POLITECNICO DI MILANO

Association Rules: Advanced Topics

Data Mining and Text Mining (UIC 583 @ Politecnico di Milano)

Lecture outline

- Frequent patterns without candidate generation
- Multilevel association rules
- Correlation rules
- Sequential rules

Is Apriori Fast Enough? Performance Bottlenecks

- The core of the Apriori algorithm
  - Use frequent (k–1)-itemsets to generate candidate frequent k-itemsets
  - Use database scan and pattern matching to collect counts for the candidate itemsets
- The bottleneck of Apriori: candidate generation
- 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., $\{a_1, a_2, \ldots, a_{100}\}$, needs to generate $2^{100} \approx 10^{30}$ candidates.
- Needs multiple (n+1) scans of database with n is the length of the longest pattern

How to Generate Candidates?

- How to generate candidates?
  - Step 1: self-joining L_k
  - Step 2: pruning
- How to count supports of candidates?
- Example of Candidate-generation
- Example:
  - L3={abc, abd, acd, ace, bcd}
  - Self-joining: L3*L3 generates abcd from abc and abd
  - Generates also acde from acd and ace
  - Pruning: acde is removed because ade is not in L_3
  - C_4={abcd}

How to Generate Candidates?

- Suppose the items in Lk-1 are listed in an order
- Step 1: self-join Lk-1
  - INSERT INTO Ck
  - SELECT p.item1, p.item2, …, p.itemk-1, q.itemk-1
  - FROM Lk-1 p, Lk-1 q
  - WHERE p.item1=q.item1, …, p.itemk-2=q.itemk-2, p.itemk-1 < q.itemk-1
- Step 2: pruning
  - For each itemset c in Ck do
    - For each (k-1)-subsets s of c do if (s is not in Lk-1) then delete c from Ck

Mining Frequent Patterns Without Candidate Generation

- Compress a large database into a compact, Frequent-Pattern tree (FP-tree) structure
  - Highly condensed, but complete for frequent pattern mining
  - Avoid costly database scans
- Develop an efficient, FP-tree-based frequent pattern mining method
- A divide-and-conquer methodology: decompose mining tasks into smaller ones
- Avoid candidate generation: sub-database test only

FP-growth…

The FP-growth Algorithm

- Leave the generate-and-test paradigm of Apriori
- Data sets are encoded using a compact structure, the FP-tree
- Frequent itemsets are extracted directly from the FP-tree

- Major Steps to mine FP-tree
  - Construct conditional pattern base for each node in the FP-tree
  - Construct conditional FP-tree from each conditional pattern-base
  - 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

Building the FP-tree: First Scan to Compute Frequent Items

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

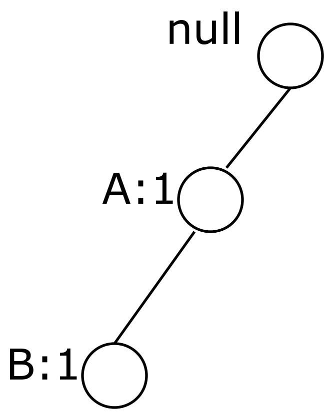
Item	Count
A	8
B	7
C	6
D	5
E	3

minsup=2

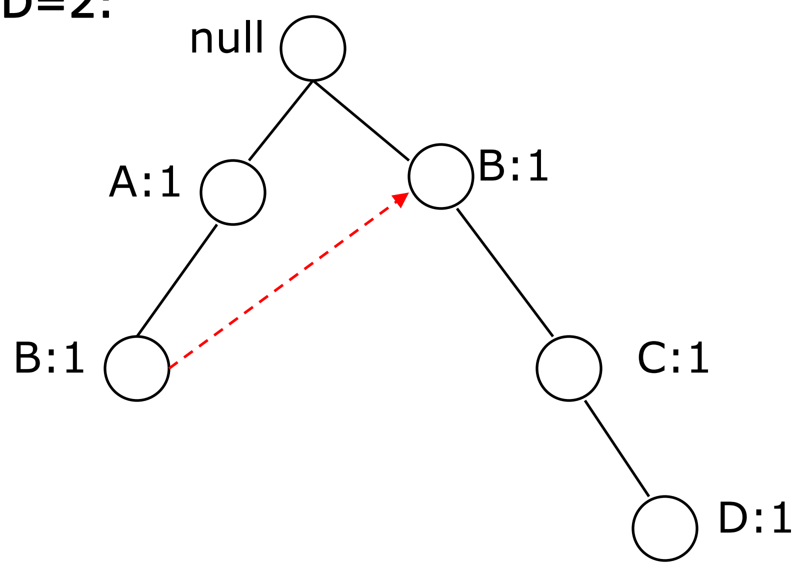
Building the FP-tree: Second Scan to Build the Tree

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

After reading TID=1:



After reading TID=2:

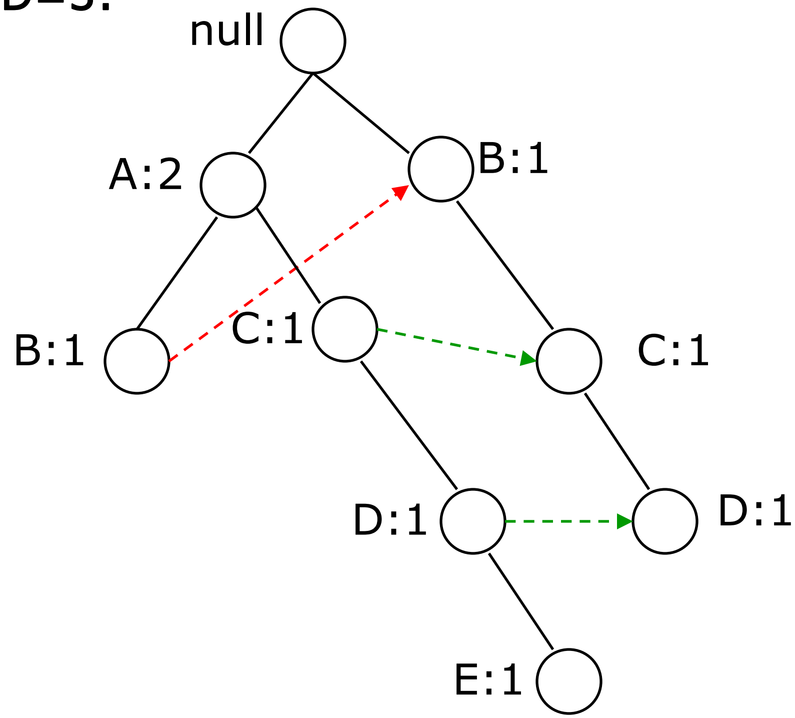


- Items are sorted in decreasing support counts

Building the FP-tree

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

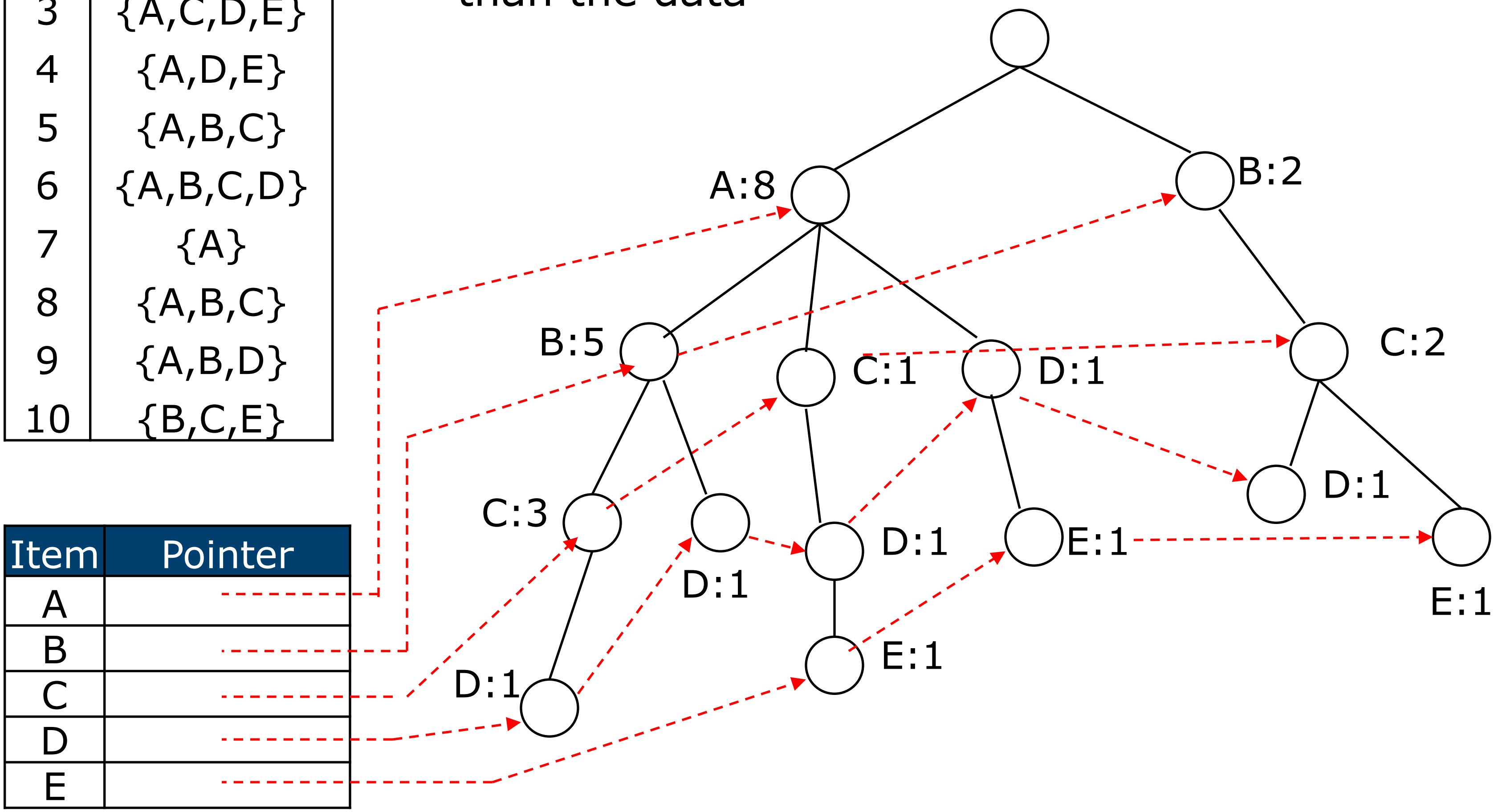
After reading TID=3:



Building the FP-tree

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

- Pointers are used to assist frequent itemset generation
- The FP-tree is typically smaller than the data



Decompose the frequent item generation into multiple subproblems

Start backward

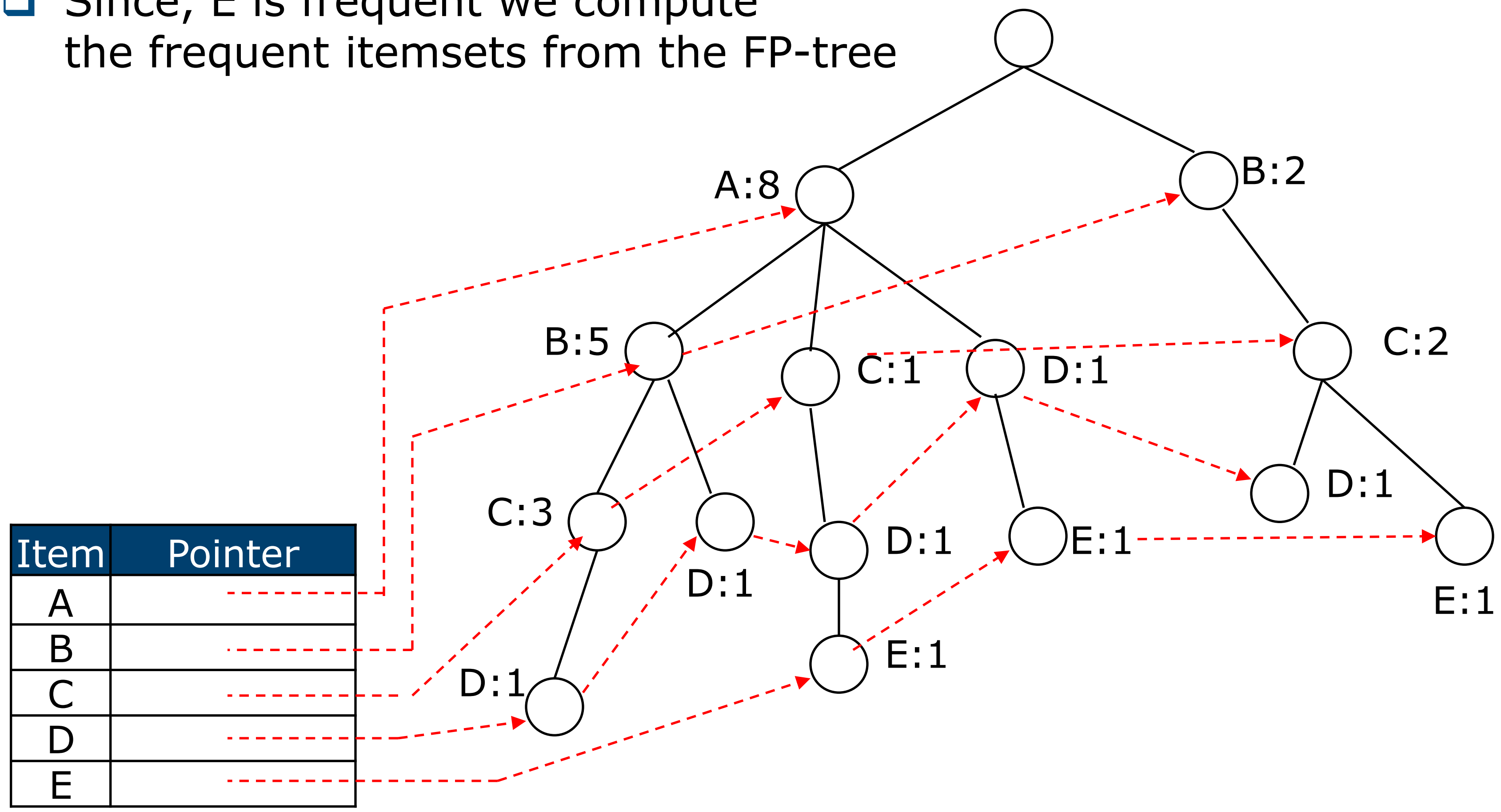
Find frequent itemsets ending in E

Find frequent itemsets ending in D

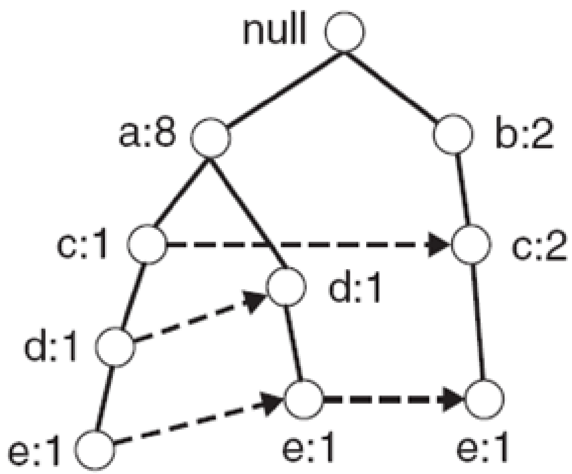
…

Mining the FP-Tree

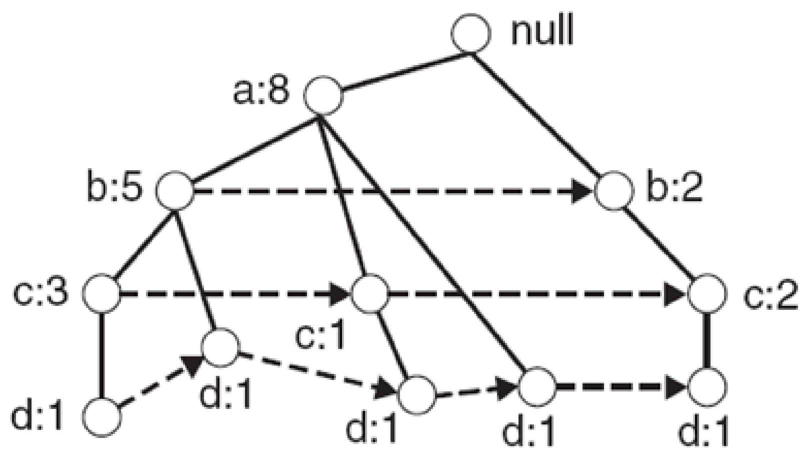
- Start from the bottom, from E
- Compute the support count, by adding the counts associated to E
- Since, E is frequent we compute the frequent itemsets from the FP-tree



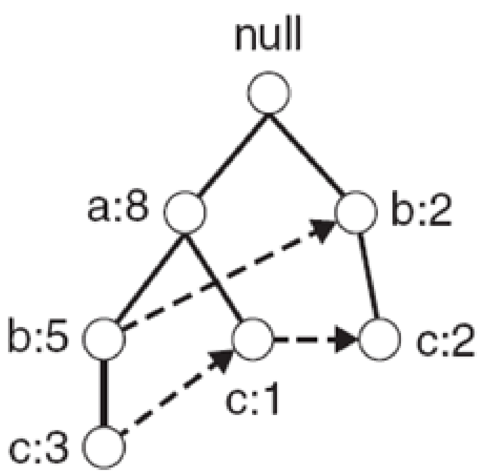
Conditional Pattern base



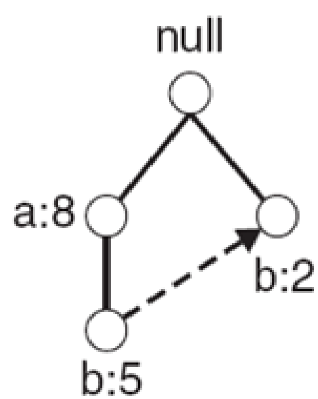
(a) Paths containing node e



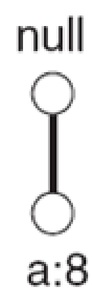
(b) Paths containing node d



(c) Paths containing node c



(d) Paths containing node b

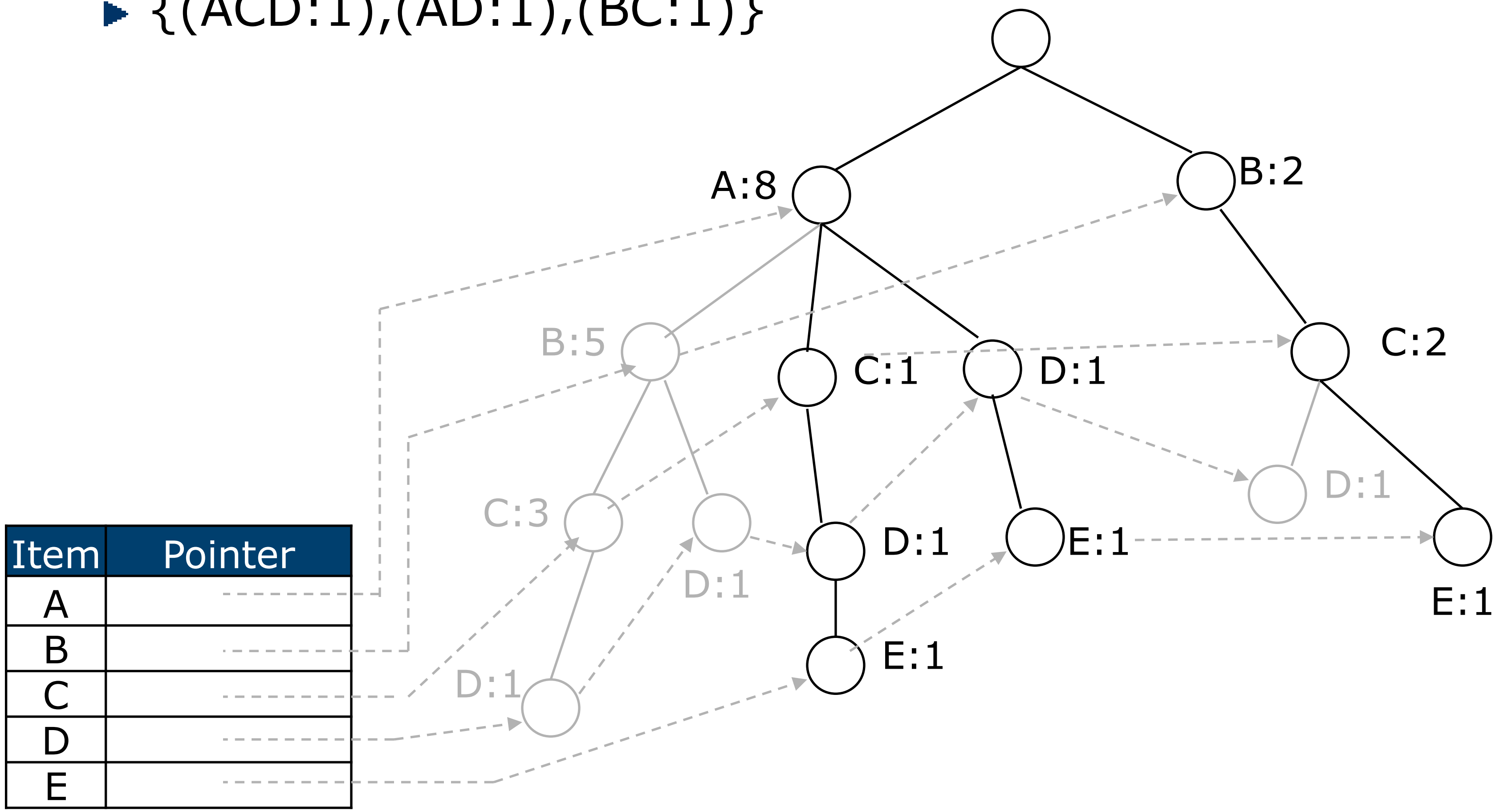


(e) Paths containing node a

Mining the FP-tree

minsup=2

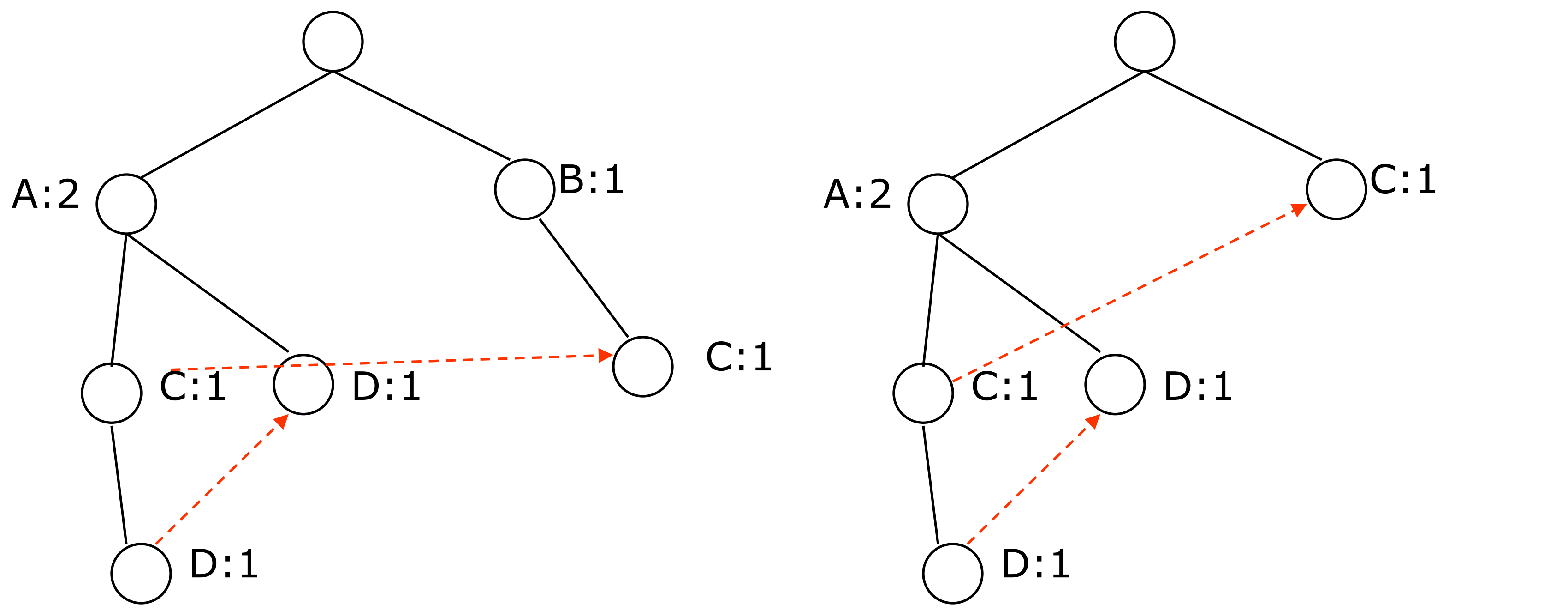
- Considers the path to E, it is frequent
- Extracts the conditional pattern base
  - {(ACD:1),(AD:1),(BC:1)}



Mining the FP-tree

minsup=2

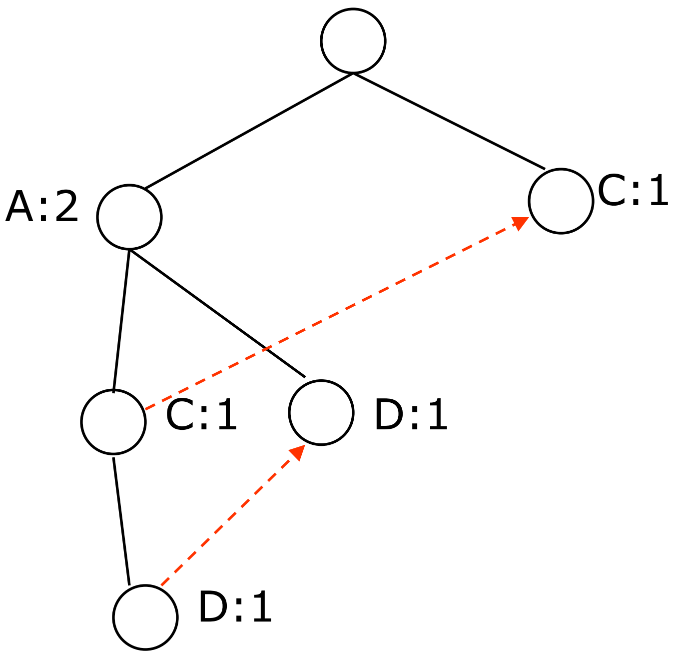
- From the conditional pattern base
  {(ACD:1),(AD:1),(BC:1)}
  build the conditional FP-tree
- Delete unfrequent items



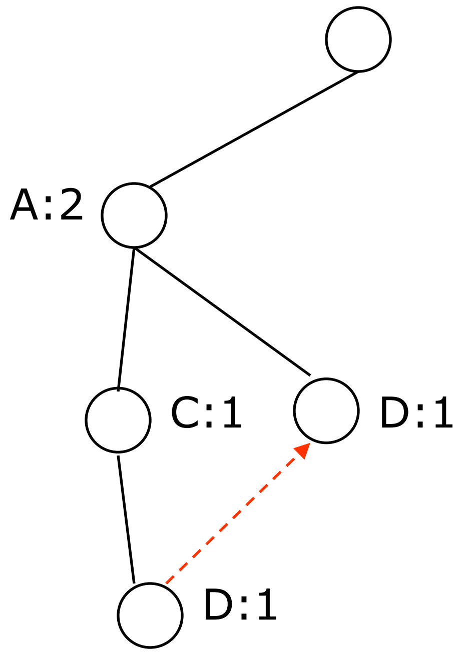
Mining the FP-tree

- Use the conditional FP-tree to extract the frequent itemsets ending with DE, CE, and AE

minsup=2



Conditional FP-tree for E

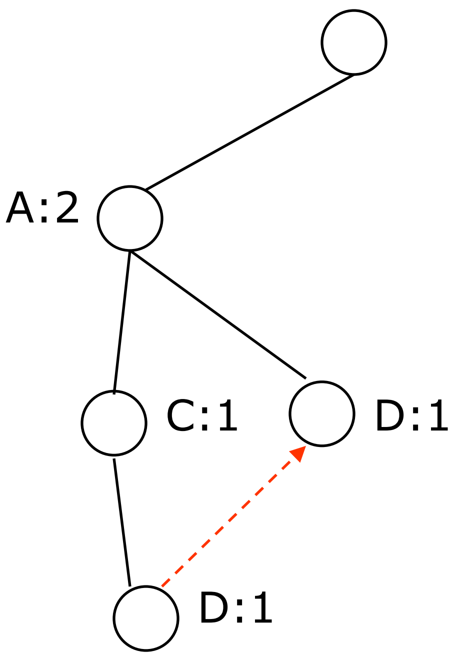


Prefix paths ending in DE

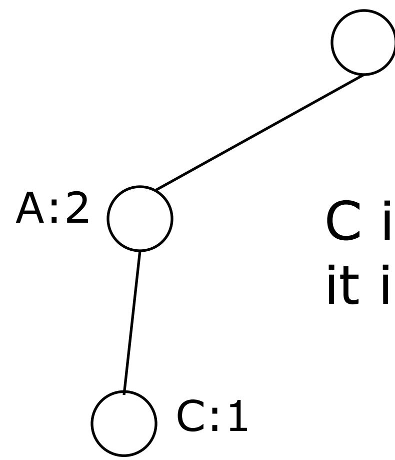
Mining the FP-tree

minsup=2

- Consider the suffix DE, it is frequent
- Build the conditional FP-tree for DE
- The last tree contains only A which is frequent
- So far we have obtained three frequent itemsets, {E, DE, ADE}



Prefix paths ending in de



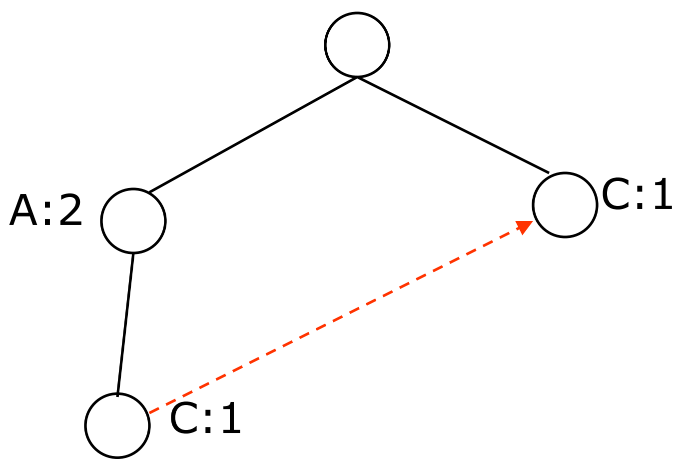
C is unfrequent so it is deleted

Conditional FP-tree for de

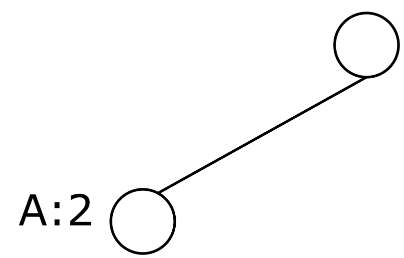
Mining the FP-tree

minsup=2

- After DE, the suffix CE is considered
- CE is frequent and thus added to the frequent itemsets
- Then, we search for the itemsets ending with AE
- At the end the frequent itemsets ending with E are {E, DE, ADE, CE, AE}



Prefix paths ending in CE



Prefix paths ending in AE

Mining Frequent Patterns Using FP-tree

- General idea (divide-and-conquer)
  - Recursively grow frequent pattern path using the FP-tree
- Method
  - For each item, construct its conditional pattern-base, and then its conditional FP-tree
  - Repeat the process on each newly created conditional FP-tree
  - Until the resulting FP-tree is empty, or it contains only one path (single path will generate all the combinations of its sub-paths, each of which is a frequent pattern)

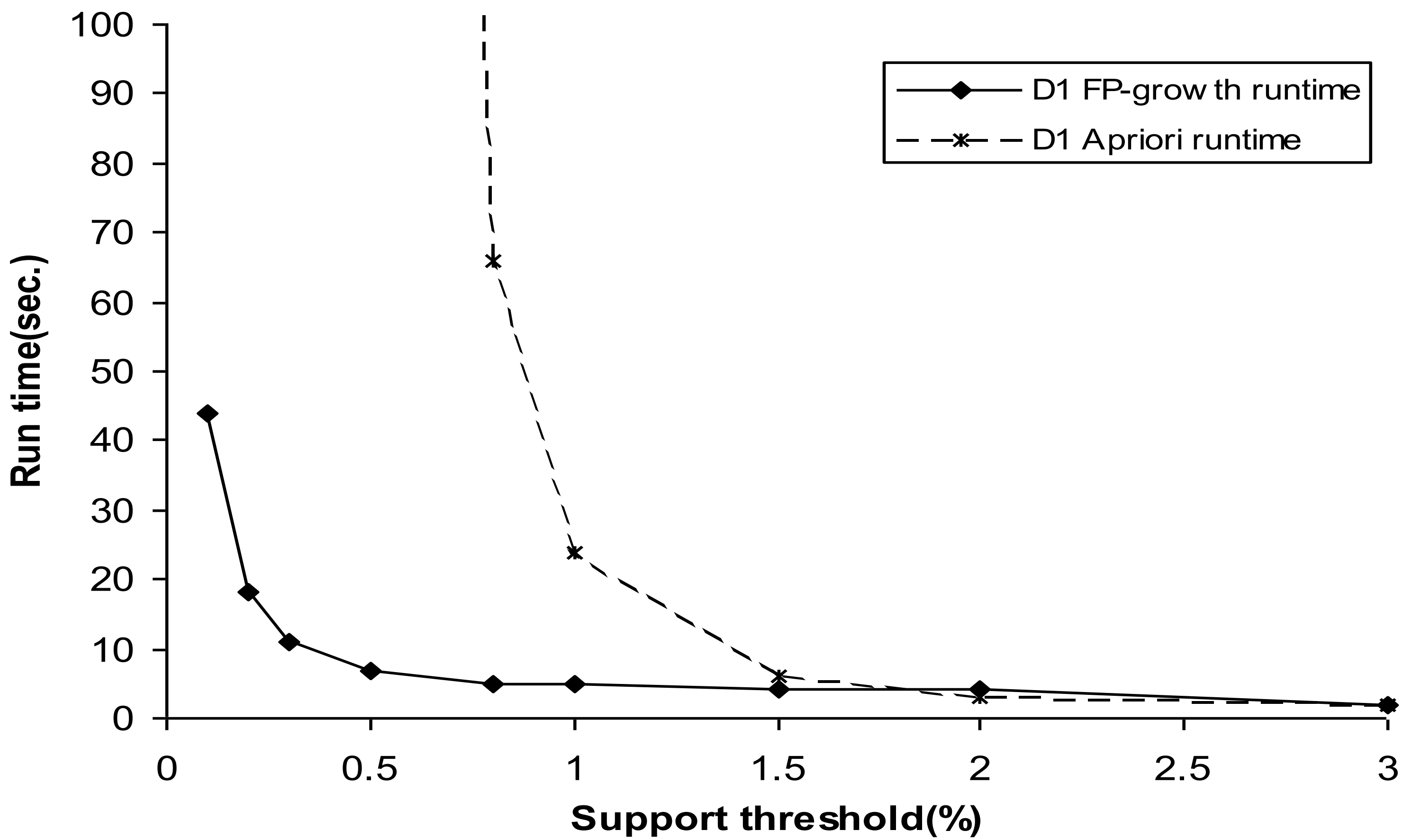
Performance of FP-growth

- FP-growth is an order of magnitude faster than Apriori, and is also faster than tree-projection
- Why?
  - No candidate generation, no candidate test
  - Use compact data structure
  - Eliminate repeated database scan
  - Basic operation is counting and FP-tree building

Mining the FP-tree

- Start at the last item in the table
- Find all paths containing item
  - Follow the node-links
- Identify conditional patterns
  - Patterns in paths with required frequency
- Build conditional FP-tree C
- Append item to all paths in C, generating frequent patterns
- Mine C recursively (appending item)
- Remove item from table and tree

FP-growth vs. Apriori: Scalability With the Support Threshold



Benefits of the FP-tree Structure

- Completeness
  - Preserve complete information for frequent pattern mining
  - Never break a long pattern of any transaction
- Compactness
  - Reduce irrelevant info—infrequent items are gone
  - Items in frequency descending order: the more frequently occurring, the more likely to be shared
  - Never be larger than the original database (not count node-links and the count field)
  - For Connect-4 DB, compression ratio could be over 100

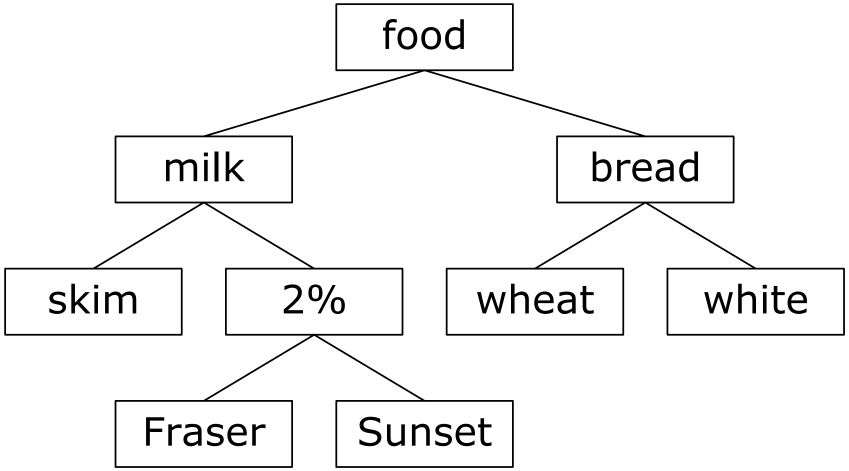
Why Is FP-Growth the Winner?

- Divide-and-conquer:
  - Decompose both the mining task and DB according to the frequent patterns obtained so far
  - Leads to focused search of smaller databases
- Other factors
  - No candidate generation, no candidate test
  - Compressed database: FP-tree structure
  - No repeated scan of entire database
  - Basic ops—counting local freq items and building sub FP-tree, no pattern search and matching

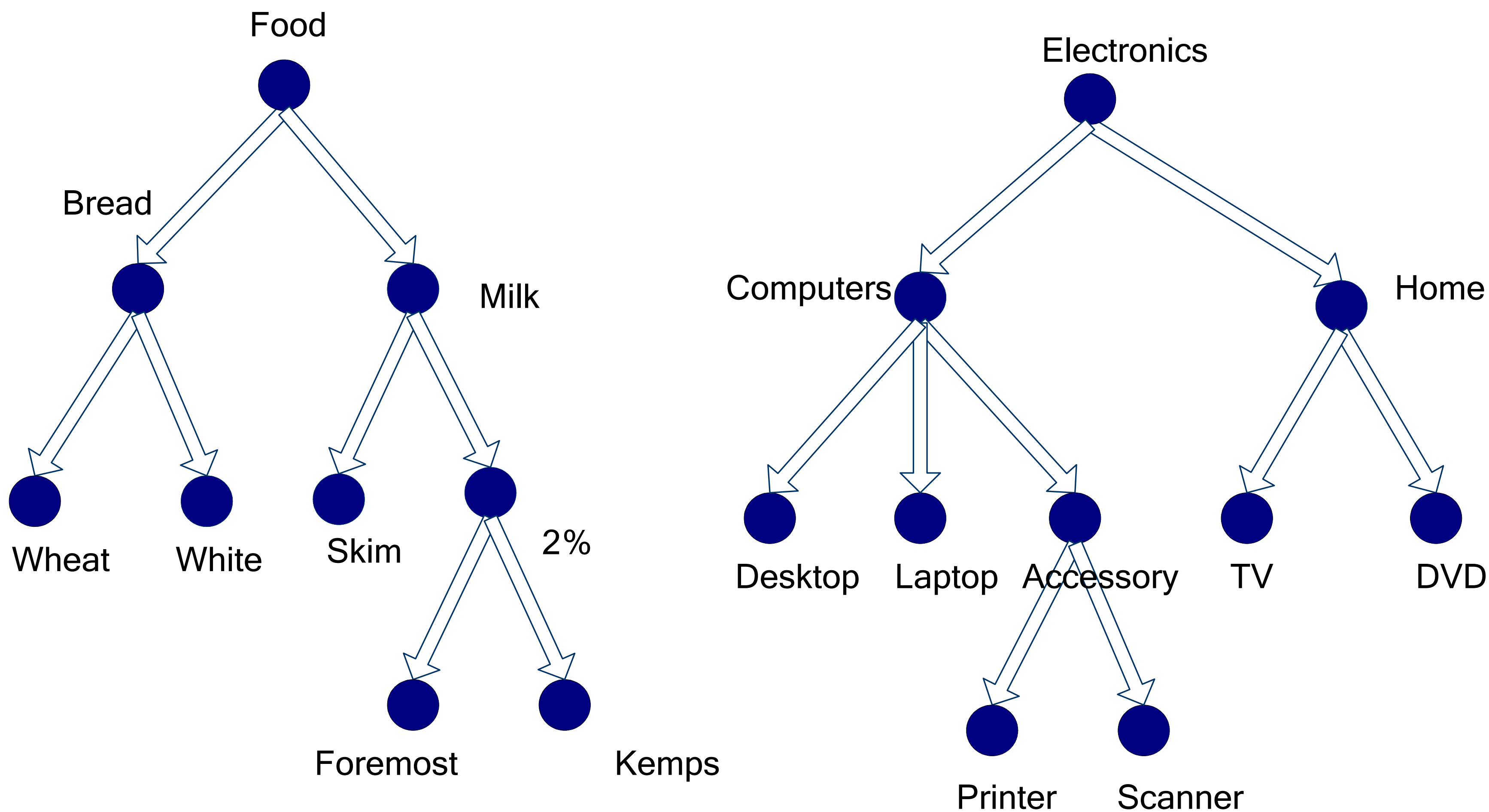
Multi-level Association Rules

Multiple-Level Association Rules

- Items often form hierarchy
- Items at the lower level are expected to have lower support
- Rules regarding itemsets at appropriate levels could be quite useful
- Transaction database can be encoded based on dimensions and levels
- We can explore shared multi-level mining



Concept Hierarchy



Why 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,

$$\{2\% \text{ milk}\} \Rightarrow \{\text{wheat bread}\}$$

is indicative of association between milk and bread

Multi-level Association Rules

- How do support and confidence vary as we traverse the concept hierarchy?
  - If X is the parent item for both X1 and X2, then $\sigma(X) \leq \sigma(X1) + \sigma(X2)$
  - If $\sigma(X1 \cup Y1) \geq minsup$,
    and X is parent of X1, Y is parent of Y1
    then $\sigma(X \cup Y1) \geq minsup$, $\sigma(X1 \cup Y) \geq minsup$
    $\sigma(X \cup Y) \geq minsup$
  - If $conf(X1 \Rightarrow Y1) \geq minconf$,
    then $conf(X1 \Rightarrow Y) \geq minconf$

Multi-level Association Rules

- 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}
- Issues:
  - Items that reside at higher levels have much higher support counts
  - If support threshold is low, too many frequent patterns involving items from the higher levels
  - Increased dimensionality of the data

Multi-level Association Rules

- Approach 2:
  - Generate frequent patterns at highest level first
  - Then, generate frequent patterns at the next highest level, and so on
- Issues:
  - I/O requirements will increase dramatically because we need to perform more passes over the data
  - May miss some potentially interesting cross-level association patterns

Mining Multi-Level Associations

- A top down, progressive deepening approach
- First find high-level strong rules:

$$\{\text{milk}\} \Rightarrow \{\text{bread}\}\ \ [20\%, 60\%]$$

- Then find their lower-level “weaker” rules:

$$\{2\%\ \text{milk}\} \Rightarrow \{\text{wheat bread}\}\ [6\%, 50\%]$$

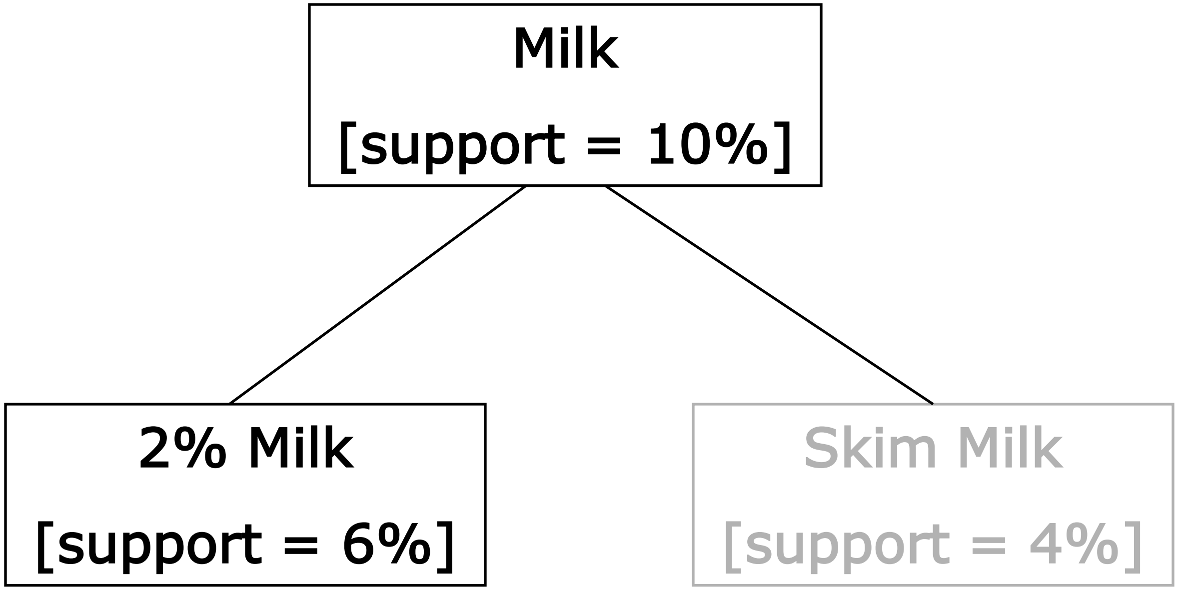
Multi-level Association: Redundancy Filtering

- Some rules may be redundant due to "ancestor" relationships between items
- A rule is redundant if its support is close to the "expected" value, based on the rule's ancestor.
- Example
  - milk $\Rightarrow$ wheat bread
    [support = 8%, confidence = 71.5%]
  - 2% milk $\Rightarrow$ wheat bread
    [support = 7.5%, confidence = 72%]
- The first rule is an ancestor of the second rule.
- One of the two rules is redundant: both rules provide the same information, in fact, they have similar support and confidence

Multi-level mining with uniform support

Level 1
min_sup = 5%

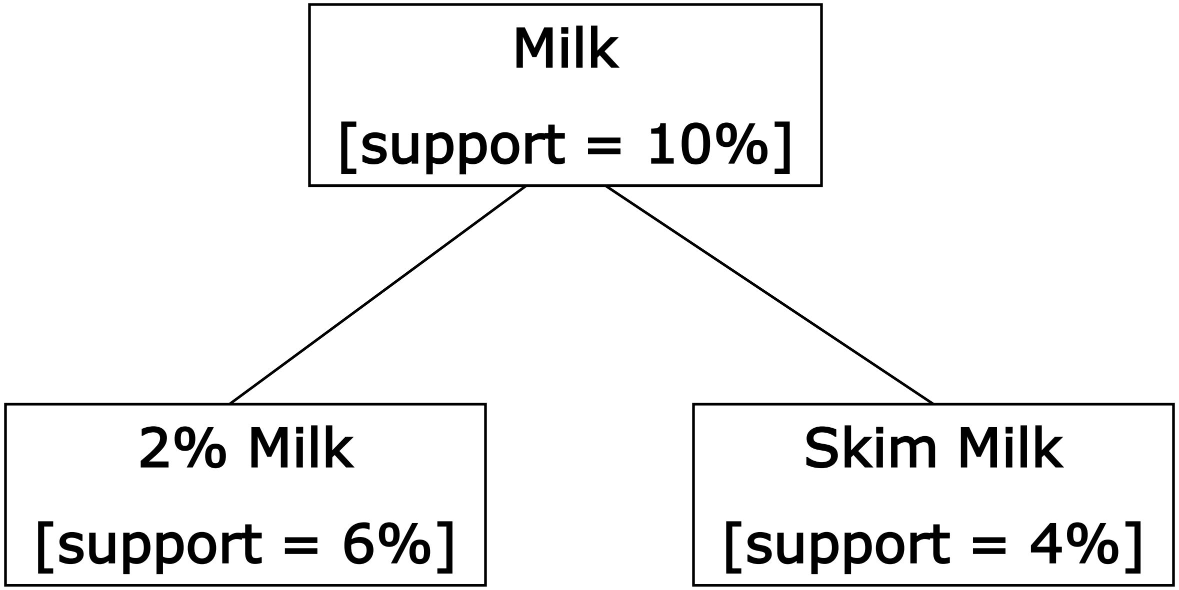
Level 2
min_sup = 5%



Multi-level mining with reduced support

Level 1
min_sup = 5%

Level 2
min_sup = 3%



Multi-level Association: Uniform Support vs. Reduced Support

- Uniform Support: the same minimum support for all levels
  - + One minimum support threshold. No need to examine itemsets containing any item whose ancestors do not have minimum support.
  - – Lower level items do not occur as frequently. If support threshold
    - too high $\Rightarrow$ miss low level associations
    - too low $\Rightarrow$ generate too many high level associations
- Reduced Support: reduced minimum support at lower levels
  - There are 4 search strategies:
    - Level-by-level independent
    - Level-cross filtering by k-itemset
    - Level-cross filtering by single item
    - Controlled level-cross filtering by single item

Multi-Level Mining: Progressive Deepening

- A top-down, progressive deepening approach
- First mine high-level frequent items:
  milk (15%), bread (10%)
- Then mine their lower-level "weaker" frequent itemsets:
  2% milk (5%), wheat bread (4%)
- Different min support threshold across
  multi-levels lead to different algorithms:
  - If adopting the same min_support across multi-levels
    then toss t if any of t's ancestors is infrequent.
  - If adopting reduced min_support at lower levels
    then examine only those descendents whose ancestor's
    support is frequent/non-negligible.

Correlation Rules

- Suppose perform association rule mining

	c	Not c	row
t	20	5	25
Not t	70	5	75
col	90	10	100

{tea} $\Rightarrow$ {coffee} [20% 80%]

- But the 90% of the customers buy coffee anyway!
- A customer who buys tea is less likely (10% less) to buy coffee than a customer about whom we have no information

A ⇒ B [support, confidence, correlation]

- One measure is to calculate correlation
- If two random variables A and B are independent,

$$lift(A,B) = \frac{P(A \wedge B)}{P(A)P(B)} = 1$$

- For tea and coffee,

$$\frac{P(t \wedge c)}{P(t)P(c)} = 0.89$$

this means that the presence of tea is negatively correlated with the presence of coffee

Correlation Rules Using χ^2

- The χ^2 statistic is defined as

$$\chi^2 = \sum_{r \in R} \frac{(O(r) - E[r])^2}{E[r]}$$

- Look up for significance value in a statistical textbook
  - There are k-1 degrees of freedom
  - If test fails cannot reject independence, otherwise contingency table represents dependence.

Correlation Rules Using χ^2

- Use the χ^2 statistical test for testing independence
- Let n be the number of baskets
- Let k be the number of items considered
- Let r be a rule consisting of item values (i_j, i_j)
- Let O(r) represent the number of baskets containing rule r (i.e. frequency)
- Let $E[i_j] = O(i_j)$ for a single item (negation is $n - E[i_j]$)
- $E[r] = n * E[r_1]/n * \ldots * E[r_k] / n$

Example

- Back to tea and coffee

	c	Not c	row
t	20	5	25
Not t	70	5	75
col	90	10	100

- E[t] = 25, E[Not t]=75, E[c]=90, E[Not c]=10
- E[tc]=100 * 25/100 * 90 /100=22.5
- O(tc) = 20
- Contrib. to χ^2 = $(20 - 22.5)^2$ / 22.5 = 0.278
- Calculate for the rest to get χ^2=2.204
- Not significant at 95% level (3.84 for k=2)
- Cannot reject independence assumption

Interest

- If χ^2 test shows significance, then want to find most interesting cell(s) in table
- I(r) = O(r)/E[r]
  - Look for values far away from 1
  - I(t c) = 20/22.5 = 0.89
  - I(t not c) = 5/2.5 = 2
  - I(not t c) = 70/67.5 = 1.04
  - I(not t not c) = 5/7.5 = 0.66

Properties of $\chi2$ statistic

- Upward closed
  - If a k-itemset is correlated, so are all it's supersets
  - Look for minimal correlated itemsets, no subset is correlated
- Can combine a-priori and $\chi2$ statistic efficiently
  - No generate and prune as in support-confidence

Sequential Rules

Sequential pattern mining

- Association rule mining does not consider the order of transactions
- In many applications such orderings are significant
- In market basket analysis, it is interesting to know whether people buy some items in sequence
- Example: buying bed first and then bed sheets later
- In Web usage mining,
  - it is useful to find navigational patterns of users in a Web site from sequences of page visits of users

Examples of Sequence

- Web sequence
  - < {Homepage} {Electronics} {Digital Cameras} {Canon Digital Camera} {Shopping Cart} {Order Confirmation} {Return to Shopping} >

- Sequence of initiating events causing the nuclear accident at 3-mile Island:
  <{clogged resin} {outlet valve closure} {loss of feedwater} {condenser polisher outlet valve shut} {booster pumps trip} {main waterpump trips} {main turbine trips} {reactor pressure increases}>

- Sequence of books checked out at a library:
  - <{Fellowship of the Ring} {The Two Towers} {Return of the King}>

- Let $I = \{i_1, i_2, \ldots, i_m\}$ be a set of items.
- A sequence is an ordered list of itemsets.
- We denote a sequence s by $\langle a_1, a_2, \ldots, a_r \rangle$, where a_i is an itemset, also called an element of s.
- An element (or an itemset) of a sequence is denoted by $\{x_1, x_2, \ldots, x_k\}$, where $x_j \in I$ is an item.
- We assume without loss of generality that items in an element of a sequence are in lexicographic order

- Size
  - The size of a sequence is the number of elements (or itemsets) in the sequence
- Length
  - The length of a sequence is the number of items in the sequence
  - A sequence of length k is called k-sequence
- Given two sequences, $s_1 = \langle a_1 a_2 \ldots a_r \rangle$ and $s_2 = \langle b_1 b_2 \ldots b_v \rangle$
- s_1 is a subsequence of s_2 if there exist integers $1 \leq j_1 < j_2 < \ldots < j_{r□-1} < j_r \leq v$ such that $a_1 \subseteq b_{j1}, a_2 \subseteq b_{j2}, \ldots, a_r \subseteq b_{jr}$
- A sequence $s_1 = \langle a_1 a_2 \ldots a_r \rangle$ is a subsequence of another sequence $s_2 = \langle b_1 b_2 \ldots b_v \rangle$, or s2 is a supersequence of s_1, if there exist integers 1 ≤ j_1 < j_2 < … < j_{r–1} < jr ≤ v such that $a_1 \subseteq$ bj1, a2 ⊆ bj2, …, ar ⊆ bjr. We also say that s2 contains s1.

Example

- Let I = {1, 2, 3, 4, 5, 6, 7, 8, 9}.
- The sequence $\langle\{3\}\{4, 5\}\{8\}\rangle$ is contained in (or is a subsequence of) $\langle\{6\}\ \{3, 7\}\{9\}\{4, 5, 8\}\{3, 8\}\rangle$
- In fact, $\{3\} \subseteq \{3, 7\}$, $\{4, 5\} \subseteq \{4, 5, 8\}$, and $\{8\} \subseteq \{3, 8\}$
- However, $\langle\{3\}\{8\}\rangle$ is not contained in $\langle\{3, 8\}\rangle$ or vice versa
- The size of the sequence $\langle\{3\}\{4, 5\}\{8\}\rangle$ is 3, and the length of the sequence is 4.

Objective of sequential pattern mining

- The input is a set S of input data sequences (or sequence database)
- The problem of mining sequential patterns is to find all the sequences that have a user-specified minimum support
- Each such sequence is called a frequent sequence, or a sequential pattern
- The support for a sequence is the fraction of total data sequences in S that contains this sequence

Terminology

- Itemset
  - Non-empty set of items
  - Each itemset is mapped to an integer
- Sequence
  - Ordered list of itemsets
- Support for a Sequence
  - Fraction of total customers that support a sequence.
- Maximal Sequence
  - A sequence that is not contained in any other sequence
- Large Sequence
  - Sequence that meets minisup

Example

Customer ID	Transaction Time	Items Bought
1	June 25 '93	30
1	June 30 '93	90
2	June 10 '93	10,20
2	June 15 '93	30
2	June 20 '93	40,60,70
3	June 25 '93	30,50,70
4	June 25 '93	30
4	June 30 '93	40,70
4	July 25 '93	90
5	June 12 '93	90

Customer ID	Customer Sequence
1	< (30) (90) >
2	< (10 20) (30) (40 60 70) >
3	< (30) (50) (70) >
4	< (30) (40 70) (90) >
5	< (90) >

Maximal seq with support > 40%
< (30) (90) >
< (30) (40 70) >

Note: Use Minisup of 40%, no less than two customers must support the sequence
< (10 20) (30) > Does not have enough support (Only by Customer #2)
< (30) >, < (70) >, < (30) (40) > … are not maximal.

Methods for sequential pattern mining

- Apriori-based Approaches
  - GSP
  - SPADE
- Pattern-Growth-based Approaches
  - FreeSpan
  - PrefixSpan

The Generalized Sequential Pattern (GSP) Mining Algorithm

- The Apriori property for sequential patterns
  - If a sequence S is not frequent, then none of the super-sequences of S is frequent
  - For instance, if <hb> is infrequent so do <hab> and <(ah)b>

- GSP (Generalized Sequential Pattern) mining algorithm
  - Initially, every item in DB is a candidate of length-1
  - for each level (i.e., sequences of length-k) do
    - scan database to collect support count for each candidate sequence
    - generate candidate length-(k+1) sequences from length-k frequent sequences using Apriori
  - repeat until no frequent sequence or no candidate can be found

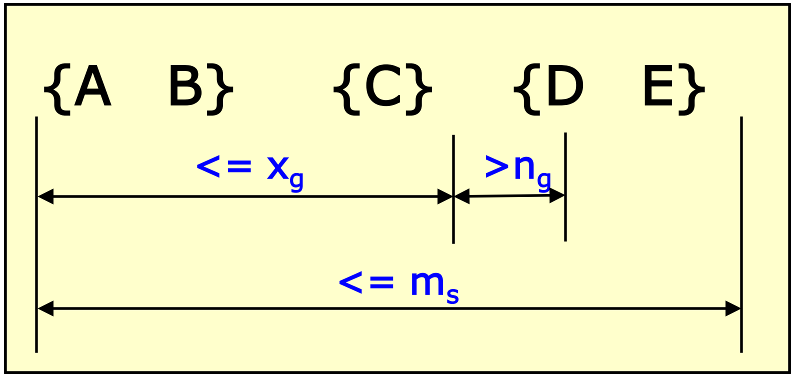
- Major strength: Candidate pruning by Apriori

The Generalized Sequential Pattern (GSP) Mining Algorithm

- **Step 1:**
  - Make the first pass over the sequence database D to yield all the 1-element frequent sequences
- **Step 2:**

  Repeat until no new frequent sequences are found
  - **Candidate Generation:**
    - Merge pairs of frequent subsequences found in the (k-1)*th* pass to generate candidate sequences that contain k items
  - **Candidate Pruning:**
    - Prune candidate *k*-sequences that contain infrequent (*k-1*)-subsequences
  - **Support Counting:**
    - Make a new pass over the sequence database D to find the support for these candidate sequences
  - **Candidate Elimination:**
    - Eliminate candidate *k*-sequences whose actual support is less than *minsup*

Timing Constraints



x_g: max-gap

n_g: min-gap

m_s: maximum span

$x_g = 2$, $n_g = 0$, $m_s = 4$

Data sequence	Subsequence	Contain?
< {2,4} {3,5,6} {4,7} {4,5} {8} >	< {6} {5} >	Yes
< {1} {2} {3} {4} {5}>	< {1} {4} >	No
< {1} {2,3} {3,4} {4,5}>	< {2} {3} {5} >	Yes
< {1,2} {3} {2,3} {3,4} {2,4} {4,5}>	< {1,2} {5} >	No

Mining Sequential Patterns with Timing Constraints

- Approach 1:
  - Mine sequential patterns without timing constraints
  - Postprocess the discovered patterns
- Approach 2:
  - Modify GSP to directly prune candidates that violate timing constraints

Sequential Rules

Summary

- FP-growth
- Multi-level association rules
- Correlation rules
- Sequential pattern mining