C | A | U Christian-Albrechts-Universität zu Kiel Technische Fakultät

Inf-KDDM: Knowledge Discovery and Data Mining

Winter Term 2019/20

Lecture 4: Frequent Itemsets and Association Rule Mining II

Lectures: Prof. Dr. Matthias Renz

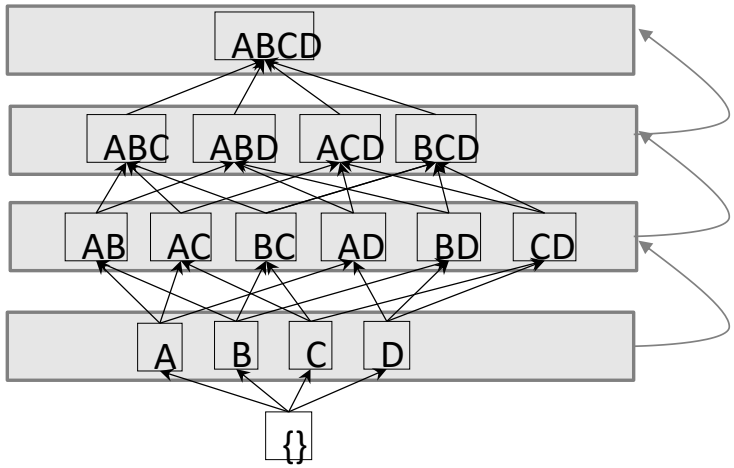
Exercises: Christian Beth

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



level –wise search (breadth-first search)

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

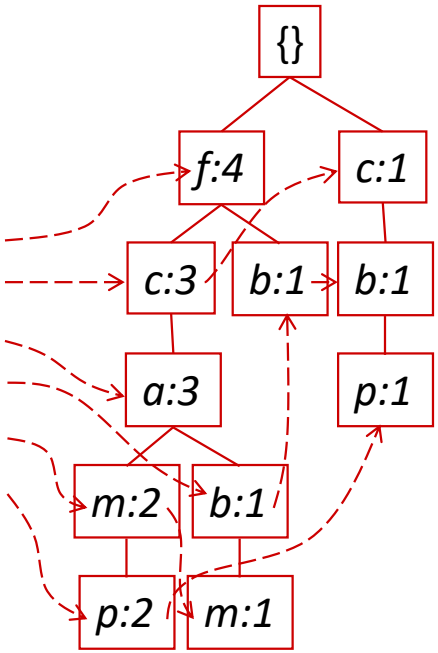
TID	Items bought	(ordered) frequent items
100	{f, a, c, d, g, i, m, p}	{f, c, a, m, p}
200	{a, b, c, f, l, m, o}	{f, c, a, b, m}
300	{b, f, h, j, o, w}	{f, b}
400	{b, c, k, s, p}	{c, b, p}
500	{a, f, c, e, l, p, m, n}	{f, c, a, m, p}

minSupport = 3

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

Header Table

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

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

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

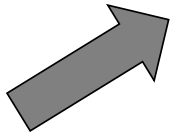
Transaction items are accessed in *f*-list order!!!

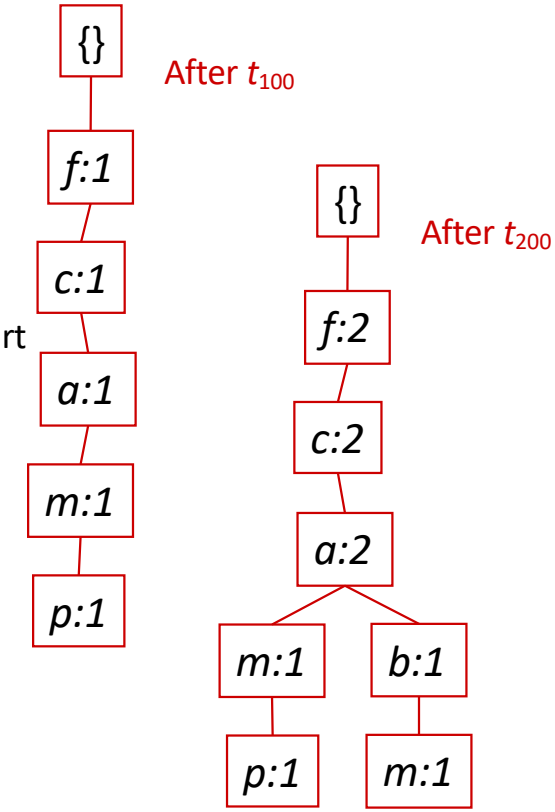
FP-tree construction 2/2

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

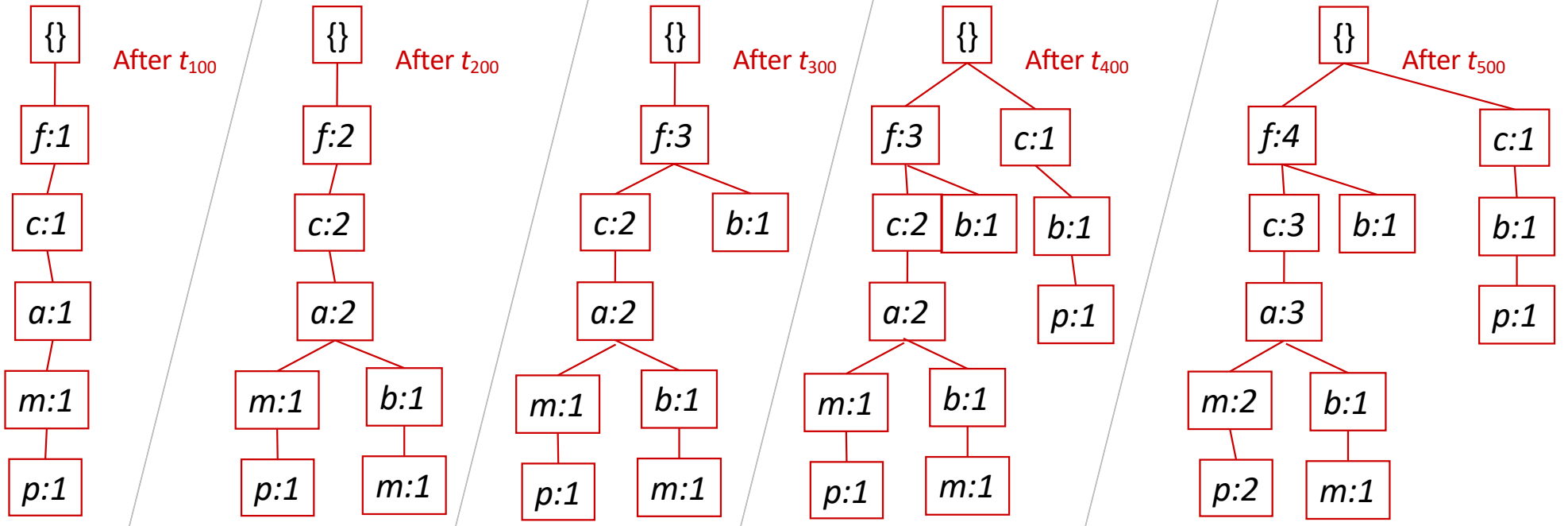
TID	(ordered) frequent items
100	{f, c, a, m, p}
200	{f, c, a, b, m}
300	{f, b}
400	{c, b, p}
500	{f, c, a, m, p}





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

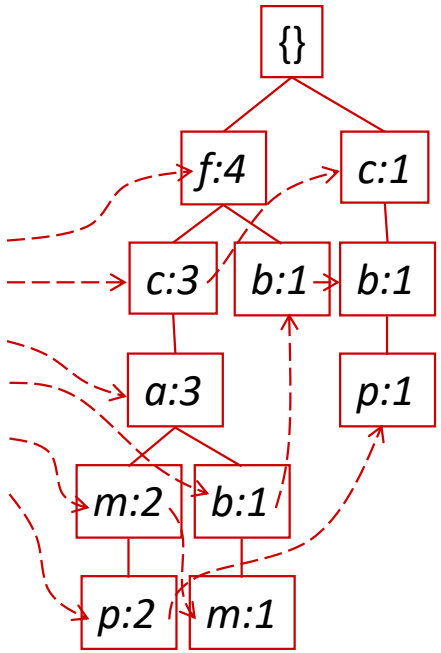
TID	Items bought	(ordered) frequent items
100	{f, a, c, d, g, i, m, p}	{f, c, a, m, p}
200	{a, b, c, f, l, m, o}	{f, c, a, b, m}
300	{b, f, h, j, o, w}	{f, b}
400	{b, c, k, s, p}	{c, b, p}
500	{a, f, c, e, l, p, m, n}	{f, c, a, m, p}

minSupport = 3

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

Header Table

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

Advantages of the FP-tree structure

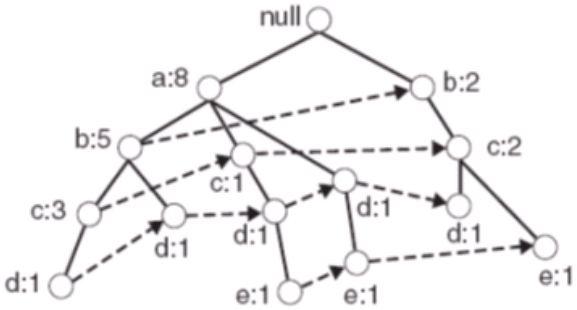
- Completeness
  - Preserves complete information for frequent pattern mining
  - Never breaks a long pattern of any transaction
- Compactness
  - Reduces irrelevant info—infrequent items are gone
  - Items in frequency descending order (*f*-list): the more frequently occurring, the more likely to be shared
  - Never is larger than the original database (not counting *node-links* and the *node-counts* fields)
  - Achieves high compression ratio


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


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

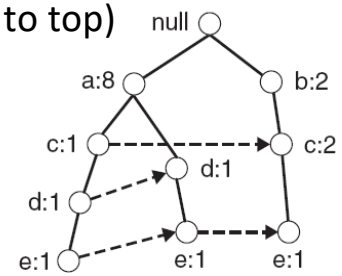
Frequent itemsets generation from FP-tree

- 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* → prefix paths ending in *e*

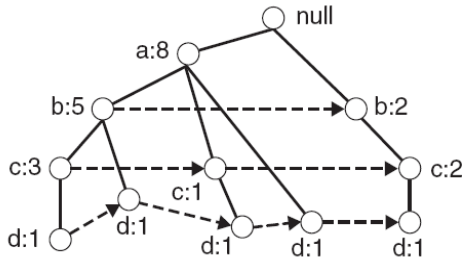


(iv) After reading TID=10

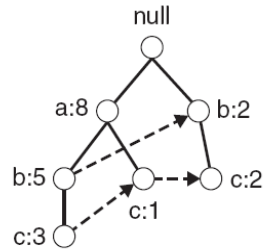
Figure 6.24. Construction of an FP-tree.



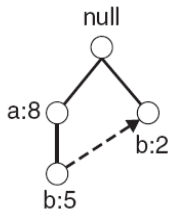
(a) Paths containing node e



(b) Paths containing node d



(c) Paths containing node c



(d) Paths containing node b

(e) Paths containing node a

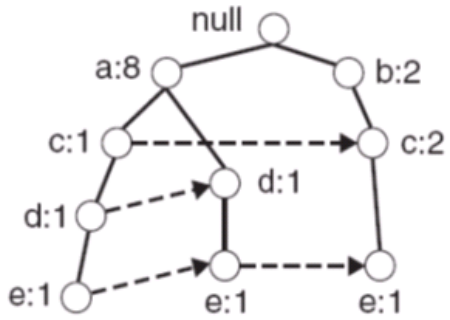
Figure 6.26. Decomposing the frequent itemset generation problem into multiple subproblems, where each subproblem involves finding frequent itemsets ending in e, d, c, b, and a.

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

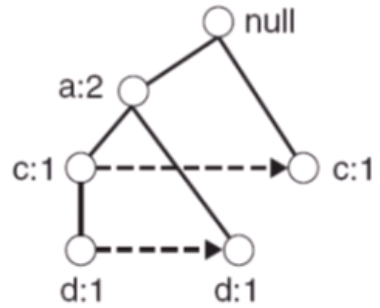
- Start with prefix paths ending in *e*
  - Update the support counts because some contain transactions without e.
    - E.g., {null-b:2-c:2-e:1} includes a transaction bc that does not contain e
  - Truncate the prefix paths by removing nodes for e
  - Remove non-frequent nodes (due to support update)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



(a) Prefix paths ending in e



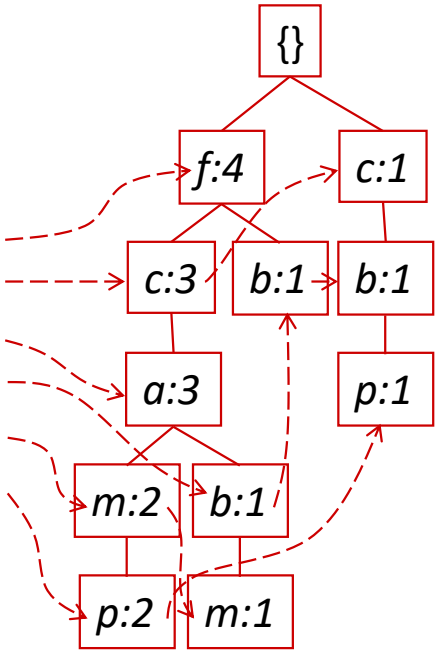
(b) Conditional FP-tree for e

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*
f	*4*	
c	*4*	
a	*3*	
b	*3*	
m	*3*	
p	*3*	

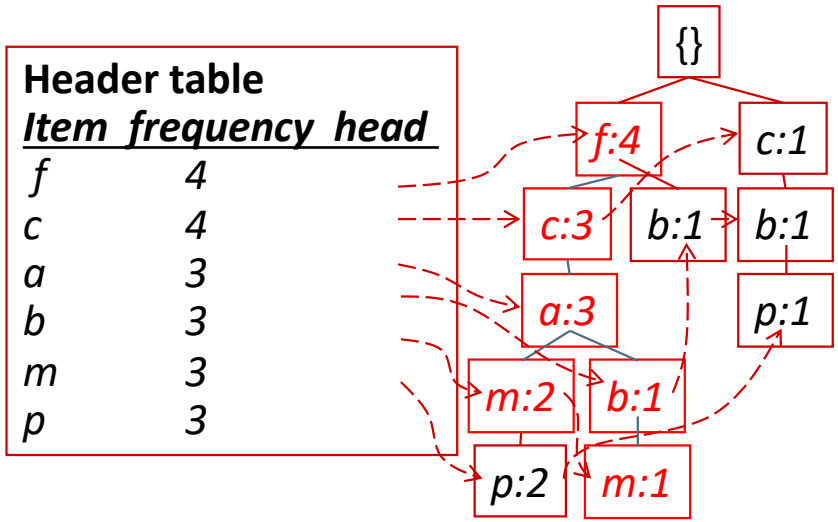


Conditional pattern bases

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*

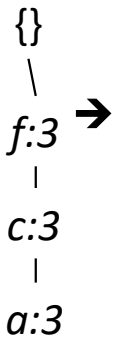
Conditional FP-trees

- 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



m-conditional pattern base:

fca:2, fcab:1



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

All frequent patterns relate to *m*

m,

fm, cm, am,

fcm, fam, cam,

fcam

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).

Partition (Savasere, Omiecinski and Navathe, *VLDB'95) 1/3*

- The reason the *DB* needs to be scanned multiple times with Apriori is because the number of possible itemsets to be tested for support is exponential in the number of items, if it must be done in a single scan of the database.
- Idea: If we had a small set of potentially frequent itemsets, we could test their support in a single scan of the database to discover the final real frequent itemsets.
  - ➔partition the database into smaller partitions that fit in main memory

Partition (Savasere, Omiecinski and Navathe, *VLDB'95) 2/3*

- Partition the *DB* into *n* non-overlapping partitions: DB_1, DB_2, …, DB_n
- Apply Apriori in each partition DB_i→ extract *local frequent itemsets*
  - local absolute *minSupport* threshold in DB_i: relative *minSupport* * $|DB_i|$

Scan 1

- Idea: Any itemset that is potentially frequent in DB must be frequent in at least one of the partitions of DB!
- The set of local frequent itemsets forms the *global candidate itemsets*
  - This is a superset of the actual global frequent itemsets, i.e., it may contain **false positives**.
- Find the actual support of each candidate → *global frequent itemsets*

Scan 2

Partition (Savasere, Omiecinski and Navathe, *VLDB'95) 3/3*

- Pseudocode

```
1.  Divide D into partitions D^1,D^2,…,D^p;
2.  For i = 1 to p do
3.      L^i = Apriori(D^i);                  // 1st pass
4.  C = L^1 ∪ … ∪ L^p;
5.  Count C on D to generate L;              // 2nd pass
```

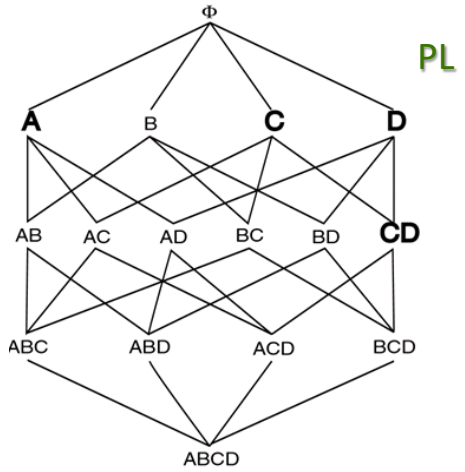
- Advantages:
  - DB partitions to be scanned adapted to fit in main memory size
  - parallel execution of DB scan enabled
  - 2 scans in DB in total to retrieve all frequent itemsets
- Disadvantages:
  - # candidates in 2nd scan might be large

Sampling (Toinoven, VLDB'96) 1/4

- Idea: Pick a random sample, find the frequent itemsets in the sample and verify the results with the rest of the database.

- Select a random sample *S* of *DB* (that fits in memory)
- Apply Apriori to the sample S

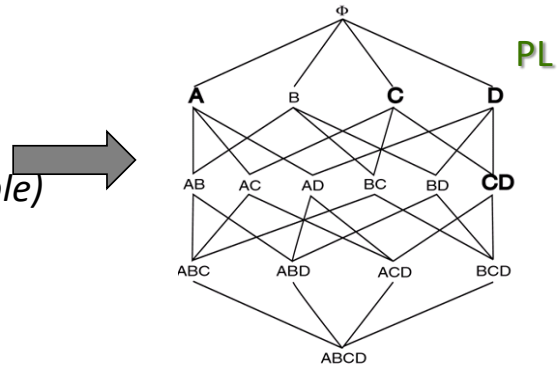
  → *PL* *(potential frequent itemsets from sample)*



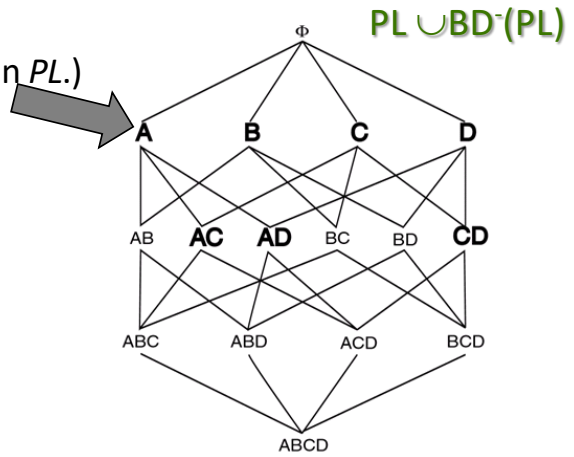
- But, since we search only in *S*, we might miss some global frequent itemsets
- Idea: Expand PL by applying the **negative border**
  - negative border: minimal set of itemsets which are not in *PL*, but whose subsets are all in *PL*.

Sampling (Toinoven, VLDB'96) 2/4

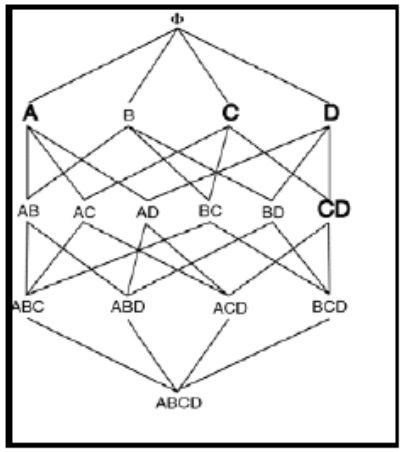
- PL from sample
  - Select a random sample *S* of *DB* (that fits in memory)
  - Apply Apriori to the sample → *PL* *(potential frequent itemsets from sample)*



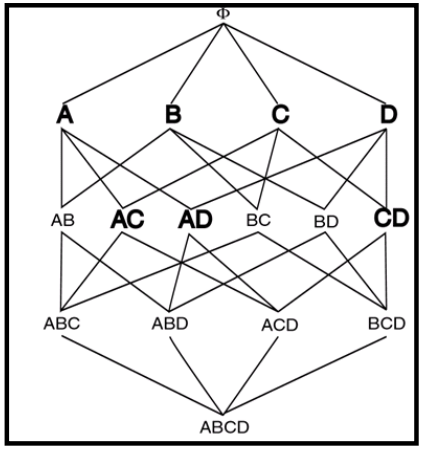
- Expand PL by applying the negative border
  - Candidate set $C = PL \cup BD^-(PL)$:
    - $BD^-(PL)$: negative border (minimal set of itemsets which are not in *PL*, but whose subsets are all in *PL*.)
  - Count the support of *C* itemsets in entire *DB*
  - If there are frequent itemsets in BD^- (by *minSupport*), expand *C* by repeatedly applying BD^-
  - Finally, count *C* in *DB*



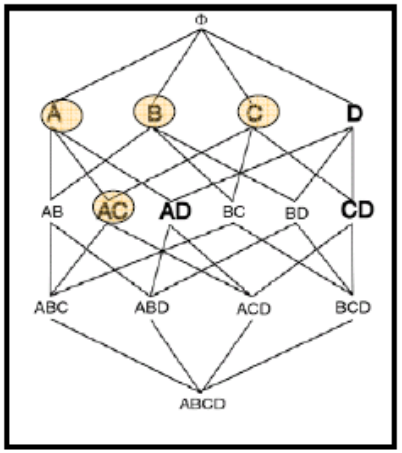
Sampling (Toinoven, VLDB'96) 3/4



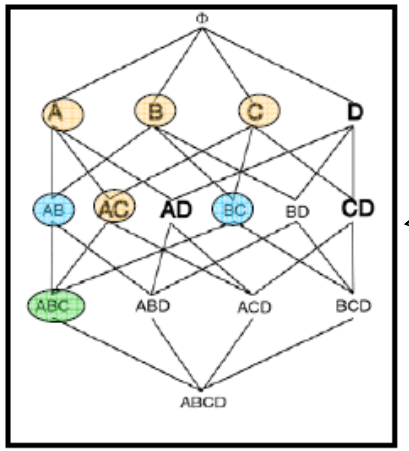
PL



C= PL ∪BD^{-}(PL)



ML:frequent itemsets



Repeated application of BD^{-}

Sampling (Toinoven, VLDB'96) 4/4

- Pseudocode

```
1.   Ds = sample of Database D;
2.   PL = Frequent itemsets in Ds using small |Ds|;
3.   C = PL ∪ BD⁻(PL);
4.   Count C in Database D; (database scan)
5.   ML = Frequent itemsets in C;
6.   repeat
7.      ML = ML ∪ BD⁻(ML); (repeated application of BD⁻)
8.   until ML does not grow;
9.   Count sets (ML-C) in Database; (database scan)
```

- Advantages:
  - Scales better than Apriori and Partition.
  - Reduces the number of database scans to 1 in the best case and 2 in worst.
- Disadvantage:
  - Provides approximate result (with error bound guaranty)

Vertical vs horizontal transaction representation

Horizontal layout: (*TID, item set*)

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 layout: *(item, TID set)*

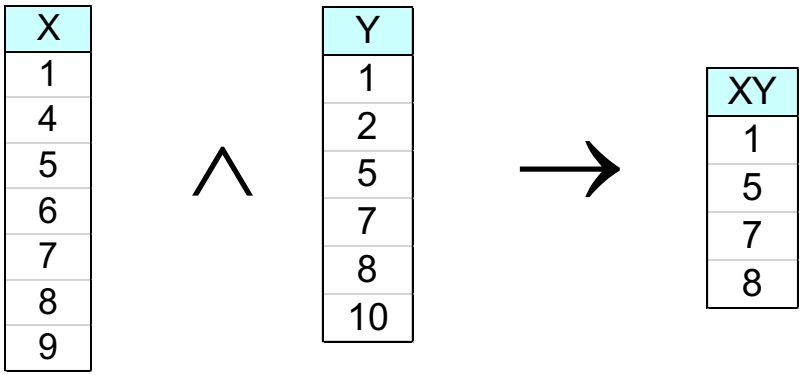
Vertical Data Layout

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				

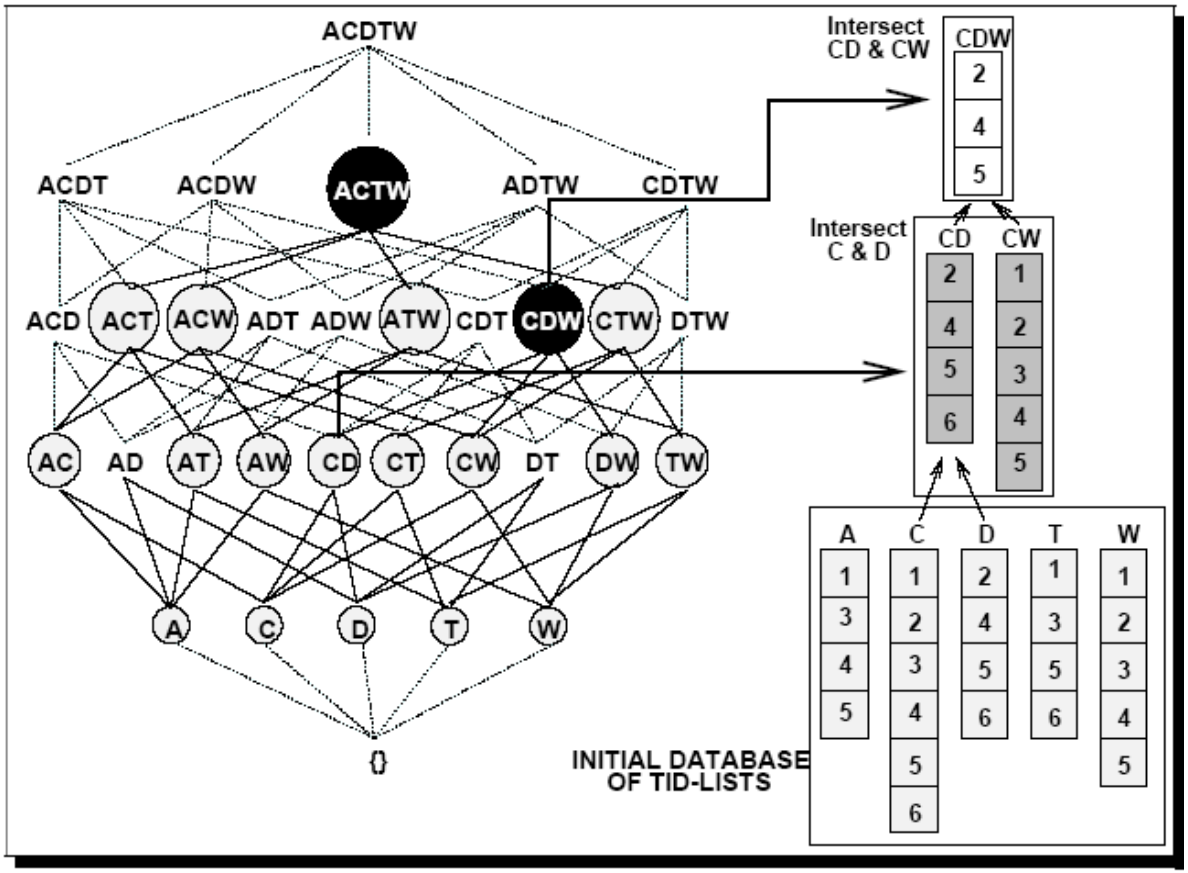
TID-list

Eclat (Zaki, TKDE'00)

- Vertical data layout
- For each itemset *X*, a list of the transaction *ids* that contain *X* is maintained:
  - *X*.tidlist ={t_1, t_4, t_5, t_6, t_7, t_8, t_9}; Y.tidilist={t_1, t_2, t_5, t_7, t_8,t_{10}}
- To find the support of *X Y*, we use their lists intersection:
  - *X*.tidlist ∩Y.tidilist={t_1, t_5, t_7, t_8}
  - Support(XY)=| *X*.tidlist ∩ Y.tidilist |=4



Example ECLAT



Eclat (Zaki, TKDE'00)

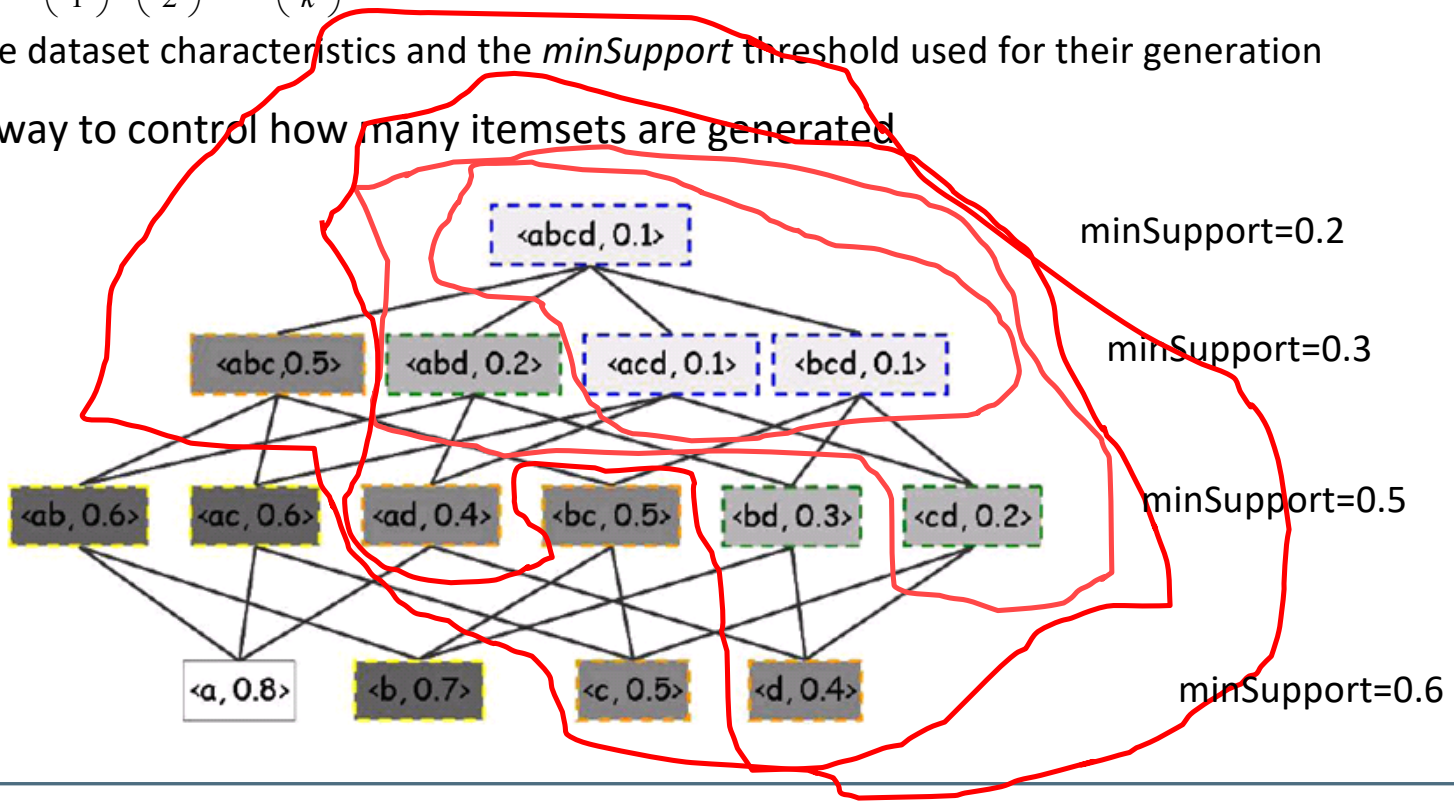
- Advantage:
  - No need to access the DB (use instead lists intersection)
  - Very fast support counting (using the lists intersection)
    - As we proceed, the size of the lists decreases, so intersection computation is faster
- Disadvantage:
  - intermediate tid-lists may become too large for memory

Outline

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

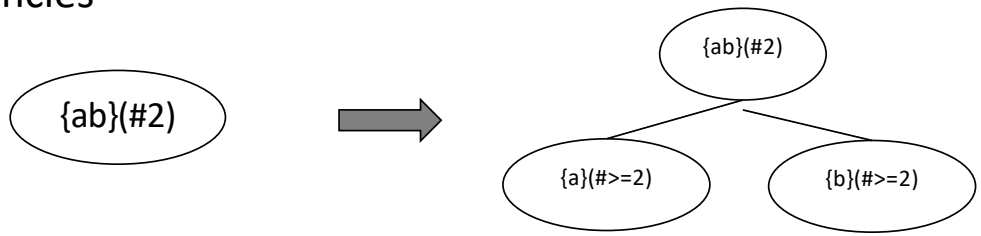
Too many frequent itemsets

- The number of frequent itemsets (FI) is too large
  - Worst-case: $\binom{|I|}{1}+\binom{|I|}{2}+\ldots+\binom{|I|}{k}=2^{|I|}-1$
  - depends on the dataset characteristics and the *minSupport* threshold used for their generation
- *minSupport* is a way to control how many itemsets are generated



Too many frequent itemsets

- Again though the resulting lattice depicts redundancies
  - Structural (i.e., in terms of itemsets items)
  - Measural (i.e., in terms of itemsets' support)



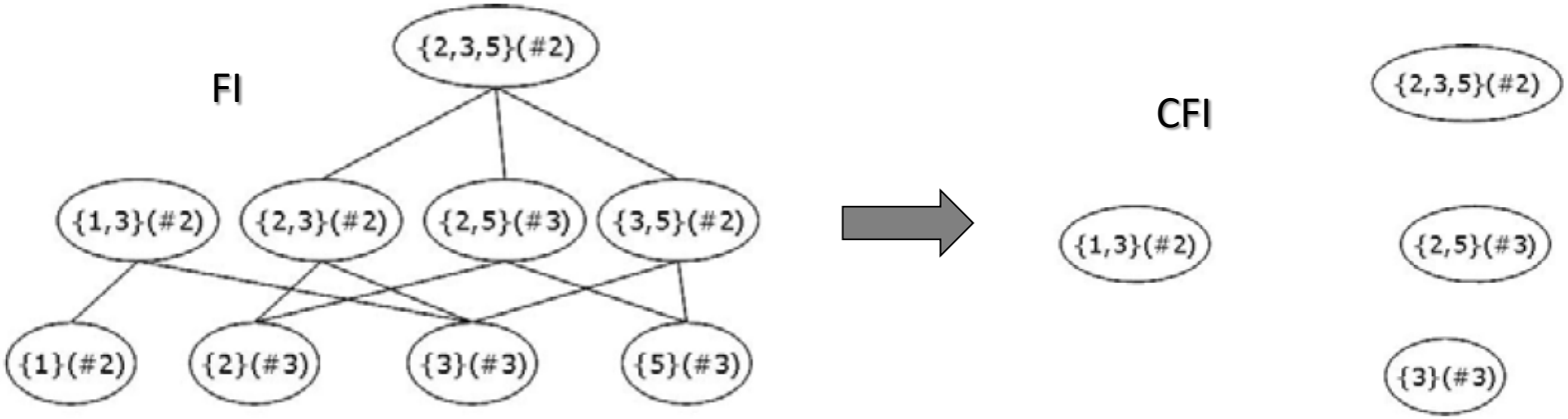
- It is useful to identify a small representative set of itemsets from which all other itemsets can be derived
- Two compressed representations
  - Closed frequent itemsets (CFI)
  - Maximal frequent itemsets (MFI)

Closed Frequent Itemsets (CFI)

A frequent itemset X is called closed if there exists no frequent superset Y ⊇ X with:

support (X) =support(Y)

- The set of closed frequent itemsets is denoted by CFI
- CFIs comprise a lossless representation of the FIs since no information is lost, neither in structure (itemsets), nor in measure (support).



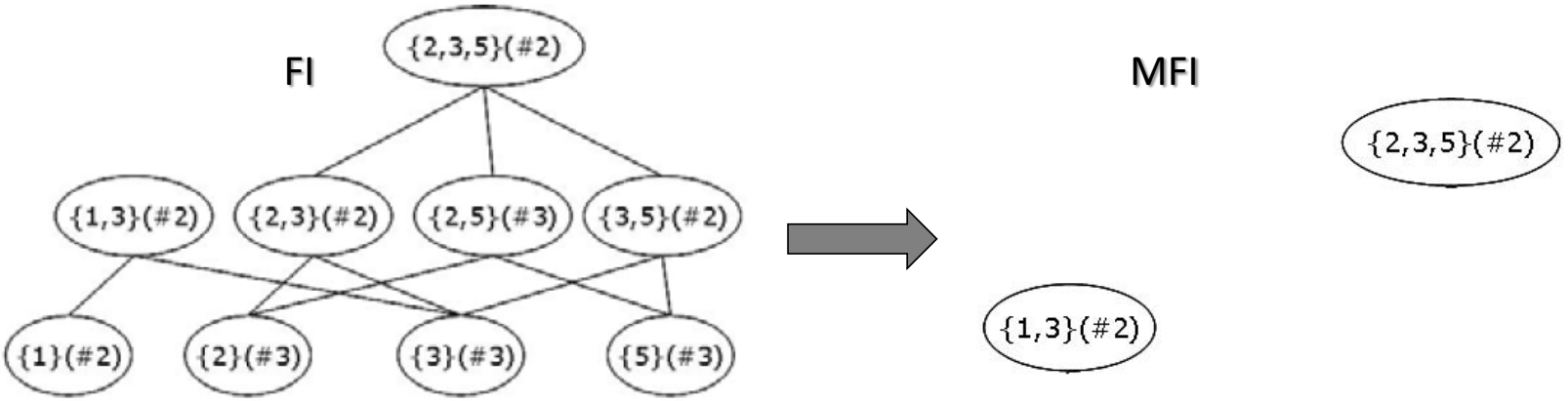
Why {2,3} is not closed?

Why {2,5} is closed?

Maximal Frequent Itemsets (MFI)

A frequent itemset is called maximal if it is not a subset of any other frequent itemset.

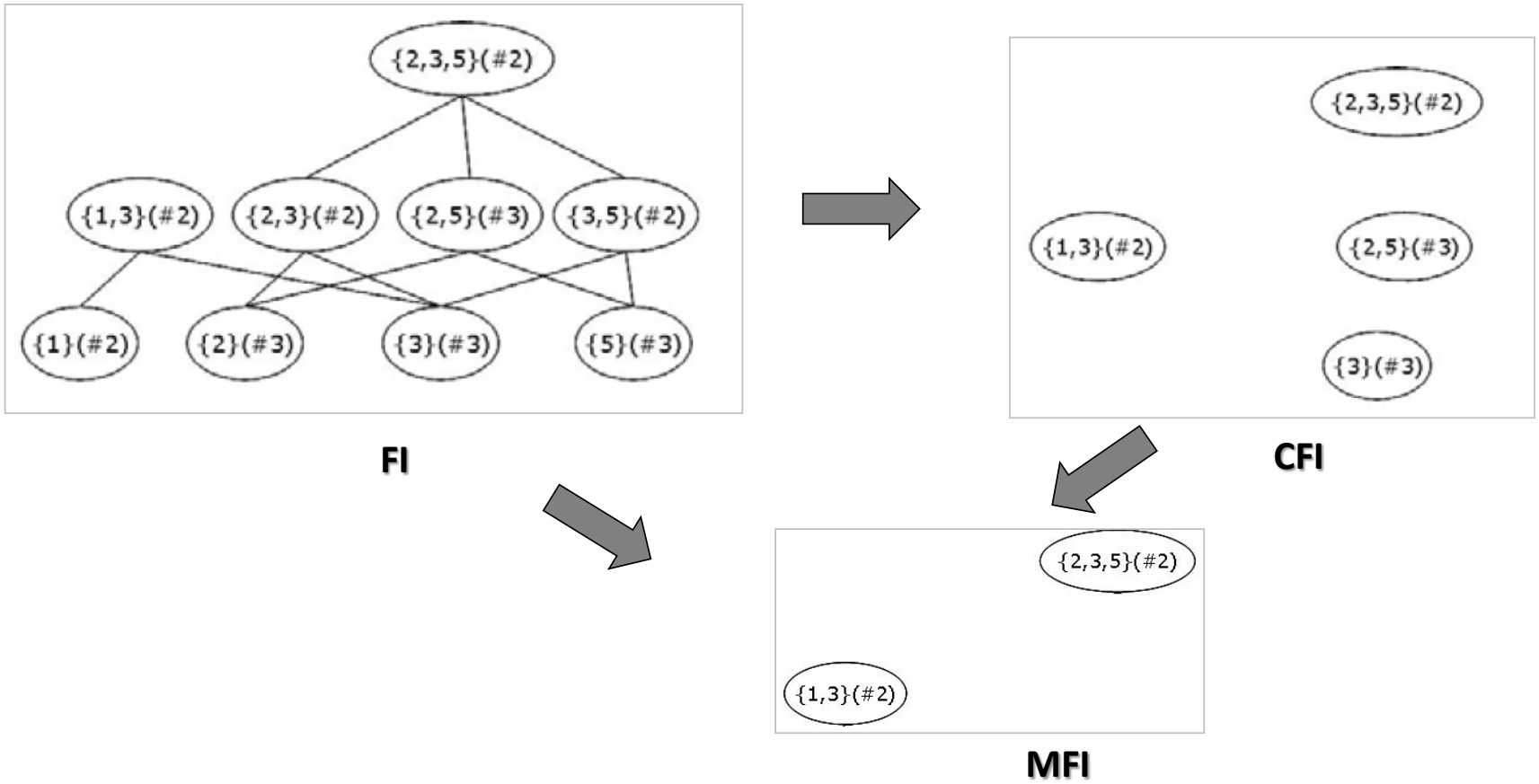
- The set of maximal frequent itemsets is denoted by MFI
- MFIs comprise a lossy representation of the FIs since it is only the lattice structure (i.e., frequent itemsets) that can be determined from MFIs whereas frequent itemsets supports are lost.



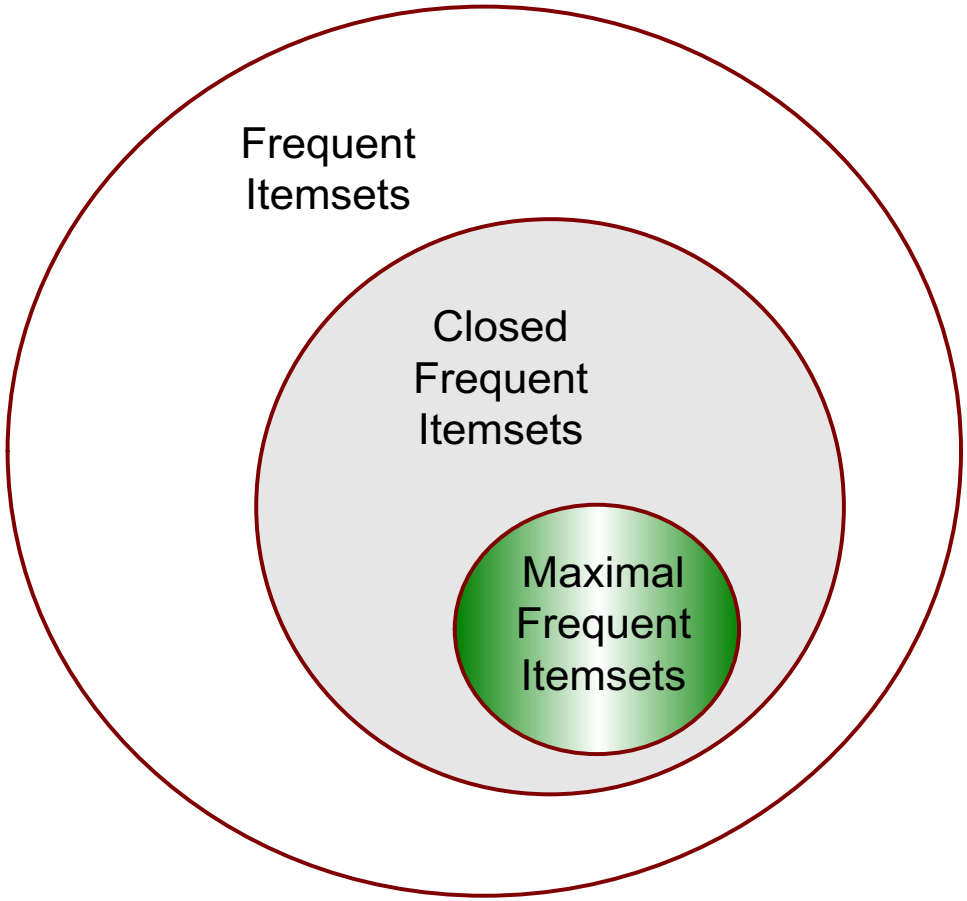
Why {1,3} is maximal?

Why {2,3} is not maximal? Why {2,5} is not maximal?

FIs vs CFIs vs MFIs



FIs vs CFIs vs MFIs



Outline

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

Thus far, FIM and ARM for binary, asymmetric data

- Binary
  - we only model the existence of an item in a transaction, e.g., t_1={A,B,C}
- Asymmetric
  - outcomes (i.e, {0,1} values) are not equally important
  - 1, i.e., the existence of an item in a transaction, is the most important

Tid	Transaction items
1	Butter, Bread, Milk, Sugar
2	Butter, Flour, Milk, Sugar
3	Butter, Eggs, Milk, Salt
4	Eggs
5	Butter, Flour, Milk, Salt, Sugar

Beyond FIM for binary data: Categorical attributes

- How to apply association analysis formulation to non-asymmetric / non-binary data ?

Gender	Level of Education	State	Computer at Home	Online Auction	Chat Online	Online Banking	Privacy Concerns
Female	Graduate	Illinois	Yes	Yes	Daily	Yes	Yes
Male	College	California	No	No	Never	No	No
Male	Graduate	Michigan	Yes	Yes	Monthly	Yes	Yes
Female	College	Virginia	No	Yes	Never	Yes	Yes
Female	Graduate	California	Yes	No	Never	No	Yes
Male	College	Minnesota	Yes	Yes	Weekly	Yes	Yes
Male	College	Alaska	Yes	Yes	Daily	Yes	No
Male	High School	Oregon	Yes	No	Never	No	No
Female	Graduate	Texas	No	No	Monthly	No	No
...	...	...	...	...	...	...	...

Example of an association rule:

{Level of Education=Graduate, Online Banking=Yes} $\rightarrow$ *{Privacy Concerns = Yes}*

Handling categorical variables

- Transform categorical attributes into (asymmetric) binary variables
- Introduce a new “item” for each distinct attribute-value pair
  - Avoid generating sets with >1 item of same attribute

Male	Female	Education = Graduate	Education = College	Education = High School	···	Privacy = Yes	Privacy = No
0	1	1	0	0	···	1	0
1	0	0	1	0	···	0	1
1	0	1	0	0	···	1	0
0	1	0	1	0	···	1	0
0	1	1	0	0	···	1	0
1	0	0	1	0	···	1	0
1	0	0	0	0	···	0	1
1	0	0	0	1	···	0	1
0	1	1	0	0	···	0	1
···	···	···	···	···	···	···	···

Beyond FIM for binary data: Continous attributes

- How to apply association analysis formulation to non-asymmetric / non-binary data ?

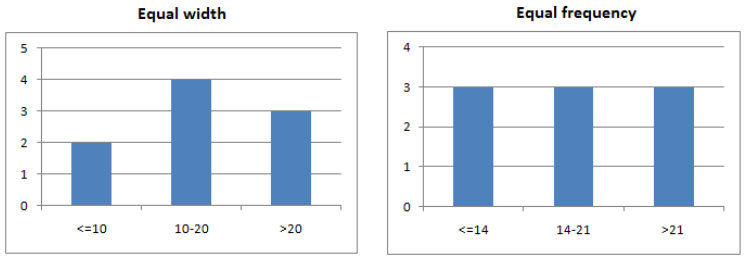
Gender	···	Age	Annual Income	No of hours spent online per week	No of email accounts	Privacy Concern
Female	···	26	90K	20	4	Yes
Male	···	51	135K	10	2	No
Male	···	29	80K	10	3	Yes
Female	···	45	120K	15	3	Yes
Female	···	31	95K	20	5	Yes
Male	···	25	55K	25	5	Yes
Male	···	37	100K	10	1	No
Male	···	41	65K	8	2	No
Female	···	26	85K	12	1	No
···	···	···	···	···	···	···

Example of an association rule:

$\{Age \in [21,30), No\_of\_hours\_online \in [10,20)\} \rightarrow \{Chat\_Online = Yes\}$

Handling continuous attributes

- Discretization
  - Equal-width binning
  - Equal-depth binning
  - …

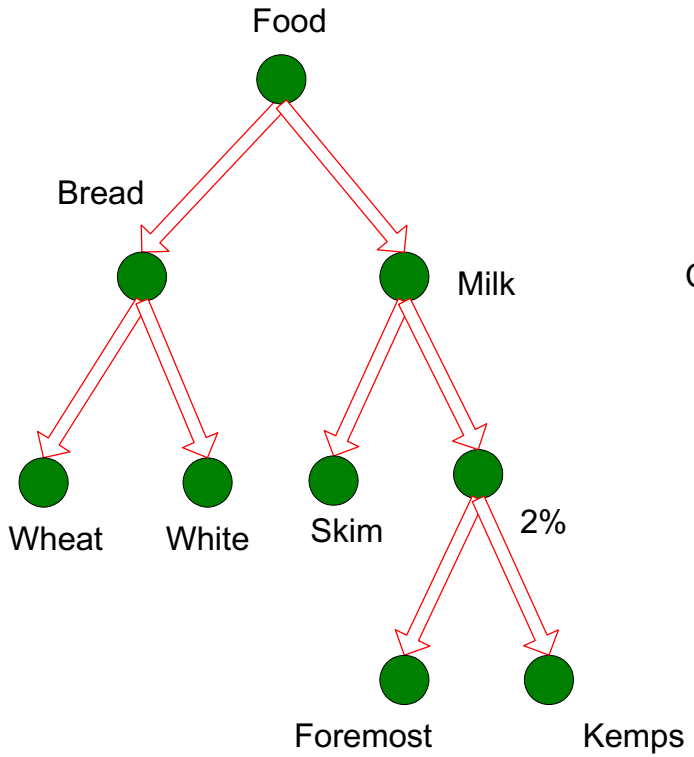


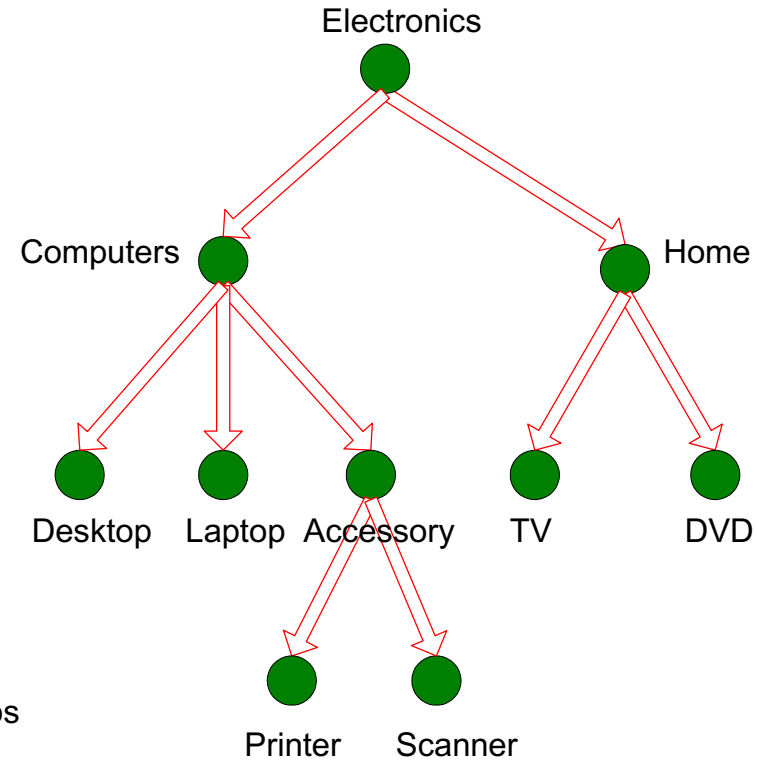
Source: http://www.saedsayad.com/images/Binning_2.png

Male	Female	⋯ ⋯	Age < 13	Age $\in [13, 21)$	Age $\in [21, 30)$	⋯ ⋯	Privacy = Yes	Privacy = No
0	1	⋯	0	0	1	⋯	1	0
1	0	⋯	0	0	0	⋯	0	1
1	0	⋯	0	0	1	⋯	1	0
0	1	⋯	0	0	0	⋯	1	0
0	1	⋯	0	0	0	⋯	1	0
1	0	⋯	0	0	1	⋯	1	0
1	0	⋯	0	0	0	⋯	0	1
1	0	⋯	0	0	0	⋯	0	1
0	1	⋯	0	0	1	⋯	0	1
⋯	⋯	⋯	⋯	⋯	⋯	⋯	⋯	⋯

Multi-level frequent itemsets

- Based on concept hierarchies like





Multi-level frequent itemsets

- Why should we incorporate concept hierarchy?
  - Rules at lower levels may not have enough support to appear in any frequent itemsets
  - Rules at lower levels of the hierarchy are overly specific
    - e.g., skim milk → white bread, 2% milk → wheat bread, skim milk → wheat bread, etc.
      are indicating an association between milk and bread, i.e., {milk}→{bread}
  - Rules at higher level of hierarchy may be too generic, e.g., {Food}→{Household items}
- Approach 1:
  - Extend current association rule formulation by augmenting each transaction with higher level items
    - Original Transaction: {skim milk, wheat bread}
    - Augmented Transaction: {skim milk, wheat bread, milk, bread, food}
- Approach 2:
  - Generate frequent patterns at highest level of concept hierarchy first
  - Then, generate frequent patterns at the next highest level of the concept hierarchy, and so on

Outline

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

Things you should know from this lecture

- Compact forms of frequent itemsets
  - Closed frequent itemsets
  - Maximal frequent itemsets
- Apriori bottlenecks and how the follow up methods deal with them
- General ideas on how to apply FIM and ARM to other types of data
  - Categorical
  - Continuous