Unsupervised Data Mining: From Batch to Stream Mining Algorithms

Prof. Dr. Stefan Kramer
Johannes Gutenberg-Universität Mainz

Outline

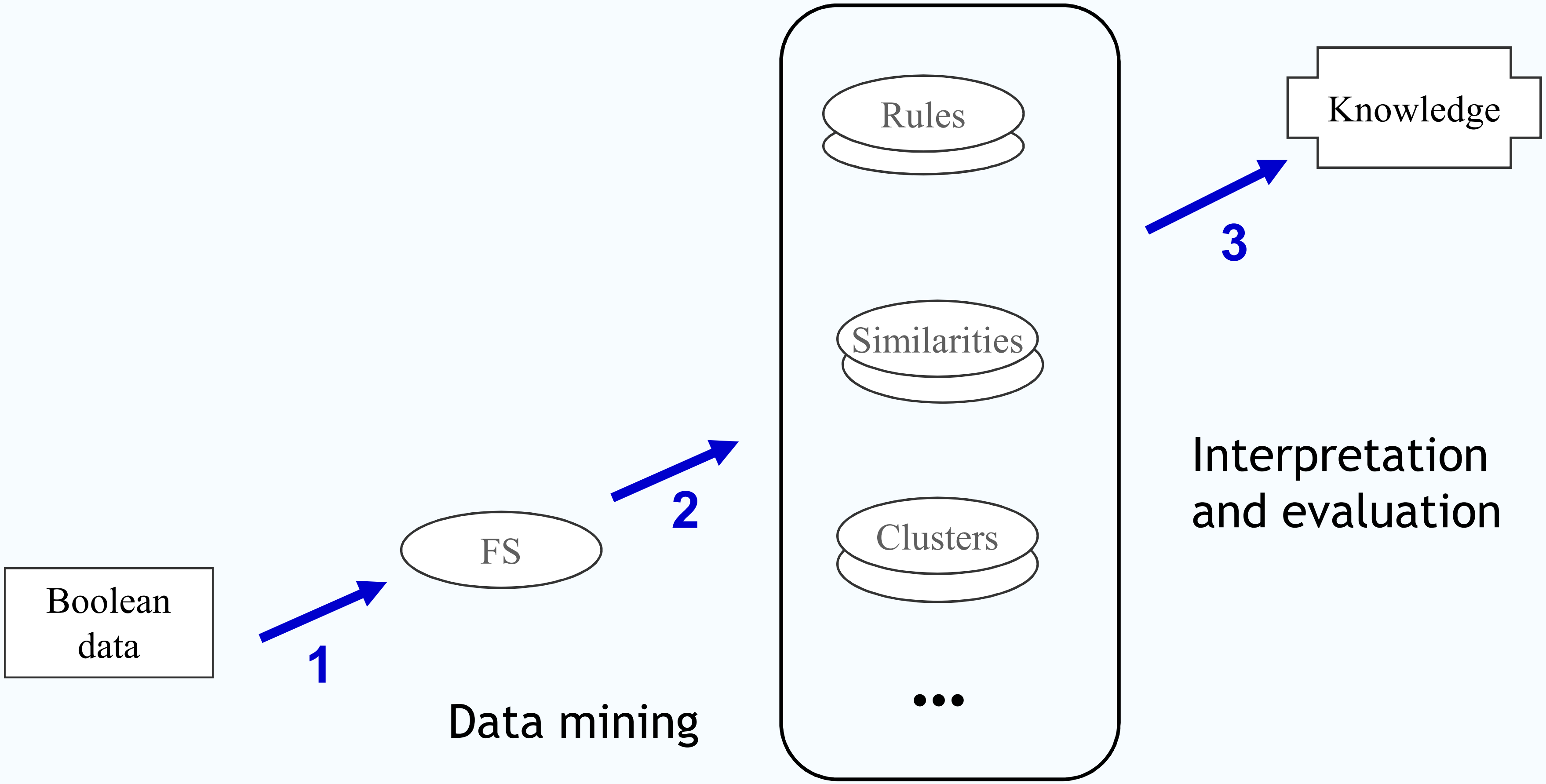
- Condensed representations: closed and free sets
- FP-trees and FP-growth

Condensed Representations: Closed and Free Sets

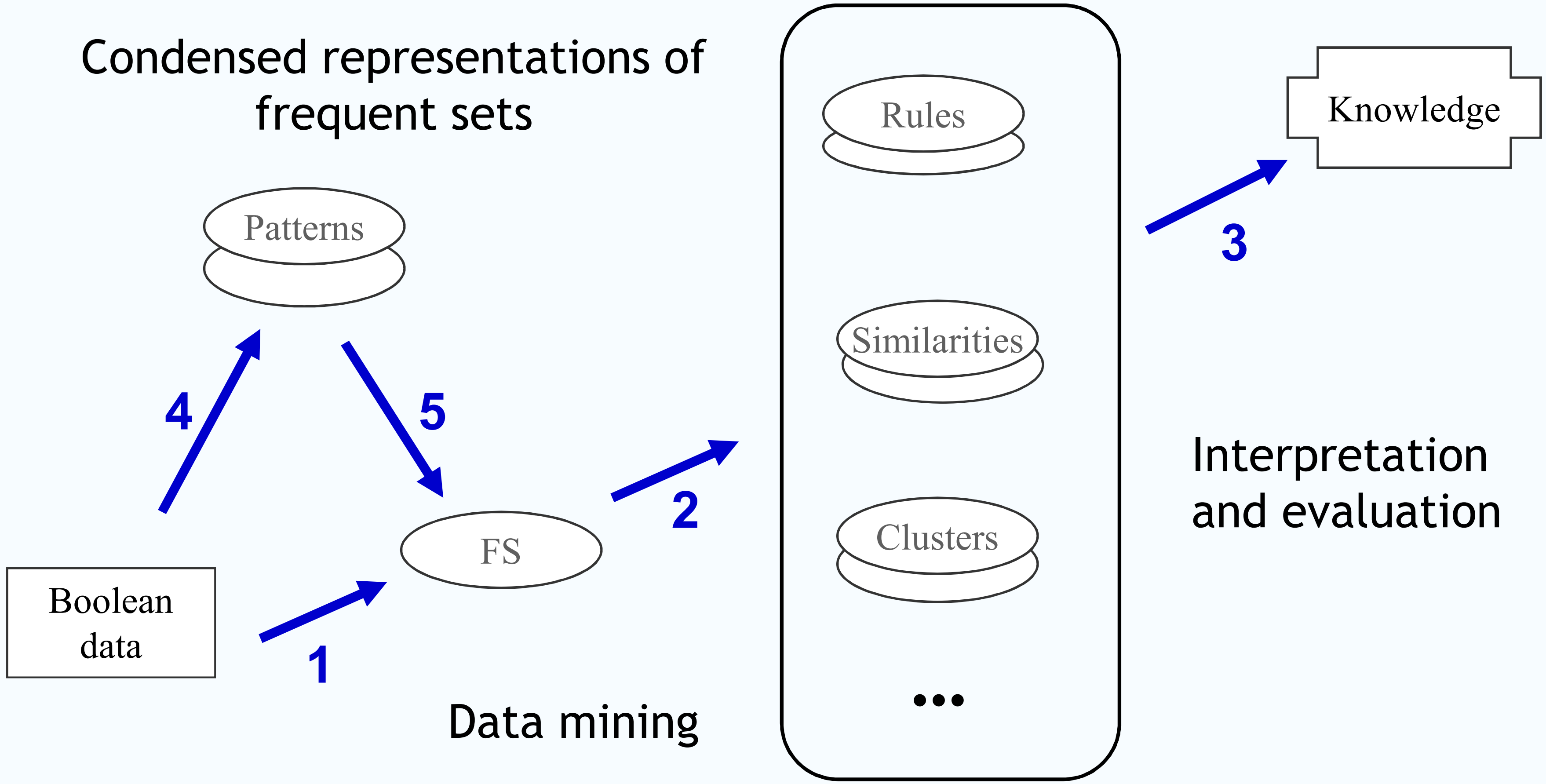
Condensed Representations: Motivation

- Problem of APriori-like approaches: computing frequent itemsets intractable in *dense* and *highly correlated Boolean* data (remember: exponential in the worst-case)
- Distinction: *sparse* and *dense* dataset
- Condensed representations:
  remove redundancy and provide more interesting patterns to the end-user

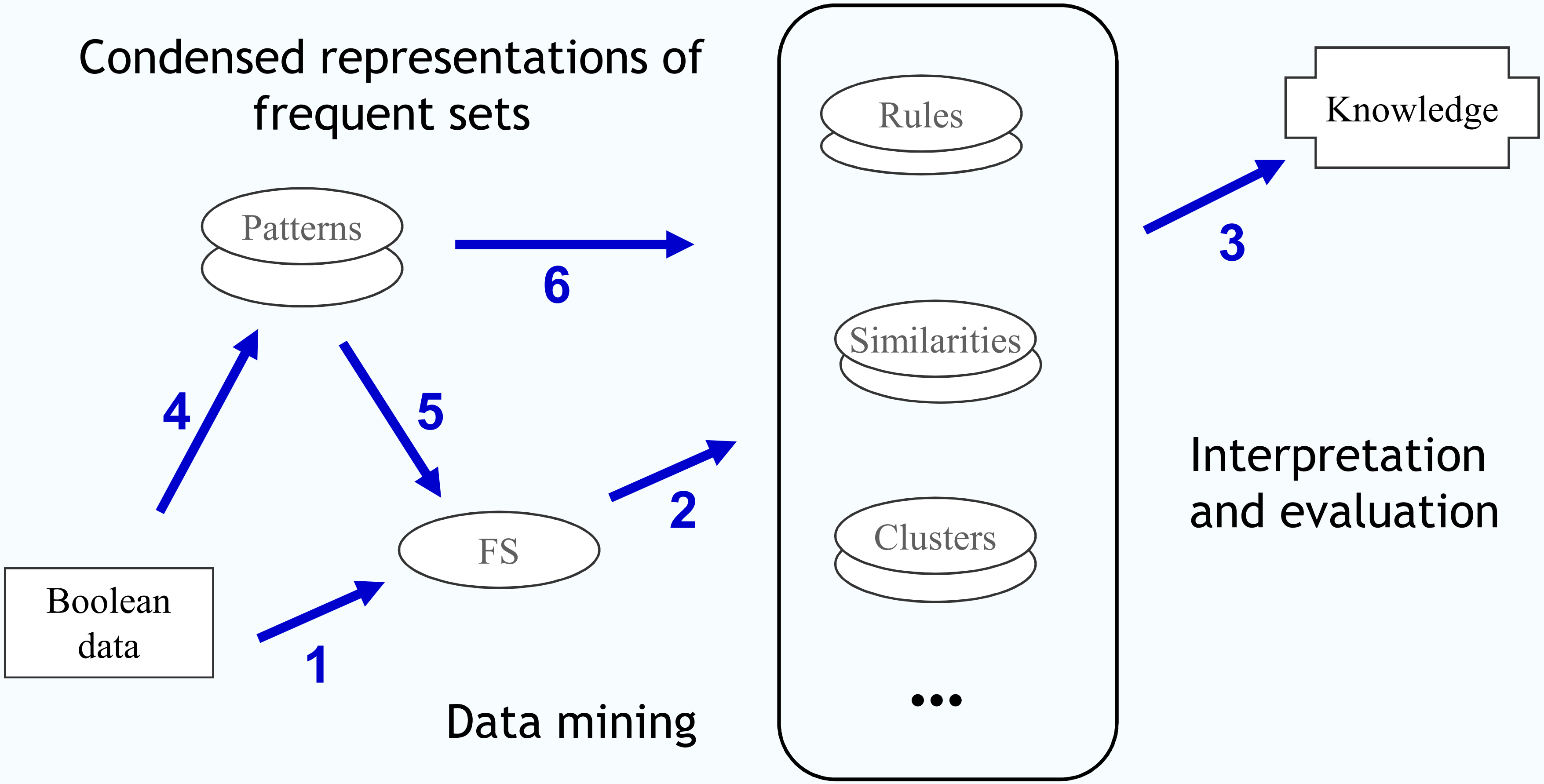
Multiples Uses of Frequent Itemsets



… Based on Condensed Representations



… Based on Condensed Representations



The “Closure” Evaluation Function

- The closure of X is the *maximal* superset of X that has exactly the same frequency as X (!)

closure(X, r) = items(objects(X, r), r)

A	B	C	D
1	0	1	0
1	1	1	0
0	1	1	1
0	1	0	1
1	1	1	0

closure({A}, r) = {A,C}

Note:
$A \Rightarrow C$ has confidence 1.0

Closed Sets

- X is a closed set iff X = closure(X, r). It is a maximal set of items that *support exactly the same transactions*.

A	B	C	D
1	0	1	0
1	1	1	0
0	1	1	1
0	1	0	1
1	1	1	0

{A,C} is closed {A,B} is not closed

$C_{Close}(S)$

- How about the empty set?
- *Closedness is* not *an anti-monotonic property!*

Closed Sets

- X is a closed set iff X = closure(X, r). It is a maximal set of items that *support exactly the same transactions*.

A	B	C	D
1	0	1	0
1	1	1	0
0	1	1	1
0	1	0	1
1	1	1	0

Frequent (MinSupport = 2)
A:3, B:4, C:4, D:2,
AB:2, AC:3, BC:3, BD:2,
ABC:2
Frequent closed:
B:4, C:4,
AC:3, BC:3, BD:2, ABC:2

Closed Sets

A	B	C	D
1	0	1	0
1	1	1	0
0	1	1	1
0	1	0	1
1	1	1	0

Frequent:
A:3, B:4, C:4, D:2,
AB:2, AC:3, BC:3, BD:2,
ABC:2
Frequent closed:
B:4, C:4,
AC:3, BC:3, BD:2, ABC:2

A	B
1	0
1	1
0	1
0	1
1	1

?

B	D
0	0
1	0
1	1
1	1
1	0

?

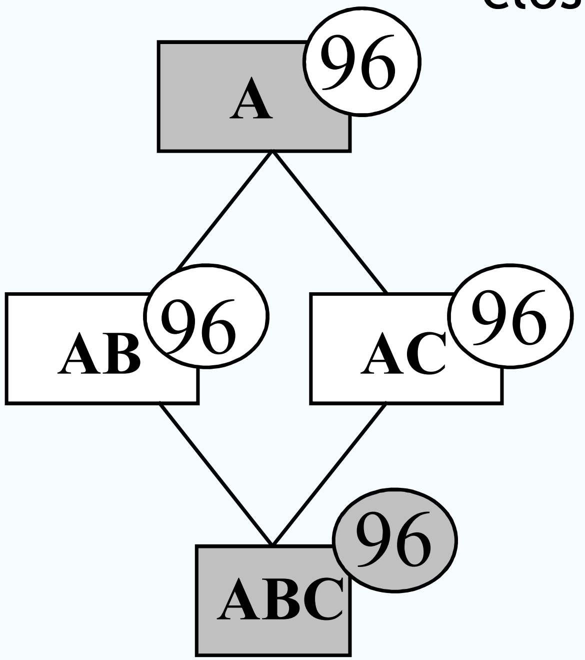
Possible: *confidence 1.0 (logical) rules:*
$A \rightarrow C$, $D \rightarrow B$, $AB \rightarrow C$

Properties of the Closure

- $X \subseteq \text{closure}(X)$
- $\text{closure}(\text{closure}(X)) = \text{closure}(X)$
- $Y \subseteq X \Rightarrow \text{closure}(Y) \subseteq \text{closure}(X)$

Using Closed Sets

closure(A)={ABC}



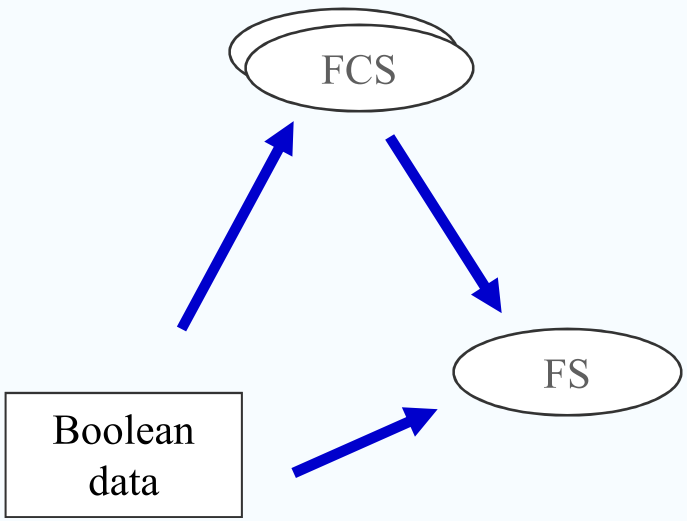
closure({ABC})={ABC}

Comparison with Maximally Specific Itemsets and Borders

- With borders/version spaces:
  possible to generate *all solution patterns*
- With frequent closed sets:
  possible to generate *all solution patterns along with their frequencies*

Closed Sets and How to Use Them

When S is frequent, choose the frequent closed set X s.t. $S \subseteq X$ that has the maximal support and return freq(S,r) = freq(X,r)



Example Frequent Closed Sets

1	ABCD
2	AC
3	AC
4	ABCD
5	BC
6	ABC

16 frequent sets

1 maximal frequent set

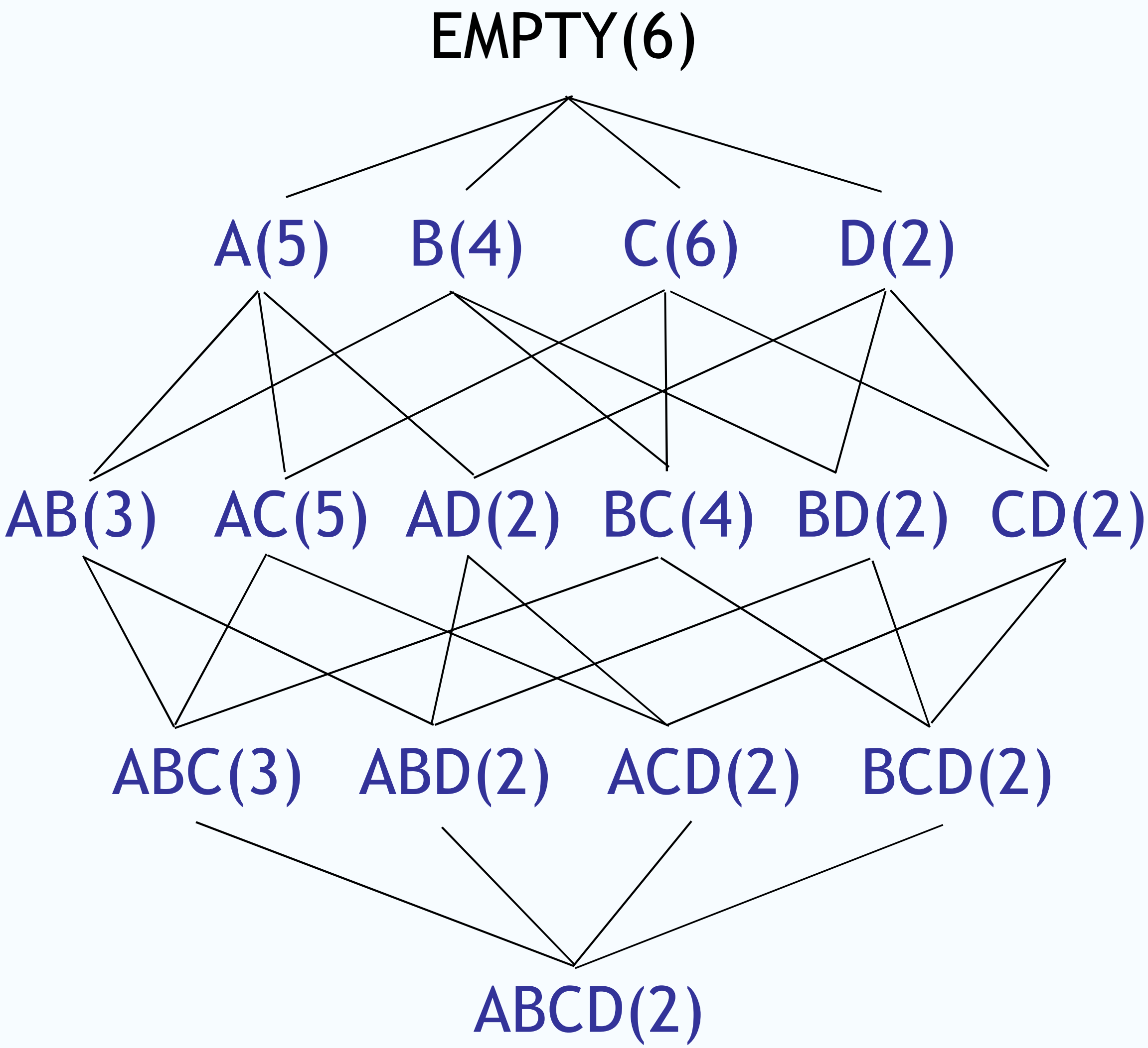
5 frequent closed sets

C, AC, BC, ABC, ABCD

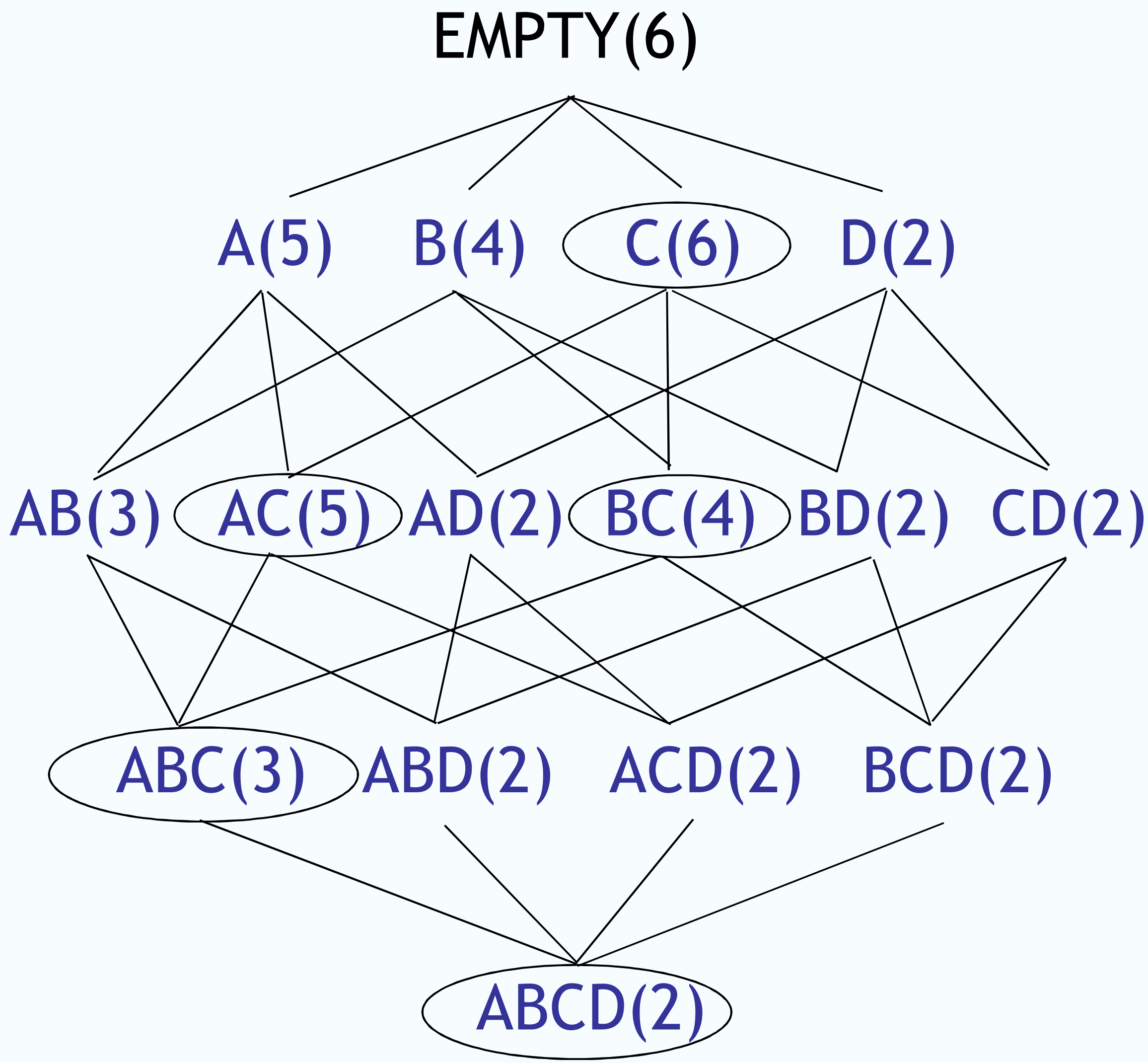
$A \rightarrow C$, $B \rightarrow C$, $AB \rightarrow C$, $ABD \rightarrow C$, etc.

Minimum frequency threshold = 2

Example: **Closed Sets?**



Example: Closed Sets!



Free Sets

- An itemset X is called *free* if *every proper subset* of X has a frequency *strictly greater* than that of X.
- In other words, X is a free set iff there is no logical (confidence 1.0) rule that holds between any of its subsets
- Free sets are special cases of δ-free sets (see next slides), *closed sets are the closures of free sets*

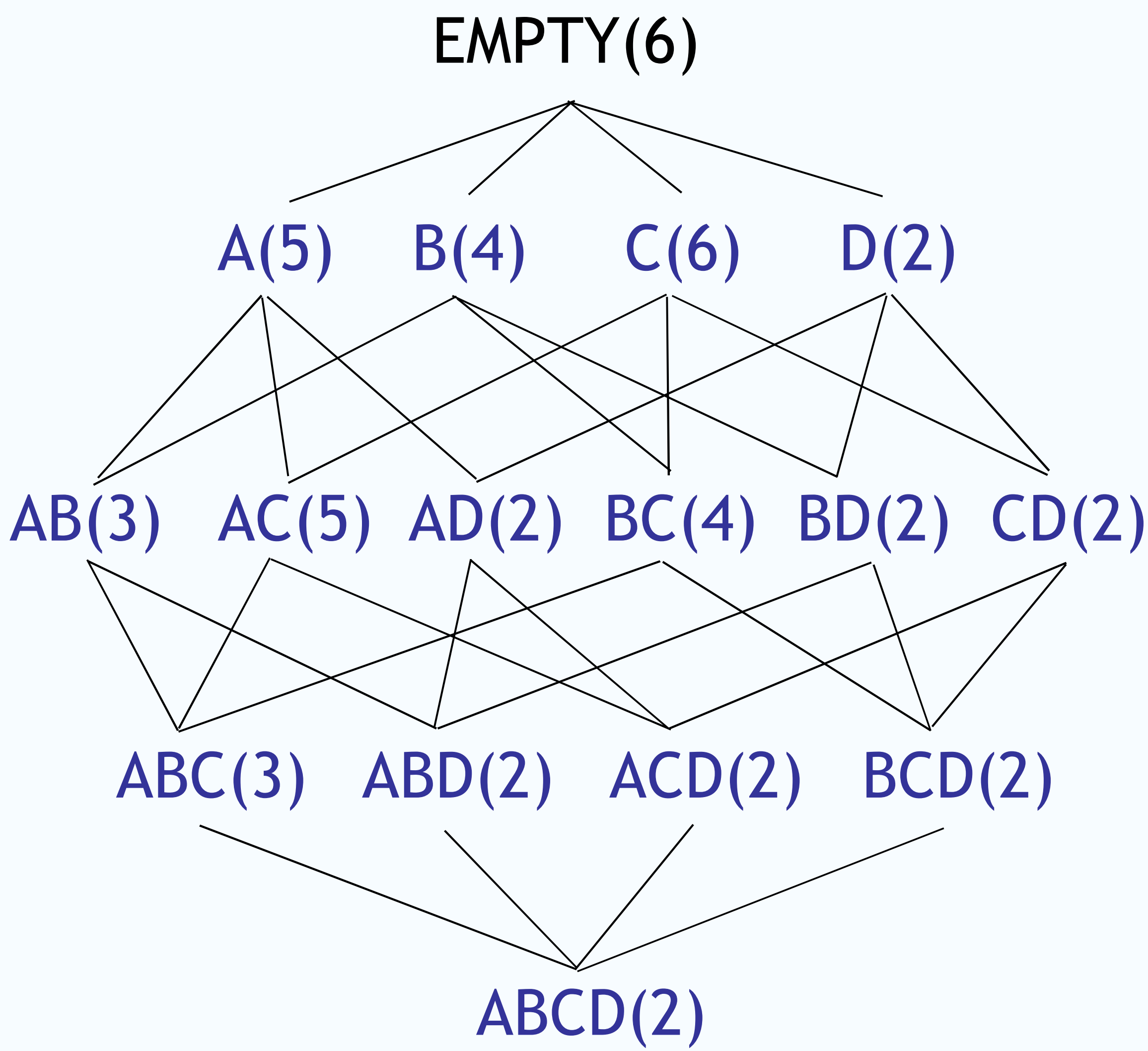
(how about the empty set?)

A	B	C	D
1	0	1	0
1	1	1	0
0	1	1	1
0	1	0	1
1	1	1	0

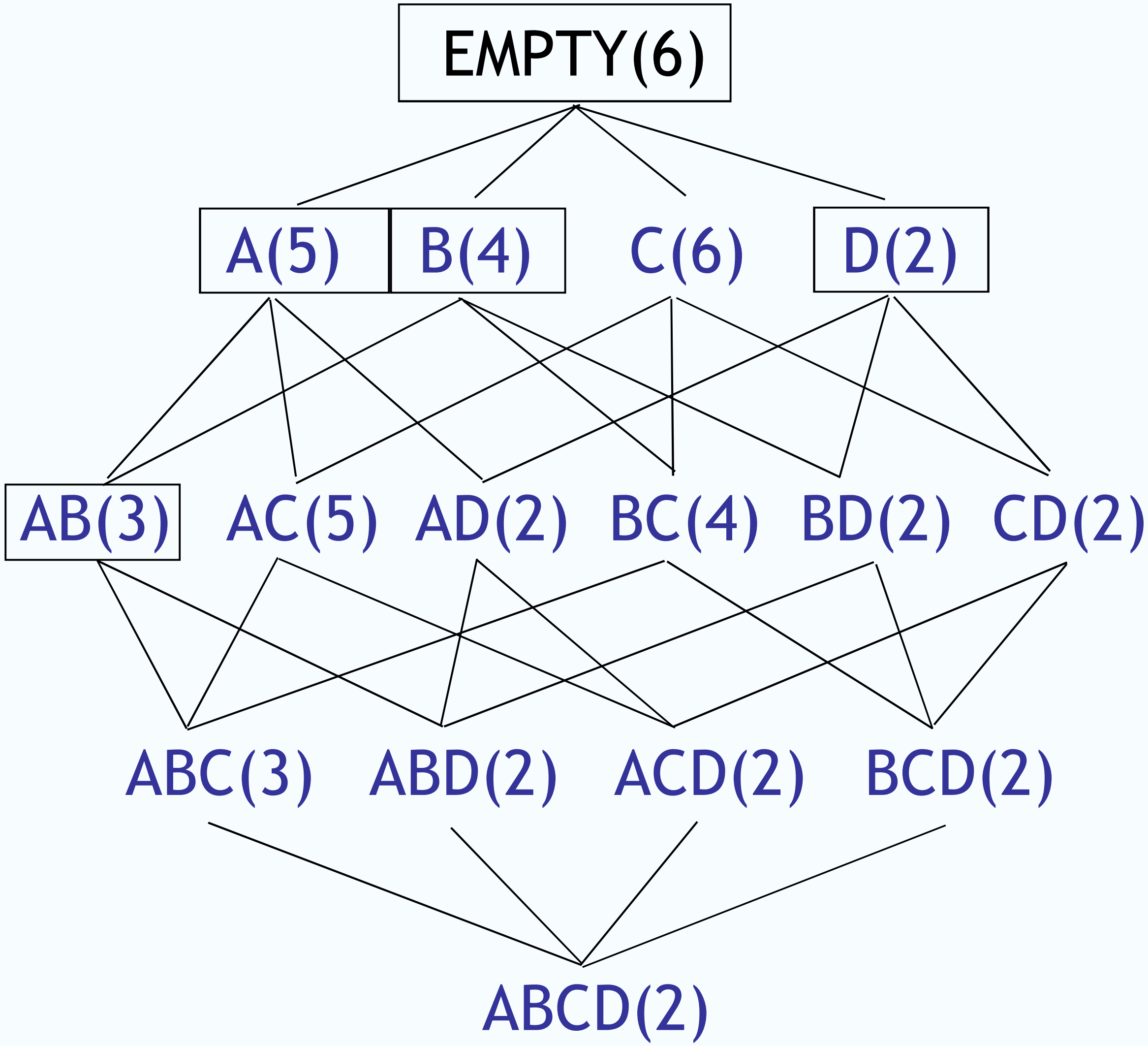
{A,B} is free {A,C} is not free

$C_{Free}(S)$ ***(anti-monotonic!)***

Example: Free Sets?

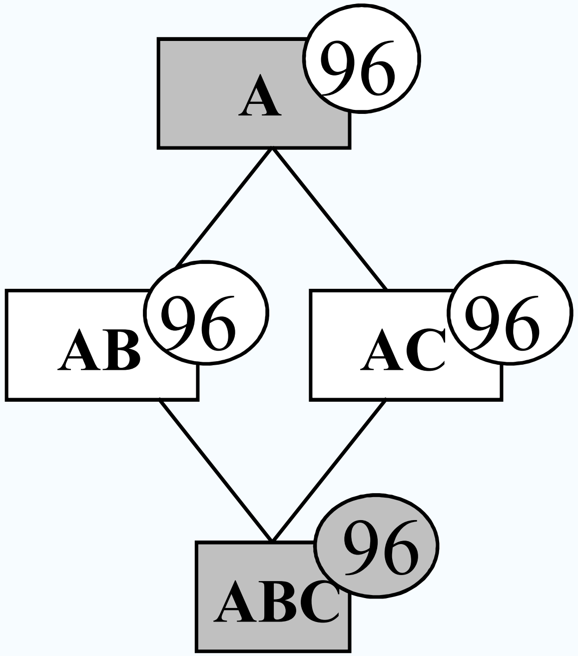


Example: Free Sets



Closed and Free Sets

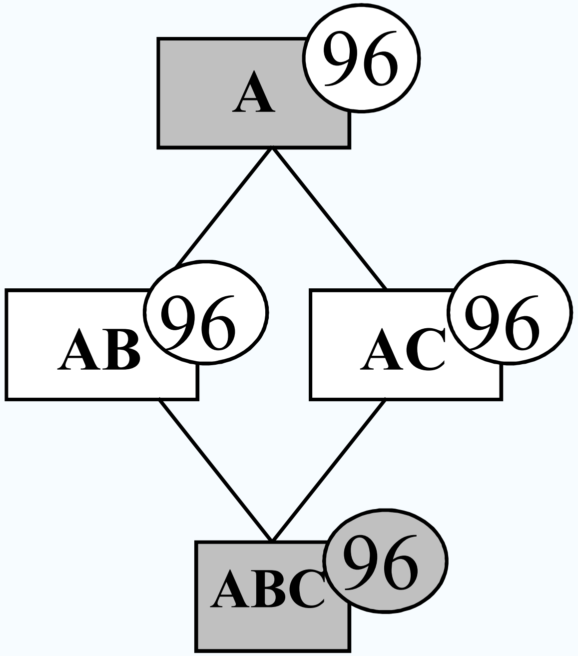
closure(A)={ABC}



closure({ABC})={ABC}

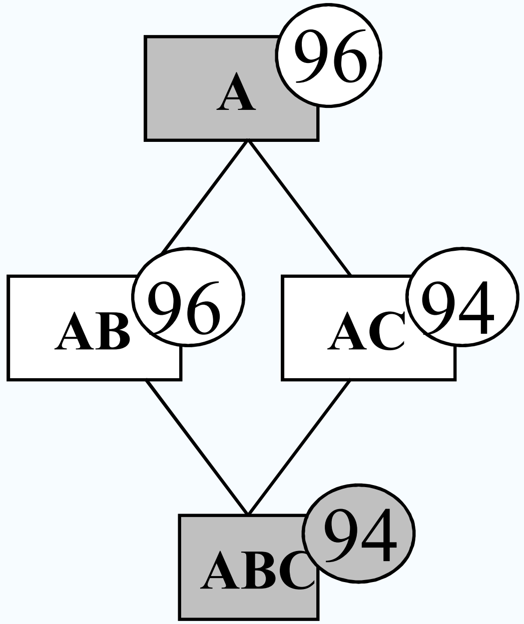
Closures and δ-Closures

closure(A)={ABC}



closure({ABC})={ABC}

B,C $\in$ closure$_\delta$(A)



δ-Freeness 1

- A δ-free-set is such that there is no δ-strong rule that holds between any of its subsets
- $X \Rightarrow_\delta Y$ is δ-strong if it has at most δ exceptions

A	B	C	D
1	0	1	0
1	1	1	0
0	1	1	1
0	1	0	1
1	1	1	0

$\{A,B\}$ is free, but not 1-free

$C_{\delta\text{-free}}(S)$ (anti-monotonic)

δ-Freeness 2

- ...is (as, e.g., the minimum frequency constraint) *anti-monotonic*! Any subset of a delta-free itemset is also delta-free
- Any superset of a non-delta-free itemset is also non-delta-free
- ...provides a *condensed representation*: frequent free itemsets are less numerous than frequent itemsets while providing almost the same information

APriori Can Be Used to Solve Any Anti-Monotonic Constraint

Most important modification here from:

$$\mathcal{F}_l(r) := \{X \in \mathcal{C}_l \mid \mathit{fr}(X, r) \geq \mathit{min\_fr}\};$$

to:

$$\mathcal{F}_l(r) := \{X \in \mathcal{C}_l \mid \mathit{fr}(X, r) \geq \mathit{min\_fr} \text{ and}$$

$$\mathit{fr}(X, r) \neq \mathit{fr}(Y, r) \text{ for all } Y \subset X\};$$

Discovery of All Frequent *Closed Sets*

- Find all frequent free sets in the described manner
- Compute closures of frequent free sets from the database
  - determine transactions, where they occur, and intersect them

Examples of Condensed Representations

1	ABCD
2	AC
3	AC
4	ABCD
5	BC
6	ABC

16 frequent sets

1 maximal frequent set

Frequent closed sets

C, AC, BC, ABC, ABCD

Frequent free sets

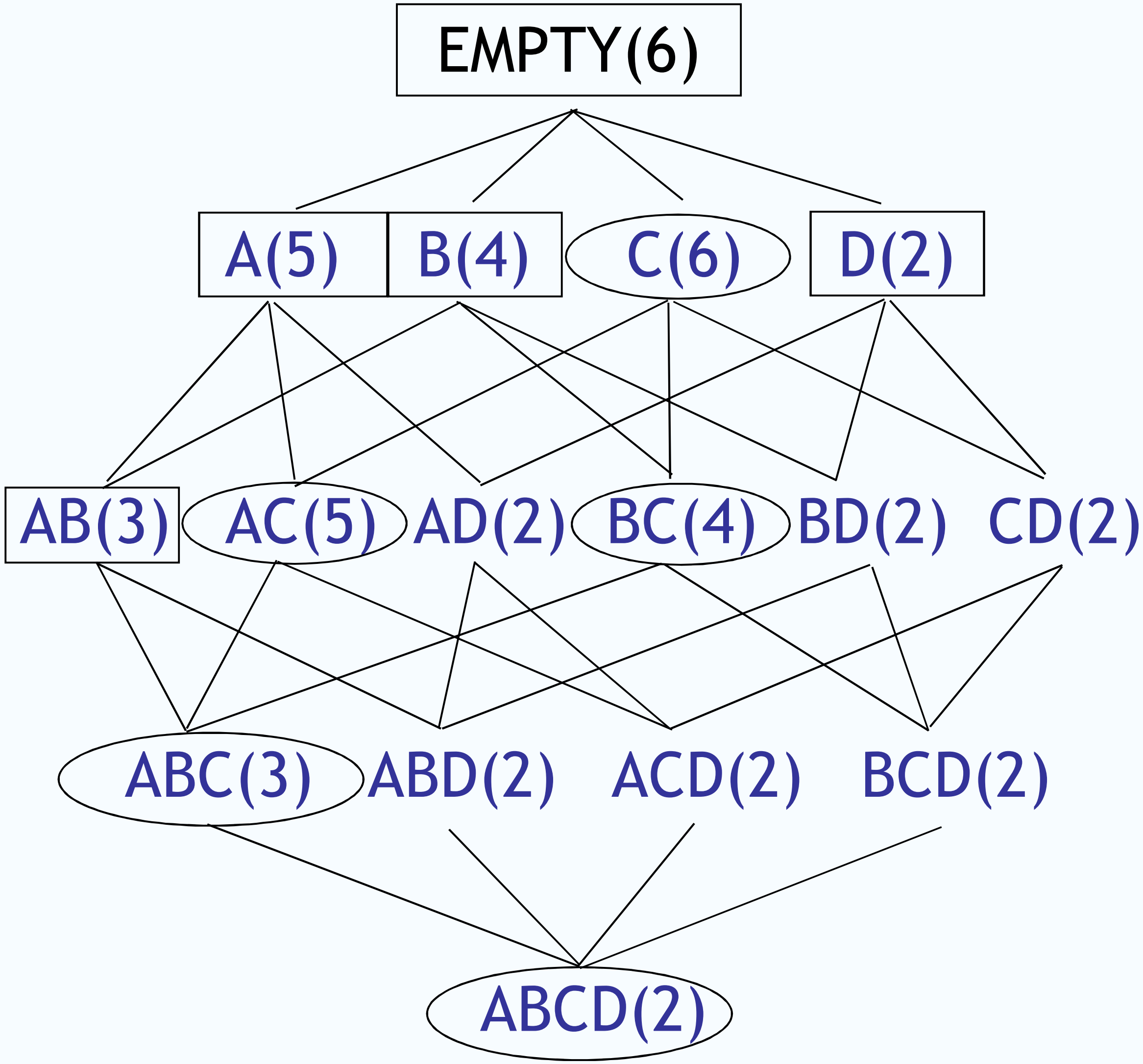
$\emptyset$, A, B, D, AB

Frequent 1-free sets

$\emptyset$, B, D

Minimum frequency threshold = 2

Example Closed and Free Sets



Different Search Strategies: FP-Trees

Motivation for FP-Growth

- Long patterns require many candidates to be tested (2^{size} -2)
- Database scans are *expensive*

- **Idea:** compress large database in a compact data structure, the so-called *FP-tree*, and extract information from that tree
- This is not done in one step: divide-and-conquer method to create smaller and smaller sub-databases and FP-trees
- This is called the *pattern growth* methodology

Construction of FP-Tree

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

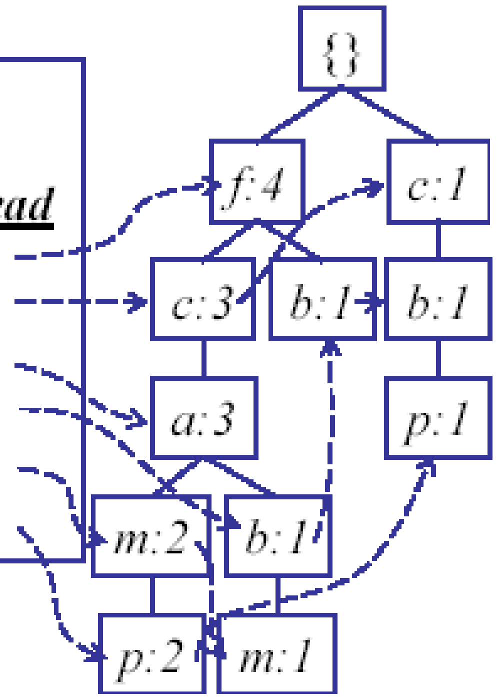
min_support = 0.5

Steps:

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

Header Table

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



Advantages

- Preserves relevant information for frequent pattern mining (long patterns are not broken; also note: infrequent information eliminated)

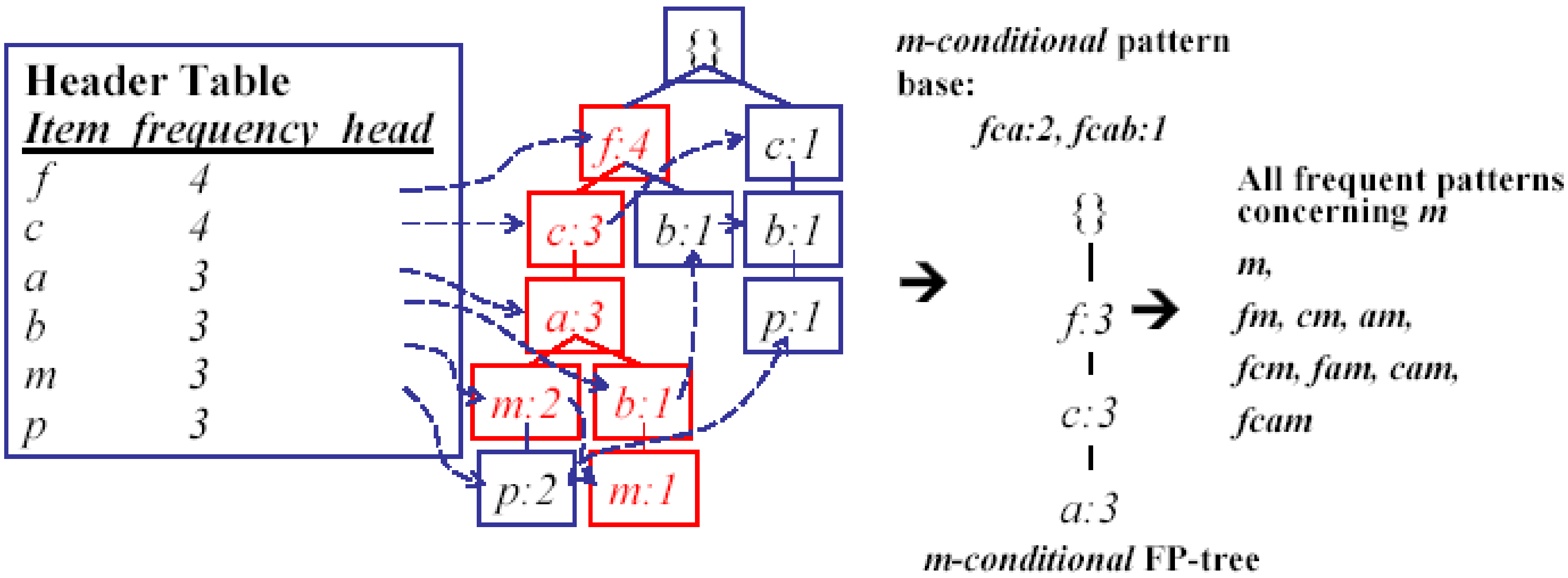
- *What is size of data structure?*
- Claims that compression takes place and expensive database scans are not repeated

Mining Frequent Patterns Using FP-Tree

- Divide-and-conquer method:
  - For each node in the FP-tree, construct its *conditional pattern-base*, and then its *conditional FP-tree*
  - Recursively mine *conditional FP-trees* until the tree is either *empty* or only a *path*.
  - If it is only a path, then enumerate all subsets as frequent patterns.

Construction of Conditional FP-Tree

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

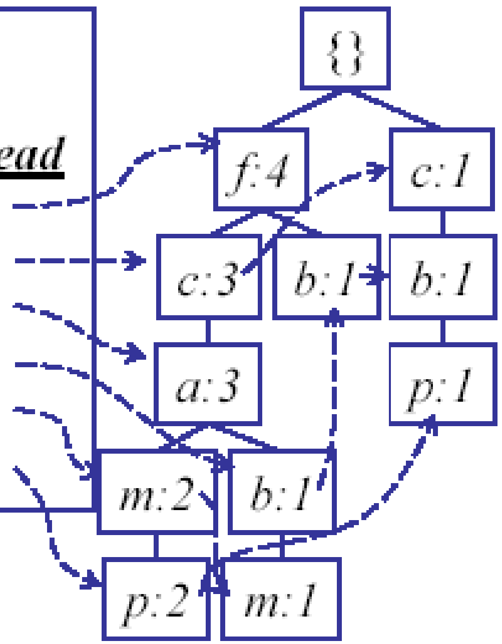


Step 1: From FP-Tree to Conditional Pattern Base

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

Header Table

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



***Conditional* pattern bases**

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

Creating Conditional Pattern Bases

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

Frequent Pattern Growth

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

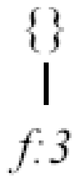
Step 3: Recursively Mine Conditional FP-Tree

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

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

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

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



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

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

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

Single FP-Tree: Path Generation

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

m-conditional **FP-tree**

→

All frequent patterns concerning *m*

m,

fm, cm, am,

fcm, fam, cam,

fcam

Summary of Frequent Pattern Growth

- FP-growth is an order of magnitude faster than APriori *if main memory is sufficient*
- Han *et al*. claim: „no candidate generation, no candidate testing“
- Quite compact and useful data structure
- Eliminate repeated database scan
- Basic operation is counting and FP-tree building

(Sub-)Algorithm FP-Growth

procedure FP_growth(*Tree*, α)
(1) **if** *Tree* contains a single path P **then**
(2) **for each** combination (denoted as β) of the nodes in the path P
(3) generate pattern $\beta \cup \alpha$ with *support_count* = *minimum support count of nodes in* β;
(4) **else for each** a_i in the header of *Tree* {
(5) generate pattern $\beta = a_i \cup \alpha$ with *support_count* = a_i.*support_count*;
(6) construct β's conditional pattern base and then β's conditional FP_tree $Tree_\beta$;
(7) **if** $Tree_\beta \neq \emptyset$ **then**
(8) call FP_growth($Tree_\beta$, β); }