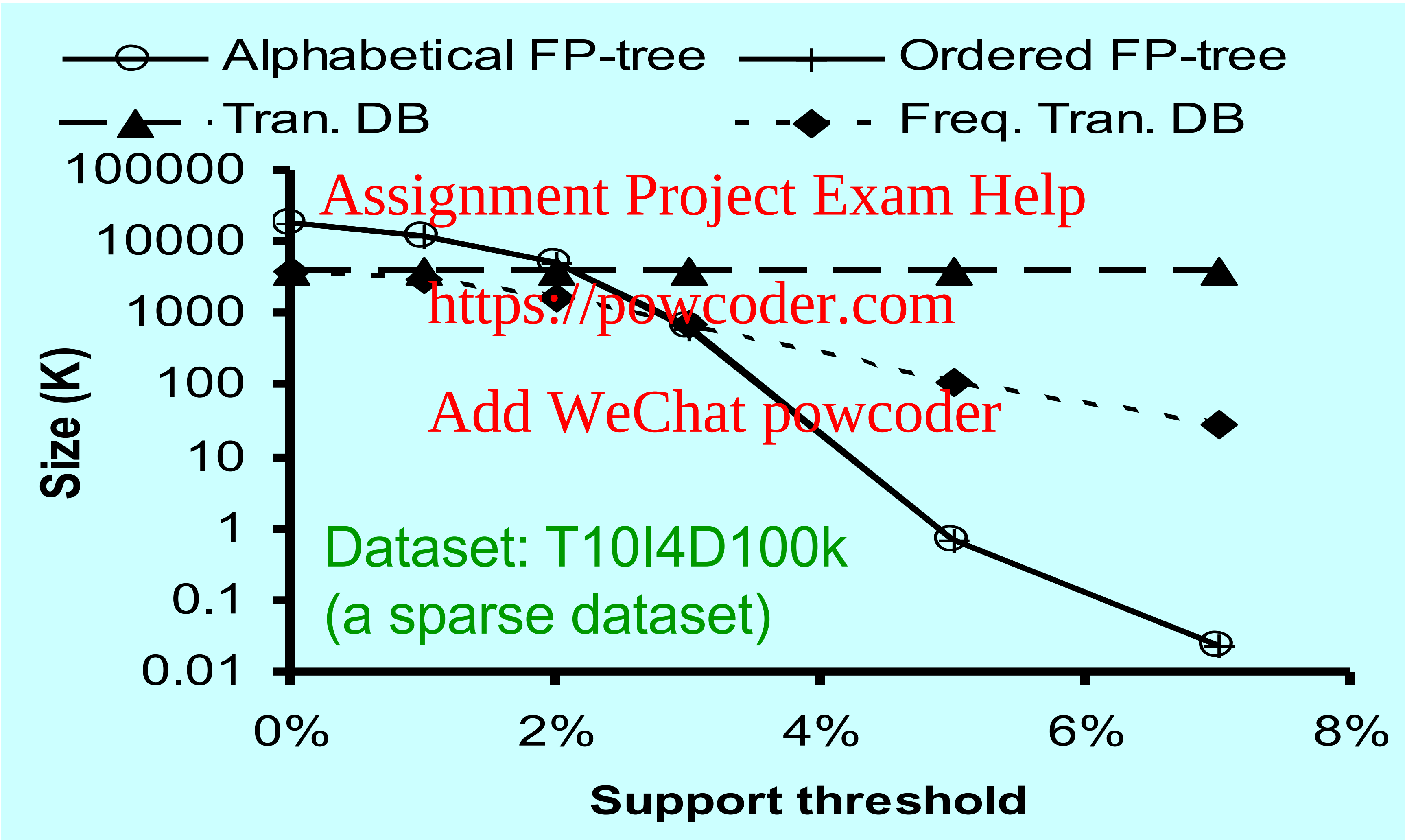
Benefits of the FP-tree Structure

- Completeness:
  - map each transaction into a path in the tree
  - preserves complete information for frequent pattern mining
    - no need to scan the database any more
- Compactness
  - reduce irrelevant information—infrequent items are gone
  - A path can store one or more transactions
  - Items in frequency descending order (*f-list*):
    - more frequent items are more likely to be shared
  - never be larger than the original database (not counting node-links and the count fields)

How Effective Is FP-tree?



Compressing Sparse Dataset



Mining Frequent Patterns from FP-tree

(Frequent pattern = frequent itemset)

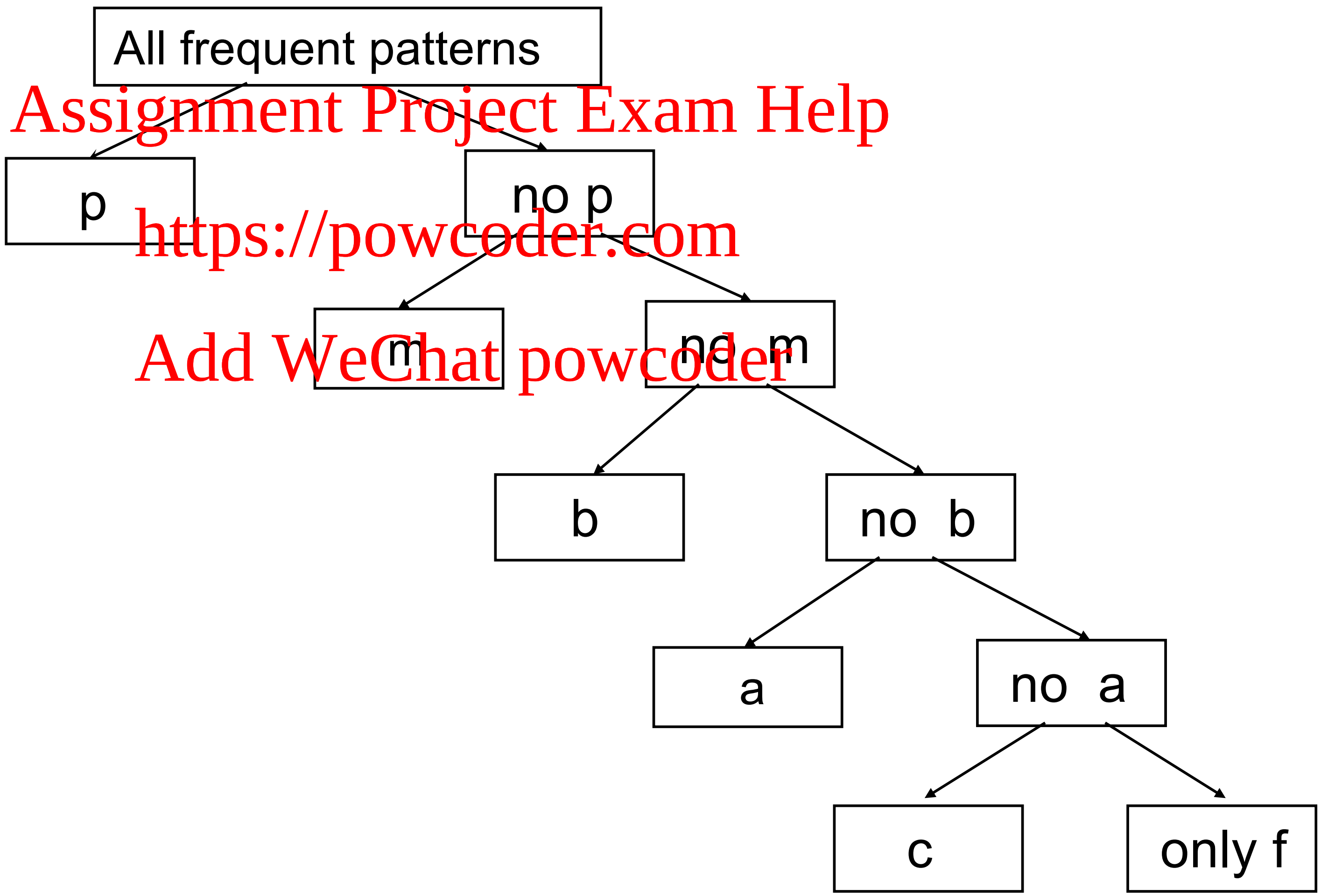
- General idea (divide-and-conquer):

  Recursively
  - partition the set of frequent patterns
  - build *conditional pattern base* and *conditional FP-tree* for each partition

- Partition the set of frequent patterns
  - Frequent patterns can be partitioned into subsets according to f-list: f-c-a-b-m-p (the list of freq. items in frequency-descending order)
    - Patterns containing p
    - Patterns having m but no p
    - …
    - Patterns having c but no a nor b, m, or p
    - Pattern f
  - The partitioning is complete and without any overlap

Partitioning Frequent Patterns

f-list: f-c-a-b-m-p



Find Frequent Patterns Having Item "p"

- Only transactions containing p are needed
- Form p-*conditional pattern base* (p-*projected database*) TDB|p Contains transactions containing p
  - Starting at entry p of header table
  - Follow the side-link of frequent item p
  - Accumulate all transformed prefix paths of p

p-conditional pattern base $TDB|_p$

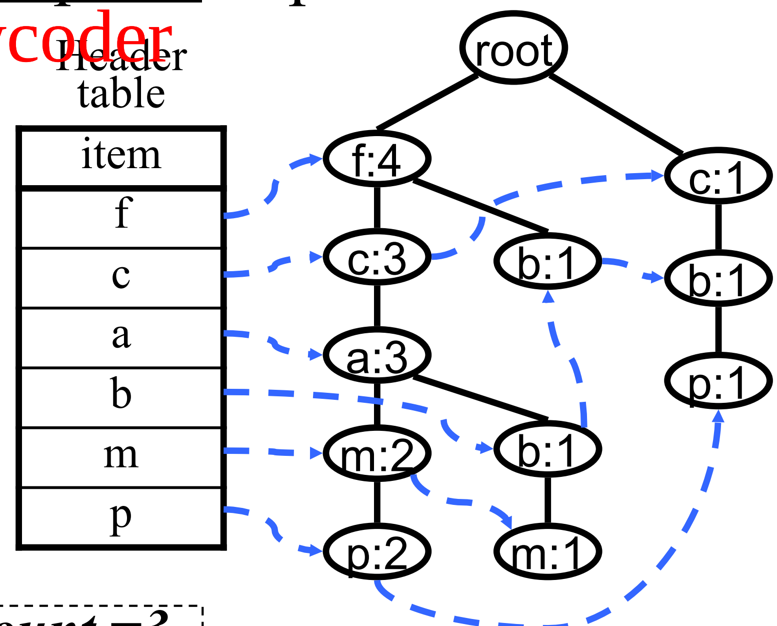
fcam: 2

cb: 1

Local frequent item: c:3

Frequent patterns containing p

cp: 3

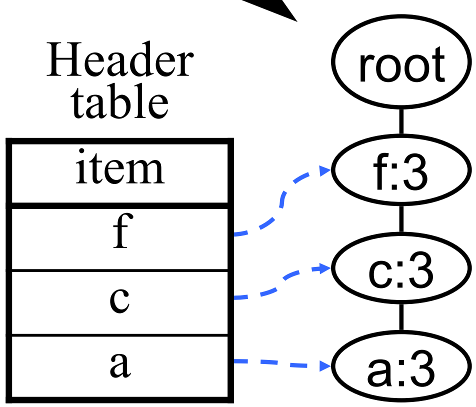


minimum suppot count =3

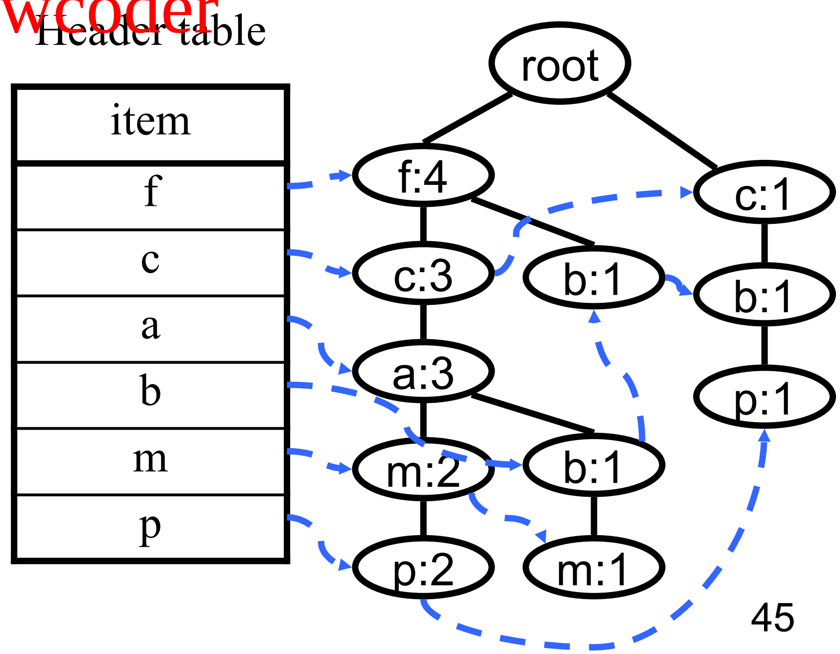
Find Frequent Patterns Having Item m But No p

- Form m-conditional pattern base (m-projected database) TDB|m
  - Item p is excluded (by looking at only the prefix paths of m)
  - TDB|m contains fca:2, fcab:1
- Recursively apply FP-growth to find freq. patterns from TDB|m
  - Local frequent items: f, c, a
  - After removing local infrequent item: fca:2, fca:1
  - Build m-conditional FP-tree from TDB|m

Frequent patterns having m:
fm, cm, am
fcm, fam, cam
fcam

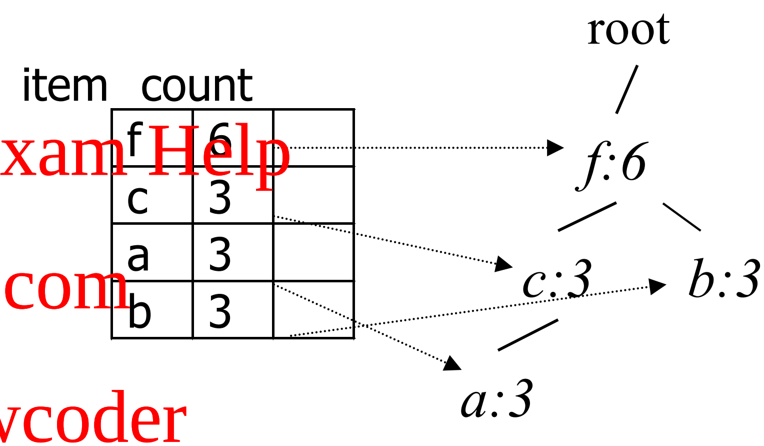


m-conditional FP-tree



Find Frequent Patterns Having Item m But No p (*more complex situation*)

- Suppose m-conditional pattern base is: fca:3, fb:3
- Local frequent items: f:6, c:3, a:3, b: 3
- Build m-conditional FP-tree
- First generate:
  - fm: 6, cm: 3, am:3, bm:3
- To learn longer patterns containing m
  - Further partition frequent patterns containing m (but no p) into
    - Patterns containing b
    - Patterns containing a but no b
    - Patterns containing c but no b or a
    - Patterns containing only f (i.e. fm)



- Compute *ym*-conditional pattern bases:

ym	conditional pattern base
bm	*f:3*
am	*fc:3*
cm	*f:3*

Find Frequent Patterns Having Item m But No p (*more complex situation*)

- Having *ym*-conditional pattern bases:

ym	conditional pattern base
bm	*f:3*
am	*fc:3*
cm	*f:3*

- Built *ym*-conditional FP-trees

bm: root — *f:3*

am: root — *f:3* — *c:3*

cm: root — *f:3*

- General frequent patterns with suffix ym:

fbm, *fam, cam, fcam*, *fcm*

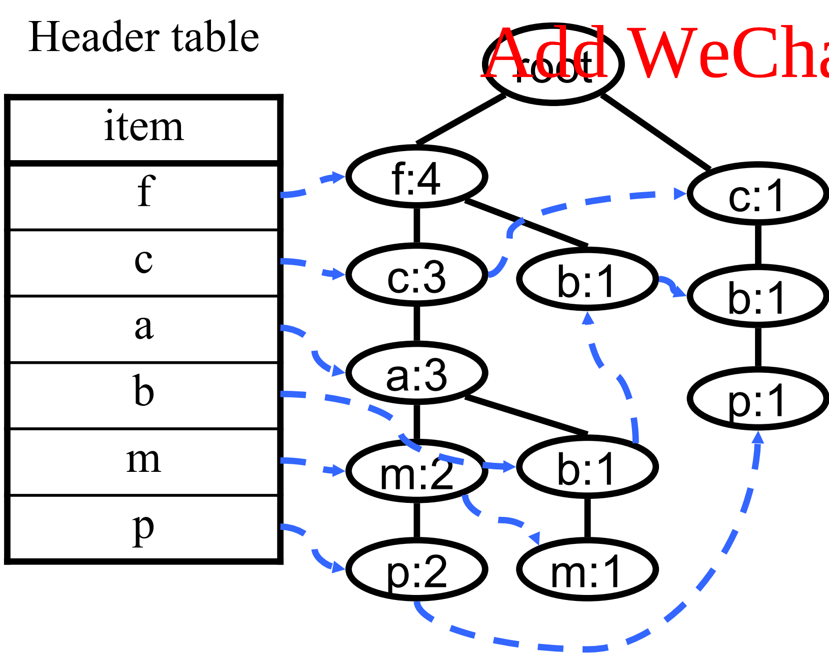
Major Steps to Mine FP-tree

For each item in the FP-tree

1. Construct *conditional pattern base*
2. Construct *conditional FP-tree* from the conditional pattern-base
3. Generate frequent patterns from the conditional FP-tree
   - If the conditional FP-tree contains a single path, simply enumerate all the patterns
   - Otherwise, recursively mine the conditional FP-tree and grow frequent patterns obtained so far

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 prefix paths of that item to form a *conditional pattern base*



***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*

Step 2: Construct Conditional FP-tree

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

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

fca:2, fcab:1

↓ **Taken out infrequent items**

fca:2, fca:1

↓

root
|
f:3
|
c:3
|
a:3

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

***p-conditional* pattern base:**

fcam:2, cb:1

↓ **Taken out infrequent items**

c:2, c:1

↓

root
|
c:3

***p-conditional* FP-tree**

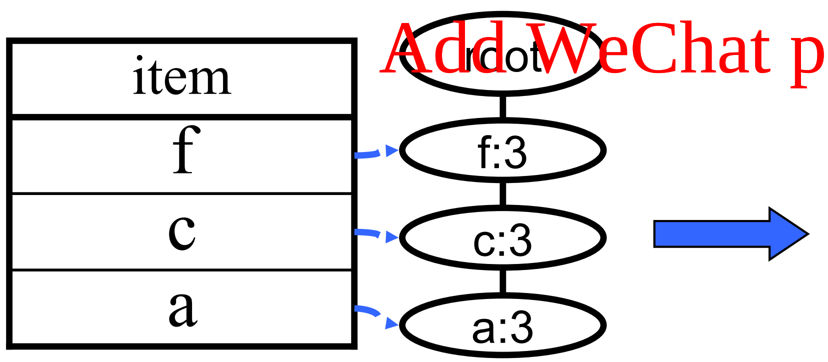
Conditional Pattern-Bases and Conditional FP-trees

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

Step 3: Generate Frequent Patterns from Conditional FP-tree

- If an x-conditional FP-tree has a single path P
  - The complete set of frequent patterns with suffix x can be generated by enumeration of all the combinations of items in P

Header table



All frequent patterns with suffix *m*

fm:3, cm:3, am:3,

fcm:3, fam:3, cam:3,

fcam:3

m-conditional FP-tree

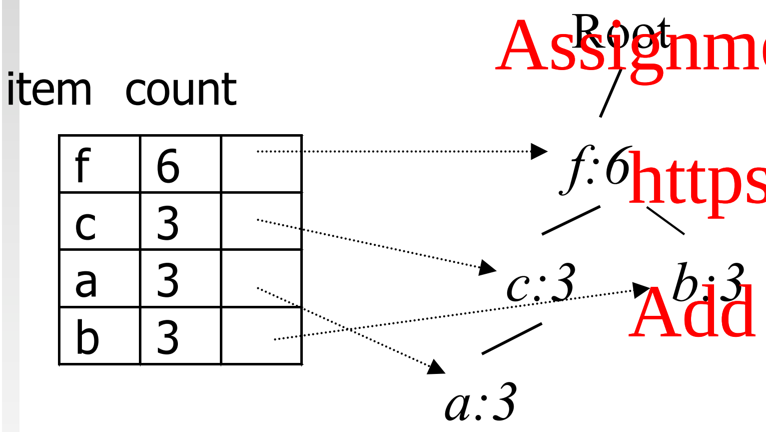
***m-conditional* FP-tree**

Step 3: Generate Frequent Patterns from Conditional FP-tree (*Contd.*)

- **If an *x*-conditional FP-tree has more than one path**
  - For each item *y* that appears in *x*-conditional FP-tree
    - Generate pattern *yx* with support = the support of y in *x*-conditional FP-tree.
    - Construct *yx*-conditional pattern base and then *yx*-conditional FP-tree to generate frequent patterns with suffix *yx* (a recursive procedure).

Step 3: Generate Frequent Patterns from Conditional FP-tree (*Contd.*)

- Suppose m-conditional FP-tree is



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

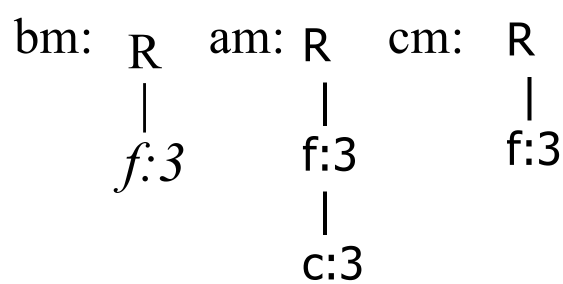
- Generate frequent 2-itemsets having m:

  fm:6, *cm*:3, *am*:3, *bm*:3

- Compute *ym*-conditional pattern bases:

ym	conditional pattern base
bm	*f:3*
am	*fc:3*
cm	*f:3*

- Built *ym*-conditional FP-trees



- General frequent patterns with suffix ym:

  fbm:3, fam:3, cam:3, fcam:3, fcm:3

FP-Growth Algorithm

- Input: *FP-tree* (a FP-tree built by scanning DB)
- Output: the complete set of frequent patterns
- Method: call FP-growth(*FP-tree, null*)
- Procedure FP-growth(*A_conditional_FP_Tree, A*)
  - if Tree contains a single path *P*
    - for each combination (denoted as *B*) of the nodes in the path *P* do
      - generate pattern *BA* with support = minimum support of nodes in *B*
  - else for each item a_i in the header table of *Tree* do
    - generate pattern $B=a_iA$ with support = the support of a_i
    - construct *B*'s conditional pattern base and then *B*'s conditional FP-tree $Tree_B$;
    - if $Tree_B$ in not empty,
      - call **FP-growth**($Tree_B$, *B*)

Exercise

- A transaction DB:

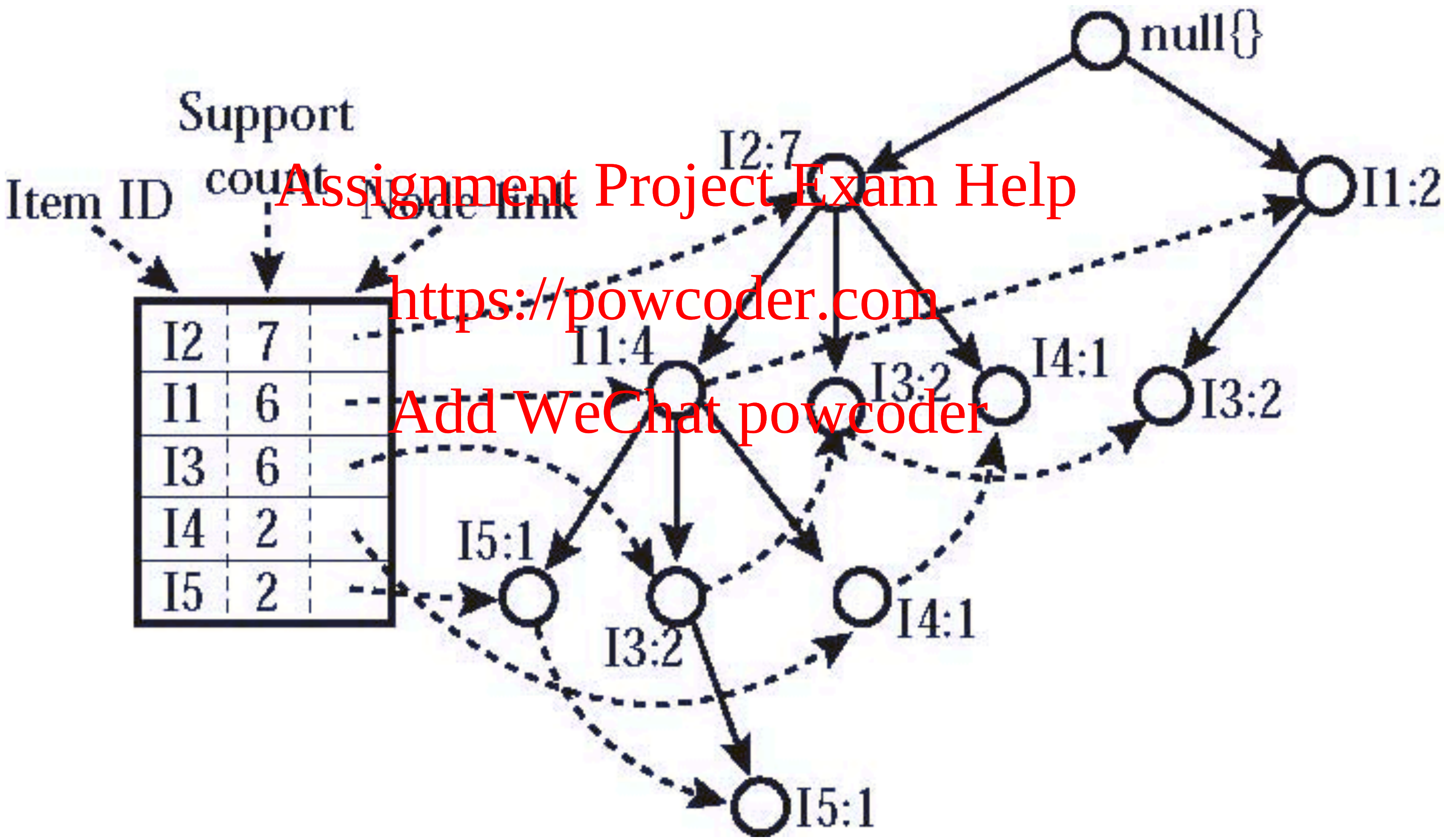
TID	items
T100	I1, I2, I5
T200	I2, I4
T300	I2, I3
T400	I1, I2, I4
T500	I1, I3
T600	I2, I3
T700	I1, I3
T800	I1, I2, I3, I5
T900	I1, I2, I3

- Support counts for single items:

Item	Sup. count
{I1}	6
{I2}	7
{I3}	6
{I4}	2
{I5}	2

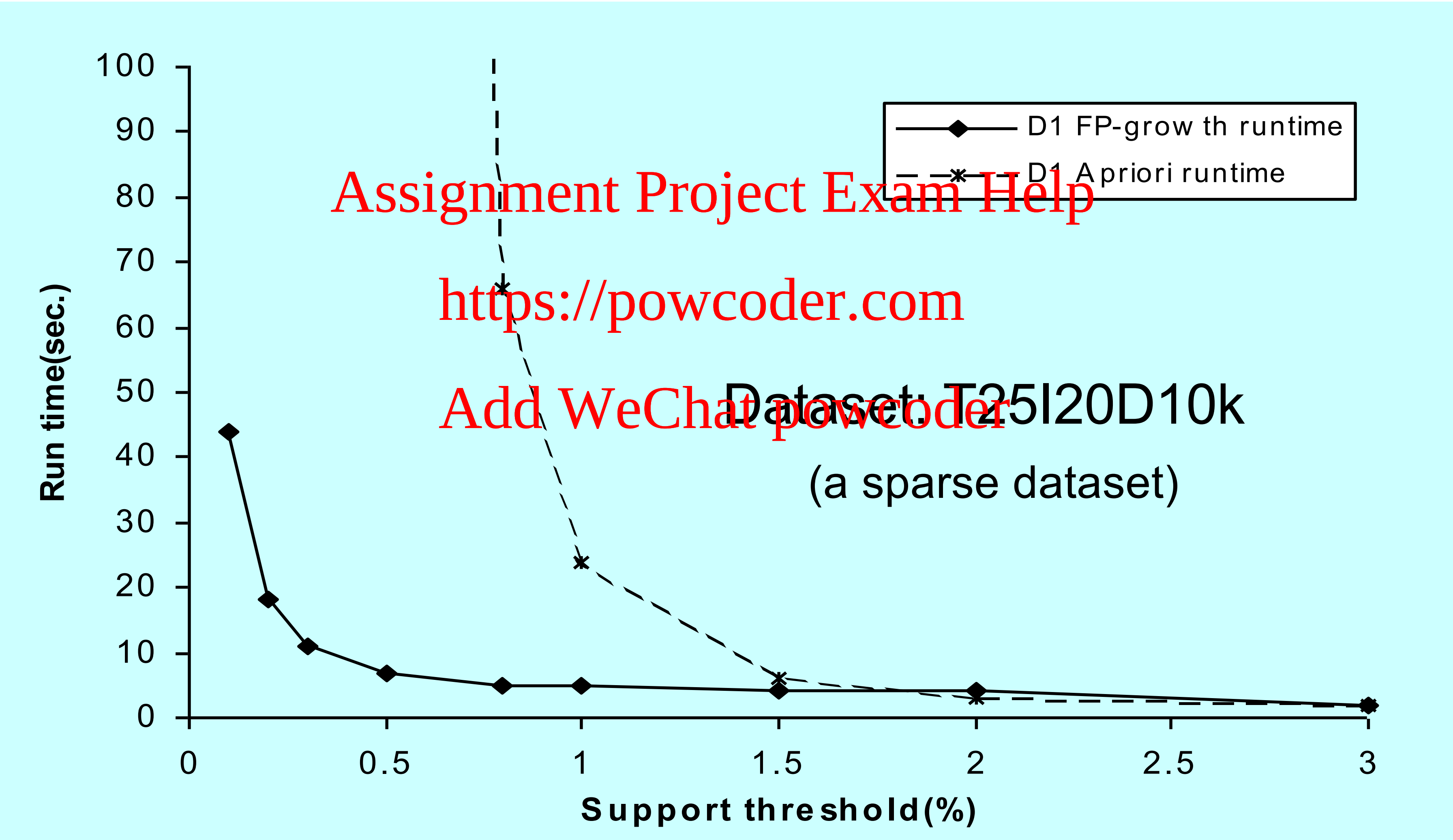
- Find all frequent patterns with minimum support count =2.

FP-tree

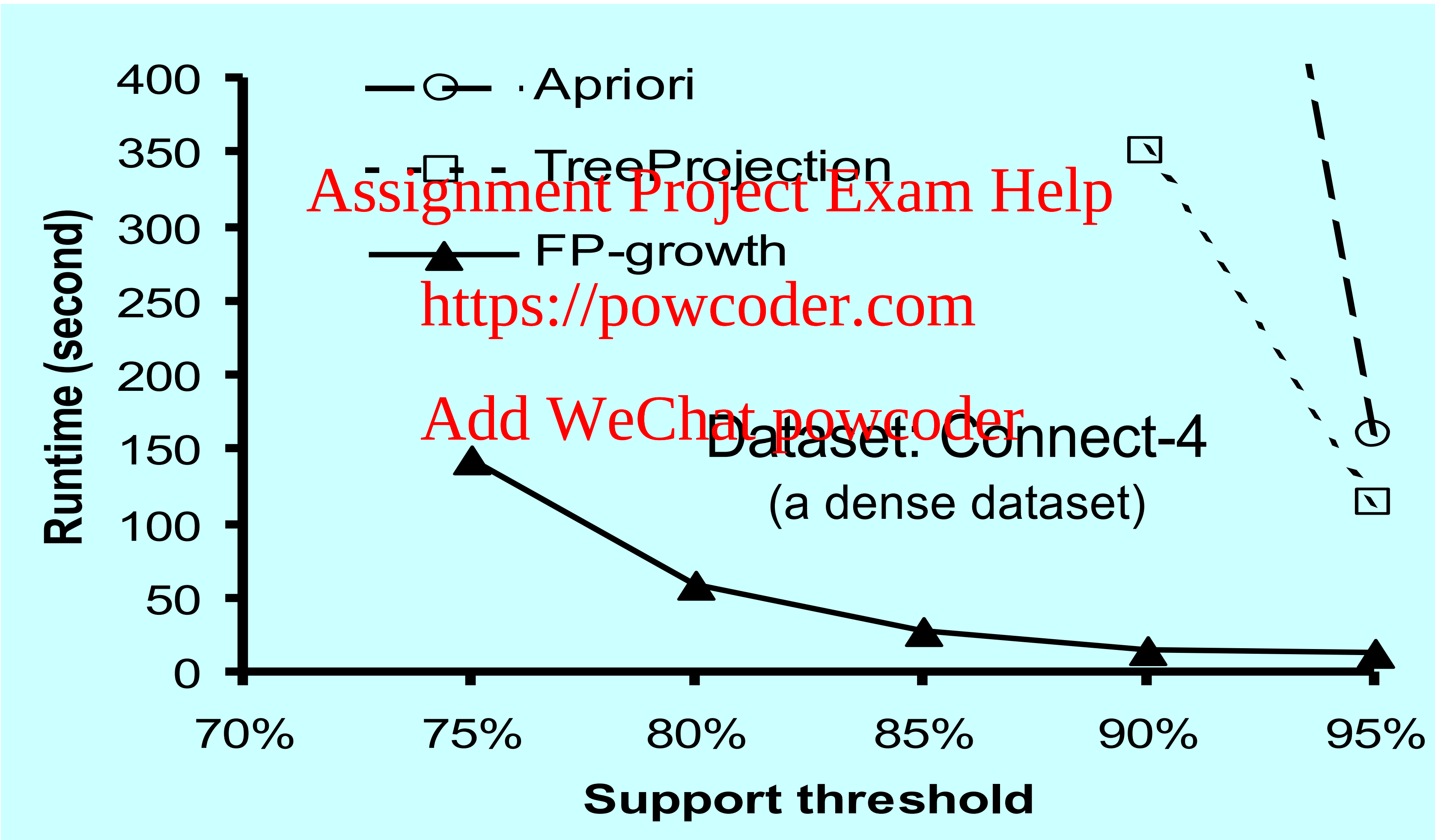


Item ID	Support count	Node-link
I2	7	
I1	6	
I3	6	
I4	2	
I5	2	

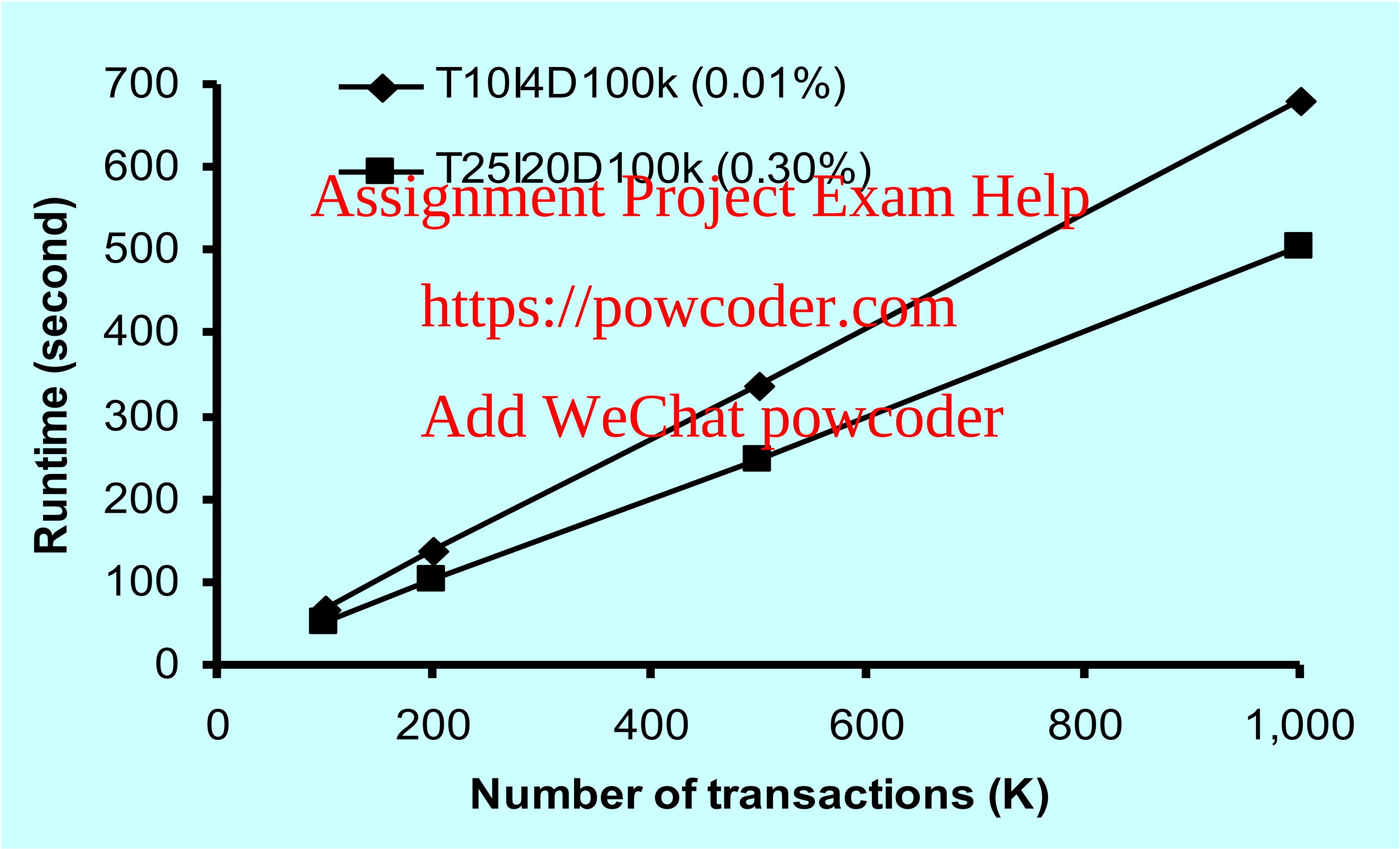
FP-growth vs. Apriori



Mining Very Dense Dataset



Scalability of FP-growth



Why Is FP-growth Efficient?

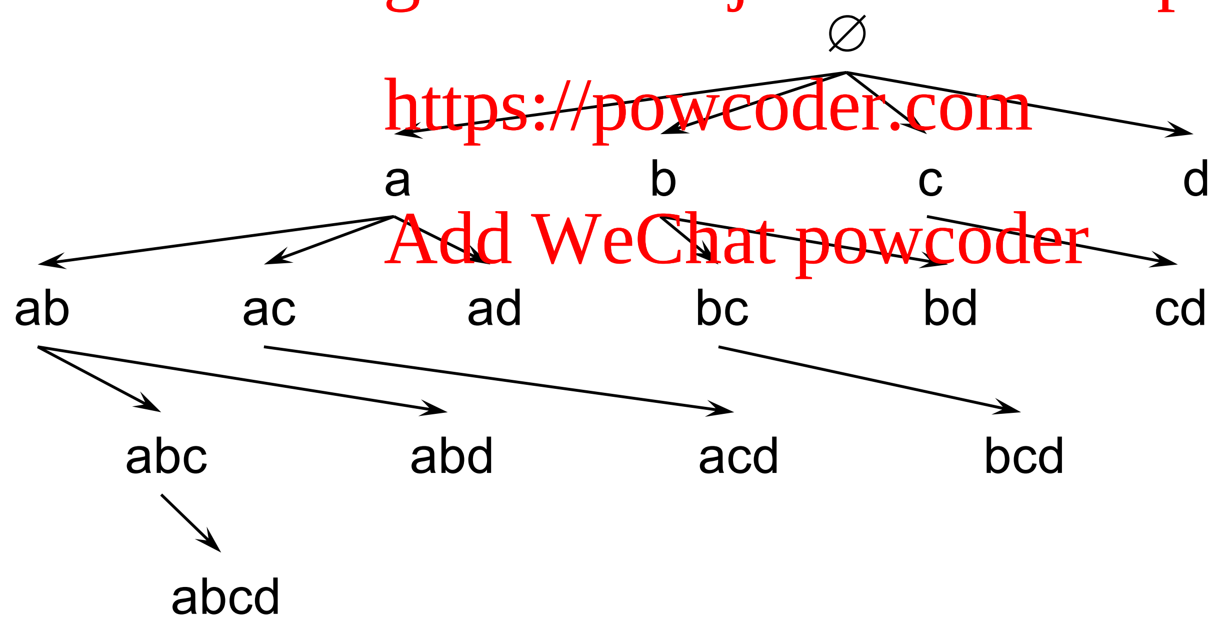
- Divide-and-conquer strategy
  - Decompose both the mining task and DB
  - Lead to focused search of smaller databases
- No candidate generation nor candidate test
- Database compression using FP-tree
  - No repeated scan of entire database
- Basic operations:
  - counting local freq items and building FP-tree, no pattern search nor pattern matching

Major Costs in FP-growth

- Building FP-trees
  - A stack of FP-trees
- Redundant information is stored in a stack of FP-trees.
- Can we avoid the redundancy?
  - H-mine (another algorithm by Pei and Han)?

Compare FP-growth to Apriori

- Search space for DB with 4 items:



- Apriori: breadth-first
- FP-growth: Depth-first

Outline

- Basic concepts of association rule learning
- Apriori algorithm
- FP-Growth Algorithm
- <u>Finding interesting rules</u>

Two Problems with Association Rule Mining

- Quantity problem
  - Too many rules can be generated
    - Given a dataset, the number of rules generated depends on the support and confidence thresholds.
      - If the support threshold is high, a small number of rules are generated. But some interesting rules are missed.
      - If the support threshold is low, a huge number of rules are generated.
- Quality problem
  - Not all the generated rules are *interesting*

Number of Generated Patterns versus Support Threshold (An Example)

Support threshold	0.02	0.01	0.008	0.005	0.003	0.0028	0.0025	0.002	0.001
Num. of rules (conf. thres.=0.5)	2	14	39	88	723	4,556	74,565	4,800,070	$>10^9$
Num. of rules (conf. thres.=0.8)	1	7	17	38	591	4,172	65,615	3,584,339	$>10^9$

Number of sessions (transactions): 30586

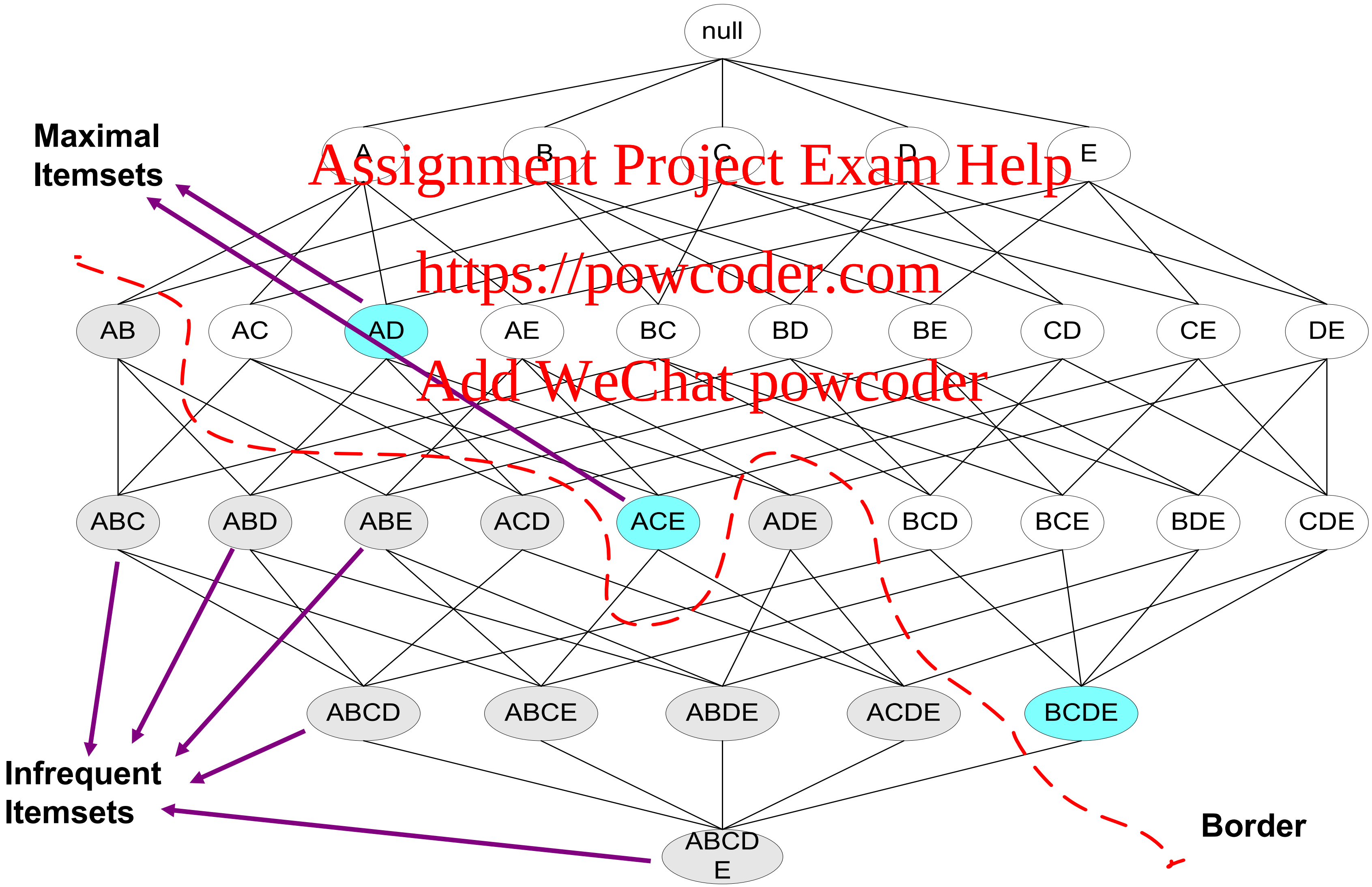
Number of objects (items): 38679

Solutions to the Problems

- Finding only *maximum* or *closed* frequent patterns
  - Other frequent patterns can be generated from them
- Constraint-based data mining
  - Applying constraints in the mining process so the search can be more focused.
- Using interestingness measures to remove or rank rules
  - Remove misleading associations and find correlation rules
  - Prune patterns using other interestingness measures
- Using rule structures
  - Eliminate structurally and semantically redundant rules.
  - Group or summarize related rules

Maximal Frequent Itemset

An itemset *X* is a ***maximal frequent itemset*** in a data set *D* if *X* is frequent and none of the proper super-set of *X* is frequent in *D*.



Maximal Frequent Patterns

- Reducing the # of patterns returned to the user
- Maximal frequent patterns are a *lossy* compression of frequent patterns
  - Given the set of all maximal frequent patterns and their supports in a data set *D*, we can generate all the frequent patterns, *but not their supports*.

- Algorithm for mining maximal frequent itemsets: MaxMiner
  - R. Bayardo. Efficiently mining long patterns from databases. *SIGMOD'98*

Closed Patterns

- Problem with maximal frequent itemsets:
  - Supports of their subsets are not known – additional DB scans are needed (to get the supports)
- An itemset is *closed* if none of its proper supersets has the same support as the itemset

TID	Items
1	{A,B}
2	{B,C,D}
3	{A,B,C,D}
4	{A,B,D}
5	{A,B,C,D}

Itemset	Support
{A}	4
{B}	5
{C}	3
{D}	4
{A,B}	4
{A,C}	2
{A,D}	3
{B,C}	3
{B,D}	4
{C,D}	3

Itemset	Support
{A,B,C}	2
{A,B,D}	3
{A,C,D}	2
{B,C,D}	2
{A,B,C,D}	2

Closed Frequent Patterns

- An itemset *X* is a ***closed frequent itemset*** in a data set *D* if *X* is both *closed* and *frequent* in *D* with respect to a support threshold.
- Closed frequent itemsets are a *lossless* compression of frequent patterns

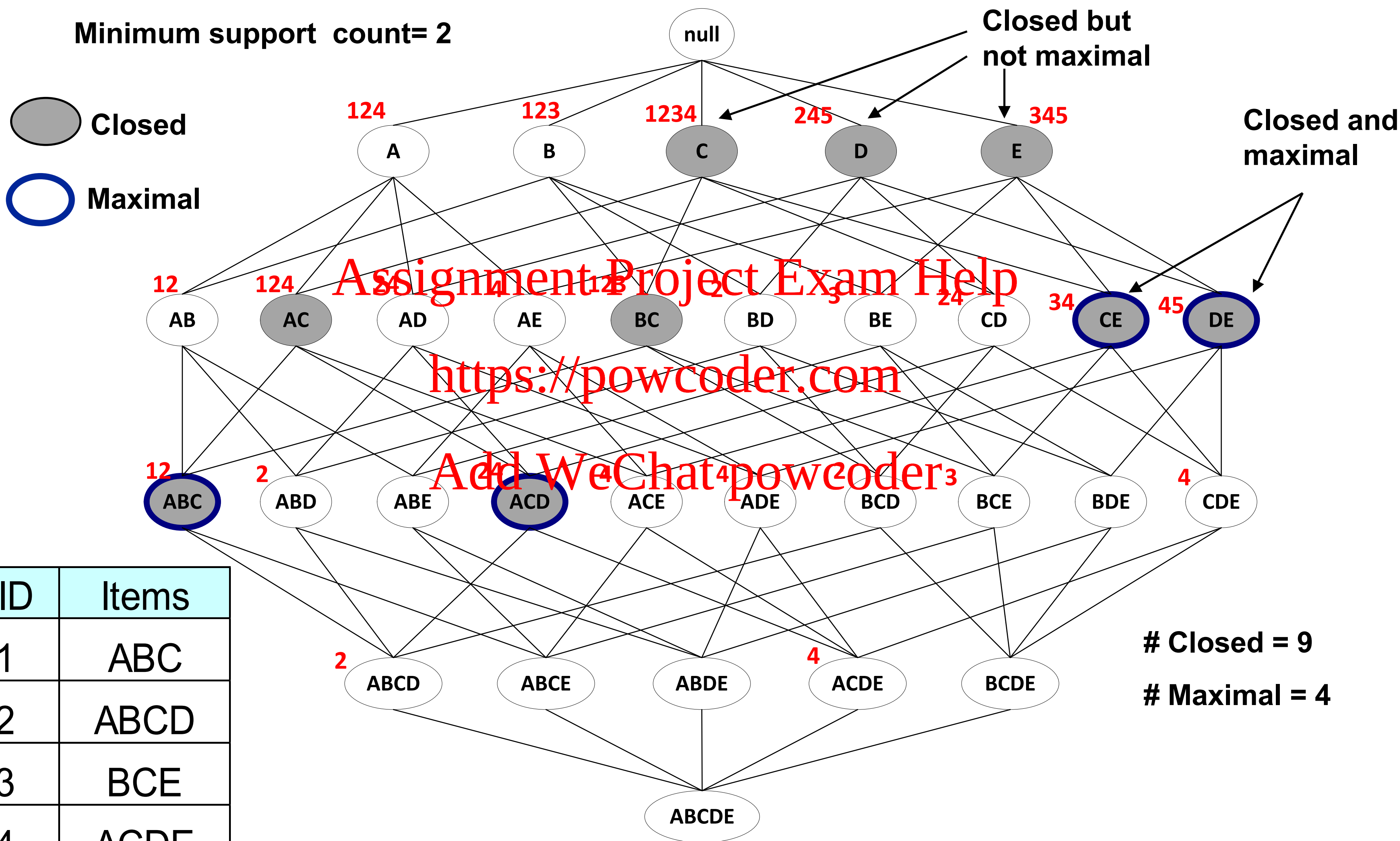
  - Reducing the # of patterns returned to the user

  - Given the set of all closed frequent patterns and their supports in a data set *D*, the user can generate all the frequent patterns and their supports.

- Algorithm for finding closed frequent patterns: CLOSET
  - J. Pei, J. Han & R. Mao. "CLOSET: An Efficient Algorithm for Mining Frequent Closed Itemsets", DMKD'00.

Maximal vs Closed Frequent Itemsets

Minimum support count= 2



 # Closed = 9

 # Maximal = 4

TID	Items
1	ABC
2	ABCD
3	BCE
4	ACDE
5	DE

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 frequent itemsets?
  - $<a_1, \ldots, a_{100}>$: 1
  - $< a_1, \ldots, a_{50}>$: 2
- What is the set of maximal frequent itemsets?
  - $<a_1, \ldots, a_{100}>$: 1
- What is the set of all frequent itemsets?
  - !!

Solutions to the Problems

- Finding only *maximum* or *closed* frequent patterns
  - Other frequent patterns can be generated from them
- *Constraint-based data mining*
  - *Applying constraints in the mining process so the search can be more focused.*
- Using interestingness measures to remove or rank rules
  - Remove misleading associations and find correlation rules
  - Prune patterns using other interestingness measures
- Using rule structures
  - Eliminate structurally and semantically redundant rules.
  - Group or summarize related rules

Constrain-based Frequent Pattern Mining

- Mining frequent patterns with constraint C
  - find all patterns satisfying not only min_sup, but also constraint C
- Examples of Constraints
  - *? → a particular product*
  - *a particular product → ?*
  - small sales (price < \$10) triggers big sales (sum > \$200)

Constrain-based Frequent Pattern Mining (Cont'd)

- A naïve solution
  - Testing frequent patterns on C as a post-processing process
- Some constraints can be incorporated into the mining process to improve the efficiency
- More efficient approaches
  - Analyze the properties of constraints comprehensively
  - Push the constraint as deeply as possible inside the frequent pattern mining
  - Example: find all frequent itemsets containing item "b"

Types of Constraints

- Anti-monotonic constraints
  - An itemset S satisfies the constraint, so does any of its subset (That is, S violates the constraint, so does any of its superset).
- Monotonic constraints
  - An itemset S satisfies the constraint, so does any of its superset
- Examples
  - Sum of the prices of items in S $\leq$ 100 is anti-monotone
  - Maximum price in S $\leq$ 15 is anti-monotone
  - Sum of the prices of items in S $\geq$100 is monotone
  - Minimum price in S $\leq$ 15 is monotone

How to Use Antimonotonic or Monotonic Constraints in Mining

- Antimonotonic constraints
  - In Apriori:
    - Use it to prune candidates in each iteration
  - In FP-growth
    - Use it to stop growing a pattern
- Monotonic constraints
  - If an itemset satisfies a monotonic constraint, no need to check its supersets on the constraint
    - Only checks their support

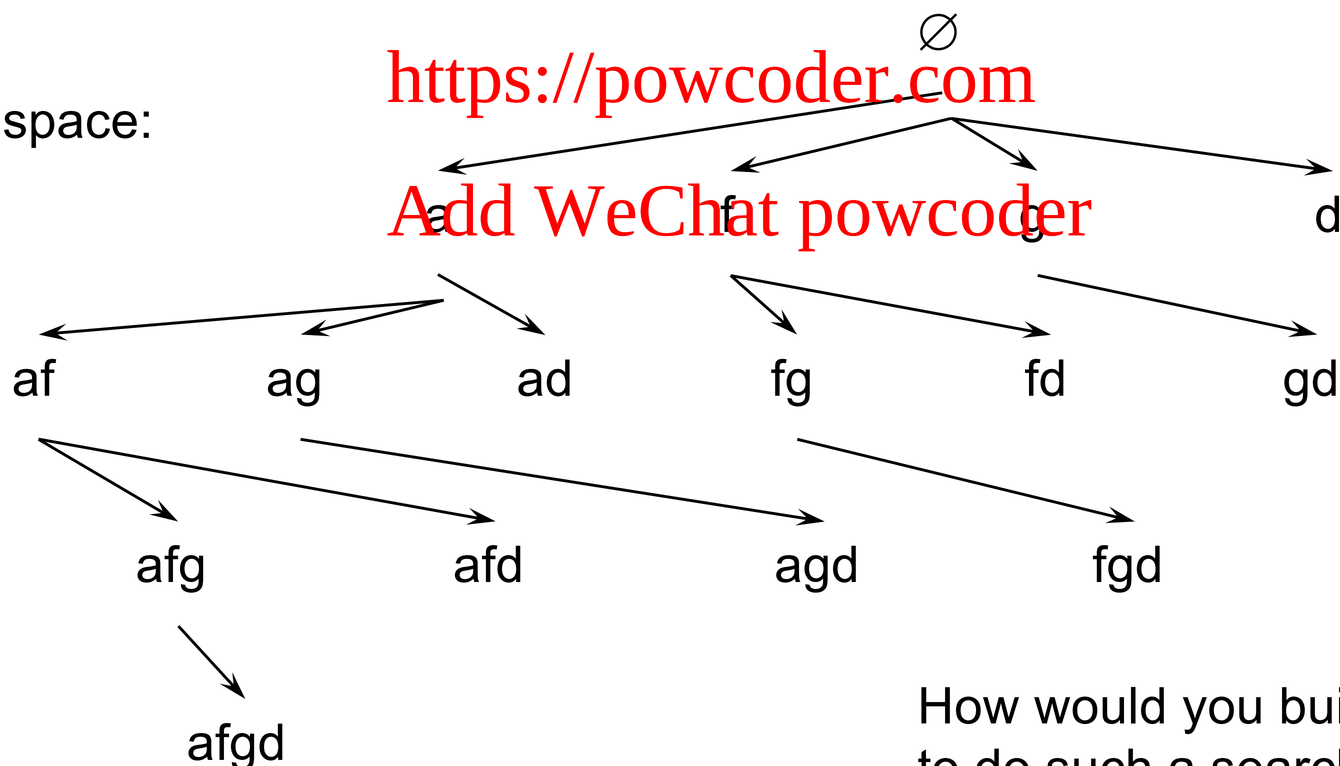
Types of Constraints (Cont'd)

- Convertible constraints
  - Some constraints are not anti-monotonic or monotonic
  - But can be converted to anti-monotonic or monotonic by properly ordering items
- Example of convertible constraint:
  - Average price of the items in S $\geq$ 25
  - Order items in price-descending order
    - <a, f, g, d, b, h, c, e>
  - If an itemset afb violates C
    - So does afbh, afb*
    - It becomes anti-monotone!

Example of Convertible Constraints

- Convertible constraint:
  - Average price of the items in S $\geq$ 25
- Price-descending order of items: a, f, g, d

Search space:



How would you build an FP-tree to do such a search?

Solutions to the Problems

- Finding only *maximum* or *closed* frequent patterns
  - Other frequent patterns can be generated from them
- Constraint-based data mining
  - Applying constraints in the mining process so the search can be more focused.
- *Using interestingness measures to remove or rank rules*
  - *Remove misleading associations and find correlation rules*
  - Prune patterns using other interestingness measures
- Using rule structures
  - Eliminate structurally and semantically redundant rules.
  - Group or summarize related rules

Misleading Association Rules

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

- *play basketball* $\Rightarrow$ *eat cereal* [40%, 66.7%] is misleading
  - 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 more accurate, although with lower support and confidence

Association ≠ Correlation !!!

Interestingness Measure: Correlation

- Correlation
  - If $P(A|B)>P(A)$, A and B are *positively correlated.*

    Note: $P(A \mid B) > P(A) \Leftrightarrow P(B \mid A) > P(B) \Leftrightarrow P(A\ B) > P(A)P(B)$

  - If $P(A|B)<P(A)$, A and B are *negatively correlated.*

    Note: $P(A \mid B) < P(A) \Leftrightarrow P(B \mid A) < P(B) \Leftrightarrow P(A\ B) < P(A)P(B)$

  - If $P(A|B)=P(A)$, A and B are *independent.*

    Note: $P(A \mid B) = P(A) \Leftrightarrow P(B \mid A) = P(B) \Leftrightarrow P(A\ B) = P(A)P(B)$

- A measure of correlation (called lift):

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

Pruning Misleading Rules: *Keep Correlation Rules*

- A measure of correlation (lift) for rule A→ B

$$lift(A \to B) = \frac{P(AB)}{P(A)P(B)}$$

- Rules whose lift ≤ 1 is ***misleading***, which should be removed
  - E.g. the following rule:

    play basketball ⇒ *eat cereal* [40%, 66.7%]

    should be removed because its lift is 0.89

Solutions to the Problems

- Finding only *maximum* or *closed* frequent patterns
  - Other frequent patterns can be generated from them
- Constraint-based data mining
  - Applying constraints in the mining process so the search can be more focused.
- *Using interestingness measures to remove or rank rules*
  - Remove misleading associations and find correlation rules
  - *Prune patterns using other interestingness measures*
- Using rule structures
  - Eliminate structurally and semantically redundant rules.
  - Group or summarize related rules

Many interestingness measures for A→ B

symbol	measure	range	formula
ϕ	ϕ-coefficient	-1...1	$\frac{P(A,B)-P(A)P(B)}{\sqrt{P(A)P(B)(1-P(A))(1-P(B))}}$
Q	Yule's Q	-1 ... 1	$\frac{P(A,B)P(\overline{A},\overline{B})-P(A,\overline{B})P(\overline{A},B)}{P(A,B)P(\overline{A},\overline{B})+P(A,\overline{B})P(\overline{A},B)}$
Y	Yule's Y	-1 ... 1	$\frac{\sqrt{P(A,B)P(\overline{A},\overline{B})}-\sqrt{P(A,\overline{B})P(\overline{A},B)}}{\sqrt{P(A,B)P(\overline{A},\overline{B})}+\sqrt{P(A,\overline{B})P(\overline{A},B)}}$
k	Cohen's	-1 ... 1	$\frac{P(A,B)+P(\overline{A},\overline{B})-P(A)P(B)-P(\overline{A})P(\overline{B})}{1-P(A)P(B)-P(\overline{A})P(\overline{B})}$
PS	Piatetsky-Shapiro's	-0.25 ... 0.25	$P(A,B)-P(A)P(B)$
F	Certainty factor	-1 ... 1	$\max(\frac{P(B\|A)-P(B)}{1-P(B)}, \frac{P(A\|B)-P(A)}{1-P(A)})$
AV	added value	-0.5 ... 1	$\max(P(B\|A)-P(B), P(A\|B)-P(A))$
K	Klosgen's Q	-0.33 ... 0.38	$\sqrt{P(A,B)}\max(P(B\|A)-P(B), P(A\|B)-P(A))$
g	Goodman-kruskal's	0 ... 1	$\frac{\Sigma_j \max_k P(A_j,B_k)+\Sigma_k \max_j P(A_j,B_k)-\max_j P(A_j)-\max_k P(B_k)}{2-\max_j P(A_j)-max_k P(B_k)}$
M	Mutual Information	0 ... 1	$\frac{\Sigma_i \Sigma_j P(A_i,B_j)\log\frac{P(A_i,B_j)}{P(A_i)P(B_J)}}{\min(-\Sigma_i P(A_i)\log P(A_i)\log P(A_i), -\Sigma_i P(B_i)\log P(B_i)\log P(B_i))}$
J	J-Measure	0 ... 1	$\max(P(A,B)\log(\frac{P(B\|A)}{P(B)}) + P(A\overline{B})\log(\frac{P(\overline{B}\|A)}{P(\overline{B})}),$ $P(A,B)\log(\frac{P(A\|B)}{P(A)}) + P(\overline{A}B)\log(\frac{P(\overline{A}\|B)}{P(\overline{A})}))$
G	Gini index	0 ... 1	$\max(P(A)[P(B\|A)^2+P(\overline{B}\|A)^2]+P(\overline{A}[P(B\|\overline{A})^2+P(\overline{B}\|\overline{A})^2]-P(B)^2-P(\overline{B})^2,$ $P(B)[P(A\|B)^2+P(\overline{A}\|B)^2]+P(\overline{B}[P(A\|\overline{B})^2+P(\overline{A}\|\overline{B})^2]-P(A)^2-P(\overline{A})^2)$
s	support	0 ... 1	$P(A,B)$
c	confidence	0 ... 1	$max(P(B\|A), P(A\|B))$
L	Laplace	0 ... 1	$\max(\frac{NP(A,B)+1}{NP(A)+2}, \frac{NP(A,B)+1}{NP(B)+2})$
IS	Cosine	0 ... 1	$\frac{P(A,B)}{\sqrt{P(A)P(B)}}$
γ	coherence(Jaccard)	0 ... 1	$\frac{P(A,B)}{P(A)+P(B)-P(A,B)}$
α	all_confidence	0 ... 1	$\frac{P(A,B)}{\max(P(A),P(B))}$
o	odds ratio	0 ... ∞	$\frac{P(A,B)P(\overline{A},\overline{B})}{P(\overline{A},B)P(A,\overline{B})}$
V	Conviction	0.5 ... ∞	$\max(\frac{P(A)P(\overline{B})}{P(A\overline{B})}, \frac{P(B)P(\overline{A})}{P(B\overline{A})})$
λ	lift	0 ... ∞	$\frac{P(A,B)}{P(A)P(B)}$
S	Collective strength	0 ... ∞	$\frac{P(A,B)+P(\overline{AB})}{P(A)P(B)+P(\overline{A})P(\overline{B})} \times \frac{1-P(A)P(B)-P(\overline{A})P(\overline{B})}{1-P(A,B)-P(\overline{AB})}$
χ^2	χ^2	0 ... ∞	$\Sigma_i \frac{(P(A_i)-E_i)^2}{E_i}$

Pruning rules with interestingness measure

- Choose a measure in your belief to assess the significance of a rule $A \rightarrow B$.

- Rank the rules according to their interestingness value.

- Remove rules with small interestingness values

Solutions to the Problems

- Finding only *maximum* or *closed* frequent patterns
  - Other frequent patterns can be generated from them
- Constraint-based data mining
  - Applying constraints in the mining process so the search can be more focused.
- Using interestingness measures to remove or rank rules
  - Remove misleading associations and find correlation rules
  - Prune patterns using other interestingness measures
- *Using rule structures to prune rules*
  - Eliminate structurally and semantically redundant rules.
  - Group or summarize related rules

Pruning Redundant Rules

- **Pruning Rule 1:** If there are two rules of the form $A \rightarrow C$ and $A \wedge B \rightarrow C$, and the interestingness value of rule $A \wedge B \rightarrow C$ is not significantly better than rule $A \rightarrow C$, then rule $A \wedge B \rightarrow C$ is redundant and should be pruned.

- **Pruning Rule 2:** If there are two rules of the form $A \rightarrow C_1$ and $A \rightarrow C_1 \wedge C_2$, and the interestingness value of rule $A \rightarrow C_1$ is not significantly better than rule $A \rightarrow C_1 \wedge C_2$, then rule $A \rightarrow C_1$ is redundant and should be pruned.

Summarizing and Grouping Association Rules

- Toivonen et al. (KDD'95)
  - Compute a subset of rules, called a structural rule cover, to reduce the number of rules and further grouped the rules in the cover using clustering
- Cristofor and Simovici (2002)
  - Define another type of rule cover, called informative cover, to group and summarize related rules.
- Khan, An and Huang (ICDM'03)
  - Proposed two algorithms
    - Objective grouping of rules according to the rule structure
    - Subjective grouping of rules according to the semantic relationship among items.

Related Topics

- Mining high utility patterns
  - Consider
    - the quantity $q(i, T_j)$ of an item i in a transaction T_j
    - the value (e.g., price $p(i)$) of an item i
  - Utility of an item i in a transaction T_j:

$$u(i, T_j) = q(i, T_j) \times p(i)$$

  - Utility of an itemset X in a transaction T_j:

$$u(X, T_j) = \sum_{i \in X} u(i, Tj)$$

  - Utility of an itemset X in a dataset D:

$$u(X, D) = \sum_{X \in T_j, T_j \in D} u(X, Tj)$$

  - *High utility pattern*: itemsets whose utility in the dataset is no less than a minimum utility threshold

Related Topics

- Mining high utility patterns (*cont'd*)
  - Challenge: utility does not have the downward closure property. That is,

The utility of a subset /superset of a set S may be smaller or larger than the utility of S

  - This means we cannot use Apriori or FP-growth to find high utility patterns directly since
    - the two algorithms use the downward closure property of support to cut down the search space

  - Solution: use an upper bound of utility with downward closure property to generate candidates first, and then scan DB to find high utility patterns from the set of candidates

Related Topics

- Mining frequent patterns over data streams
  - A continuous flow of data generated often at high-speed in a dynamic, time-changing environment
  - Memory is limited to hold all the data
  - Processing time may be limited by the rate of arrival of instances
  - One scan of data set is required for online mining
  - Pattern changes over time
    - Incremental learning
    - Change detection
    - etc

Related Topics

- Contrast pattern mining
  - Finding patterns and models contrasting two or more classes or conditions
  - Contrasting groups:
    - Objects at different time periods
    - Objects at different spatial locations
    - Objects across different classes.
  - Measures for measuring the difference
    - Frequent/infrequent
    - Frequency ratio
    - Odds ratio, etc.
  - A challenge: need to find infrequent itemsets in a group.

Next Class

- Sequential pattern mining (papers on the supplementary reading list)