Principles of Knowledge Discovery in Data

Winter 2007

Chapter 2: Mining Association Rules

Dr. Osmar R. Zaïane

University of Alberta

Course Content

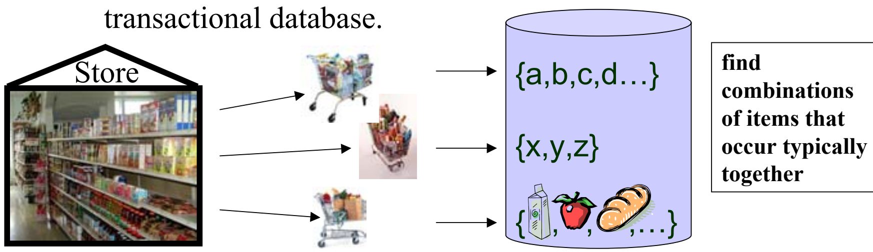
- Introduction to Data Mining
- Association Analysis
- Sequential Pattern Analysis
- Classification and prediction
- Contrast Sets
- Data Clustering
- Outlier Detection
- Web Mining
- Other topics if time permits (spatial data, biomedical data, etc.)

Chapter 6 Objectives

Understand association analysis in large datasets and get a brief introduction to the different types of association rule mining

What Is Association Rule Mining?

- **Association rule mining searches for relationships between items in a dataset:**
  - aims at discovering associations between items in a transactional database.

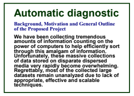
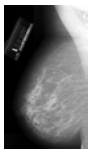


- Rule form: "**Body → Head [support, confidence**

buys(x, "bread") → buys(x, "milk") [0.6%, 65%]

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

Transactional Databases

	Transaction	Frequent itemset	Rule
	{bread, milk, beer,…}	(Bread, milk)	Bread → milk
	{term$_1$, term$_2$,…,term$_n$}	(term$_2$, term$_{25}$)	term2 → term25
	{f1, f2,…,Ca}	(f3, f5, fα)	f3^f5 → fα

Association Rule Mining

- mining association rules (Agrawal et. al SIGMOD93)
- Fast algorithm (Agrawal et. al VLDB94)
- Partitioning (Navathe et. al VLDB95)
- Hash-based (Park et. al SIGMOD95)
- Multilevel A.R. (Han et. al. VLDB95)
- Generalized A.R. (Srikant et. Al. VLDB95)
- Quantitative A.R. (Srikant et. al SIGMOD96)
- Incremental mining (Cheung et. al ICDE96)
- Parallel mining (Agrawal et. al TKDE96)
- Distributed mining (Cheung et. al PDIS96)
- Meta-ruleguided mining (Kamber et al. KDD97)
- Direct Itemset Counting (Brin et. al SIGMOD97)
- N-dimensional A.R. (Lu et. al DMKD'98)
- Constraint A.R. (Ng et. al SIGMOD'98)
- A.R. with recurrent items (Zaïane et. al ICDE'00)
- FP without Candidate gen. (Han et. al SIGMOD'00)
- DualMiner (Bucil, et. al KDD'02)
- COFI algorithm (El-Hajj, et. al Dawak'03)

And many many others:
Spatial AR; Sequence Associations;AR for multimedia; AR in time series;AR with progressive refinement; etc.

Lecture Outline

Part I: Concepts *(30 minutes)*
- Basic concepts
  - Support and Confidence
- Naïve approach

Part II: The Apriori Algorithm *(30 minutes)*
- Principles
- Algorithm
- Running Example

Part III: The FP-Growth Algorithm *(30 minutes)*
- FP-tree structure
- Running Example

Part IV: More Advanced Concepts *(30 minutes)*
- Database layout and space search approach
- Other types of patterns and constraints

Finding Rules in Transaction Data Set

- 6 transactions
- 5 items: {Beer, Bread, Jelly, Milk, PeanutButter}

Transactions	Items
T1	Bread, Jelly, PeanutButter
T2	Bread, PeanutButter
T3	Bread, Milk, PeanutButter
T4	Beer, Bread
T5	Beer, Milk
T6	Bread, Milk

- Searching for rules of the form X→Y, where X and Y are sets of items
  - e.g. Bread → Jelly; Bread, Jelly → PeanutButter
- Design an efficient algorithm for mining association rules in large data sets
- Develop an effective approach for distinguishing interesting rules from irrelevant ones

Basic Concepts

A transaction is a set of items: $T=\{i_a, i_b,\ldots i_t\}$

$T \subset I$, where I is the set of all possible items $\{i_1, i_2,\ldots i_d\}$

D, the task relevant data, is a set of transactions $D=\{T_1, T_2,\ldots T_n\}$.

An association rule is of the form:
$P \rightarrow Q$, where $P \subset I$, $Q \subset I$, and $P \cap Q = \emptyset$

Basic Concepts (con't)

A set of items is referred to as itemset.

An itemset containing k items is called **k-itemset**.
{Jelly, Milk, Bread} is a 3-itemset example

An items set can also be seen as a conjunction of items (or a predicate)

$P \rightarrow Q$ holds in D with support s
and
$P \rightarrow Q$ has a confidence c in the transaction set D.

Support($P \rightarrow Q$) = Probability($P \cup Q$)
Confidence($P \rightarrow Q$) = Probability(Q/P)

Support of an Itemset

- **Support** of $P = P_1 \wedge P_2 \wedge \ldots \wedge P_k$ in $\boldsymbol{D}$ $\sigma(P/D)$ is the probability that P occurs in D: it is the percentage of transactions T in D satisfying P.

- **I.e. the *support of an item (or itemset) X* is the percentage of transactions in which that item (or items) occurs:** (number of T by cardinality of D).

$$\text{support}(X) = \frac{\#X}{n}$$

- **Support for all subsets of items**
  - **Note the exponential growth in the set of items – 5 items: 31 sets**

Transactions	Items
T1	Bread, Jelly, PeanutButter
T2	Bread, PeanutButter
T3	Bread, Milk, PeanutButter
T4	Beer, Bread
T5	Beer, Milk
T6	Bread, Milk

Itemset	Support	Itemset	Support
Beer	33%	Beer, Bread, Milk	0%
Bread	66%	Beer, Bread, PeanutButter	0%
Jelly	16%	Beer, Jelly, Milk	0%
Milk	50%	Beer, Jelly, PeanutButter	0%
PeanutButter	50%	Beer, Milk, PeanutButter	0%
Beer, Bread	16%	Bread, Jelly, Milk	0%
Beer, Jelly	0%	Bread, Jelly, PeanutButter	16%
Beer, Milk	16%	Bread, Milk, PeanutButter	16%
Beer, PeanutButter	0%	Jelly, Milk, PeanutButter	0%
Bread, Jelly	16%	Beer, Bread, Jelly, Milk	0%
Bread, Milk	33%	Beer, Bread, Jelly, PeanutButter	0%
Bread, PeanutButter	50%	Beer, Bread, Milk, PeanutButter	0%
Jelly, Milk	0%	Beer, Jelly, Milk, PeanutButter	0%
Jelly, PeanutButter	16%	Bread, Jelly, Milk, PeanutButter	0%
Milk, PeanutButter	16%	Beer, Bread, Jelly, Milk, PeanutButter	0%
Beer, Bread, Jelly	0%		

Support and Confidence of an Association Rule

- **The *support of an association rule X→Y* is the percentage of transactions that contain X ∪Y**

$$\text{support}(X->Y) = \frac{\#(X \cup Y)}{n}$$

- **The *confidence of an association rule X→Y* is the ratio of the number of transactions that contain X ∪Y to the number of transactions that contain X**

$$\text{confidence}(X->Y) = \frac{\#(X \cup Y)}{\#X}$$

- **Confidence** of a rule $P \rightarrow Q$ in database D $\varphi(P \rightarrow Q/D)$ is the ratio $\sigma((P \wedge Q)/D)$ by $\sigma(P/D)$

$$\text{confidence}(X->Y) = \frac{\text{support}(X->Y)}{\text{support}(X)}$$

Support and Confidence – cont.

- **What is the support and confidence of the following rules?**
  - **Beer→Bread**
  - **{Bread, PeanutButter}→Jelly**

Transactions	Items
T1	Bread, Jelly, PeanutButter
T2	Bread, PeanutButter
T3	Bread, Milk, PeanutButter
T4	Beer, Bread
T5	Beer, Milk
T6	Bread, Milk

- **Support and confidence for some association rules**

Rule	Support	Confidence
Bread → PeanutButter	50%	60%
PeanutButter → Bread	50%	100%
Beer → Bread	16%	50%
PeanutButter → Jelly	16%	33%
Jelly → PeanutButter	16%	100%
Jelly → Milk	0%	0%
{Bread, PeanutButter} → Jelly	16%	33%

Why the difference?

- **Support measures how often the rule occurs in the database.**
- **Confidence measures the strength of the rule.**

Frequent Itemsets and Strong Rules

Support and Confidence are bound by Thresholds:

- *minimum support* σ'
- *minimum confidence* φ'

A **Frequent (or large) itemset** I in D is an itemset with a support larger than the minimum support;
A **strong rule** X→Y is a rule that is frequent (i.e. support higher than minimum support) and its confidence is higher than the minimum confidence threshold.

Association Rule Problem Definition

- Given I={i_1, i_2,…,i_m}, D={t_1, t_2, …,t_n} and the support and confidence thresholds, the *association rule mining problem* is to identify **all strong** association rules X→Y.

Naïve Approach to Generate Association Rules

- **Enumerate all possible rules and select those of them that satisfy the minimum support and confidence thresholds**
- **Not practical for large databases**
  - **For a given dataset with *m* items, the total number of possible rules is $3^m-2^{m+1}+1$**
  - **For our example: 3^5-2^6+1= 180**
  - **More than 80% of these rules are discarded if σ'=0.2 and φ'=0.5**
- **We need a strategy for rule generation - generate only the promising rules**

Better Approach

- Find the *frequent itemsets*: the sets of items that have minimum support
- Use the frequent itemsets to generate association rules. Keep only strong rules.

Frequent Itemset Mining

FIM 1 abc

Bound by a **support** threshold

Association Rules Generation

2 ab→c
b→ac

Bound by a **confidence** threshold

Generating Association Rules from Frequent Itemsets

•Only strong association rules are generated.
•Frequent itemsets satisfy minimum support threshold.
•Strong AR satisfy minimum confidence threshold.

•Confidence(A→B) = Prob(B/A) = $\frac{\text{Support}(A \cup B)}{\text{Support}(A)}$

For each frequent itemset, **f**, generate all non-empty subsets of **f**.
For every non-empty subset **s** of **f** **do**
  output rule **s→(f-s)** if support(**f**)/support(**s**) ≥ min_confidence
end

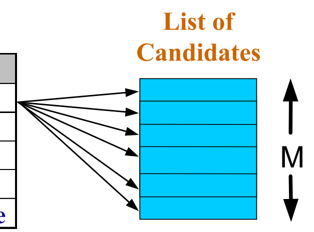
Naïve Frequent Itemset Generation

- Brute-force approach (Basic approach):
  - Each itemset in the lattice is a candidate frequent itemset
  - Count the support of each candidate by scanning the database

Transactions

TID	Items
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

N, w



  - Match each transaction against every candidate
  - Complexity ~ O(NMw) => **Expensive since M = 2^d** !!!

Lecture Outline

Part I: Concepts *(30 minutes)*
- **Basic concepts**
  - **Support and Confidence**
- **Naïve approach**

Part II: The Apriori Algorithm *(30 minutes)*
- **Principles**
- **Algorithm**
- **Running Example**

Part III: The FP-Growth Algorithm *(30 minutes)*
- **FP-tree structure**
- **Running Example**

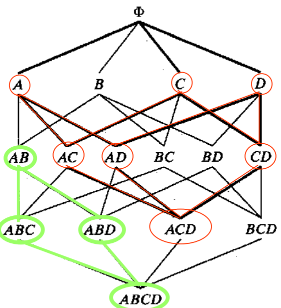
Part IV: More Advanced Concepts *(30 minutes)*
- **Database layout and space search approach**
- **Other types of patterns and constraints**

An Influential Mining Methodology — The *Apriori* Algorithm

- The *Apriori* method:
  - Proposed by Agrawal & Srikant 1994
  - A similar level-wise algorithm by Mannila et al. 1994
- Major idea (*Apriori Principle*):
  - A subset of a frequent itemset must be frequent
    - E.g., if **{beer, diaper, nuts}** is frequent, **{beer, diaper}** must be. Any itemset that is infrequent, its superset cannot be frequent!
  - **A powerful, scalable candidate set pruning technique:**
    - **It reduces candidate k-itemsets dramatically (for k > 2)**

Apriori Algorithm

- Apriori principle:
  - *A subset of any frequent (large) itemset is also frequent*
  - This also implies that *if an itemset is not frequent (small), a superset of it is also not frequent*
    - If we know that an itemset is infrequent, we need not generate any subsets of it as they will be infrequent



- Lines represent "subset" relationship
- If ACD is frequent, than AC,AD,CD,A,C,D are also frequent, i.e. if an itemset is frequent than any set in a path above it is also frequent
- If AB is infrequent, than ABC, ABD, ABCD will also be infrequent, i.e. if an itemset is infrequent than any set in the path below is also infrequent
- If any of A, C, D, AC, AD, CD, is infrequent than ACD is infrequent (no need to check).

Mining Association rules: the Key Steps

- 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 frequent itemsets
  - Iteratively find frequent itemsets with cardinality from 1 to k (k-itemsets)
- Use the frequent itemsets to generate strong association rules.

Apriori Algorithm – Idea

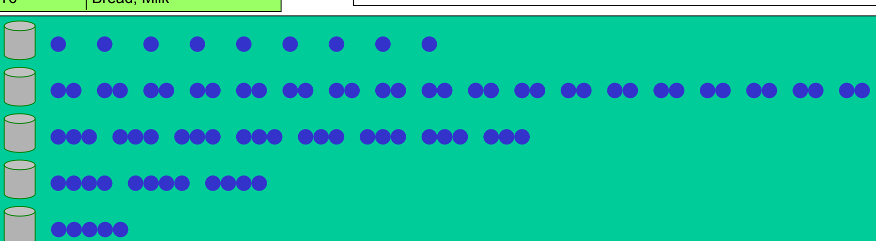
- 1. Generate candidate itemsets of a particular size
- 2. Scan the database to see which of them are frequent
  - An itemset is frequent if all its subsets are frequent
- 3. Use only these frequent itemsets to generate the set of candidates with size=size+1

For our example if σ'=50%

Transactions	Items
T1	Bread, Jelly, PeanutButter
T2	Bread, PeanutButter
T3	Bread, Milk, PeanutButter
T4	Beer, Bread
T5	Beer, Milk
T6	Bread, Milk

itemset size

Pass	Candidates	Frequent itemsets
1	{Beer}, {Bread}, {Jelly}, {Milk}, {PeanutButter}	{Bread}(66%), {Milk}(50%) {PeanutButter}(50%)
2	{Bread, Milk}, {Bread, PeanutButter} {Milk, PeanutButter}	{Bread, PeanutButter}(50%)



The Apriori Algorithm

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 $\cup_k L_k$;

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

Support > 1

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

Note: {1,2,3} {1,2,5} and {1,3,5} not in C_3

Apriori-Gen Algorithm – Clothing Example

- **Given: 20 clothing transactions; *s*=20%, c=50%**
- **Generate association rules using the Apriori algorithm**

Transaction	Items	Transaction	Items
t_1	Blouse	t_{11}	TShirt
t_2	Shoes, Skirt, TShirt	t_{12}	Blouse, Jeans, Shoes, Skirt, TShirt
t_3	Jeans, TShirt	t_{13}	Jeans, Shoes, Shorts, TShirt
t_4	Jeans, Shoes, TShirt	t_{14}	Shoes, Skirt, TShirt
t_5	Jeans, Shorts	t_{15}	Jeans, TShirt
t_6	Shoes, TShirt	t_{16}	Skirt, TShirt
t_7	Jeans, Skirt	t_{17}	Blouse, Jeans, Skirt
t_8	Jeans, Shoes, Shorts, TShirt	t_{18}	Jeans, Shoes, Shorts, TShirt
t_9	Jeans	t_{19}	Jeans
t_{10}	Jeans, Shoes, TShirt	t_{20}	Jeans, Shoes, Shorts, TShirt

- **Scan1: Find all 1-itemsets. Identify the frequent ones.**

Candidates: ~~Blouse~~, Jeans, Shoes, Shorts, Skirt, Tshirt

Support: ~~3/20~~ 14/20 10/20 5/20 6/20 14/20

Frequent (Large): Jeans, Shoes, Shorts, Skirt, Tshirt

Join the frequent items – combine items with each other to generate candidate pairs

Clothing Example – cont.1

- **Scan2: 10 candidate 2-itemsets were generated. Find the frequent ones.**

{Jeans, Shoes}:**7/20** {Shoes, Short}:**4/20** ~~{Short, Skirt}: 0/20~~ {Skirt, TShirt}: **4/20**

{Jeans, Short} :**5/20** ~~{Shoes, Skirt}: 3/20~~ {Short, TShirt}: **4/20**

~~{Jeans, Skirt}~~ :**3/20** {Shoes, TShirt}: **10/20**

{Jeans, TShirt}:**9/20 4/20** ***7 frequent itemsets are found out of 10.***

Scan	Candidates	Large Itemsets
1	{Blouse}, {Jeans}, {Shoes}, {Shorts}, {Skirt}, {TShirt}	{Jeans}, {Shoes}, {Shorts} {Skirt}, {Tshirt}
2	{Jeans, Shoes}, {Jeans, Shorts}, {Jeans, Skirt}, {Jeans, TShirt}, {Shoes, Shorts}, {Shoes, Skirt}, {Shoes, TShirt}, {Shorts, Skirt}, {Shorts, TShirt}, {Skirt, TShirt}	{Jeans, Shoes}, {Jeans, Shorts}, {Jeans, TShirt}, {Shoes, Shorts}, {Shoes, TShirt}, {Shorts, TShirt}, {Skirt, TShirt}
3	{Jeans, Shoes, Shorts}, {Jeans, Shoes, TShirt}, {Jeans, Shorts, TShirt}, {Jeans, Skirt, TShirt}, {Shoes, Shorts, TShirt}, {Shoes, Skirt, TShirt}, {Shorts, Skirt, TShirt}	{Jeans, Shoes, Shorts}, {Jeans, Shoes, TShirt}, {Jeans, Shorts, TShirt}, {Shoes, Shorts, TShirt}
4	{Jeans, Shoes, Shorts, TShirt}	{Jeans, Shoes, Shorts, TShirt}
5	Ø	Ø

Everyone is combined with each other

2 sets are joined if they have 1 item in common (i,.e. 1 item different)

2 sets are joined if they have 2 item in common (i,.e. 1 item different)

Clothing Example – cont.2

- **The next step is to use the large itemsets and generate association rules**
- **c=50%**
- **The set of large itemsets is**

L={{Jeans},{Shoes}, {Shorts}, {Skirt}, {TShirt}, {Jeans, Shoes}, {Jeans, Shorts}, {Jeans, TShirt}, {Shoes, Shorts}, {Shoes, TShirt}, {Shorts, TShirt}, {Skirt, TShirt}, {Jeans, Shoes, Shorts}, {Jeans, Shoes, TShirt}, {Jeans, Shorts, TShirt},{Shoes, Shorts, TShirt}, {Jeans, Shoes, Shorts,TShirt} }

- **We ignore the first 5 as they do not consists of 2 nonempty subsets of large itemsets. We test all the others, e.g.:**

$$confidence(Jeans->Shoes) = \frac{\text{support}(\{Jeans, Shoes\})}{\text{support}(\{Jeans\})} = \frac{7/20}{14/20} = 50\% \geq c$$

etc.

Lecture Outline

Part I: Concepts *(30 minutes)*

- **Basic concepts**
  - **Support and Confidence**
- **Naïve approach**

Part II: The Apriori Algorithm *(30 minutes)*

- **Principles**
- **Algorithm**
- **Running Example**

Part III: The FP-Growth Algorithm *(30 minutes)*

- **FP-tree structure**
- **Running Example**

Part IV: More Advanced Concepts *(30 minutes)*

- **Database layout and space search approach**
- **Other types of patterns and constraints**

Problems with Apriori

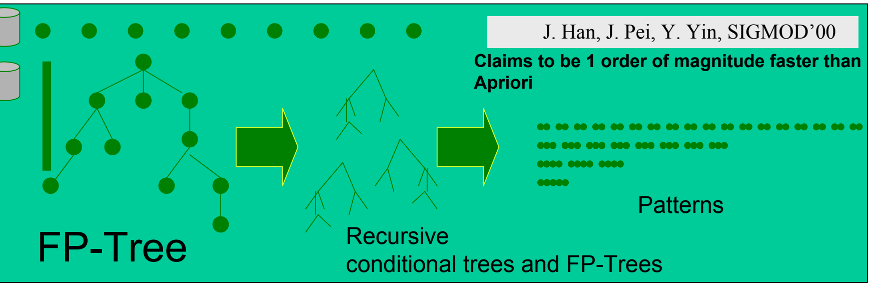
- Generation of candidate itemsets are expensive (Huge candidate sets)
  - 10^4 frequent 1-itemset will generate 10^7 candidate 2-itemsets
  - To discover a frequent pattern of size 100, e.g., $\{a_1, a_2, \ldots, a_{100}\}$, one needs to generate $2^{100} \approx 10^{30}$ candidates.
- High number of data scans

Frequent Pattern Growth

- First algorithm that allows frequent pattern mining without generating candidate sets
- Requires Frequent Pattern Tree

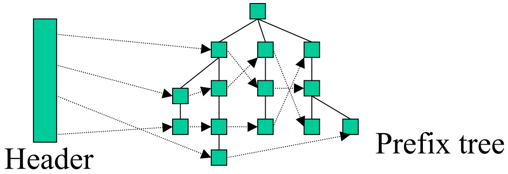
FP-Growth

- Grow long patterns from short ones using local frequent items
  - "abc" is a frequent pattern
  - Get all transactions having "abc": DB|abc
  - "d" is a local frequent item in DB|abc → abcd is a frequent pattern

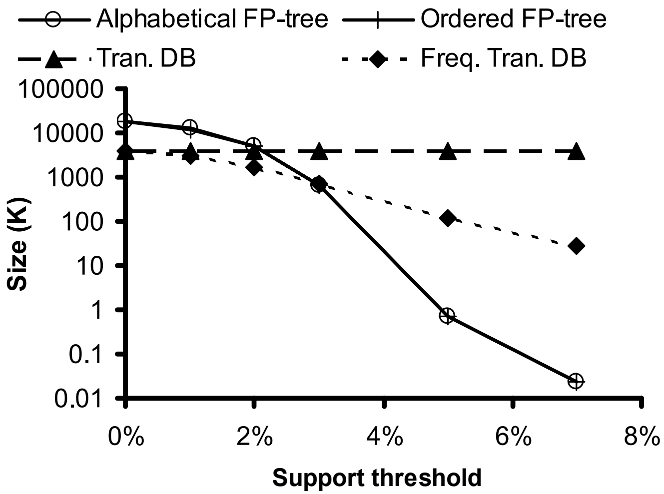


Frequent Pattern Tree

- Prefix tree.
- Each node contains the item name, frequency and pointer to another node of the same kind.
- Frequent item header that contains item names and pointer to the first node in FP tree.



Database Compression Using FP-tree (on T10I4D100k)



Frequent Pattern Tree

F, A, C, D, G, I, M, P
A, B, C, F, L, M, O
B, F, H, J, O
A, F, C, E, L, P, M, N
B, C, K, S, P
F, M, C, B, A

Required Support: **3**

F:5, C:5, A:4, B:4, M:4, P:3 ~~D:1 E:1 G:1 H:1 I:1 J:1 K:1 L:1 O:1~~

Frequent Pattern Tree

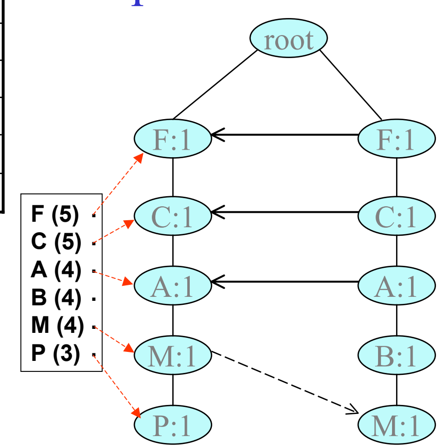
Original Transaction	Ordered frequent items
F, A, C, D, G, I, M, P	F, C, A, M, P
A, B, C, F, L, M, O	F, C, A, B, M
B, F, H, J, O	F, B
A, F, C, E, L, P, M, N	C, B, P
B, C, K, S, P	F, C, A, M, P
F, M, C, B, A	F, C, A, M
F, B, D	F, B

F:5, C:5, A:4, B:4, M:4, P:3

Required Support: **3**

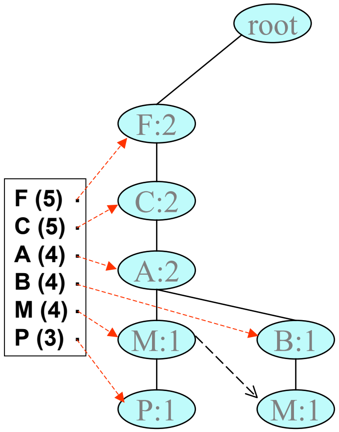
Frequent Pattern Tree

F, C, A, M, P
F, C, A, B, M
F, B
C, B, P
F, C, A, M, P
C, A, M
F, B



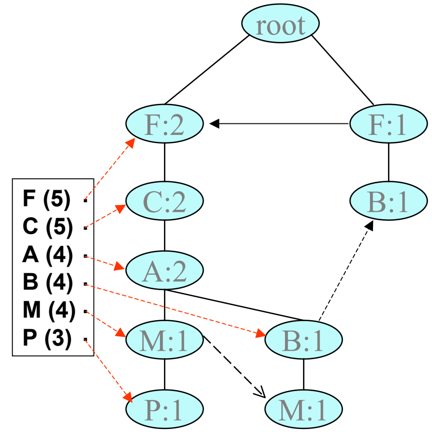
Frequent Pattern Tree

F, C, A, M, P
F, C, A, B, M
F, B
C, B, P
F, C, A, M, P
C, A, M
F, B



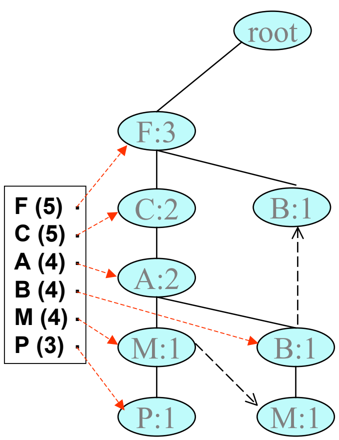
Frequent Pattern Tree

F, C, A, M, P
F, C, A, B, M
F, B
C, B, P
F, C, A, M, P
C, A, M
F, B



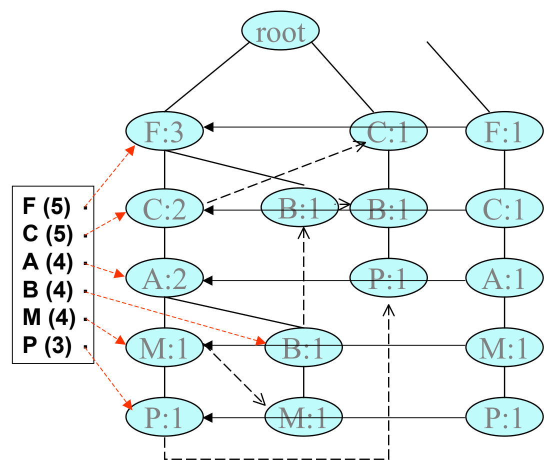
Frequent Pattern Tree

F, C, A, M, P
F, C, A, B, M
F, B
C, B, P
F, C, A, M, P
C, A, M
F, B



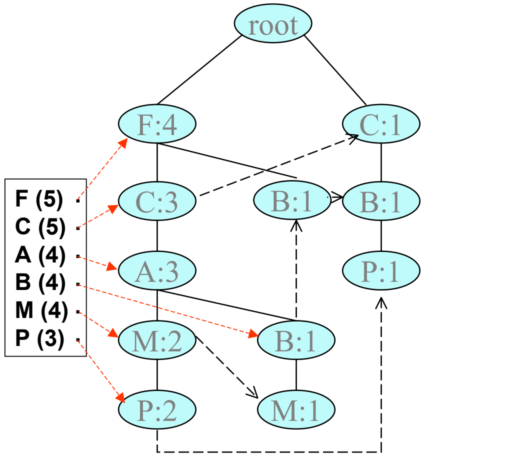
Frequent Pattern Tree

F, C, A, M, P
F, C, A, B, M
F, B
C, B, P
F, C, A, M, P
C, A, M
F, B



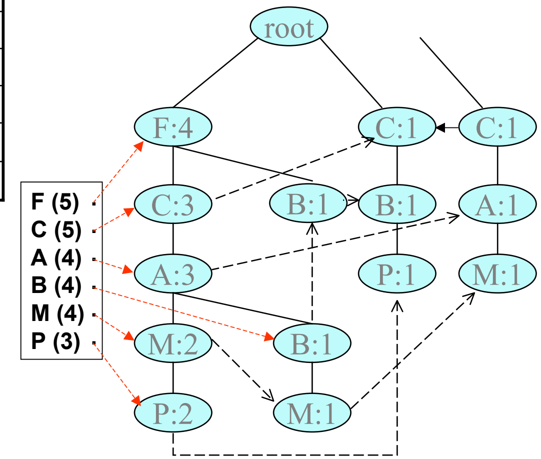
Frequent Pattern Tree

F, C, A, M, P
F, C, A, B, M
F, B
C, B, P
F, C, A, M, P
C, A, M
F, B



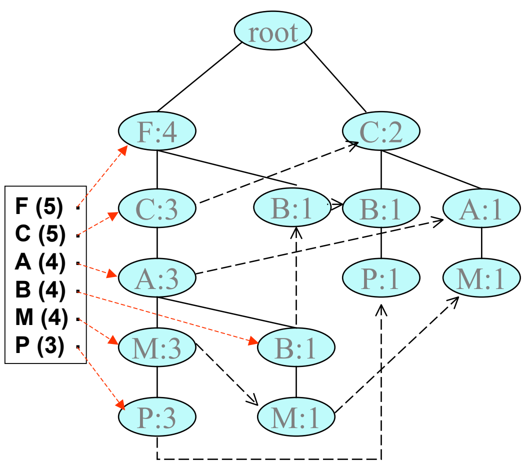
Frequent Pattern Tree

F, C, A, M, P
F, C, A, B, M
F, B
C, B, P
F, C, A, M, P
C, A, M
F, B



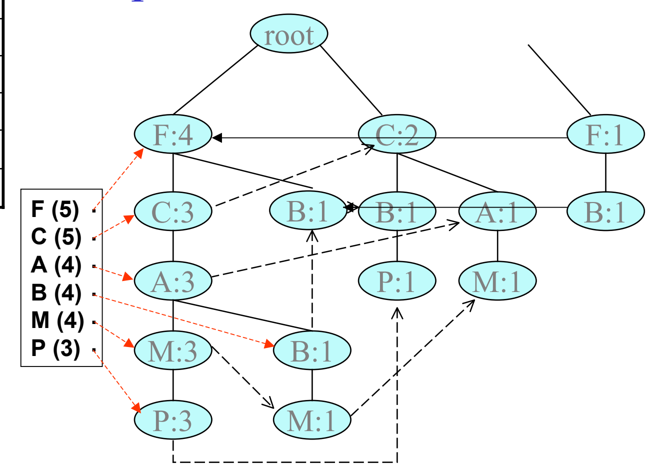
Frequent Pattern Tree

F, C, A, M, P
F, C, A, B, M
F, B
C, B, P
F, C, A, M, P
C, A, M
F, B



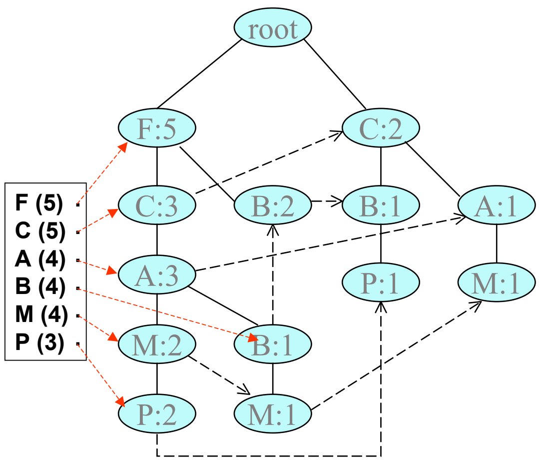
Frequent Pattern Tree

F, C, A, M, P
F, C, A, B, M
F, B
C, B, P
F, C, A, M, P
C, A, M
F, B



Frequent Pattern Tree

F, C, A, M, P
F, C, A, B, M
F, B
C, B, P
F, C, A, M, P
C, A, M
F, B



Mining Frequent Patterns with FP-Tree 3 Major Steps

Starting the processing from the end of list L:

Step 1:

Construct conditional pattern base for each item in the header table

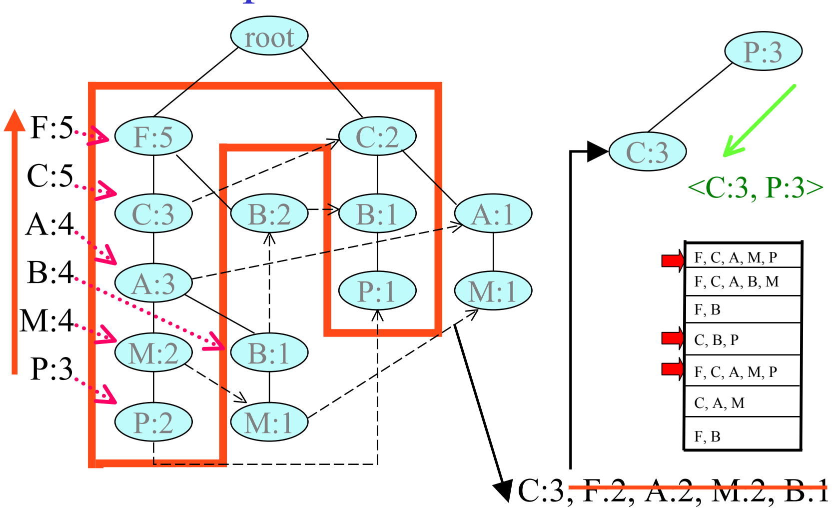
Step 2

Construct conditional FP-tree from each conditional pattern base

Step 3

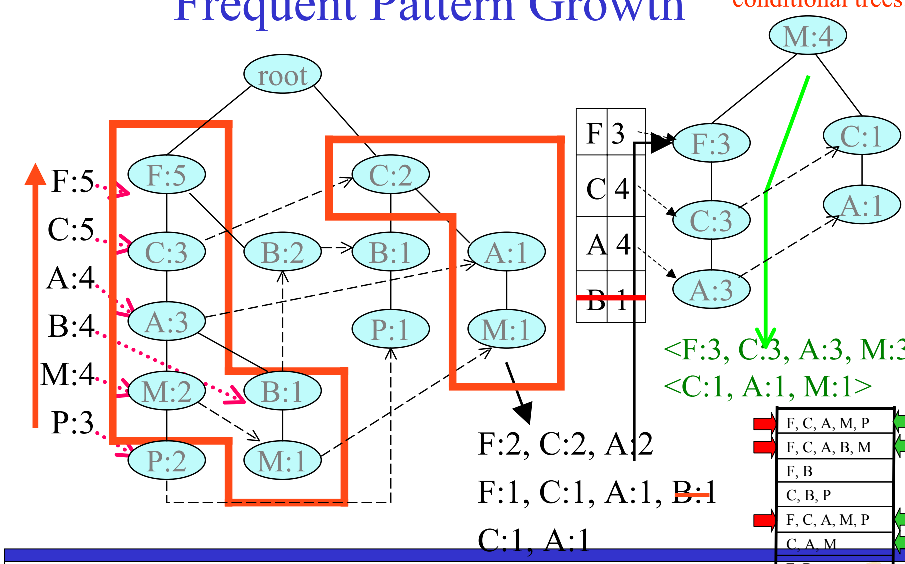
Recursively mine conditional FP-trees and grow frequent patterns obtained so far. If the conditional FP-tree contains a single path, simply enumerate all the patterns

Frequent Pattern Growth



Frequent Pattern Growth

Recursively build the A, C and F conditional trees



Another Example: Construct FP-tree from a Transaction Database

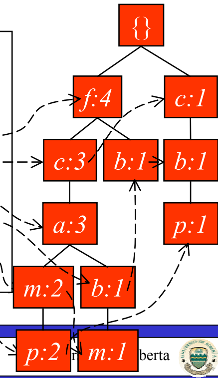
TID	*Items bought*	*(ordered) frequent items*
100	{*f, a, c, d, g, i, m, p*}	{*f, c, a, m, p*}
200	{*a, b, c, f, l, m, o*}	{*f, c, a, b, m*}
300	{*b, f, h, j, o, w*}	{*f, b*}
400	{*b, c, k, s, p*}	{*c, b, p*}
500	{*a, f, c, e, l, p, m, n*}	{*f, c, a, m, p*}

min_support = 3

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

Header Table

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



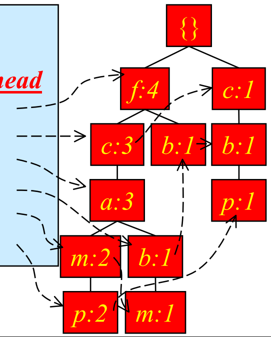
F-list=f-c-a-b-m-p

Step 1: Construct Conditional Pattern Base

- Starting at the frequent-item 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*
p	*fcam:2, cb:1*
m	*fca:2, fcab:1*
b	*fca:1, f:1, c:1*
a	*fc:3*
c	*f:3*

Properties of Step 1

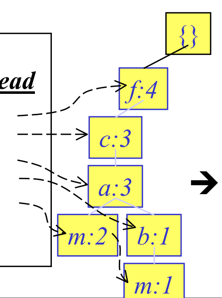
- Node-link property
  - For any frequent item a_i, all the possible frequent patterns that contain a_i can be obtained by following a_i's node-links, starting from a_i's head in the FP-tree header.
- Prefix path property
  - To calculate the frequent patterns for a node a_i in a path P, only the prefix 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

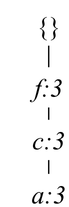
- For each pattern base
  - Accumulate the count for each item in the base
  - Construct the conditional FP-tree for the frequent items of the pattern base

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 ➔



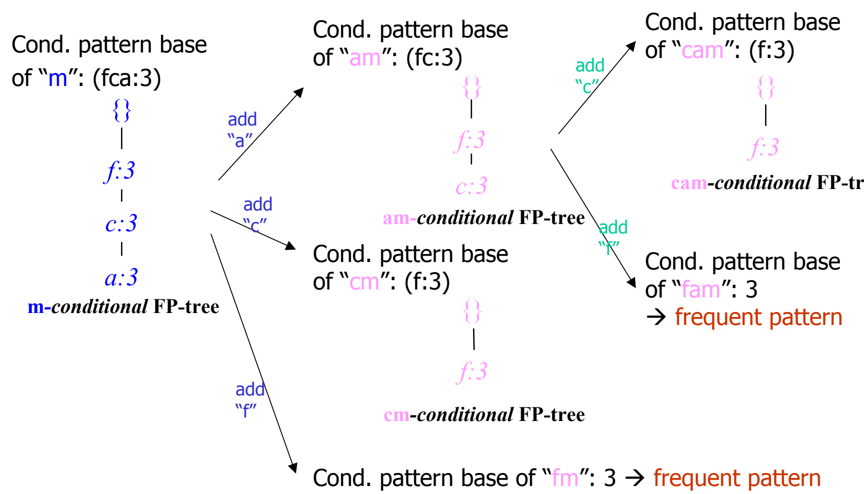
***m-conditional* FP-tree**

Conditional Pattern Bases and Conditional FP-Tree

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

order of L

Step 3: Recursively mine the conditional FP-tree

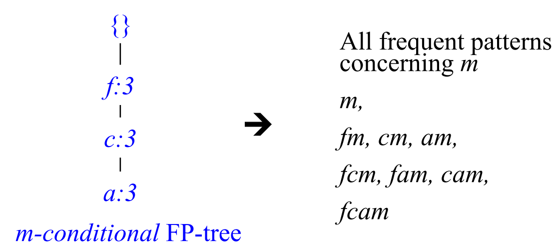


Principles of FP-Growth

- 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.
- Is "*fcabm* " a frequent pattern?
  - "fcab" is a branch of m's conditional pattern base
  - "b" is **NOT** frequent in transactions containing "fcab"
  - "bm" is **NOT** a frequent itemset.

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



Discussion (1/2)

- **Association rules are typically sought for very large databases → efficient algorithms are needed**
- **The Apriori algorithm makes 1 pass through the dataset for each different itemset size**
  - **The maximum number of database scans is k+1, where k is the cardinality of the largest large itemset (4 in the clothing ex.)**
  - **potentially large number of scans – weakness of Apriori**
- **Sometimes the database is too big to be kept in memory and must be kept on disk**
- **The amount of computation also depends on the min.support; the confidence has less impact as it does not affect the number of passes**
- **Variations**
  - **Using sampling of the database**
  - **Using partitioning of the database**
  - **Generation of incremental rules**

Discussion (2/2)

- Choice of minimum support threshold
  - lowering support threshold results in more frequent itemsets
  - this may increase number of candidates and max length of frequent itemsets
- Dimensionality (number of items) of the data set
  - more space is needed to store support count of each item
  - if number of frequent items also increases, both computation and I/O costs may also increase
- Size of database
  - since Apriori makes multiple passes, run time of algorithm may increase with number of transactions
- Average transaction width
  - transaction width increases with denser data sets
  - This may increase max length of frequent itemsets and traversals of hash tree (number of subsets in a transaction increases with its width)

Lecture Outline

Part I: Concepts *(30 minutes)*
- **Basic concepts**
  - **Support and Confidence**
- **Naïve approach**

Part II: The Apriori Algorithm *(30 minutes)*
- **Principles**
- **Algorithm**
- **Running Example**

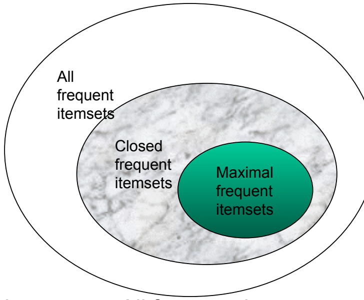
Part III: The FP-Growth Algorithm *(30 minutes)*
- **FP-tree structure**
- **Running Example**

Part IV: More Advanced Concepts *(30 minutes)*
- **Database layout and space search approach**
- **Other types of patterns and constraints**

Other Frequent Patterns

- Frequent pattern $\{a_1, \ldots, a_{100}\} \rightarrow \binom{100}{1} + \binom{100}{2} + \ldots + \binom{100}{100} = 2^{100}-1 = 1.27*10^{30}$ frequent sub-patterns!

- Frequent Closed Patterns
- Frequent Maximal Patterns
- All Frequent Patterns



Maximal frequent itemsets ⊆ Closed frequent itemsets ⊆ All frequent itemset

Frequent Closed Patterns

- For frequent itemset X, if there exists no item y such that every transaction containing X also contains y, then X is a frequent closed pattern
- In other words, frequent itemset X is closed if there is no item y, not already in X, that always accompanies X in all transactions where X occurs.
- Concise representation of frequent patterns. Can generate all frequent patterns with their support from frequent closed ones.
- Reduce number of patterns and rules
- N. Pasquier et al. In ICDT'99

{abcd}
{abc}
{bd}
Transactions
Support = 2

a	2
b	3
c	2
d	2
ab	2
ac	2
bc	2
bd	2
abc	2

Frequent itemsets

b	3
bd	2
abc	2

Frequent Closed itemsets

Frequent Maximal Patterns

- Frequent itemset X is maximal if there is no other frequent itemset Y that is superset of X.
- In other words, there is no other frequent pattern that would include a maximal pattern.
- More concise representation of frequent patterns but the information about supports is lost.
- Can generate all frequent patterns from frequent maximal ones but without their respective support.
- R. Bayardo. In SIGMOD'98

{abcd}
{abc}
{bd}
Transactions
Support = 2

a	2
b	3
c	2
d	2
ab	2
ac	2
bc	2
bd	2
abc	2

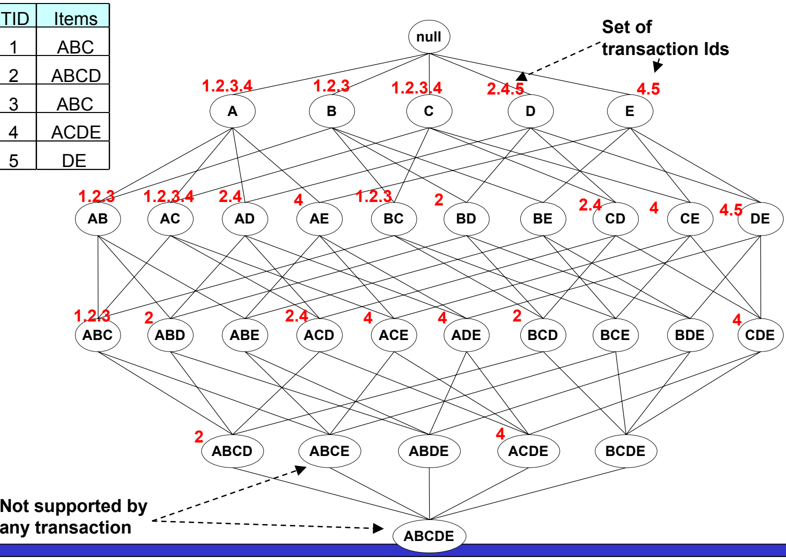
Frequent itemsets

bd	2
abc	2

Frequent Maximal itemse

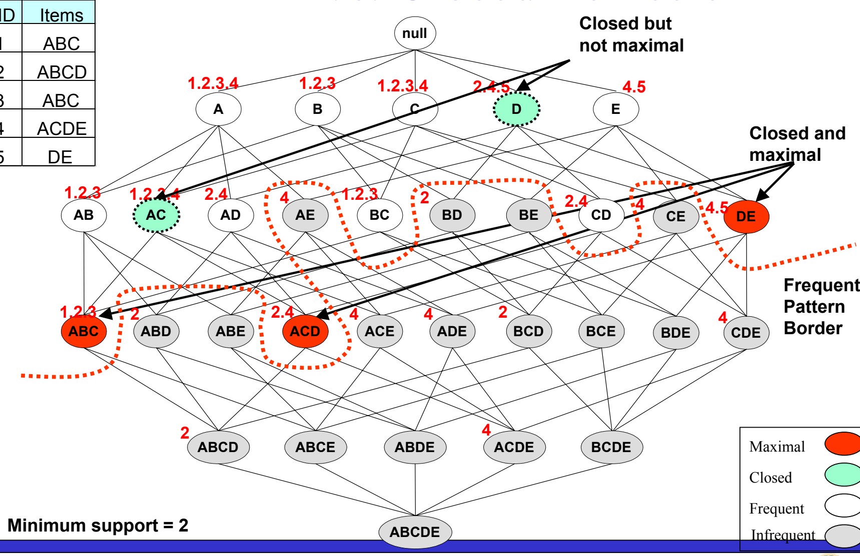
Maximal vs. Closed Itemsets

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



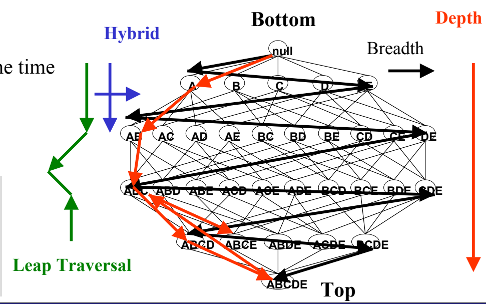
Maximal vs. Closed Itemsets

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



Mining the Pattern Lattice

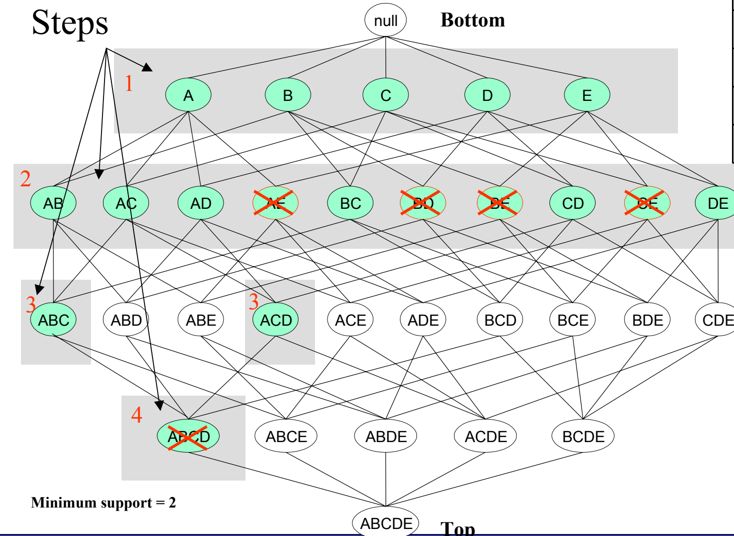
- Breadth-First
  - It uses current items at level k to generate items of level k+1 (many database scans)
- Depth-First
  - It uses a current item at level k to generate all its supersets (favored when mining long frequent patterns)
- Hybrid approach
  - It mines using both direction at the same time
- Leap traversal approach
  - Jumps to selected nodes



There is also the notion of :
Top-down (level k then level k+1)
Bottom-up (level k+1 then level k)

Breadth- First (Bottom-Up Example)

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

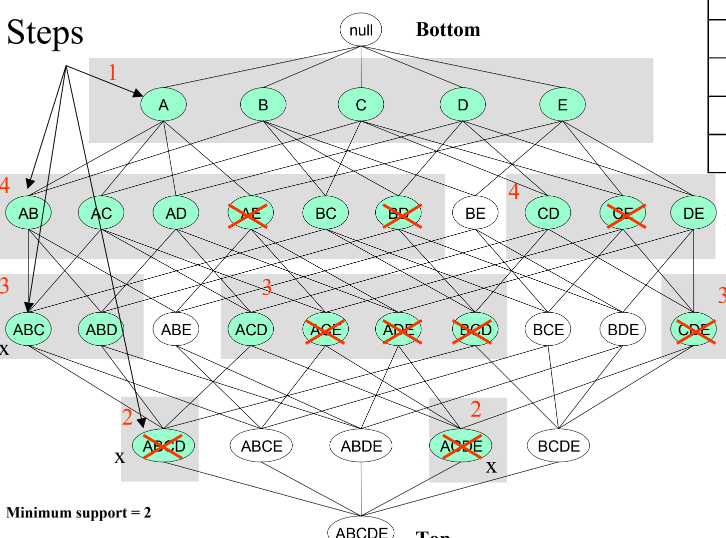


Superset is candidate if ALL its subset are frequent

18 candidate to check

Depth First (Top-Down Example)

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

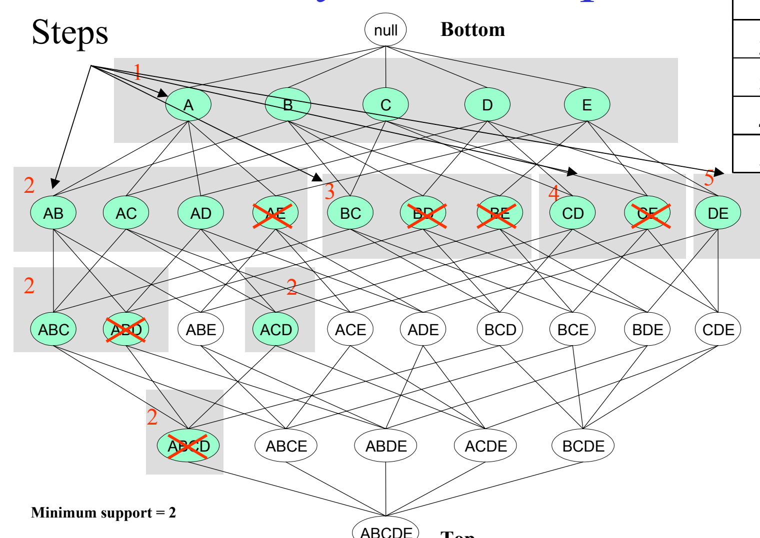


Subset is candidate if it is marked or if one of its supersets is candidate

23 candidates to check

One Hybrid Example

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

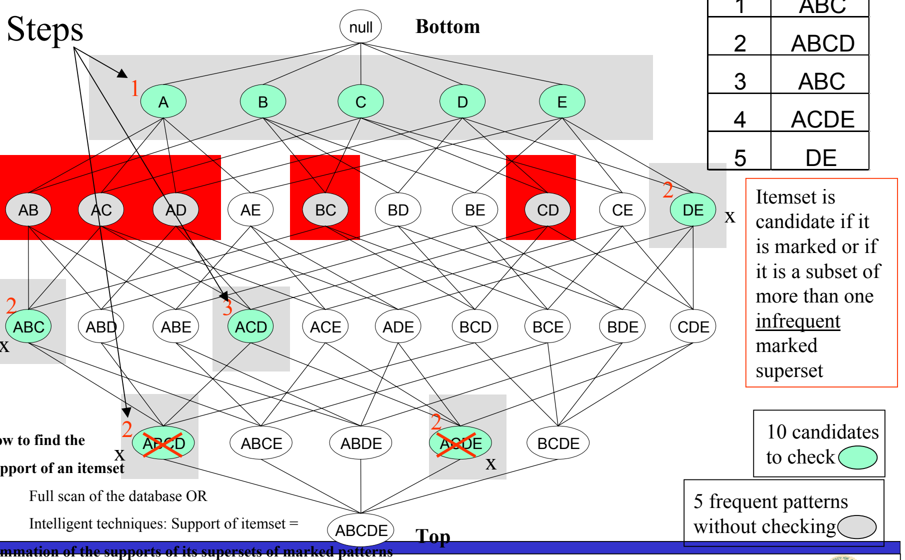


Superset is candidate if ALL its subset are frequent

19 candidate to check

Leap Traversal Example

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



Constraint-based Data Mining

- Finding all the patterns in a database autonomously? — unrealistic!
  - The patterns could be too many but not focused!
- Data mining should be an interactive process
  - User directs what to be mined using a data mining query language (or a graphical user interface)
- Constraint-based mining
  - User flexibility: provides constraints on what to be mined
  - System optimization: explores such constraints for efficient mining—**constraint-based mining**

Restricting Association Rules

• Useful for interactive and ad-hoc mining
• Reduces the set of association rules discovered and confines them to more relevant rules.
• **Before mining**
✓ Knowledge type constraints: classification, etc.
✓ Data constraints: SQL-like queries (DMQL)
✓ Dimension/level constraints: relevance to some dimensions and some concept levels.
• **While mining**
✓ Rule constraints: form, size, and content.
✓ Interestingness constraints: support, confidence, correlation.
• **After mining**
✓ Querying association rules

Constrained Frequent Pattern Mining: A Mining Query Optimization Problem

- Given a frequent pattern mining query with a set of constraints C, the algorithm should be
  - sound: it only finds frequent sets that satisfy the given constraints *C*
  - complete: all frequent sets satisfying the given constraints *C* are found
- A naïve solution
  - First find all frequent sets, and then test them for constraint satisfaction
- More efficient approaches:
  - Analyze the properties of constraints comprehensively
  - Push them as deeply as possible inside the frequent pattern computation.

Rule Constraints in Association Mining

- Two kind of rule constraints:
  - Rule form constraints: meta-rule guided mining.
    - P(x, y) ^ Q(x, w) –> takes(x, "database systems").
  - Rule content constraint: constraint-based query optimization (where and having clauses) (Ng, et al., SIGMOD'98).
    - sum(LHS) < 100 ^ min(LHS) > 20 ^ count(LHS) > 3 ^ sum(RHS) > 1000
- **1-variable vs. 2-variable constraints**

(Lakshmanan, et al. SIGMOD'99):

  - 1-var: A constraint confining only one side (L/R) of the rule, e.g., as shown above.
  - 2-var: A constraint confining both sides (L and R).
    - sum(LHS) < min(RHS) ^ max(RHS) < 5* sum(LHS)

Anti-Monotonicity in Constraint-Based Mining

TDB (min_sup=2)

TID	Transaction
10	a, b, c, d, f
20	b, c, d, f, g, h
30	a, c, d, e, f
40	c, e, f, g

Item	Profit
a	40
b	0
c	-20
d	10
e	-30
f	30
g	20
h	-10

- Anti-monotonicity
  - *When an intemset S **violates** the constraint, so does any of its supersets*
  - *sum(S.Price)* ≤ *v* is anti-monotone
  - *sum(S.Price)* ≥ *v* is not anti-monotone
- Example. C: range(S.profit) ≤ 15 is anti-monotone
  - Itemset *ab* violates C
  - So does every superset of *ab*

Monotonicity in Constraint-Based Mining

TDB (min_sup=2)

TID	Transaction
10	a, b, c, d, f
20	b, c, d, f, g, h
30	a, c, d, e, f
40	c, e, f, g

Item	Profit
a	40
b	0
c	-20
d	10
e	-30
f	30
g	20
h	-10

- Monotonicity
  - *When an intemset S **satisfies** the constraint, so does any of its supersets*
  - *sum(S.Price)* ≥ *v* is monotone
  - *min(S.Price)* ≤ *v* is monotone
- Example. C: range(S.profit) ≥ 15
  - Itemset *ab* satisfies C
  - So does every superset of *ab*

Which Constraints Are Monotone or Anti-Monotone?

SQL-based Constraints

Constraint	Monotone	Anti-Monotone
v ∈ S	yes	no
S ⊇ V	yes	no
S ⊆ V	no	yes
min(S) ≤ v	yes	no
min(S) ≥ v	no	yes
max(S) ≤ v	no	yes
max(S) ≥ v	yes	no
count(S) ≤ v	no	yes
count(S) ≥ v	yes	no
sum(S) ≤ v (a ∈ S, a ≤ 0)	no	yes
sum(S) ≥ v (a ∈ S, a ≤ 0)	yes	no
range(S) ≤ v	no	yes
range(S) ≥ v	yes	no
support(S) ≥ ξ	no	yes
support(S) ≤ ξ	yes	no

State Of The Art

- Constraint pushing techniques have been proven to be effective in reducing the explored portion of the search space in constrained frequent pattern mining tasks.
- Anti-monotone constraints:
  - Easy to push …
  - Always profitable to do …

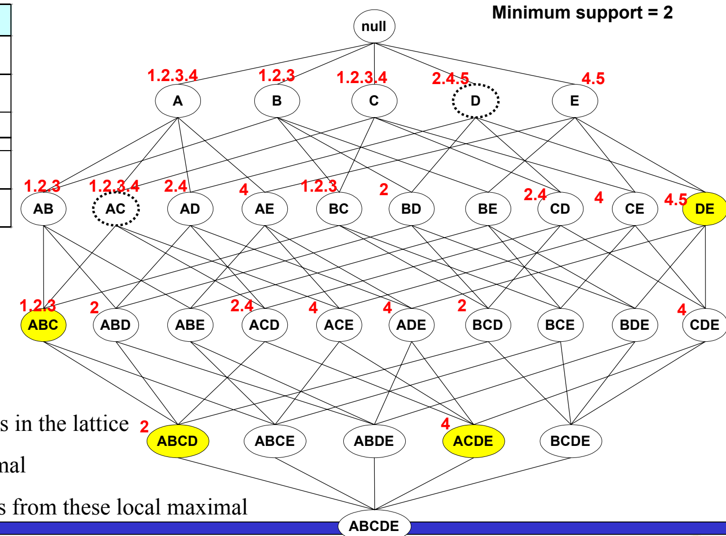
FP-Growth with Constraints:
J. Pei, J. Han, L. Lakshmanan, ICDE'01

- Monotone constraints:
  - Hard to push …
  - Should we push them, or not?

• Dual Miner: C. Bucil, J. Gherke, D. Kiefer and W. White, SIGKDD'02
• FP-Bonsai: F. Bonchi anf B. Goethals, PAKDD'04
• COFI with constraints: M. El-Hajj and O. Zaiane, AI'05
• BifoldLeap: M. El-Hajj and O. Zaiane, ICDM'05

Finding Maximal using leap traversal approach

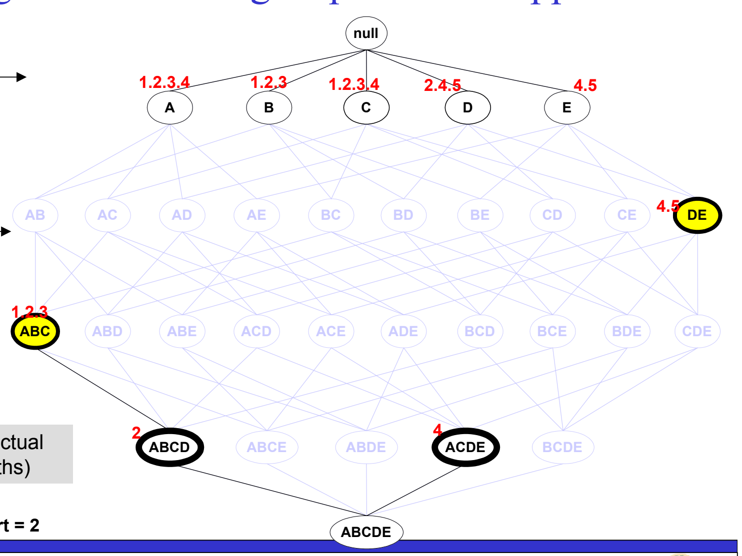
TID	Items
1,3	ABC
2	ABCD
~~3~~	~~ABC~~
4	ACDE
5	DE



-Selectively jumps in the lattice
-Find local Maximal
-Generate patterns from these local maximal

Finding Maximal using leap traversal approach

TID	Items
~~1,3~~	~~ABC~~
2	ABCD
4	ACDE
~~5~~	~~DE~~



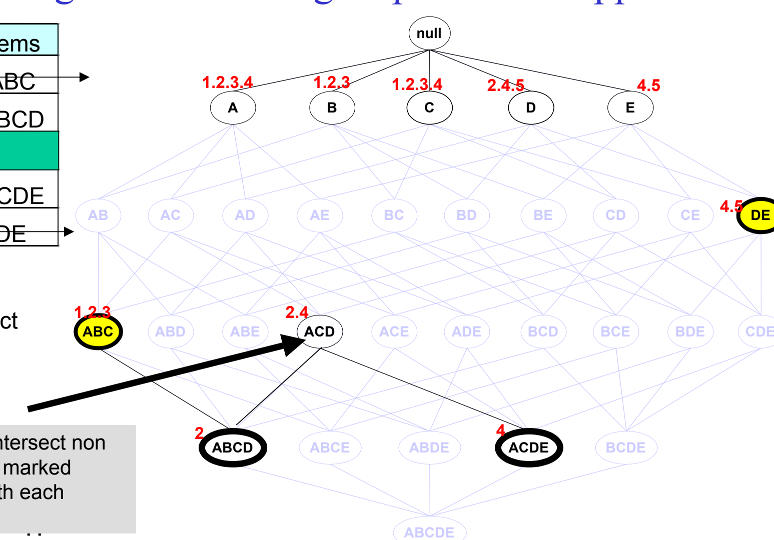
Step1: Define actual paths (Mark paths)

Minimum support = 2

Finding Maximal using leap traversal approach

TID	Items
~~1,3~~	~~ABC~~
2	ABCD
4	ACDE
~~5~~	~~DE~~

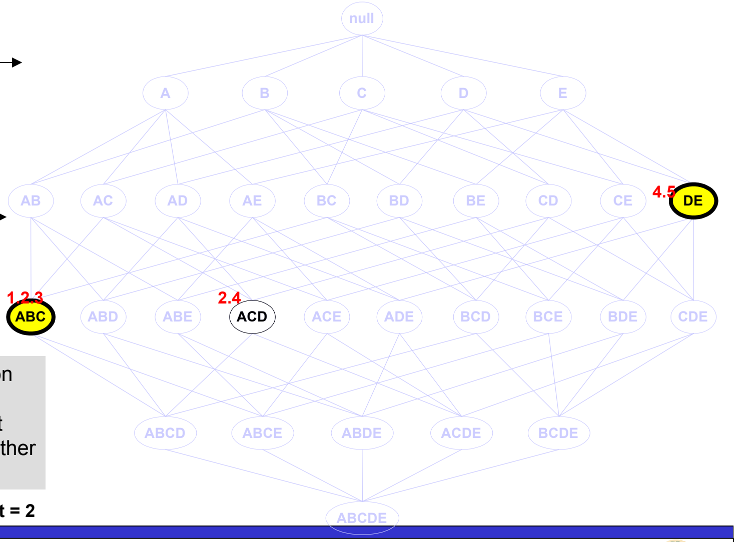
Intersect ABCD with ACDE



Step2: Intersect non frequent marked paths with each others

Finding Maximal using leap traversal approach

TID	Items
1,3	~~ABC~~
2	ABCD
4	ACDE
5	~~DE~~

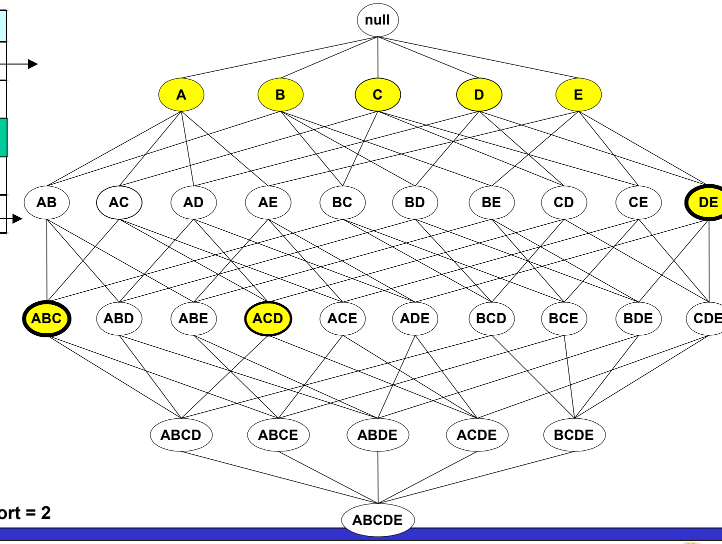


Step3: Remove non frequent paths, or frequent paths that have superset of other frequent paths

Minimum support = 2

Finding Maximal using leap traversal approach

TID	Items
1,3	~~ABC~~
2	ABCD
4	ACDE
5	~~DE~~



Minimum support = 2

When to use a Given Strategy

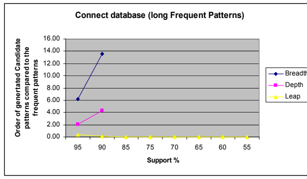
- Breadth First
  - Suitable for short frequent patterns
  - Unsuitable for long frequent patterns
- Depth First
  - Suitable for long frequent patterns
  - In general not scalable when long candidate patterns are not frequent
- Leap Traversal
  - Suitable for cases having short and long frequent patterns simultaneously

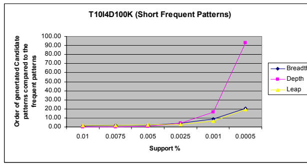
Empirical Tests

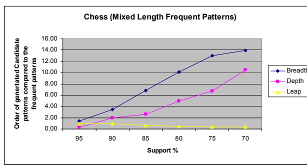
Connect				Total Candidates Created		
Support	Size of Largest	F1-Size	Total Frequent	Breadth	Depth	Leap
95	9	17	2205	15626	6654	3044
90	12	21	27127	394648	144426	29263
80	15	28	119418			120177
75	17	30	176365			177244
70	18	31	627952			628963
65	19	33	1368337			1369585
60	20	36	2908632			2910175
55	21	37	5996892			5998715

T10I4D100K				Total Candidates Created		
Support	Size of Largest	F1-Size	Total Frequent	Breadth	Depth	Leap
1000	3	375	385	20	10	29
750	4	463	561	262	124	291
500	5	569	1073	1492	731	1546
250	8	717	7703	36994	36309	26666
100	10	797	27532	264645	458275	200113
50	10	839	53385	1129779	4923723	1067050

Chess				Total Candidates Created		
Support	Size of Largest	F1-Size	Total Frequent	Breadth	Depth	Leap
95	5	9	78	165	90	136
90	7	13	628	2768	1842	1191
85	8	16	2690	20871	9667	4318
80	10	20	8282	91577	49196	12021
75	11	23	20846	292363	160362	28986
70	13	24	48939	731740	560103	65093







Transactional Layouts

• Horizontal Layout

Each transaction is recorded as a list of items

Transaction ID	Items				
1	A	G	D	C	B
2	B	C	H	E	D
3	B	D	E	A	M
4	C	E	F	A	N
5	A	B	N	O	P
6	A	C	Q	R	G
7	A	C	H	I	G
8	L	E	F	K	B
9	A	F	M	N	O
10	C	F	P	J	R
11	A	D	B	H	I
12	D	E	B	K	L
13	M	D	C	G	O
14	C	F	P	Q	J
15	B	D	E	F	I
16	J	E	B	A	D
17	A	K	E	F	C
18	C	D	L	B	A

Candidacy generation can be removed (FP-Growth)

Superfluous Processing

Transactional Layouts

• Vertical Layout

Tid-list is kept for each item

Transaction ID	Items				
1	A	G	D	C	B
2	B	C	H	E	D
3	B	D	E	A	M
4	C	E	F	A	N
5	A	B	N	O	P
6	A	C	Q	R	G
7	A	C	H	I	G
8	L	E	F	K	B
9	A	F	M	N	O
10	C	F	P	J	R
11	A	D	B	H	I
12	D	E	B	K	L
13	M	D	C	G	O
14	C	F	P	Q	J
15	B	D	E	F	I
16	J	E	B	A	D
17	A	K	E	F	C
18	C	D	L	B	A

Items	Transaction ID										
A	1	3	4	5	6	7	9	11	16	17	18
B	1	2	3	5	8	11	12	15	16	18	
C	1	2	4	6	7	10	13	14	17	18	
D	1	2	3	11	12	13	15	16	18		
E	2	3	4	8	12	15	16	17			
F	4	8	9	10	14	15	17				
G	1	6	7	13							
H	2	7	11								
I	7	11	15								
J	10	14	16								
K	8	12	17								
L	8	12	18								
M	3	9	13								
N	4	5	9								
O	5	9	13								
P	5	9	13								
Q	6	14									
R	6	10									

Minimize Superfluous Processing

Candidacy generation is required

Transactional Layouts

• Bitmap Layout

Matrix : Rows represent transactions
Columns represent item
If item exists in transaction
then cell value = 1 else cell value = 0

T#	Items				
T1	A	G	D	C	B
T2	B	C	H	E	D
T3	B	D	E	A	M
T4	C	E	F	A	N
T5	A	B	N	O	P
T6	A	C	Q	R	G
T7	A	C	H	I	G
T8	L	E	F	K	B
T9	A	F	M	N	O
T10	C	F	P	J	R
T11	A	D	B	H	I
T12	D	E	B	K	L
T13	M	D	C	G	O
T14	C	F	P	Q	J
T15	B	D	E	F	I
T16	J	E	B	A	D
T17	A	K	E	F	C
T18	C	D	L	B	A

Transaction ID	items																	
T#	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T1	1	1	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0
T2	0	1	1	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0
T3	1	1	0	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0
T4	1	0	1	0	1	1	0	0	0	0	0	0	0	1	0	0	0	0
T5	1	1	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0
T6	1	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	1	1
T7	1	0	1	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0
T8	0	1	0	0	1	1	0	0	0	0	1	1	0	0	0	0	0	0
T9	1	0	0	0	0	1	0	0	0	0	0	0	1	1	1	0	0	0
T10	0	0	1	0	0	1	0	0	0	1	0	0	0	0	0	1	0	1
T11	1	1	0	1	0	0	0	1	1	0	0	0	0	0	0	0	0	0
T12	0	1	0	1	1	0	0	0	0	0	1	1	0	0	0	0	0	0
T13	0	0	1	1	0	0	1	0	0	0	0	0	1	0	1	0	0	0
T14	0	0	1	0	0	1	0	0	0	1	0	0	0	0	0	1	1	0
T15	0	1	0	1	1	1	0	0	1	0	0	0	0	0	0	0	0	0
T16	1	1	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0
T17	1	0	1	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0
T18	1	1	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0

Similar to horizontal layout.

Suitable for datasets with small dimensionality

Transactional Layouts

• Inverted Matrix Layout

El-Hajj and Zaiane, ACM SIGKDD'03

Minimize Superfluous Processing

Candidacy generation can be reduced

Appropriate for Interactive Mining

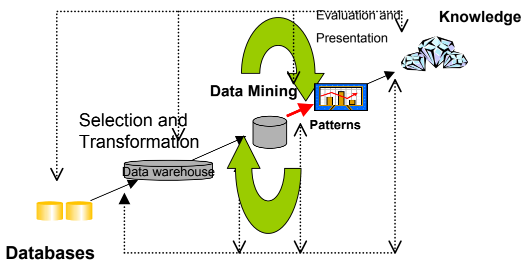
T#	Items				
T1	A	G	D	C	B
T2	B	C	H	E	D
T3	B	D	E	A	M
T4	C	E	F	A	N
T5	A	B	N	O	P
T6	A	C	Q	R	G
T7	A	C	H	I	G
T8	L	E	F	K	B
T9	A	F	M	N	O
T10	C	F	P	J	R
T11	A	D	B	H	I
T12	D	E	B	K	L
T13	M	D	C	G	O
T14	C	F	P	Q	J
T15	B	D	E	F	I
T16	J	E	B	A	D
T17	A	K	E	F	C
T18	C	D	L	B	A

Loc	Index		Transactional Array 1	2	3	4	5	6	7	8	9	10	11
1	R	2	(2,1)	(3,2)									
2	Q	2	(12,2)	(3,3)									
3	P	3	(4,1)	(9,1)	(9,2)								
4	O	3	(5,2)	(5,3)	(6,3)								
5	N	3	(13,1)	(17,4)	(6,2)								
6	M	3	(14,2)	(13,3)	(12,4)								
7	L	3	(8,1)	(8,2)	(15,9)								
8	K	3	(13,2)	(14,5)	(13,7)								
9	J	3	(13,4)	(13,5)	(14,7)								
10	I	3	(11,2)	(11,3)	(13,6)								
11	H	3	(14,1)	(12,3)	15,4)								
12	G	4	**(15,1)**	(16,4)	(16,5)	(15,6)							
13	F	7	(14,3)	(14,4)	(18,7)	(16,6)	(16,8)	(14,6)	(14,8)				
14	E	8	(15,2)	(15,3)	(16,3)	(17,5)	(15,5)	(15,7)	(15,8)	(16,9)			
15	D	9	**(16,1)**	(16,2)	(17,2)	(17,6)	(17,7)	(16,7)	(17,8)	(17,9)	(16,10)		
16	C	10	**(17,1)**	(17,2)	(18,3)	(18,5)	(18,6)	**(¤, ¤)**	**(¤, ¤)**	**(¤, ¤)**	(18,10)	(17,10)	
17	B	10	**(18,1)**	**(¤, ¤)**	(18,2)	(18,4)	**(¤, ¤)**	(18,8)	**(¤, ¤)**	**(¤, ¤)**	(18,9)	(18,11)	
18	A	11	**(¤, ¤)**	**(¤, ¤)**	**(¤, ¤)**	**(¤, ¤)**	**(¤, ¤)**	**(¤, ¤)**	**(¤, ¤)**	**(¤, ¤)**	**(¤, ¤)**	**(¤, ¤)**	**(¤, ¤)**

Why The Matrix Layout?

Interactive mining

Changing the support level means expensive steps (whole process is redone)



Why The Matrix Layout?

Repetitive tasks, (I/O) readings (Superfluous Processing)

T#	Items				
T1	A	G	D	C	B
T2	B	C	H	E	D
T3	B	D	E	A	M
T4	C	E	F	A	N
T5	A	B	N	O	P
T6	A	C	Q	R	G
T7	A	C	H	I	G
T8	L	E	F	K	B
T9	A	F	M	N	O
T10	C	F	P	J	R
T11	A	D	B	H	I
T12	D	E	B	K	L
T13	M	D	C	G	O
T14	C	F	P	Q	J
T15	B	D	E	F	I
T16	J	E	B	A	D
T17	A	K	E	F	C
T18	C	D	L	B	A

Support > 4

Frequent 1-itemsets {A, B, C, D, E, F}
Non frequent items {G, H, I, J, K, L, M, N, O, P, Q, R}

T#	Items				
T1	A		D	C	B
T2	B	C		E	D
T3	B	D	E	A	
T4	C	E	F	A	
T5	A	B			
T6	A	C			
T7	A	C			
T8		E	F		B
T9	A	F			
T10	C	F			
T11	A	D	B		
T12	D	E	B		
T13		D	C		
T14	C	F			
T15	B	D	E	F	
T16		E	B	A	D
T17	A		E	F	C
T18	C	D		B	A

Why The Matrix Layout?

Repetitive tasks, (I/O) readings (Superfluous Processing)

T#	Items				
T1	A	G	D	C	B
T2	B	C	H	E	D
T3	B	D	E	A	M
T4	C	E	F	A	N
T5	A	B	N	O	P
T6	A	C	Q	R	G
T7	A	C	H	I	G
T8	L	E	F	K	B
T9	A	F	M	N	O
T10	C	F	P	J	R
T11	A	D	B	H	I
T12	D	E	B	K	L
T13	M	D	C	G	O
T14	C	F	P	Q	J
T15	B	D	E	F	I
T16	J	E	B	A	D
T17	A	K	E	F	C
T18	C	D	L	B	A

Support > 9

Frequent 1-itemsets {A, B, C}
Non frequent items
{D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R}

T#	Items				
T1	A			C	B
T2	B	C			
T3	B			A	
T4	C			A	
T5	A	B			
T6	A	C			
T7	A	C			
T8					B
T9	A				
T10	C				
T11	A		B		
T12			B		
T13			C		
T14	C				
T15	B				
T16			B	A	
T17	A				C
T18	C			B	A

Transactional Layouts

- Inverted Matrix Layout

Support > 4

Loc	Index		Transactional Array										
			1	2	3	4	5	6	7	8	9	10	11
1	R	2	(2,1)	(3,2)									
2	Q	2	(12,2)	(3,3)									
3	P	3	(4,1)	(9,1)	(9,2)								
4	O	3	(5,2)	(5,3)	(6,3)								
5	N	3	(13,1)	(17,4)	(6,2)								
6	M	3	(14,2)	(13,3)	(12,4)								
7	L	3	(8,1)	(8,2)	(15,9)								
8	K	3	(13,2)	(14,5)	(13,7)								
9	J	3	(13,4)	(13,5)	(14,7)								
10	I	3	(11,2)	(11,3)	(13,6)								
11	H	3	(14,1)	(12,3)	15,4)								
12	G	4	(15,1)	(16,4)	(16,5)	(15,6)							
13	F	7	(14,3)	(14,4)	(18,7)	(16,6)	(16,8)	(14,6)	(14,8)				
14	E	8	(15,2)	(15,3)	(16,3)	(17,5)	(15,5)	(15,7)	(15,8)	(16,9)			
15	D	9	(16,1)	(16,2)	(17,2)	(17,6)	(17,7)	(16,7)	(17,8)	(17,9)	(16,10)		
16	C	10	(17,1)	(17,2)	(18,3)	(18,5)	(18,6)	(☼,☼)	(☼,☼)	(☼,☼)	(18,10)	(17,10)	
17	B	10	(18,1)	(☼,☼)	(18,2)	(18,4)	(☼,☼)	(18,8)	(☼,☼)	(☼,☼)	(18,9)	(18,11)	
18	A	11	(☼,☼)	(☼,☼)	(☼,☼)	(☼,☼)	(☼,☼)	(☼,☼)	(☼,☼)	(☼,☼)	(☼,☼)	(☼,☼)	(☼,☼)

Border Support

Transactional Layouts

- Inverted Matrix Layout

Loc	Index		Transactional Array										
			1	2	3	4	5	6	7	8	9	10	11
13	F	7	(14,3)	(14,4)	(18,7)	(16,6)	(16,8)	(14,6)	(14,8)				
14	E	8	(15,2)	(15,3)	(16,3)	(17,5)	(15,5)	(15,7)	(15,8)	(16,9)			
15	D	9	(16,1)	(16,2)	(17,2)	(17,6)	(17,7)	(16,7)	(17,8)	(17,9)	(16,10)		
16	C	10	(17,1)	(17,2)	(18,3)	(18,5)	(18,6)	(☼,☼)	(☼,☼)	(☼,☼)	(18,10)	(17,10)	
17	B	10	(18,1)	(☼,☼)	(18,2)	(18,4)	(☼,☼)	(18,8)	(☼,☼)	(☼,☼)	(18,9)	(18,11)	
18	A	11	(☼,☼)	(☼,☼)	(☼,☼)	(☼,☼)	(☼,☼)	(☼,☼)	(☼,☼)	(☼,☼)	(☼,☼)	(☼,☼)	(☼,☼)

Transactional Layouts

- Inverted Matrix Layout

Loc	Index		Transactional Array										
			1	2	3	4	5	6	7	8	9	10	11
13	F	7	(14,3)	(14,4)	(18,7)	(16,6)	(16,8)	(14,6)	(14,8)				
14	E	8	(15,2)	(15,3)	(16,3)	(17,5)	(15,5)	(15,7)	(15,8)	(16,9)			
15	D	9	(16,1)	(16,2)	(17,2)	(17,6)	(17,7)	(16,7)	(17,8)	(17,9)	(16,10)		
16	C	10	(17,1)	(17,2)	(18,3)	(18,5)	(18,6)	(☼,☼)	(☼,☼)	(☼,☼)	(18,10)	(17,10)	
17	B	10	(18,1)	(☼,☼)	(18,2)	(18,4)	(☼,☼)	(18,8)	(☼,☼)	(☼,☼)	(18,9)	(18,11)	
18	A	11	(☼,☼)	(☼,☼)	(☼,☼)	(☼,☼)	(☼,☼)	(☼,☼)	(☼,☼)	(☼,☼)	(☼,☼)	(☼,☼)	(☼,☼)

T#	Items			
T1	A	D	C	B
T2	B	C	E	D
T3	B	D	E	A
T4	C	E	F	A
T5	A	B		
T6	A	C		
T7	A	C		
T8	E	F	B	
T9	A	F		
T10	C	F		
T11	A	D	B	
T12	D	E	B	
T13	D	C		
T14	C	F		
T15	B	D	E	F
T16	E	B	A	D
T17	A	E	F	C
T18	C	D	B	A

The Algorithms
(State of the Art)

All

Apriori, FP-Growth, COFI*, ECLAT

Closed

CHARM, CLOSET+,COFI-CLOSED

Maximal

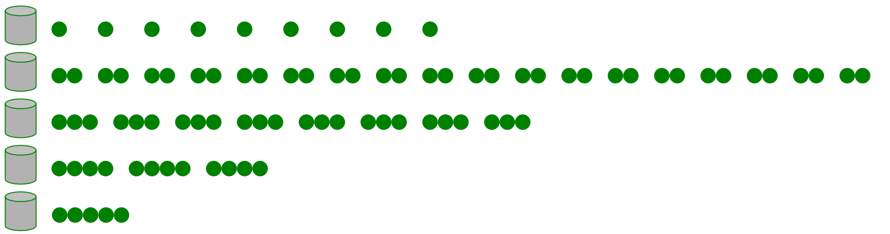
MaxMiner, MAFIA, GENMAX, COFI-MAX

All-Apriori

Apriori

Repetitive I/O scans

Huge Computation to generate candidate items



R. Agrawal, R. Srikant, VLDB'9

All-Apriori

Problems with Apriori

- Generation of candidate itemsets are expensive (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.
- High number of data scans

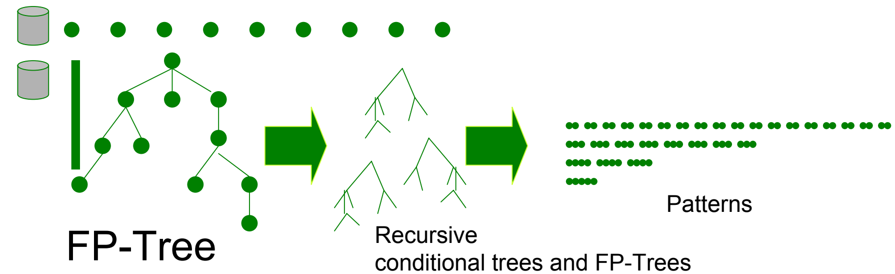
Frequent Pattern Growth

- First algorithm that allows frequent pattern mining without generating candidate sets
- Requires Frequent Pattern Tree

All-FP-Growth

FP-Growth

2 I/O scans
Reduced candidacy generation
High memory requirements
Claims to be 1 order of magnitude faster than Apriori



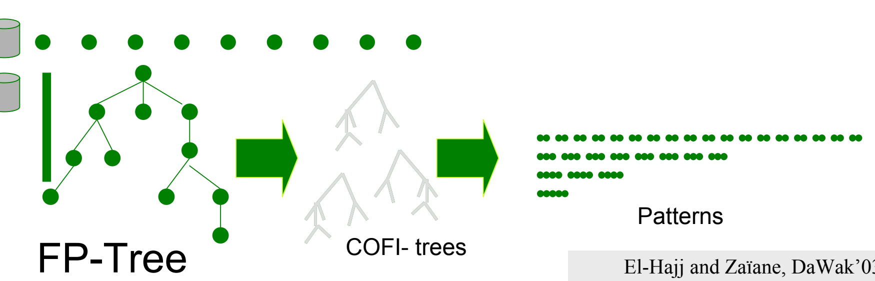
J. Han, J. Pei, Y. Yin, SIGMOD'00

All-COFI

COFI algorithm big picture

COFI

2 I/O scans
reduced candidacy generation
Small memory footprint



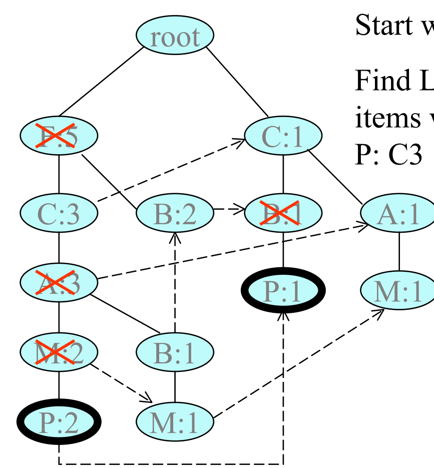
El-Hajj and Zaïane, DaWak'03

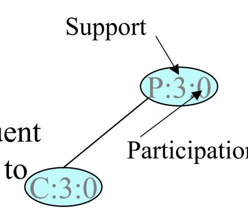
All-COFI

Co-Occurrences Frequent Item tree

Start with item P:

Find Locally frequent items with respect to P: C3

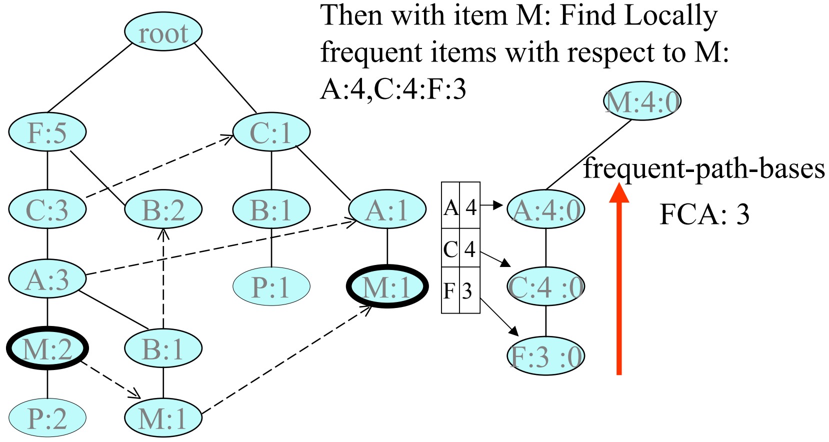




PC:3 frequent-path-base
All subsets of PC:3 are frequent and have the same support

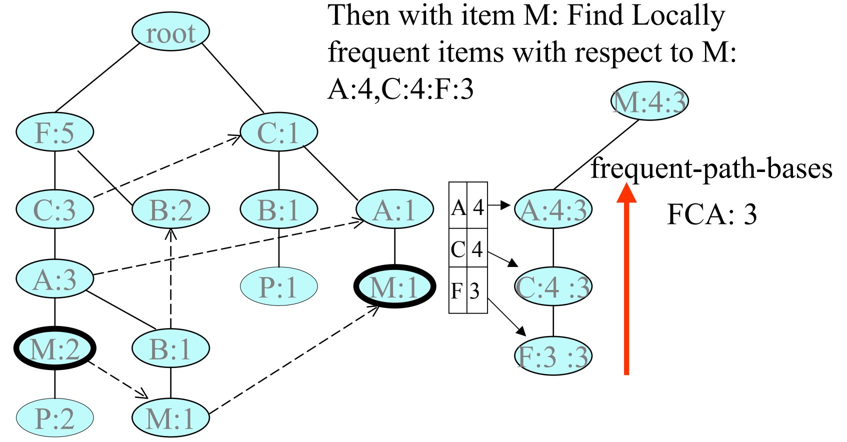
Co-Occurrences Frequent Item tree

Then with item M: Find Locally frequent items with respect to M: A:4,C:4:F:3



Co-Occurrences Frequent Item tree

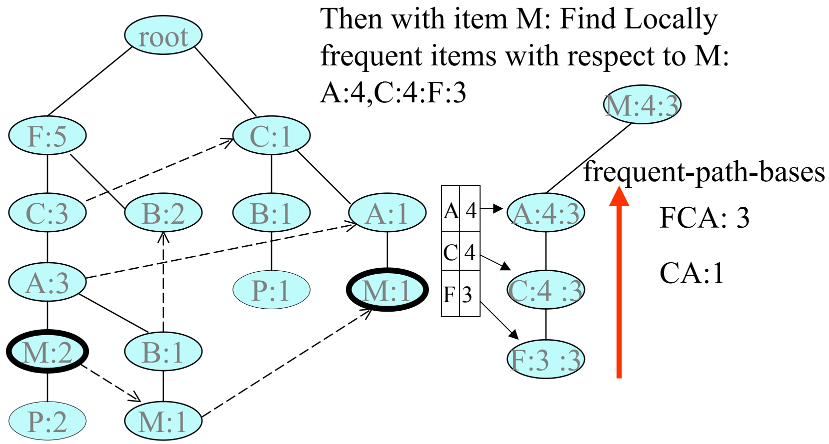
Then with item M: Find Locally frequent items with respect to M: A:4,C:4:F:3



ll-COFI

Co-Occurrences Frequent Item tree

Then with item M: Find Locally frequent items with respect to M: A:4,C:4:F:3



All-COFI

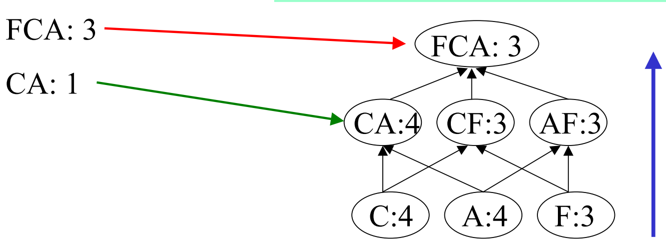
Co-Occurrences Frequent Item tree

How to mine frequent-path-bases

Three approaches:

1: Bottom-Up

Support of any pattern is the summation of the supports of its supersets of frequent-path-bases



Co-Occurrences Frequent Item tree

How to mine frequent-path-bases

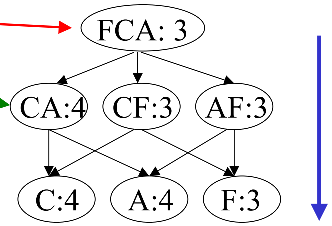
Three approaches:

2: Top-down

Support of any pattern is the summation of the supports of its supersets of frequent-path-bases

FCA: 3

CA: 1



Co-Occurrences Frequent Item tree

How to mine frequent-path-bases

Three approaches:

3: Leap-Traversal

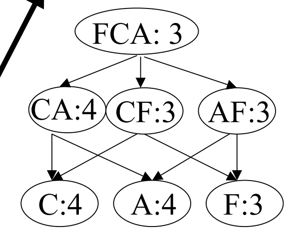
Support of any pattern is the summation of the supports of its supersets of frequent-path-bases

1) Intersect non frequent path bases

FCA: 3 ∩ CA:1 = CA

2) Find subsets of the only frequent paths (sure to be frequent)

3) Find the support of each pattern



All-ECLAT

ECLAT

- For each item, store a list of transaction ids (tids)

Horizontal Data Layout

TID	Items
1	A,B,E
2	B,C,D
3	C,E
4	A,C,D
5	A,B,C,D
6	A,E
7	A,B
8	A,B,C
9	A,C,D
10	B

Vertical Data Layout

A	B	C	D	E
1	1	2	2	1
4	2	3	4	3
5	5	4	5	6
6	7	8	9	
7	8	9		
8	10			
9				

TID-list

M.J.Zaki IEEE transactions on Knowledge and data Engineering 00

All-ECLAT

ECLAT

- Determine support of any k-itemset by intersecting tid-lists of two of its (k-1) subsets.

A
1
4
5
6
7
8
9

$\wedge$

B
1
2
5
7
8
10

$\rightarrow$

AB
1
5
7
8

ll-ECLAT

ECLAT

Find all frequent patters with respect to item A

AB, AC, …. ABC, ABD, ACD, ABCD ……..

Then it finds all frequent patters with respect to item B

BC, BD, …. BCD, BDE, BCDE ……..

- 3 traversal approaches:
  - top-down, bottom-up and hybrid
- Advantage: very fast support counting, Few scans of database (best case 2)
- Disadvantage: intermediate tid-lists may become too large for memory

Other Algorithms for Other Patterns

Algorithms for Closed Patterns and Maximal Patterns will be discussed in class with paper presentations.

Which algorithm is the winner?

Not clear yet

With relatively small datasets we can find different winners

1. By using different datasets
2. By changing the support level
3. By changing the implementations

Which algorithm is the winner?

What about Extremely large datasets (hundreds of millions of transactions)?

Most of the existing algorithms do not run on such sizes

Vertical approaches and Bitmaps approaches cannot load the transactions in Main Memory

Reparative approaches cannot keep scanning these huge databases many times

Requirements: We need algorithms that

1) do not require multiple scans of the database

2) Leave small foot print in Main Memory at any given time