

Inf-KDDM: Knowledge Discovery and Data Mining

Winter Term 2018/19

Lecture 4: Frequent Itemsets and Association Rule Mining II

Lectures: Prof. Dr. Matthias Renz

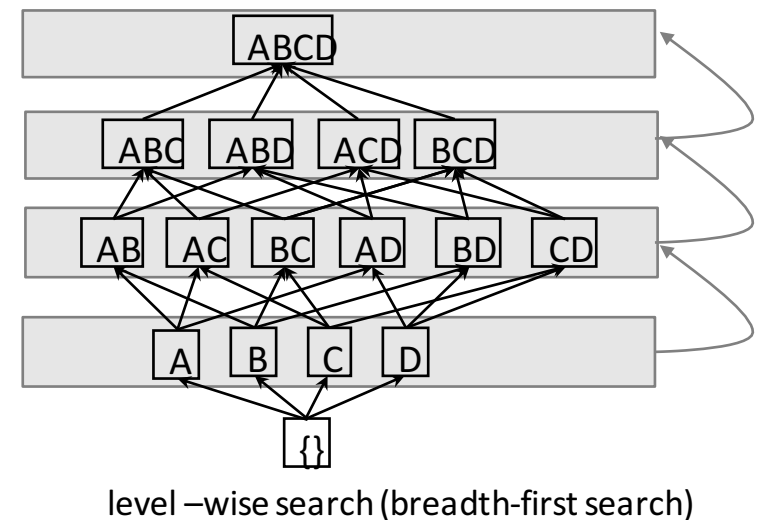
Exercises: Lukas Galke and Falk Bösch

Outline

- Apriori improvements
- Closed frequent itemsets (CFI) & Maximal frequent itemsets (MFI)
- Beyond FIM for binary data
- Things you should know from this lecture

Apriori improvements

- Major computational challenges in Apriori:
 - Multiple scans of the DB: For each level k (i.e., k -itemsets), a database scan is required
 - Huge number of candidates (first generate, then test)
 - Tedious workload of support counting for candidates
 - Too many candidates.
 - One transaction may contain many candidates.
- Improving Apriori directions:
 - Reduce passes of transaction database scans
 - Shrink number of candidates
 - Facilitate support counting of candidates
 - In this lecture:
 - (FPGrowth), Partition, Sampling, ECLAT



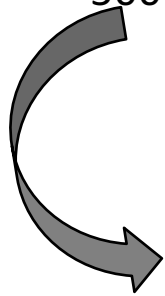
FPGrowth (Han, Pei & Yin, SIGMOD'00)

- The FPGrowth (frequent pattern growth) approach
 - Compresses the database using FP-tree, an extension of prefix-tree
 - It retains the transactions information
 - Never breaks a long pattern of any transaction
 - Depth-first search (DFS)
 - Avoids explicit candidate generation
 - Frequent itemsets are generated directly from the FP-tree.

FP-tree

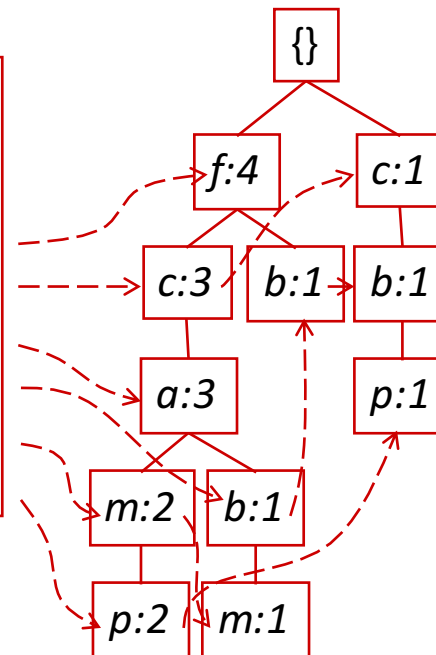
TID	Items bought	(ordered) frequent items
100	{f, a, c, d, g, i, m, p}	{f, c, a, m, p}
200	{a, b, c, f, l, m, o}	{f, c, a, b, m}
300	{b, f, h, j, o, w}	{f, b}
400	{b, c, k, s, p}	{c, b, p}
500	{a, f, c, e, l, p, m, n}	{f, c, a, m, p}

minSupport = 3



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

Header Table	
<u>Item frequency head</u>	
<i>f</i>	4
<i>c</i>	4
<i>a</i>	3
<i>b</i>	3
<i>m</i>	3
<i>p</i>	3



- Each transaction is mapped into a path in the FP-tree
- To facilitate tree traversal, each item in the header table points to its occurrences in the tree via a chain of node-links
- Most common items appear close to the root

FP-tree construction 1/2

Method:

1. Scan DB once and find the frequent 1-itemsets. Scan 1
2. Discard infrequent items
3. Sort frequent items in frequency descending order → *f*-list

<u>TID</u>	<u>Items bought</u>	<u>(ordered) frequent items</u>
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}

minSupport = 3

in our example:
a-b-c-f-m-p

in our example:
f-list = f-c-a-b-m-p

4. Scan DB again, construct FP-tree

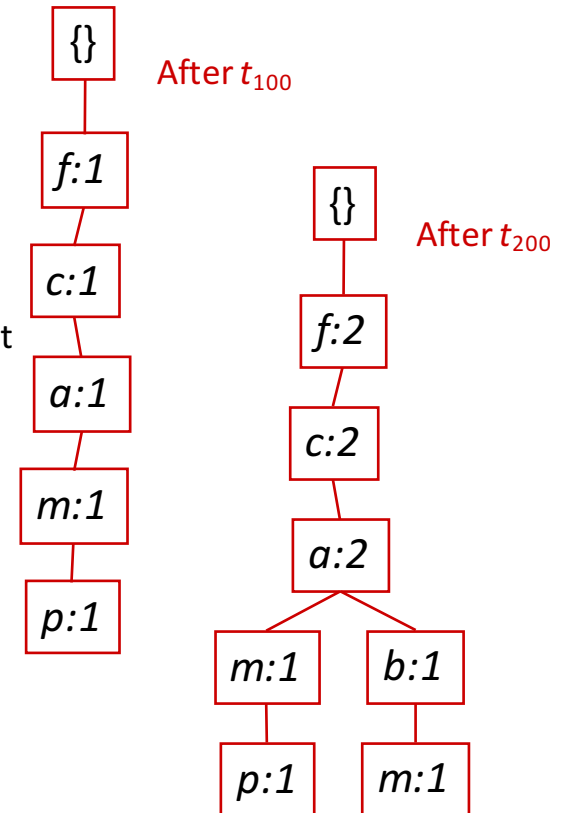
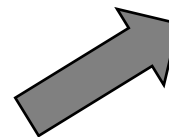
Transaction items are
accessed in *f*-list order!!!

FP-tree construction 2/2

4. (Cont') Scan DB again, construct FP-tree Scan 2

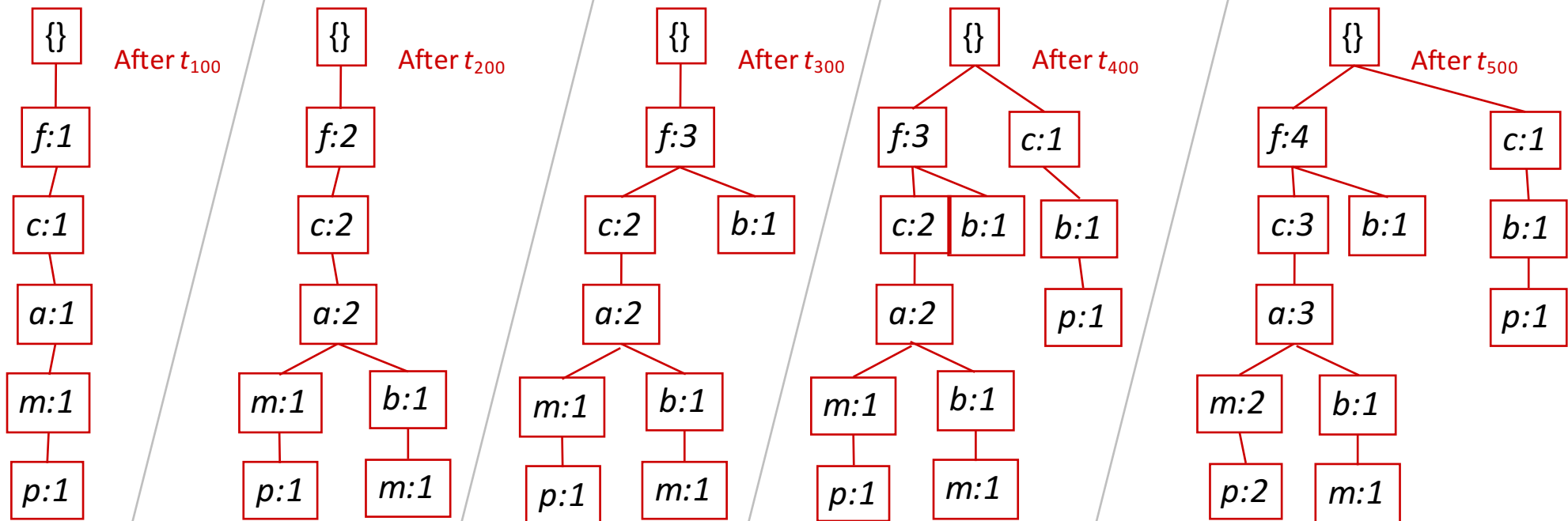
- Create the root of the tree, labeled with “*null*”
- Insert first transaction (t_{100}) in the tree
- Insert the next transaction (t_{200}) in the tree
 - If they are identical, just update the nodes along the path
 - If they share a common prefix (in our case *fca*), update nodes in the shared part (i.e., *fca*) and create a new branch for the rest of the transaction t_{200} (i.e., (*bm*)).
 - If nothing in common, start a new branch from the root
- Repeat for all transactions

TID	(ordered) frequent items
100	{f, c, a, m, p}
200	{f, c, a, b, m}
300	{f, b}
400	{c, b, p}
500	{f, c, a, m, p}



FP-tree construction step by step

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}



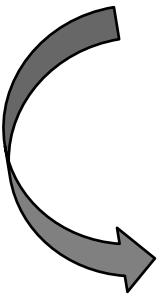
The complete FP-tree

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, 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}

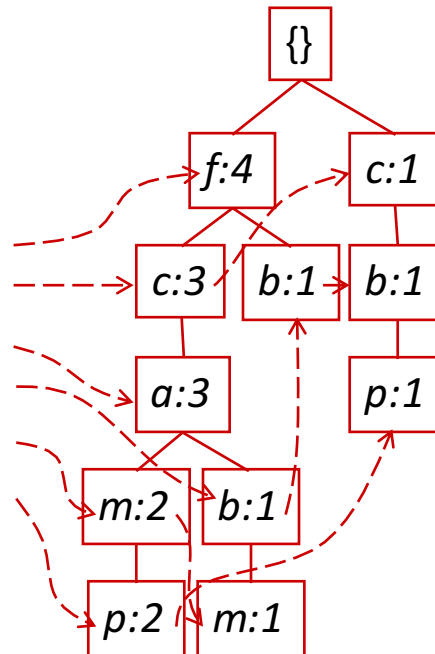
minSupport = 3

Header Table

<u>Item frequency head</u>	
<i>f</i>	4
<i>c</i>	4
<i>a</i>	3
<i>b</i>	3
<i>m</i>	3
<i>p</i>	3



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



- Each transaction is mapped into a path in the FP-tree
- To facilitate tree traversal, each item in the header table points to its occurrences in the tree via a chain of node-links
- most common items appear close to the root

Advantages of the FP-tree structure

- Completeness

- Preserves complete information for frequent pattern mining
- Never breaks a long pattern of any transaction

- Compactness

- Reduces irrelevant info—infrequent items are gone
- Items in frequency descending order (*f*-list): the more frequently occurring, the more likely to be shared
- Never is larger than the original database (not counting *node-links* and the *node-counts* fields)
- Achieves high compression ratio

① What is the best-case compression scenario for an FP-tree?

② What is the worse-case compression scenario for an FP-tree?

Frequent itemsets generation from FP-tree

- Explore the tree in a bottom-up fashion, finding all frequent itemsets ending with a particular suffix
 - Rationale: all the patterns containing frequent items that a node a_i participates can be collected by starting at a_i 's node-link head and following its node-links.
 - Start with the frequent item header table (bottom to top) in the FP-tree: first look for e, d, c, b, a
 - Traverse the FP-tree by following the link of each frequent item $e \rightarrow$ **prefix paths** ending in e

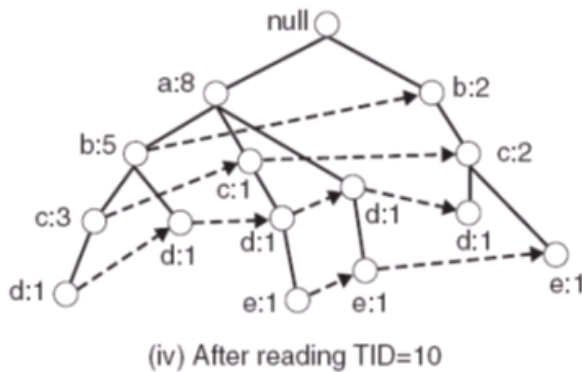


Figure 6.24. Construction of an FP-tree.

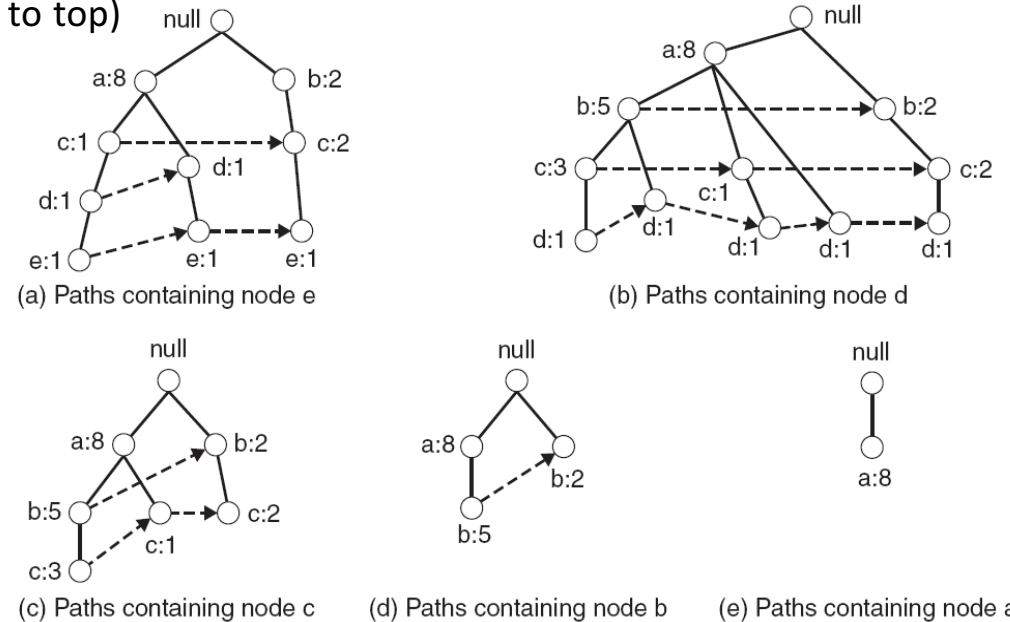


Figure 6.26. Decomposing the frequent itemset generation problem into multiple subproblems, where each subproblem involves finding frequent itemsets ending in $e, d, c, b,$ and a .

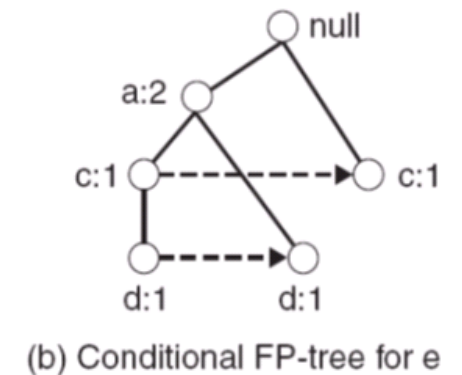
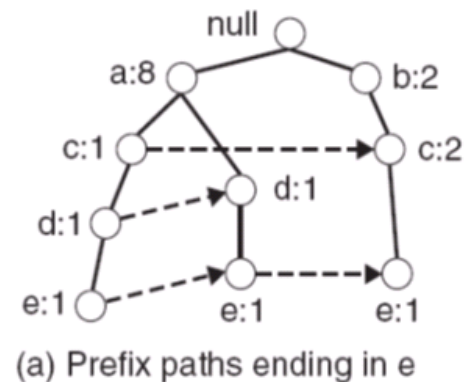
From prefix paths ending in e to conditional FP-tree for e

- Start with prefix paths ending in e
 - Update the support counts because some contain transactions without e .
 - E.g., {null-b:2-c:2-e:1} includes a transaction bc that does not contain e
 - Truncate the prefix paths by removing nodes for e
 - Remove non-frequent nodes (due to support update)
 - E.g., b:1 is not frequent anymore

■ The result is e' conditional pattern base

■ Conditional FP-tree

- similar to FP-tree but encodes items ending with a specific suffix, e in our case



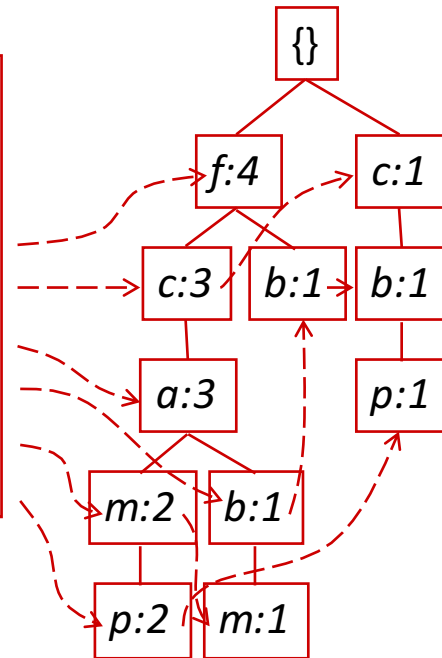
Conditional pattern bases

TID	(ordered) frequent items
100	{f, c, a, m, p}
200	{f, c, a, b, m}
300	{f, b}
400	{c, b, p}
500	{f, c, a, m, p}

Header Table

Item frequency head

<i>f</i>	4
<i>c</i>	4
<i>a</i>	3
<i>b</i>	3
<i>m</i>	3
<i>p</i>	3



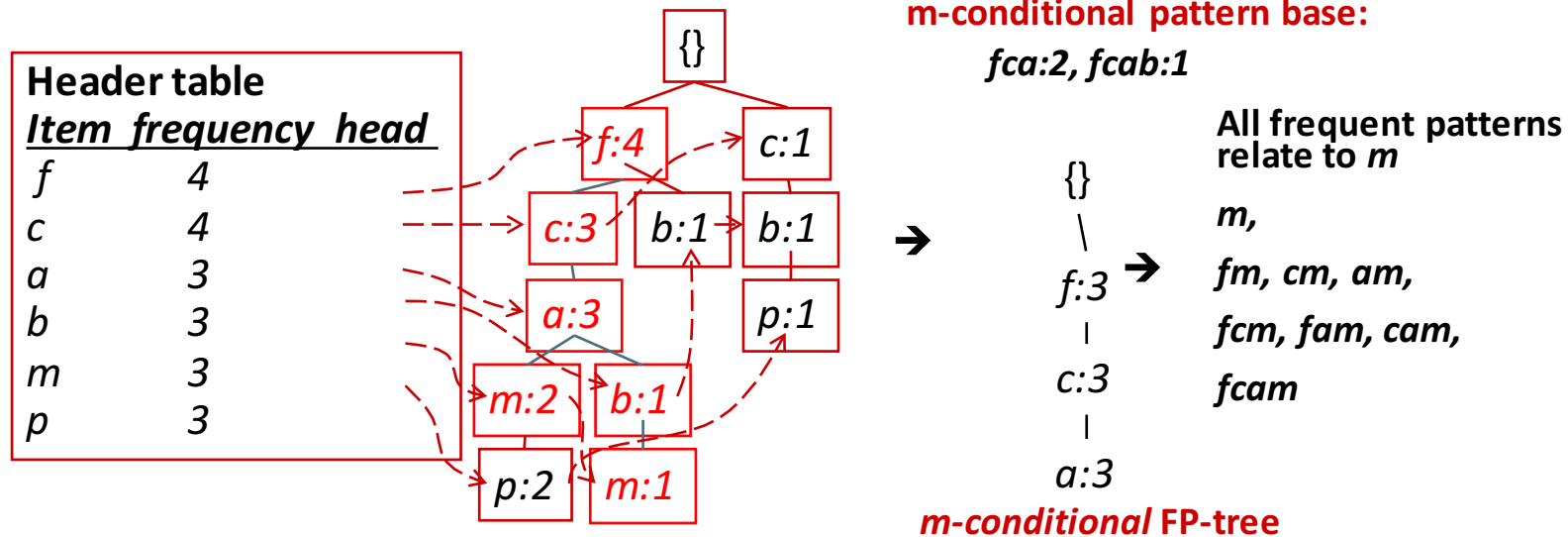
Conditional pattern bases

item cond. pattern base

<i>c</i>	<i>f</i> :3
<i>a</i>	<i>fc</i> :3
<i>b</i>	<i>fca</i> :1, <i>f</i> :1, <i>c</i> :1
<i>m</i>	<i>fca</i> :2, <i>fcab</i> :1
<i>p</i>	<i>fcam</i> :2, <i>cb</i> :1

Conditional FP-trees

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



FP-Growth: Frequent itemsets generation from FP-tree

FP-Growth procedure

- Starting with the least frequent item, construct its conditional pattern-base, and then its conditional FP-tree
- After removing the infrequent items from the conditional FP-tree, retrieve all the frequent itemsets that involves item x.
- Repeat the above process for each frequent item in the order of increasing frequency (i.e. accessing the items of the Header table bottom up).