Session 5: FPGrowth: A Pattern Growth Approach

FPGrowth: Mining Frequent Patterns by Pattern Growth

- Idea: Frequent pattern growth (FPGrowth)
  - Find frequent single items and partition the database based on each such item
  - Recursively grow frequent patterns by doing the above for each partitioned database (also called *conditional database*)
  - To facilitate efficient processing, an efficient data structure, FP-tree, can be constructed
- Mining becomes
  - Recursively construct and mine (conditional) FP-trees
  - Until the resulting FP-tree is empty, or until it contains only one path—single path will generate all the combinations of its sub-paths, each of which is a frequent pattern

Example: Construct FP-tree from a Transational DB

TID	Items in the Transaction	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*}

1. Scan DB once, find single item frequent pattern: **Let min_support = 3**

   f:4, a:3, c:4, b:3, m:3, p:3

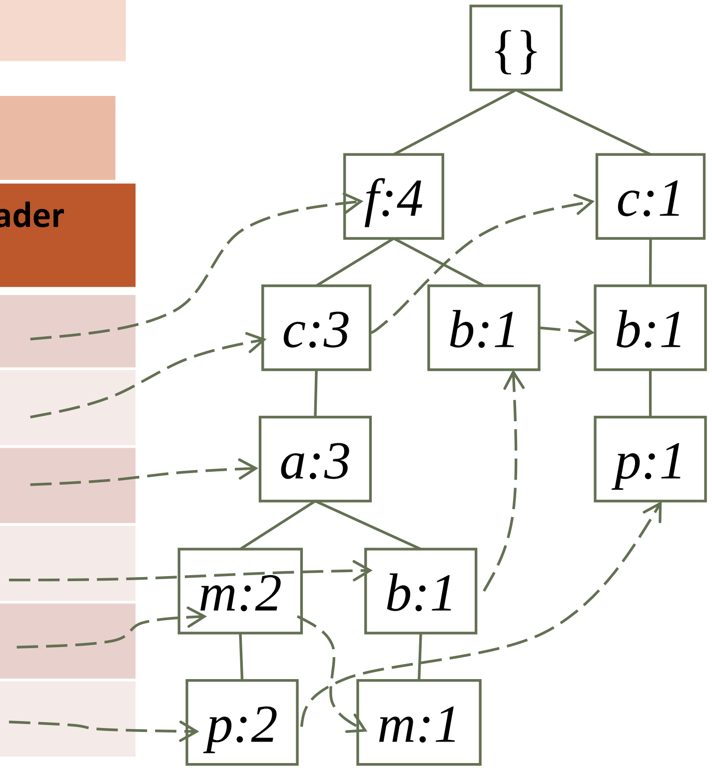
2. Sort frequent items in frequency descending order, f-list

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

3. Scan DB again, construct FP-tree

Header Table

Item	Frequency	header
f	4	
c	4	
a	3	
b	3	
m	3	
p	3	

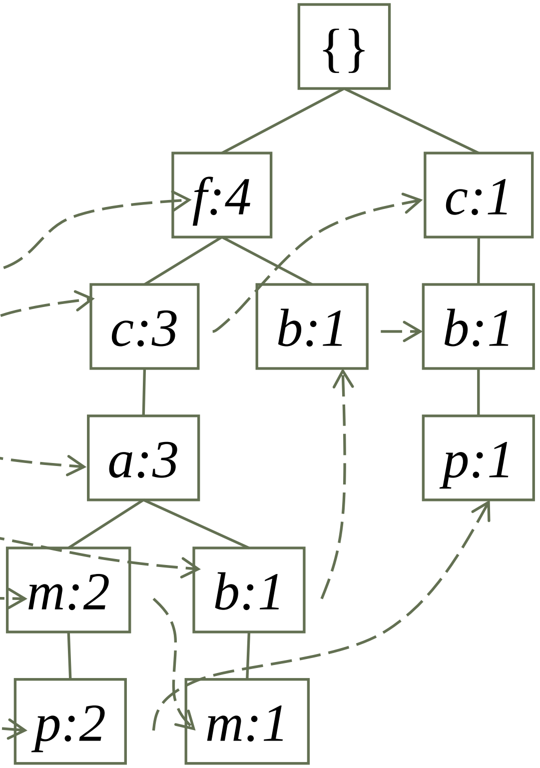


Divide and Conquer Based on Patterns and Data

- Pattern mining can be partitioned according to current patterns
  - Patterns containing p: p's conditional database: *fcam:2, cb:1*
  - Patterns having m but no p: m's conditional database: *fca:2, fcab:1*
  - …… ……
- *p*'s conditional pattern base: *transformed prefix paths* of item *p*

min_support = 3

Item	Frequency	Header
f	4	
c	4	
a	3	
b	3	
m	3	
p	3	



***Conditional* pattern bases**

Item	*Conditional 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***

Mine Each Conditional Pattern-Base Recursively

***Conditional* pattern bases**

min_support = 3

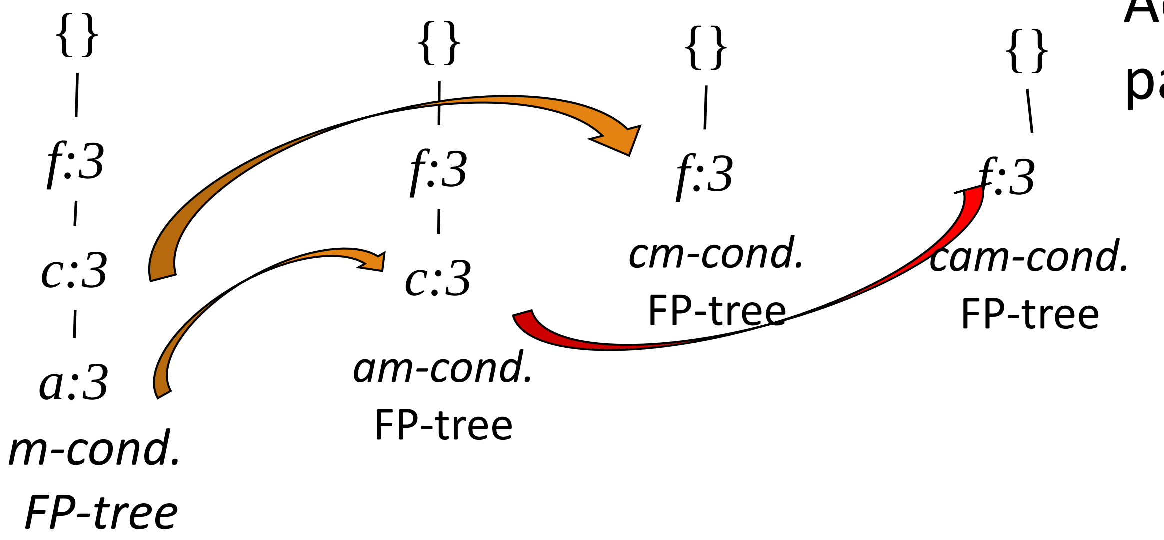
item	*cond. pattern base*
c	*f:3*
a	*fc:3*
b	*fca:1, f:1, c:1*
m	*fca:2, fcab:1*
p	*fcam:2, cb:1*

- For each conditional pattern-base
  - Mine single-item patterns
  - Construct its FP-tree & mine it

p-conditional PB: ***fcam:2, cb:1*** → ***c: 3***

m-conditional PB: ***fca:2, fcab:1*** → ***fca: 3***

b-conditional PB: ***fca:1, f:1, c:1*** → ***ϕ***

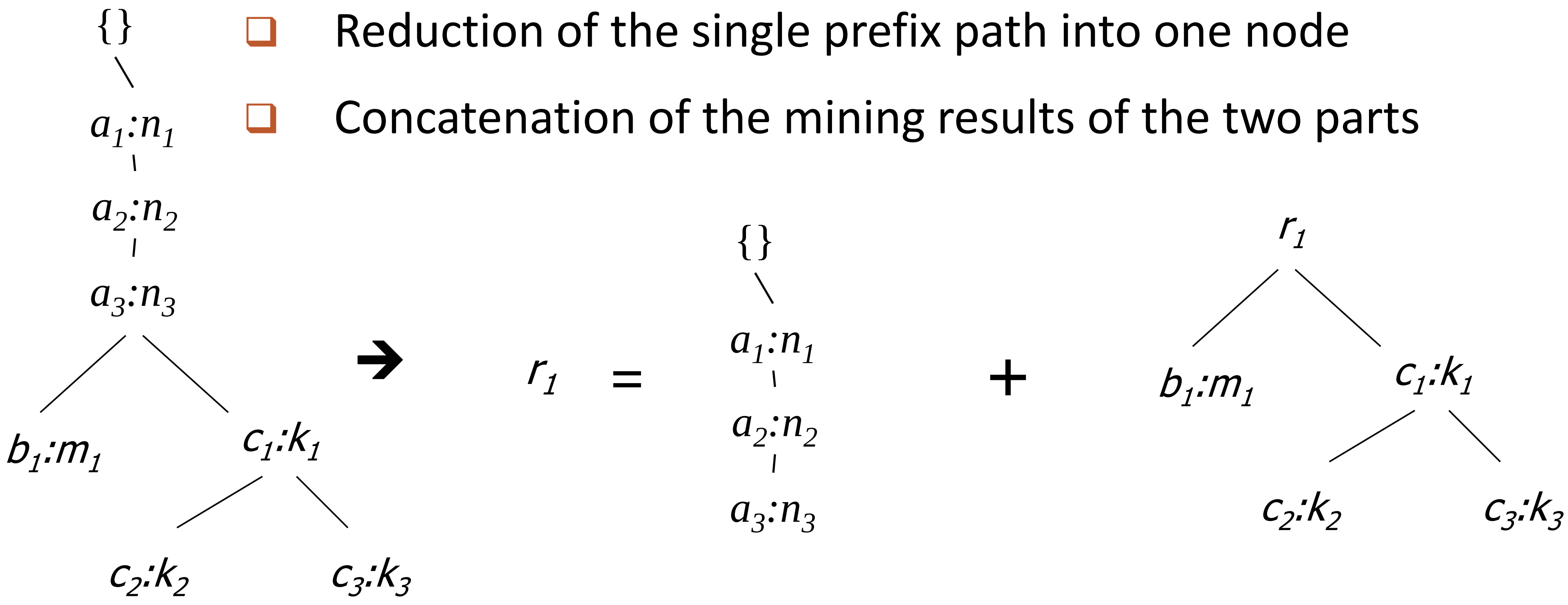


Actually, for single branch FP-tree, all frequent patterns can be generated in one shot

m: 3
fm: 3, cm: 3, am: 3
fcm: 3, fam:3, cam: 3
fcam: 3

A Special Case: Single Prefix Path in FP-tree

- Suppose a (conditional) FP-tree T has a shared single prefix-path P
- Mining can be decomposed into two parts
  - Reduction of the single prefix path into one node
  - Concatenation of the mining results of the two parts



Scaling FP-growth by Database Projection

- What if FP-tree cannot fit in memory? — DB projection
  - Project the DB based on patterns
  - Construct & mine FP-tree for each projected DB
- Parallel projection vs. partition projection
  - Parallel projection: Project the DB on each frequent item
    - Space costly, all partitions can be processed in parallel
  - Partition projection: Partition the DB in order
    - Passing the unprocessed parts to subsequent partitions

Trans. DB
f_2 f_3 f_4 g h
f_3 f_4 i j
f_2 f_4 k
f_1 f_3 h
…

Assume only f's are freq. & the freq. item ordering is: f_1-f_2-f_3-f_4

Parallel projection

f_4-proj. DB	**f_3-proj. DB**
f_2 f_3	f_2
f_3	f_1
f_2	…
…	

Partition projection

f_4-proj. DB	**f_3-proj. DB**
f_2 f_3	f_1
f_3	…
f_2	
…	

f_2 will be projected to f_3-proj. DB only when processing f_4-proj. DB