



Association Rules: Advanced

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

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

Frequent Patterns Mining Without Candidate Generation

- 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
- 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., $\{a_1, a_2, \dots, a_{100}\}$, needs to generate $2^{100} \approx 10^{30}$ candidates.
 - Multiple scans of database, it needs $(n+1)$ scans, n is the length of the longest pattern

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
- Use 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

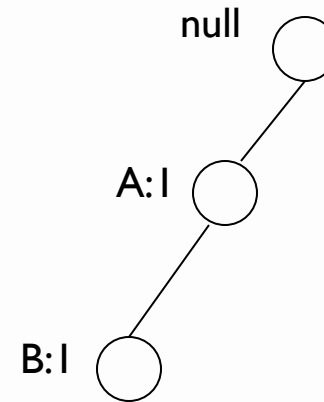
- Leave the generate-and-test paradigm of Apriori
- Data sets are encoded using a compact structure, the FP-tree
- Frequent itemsets are extracted directly from the FP-tree

- 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

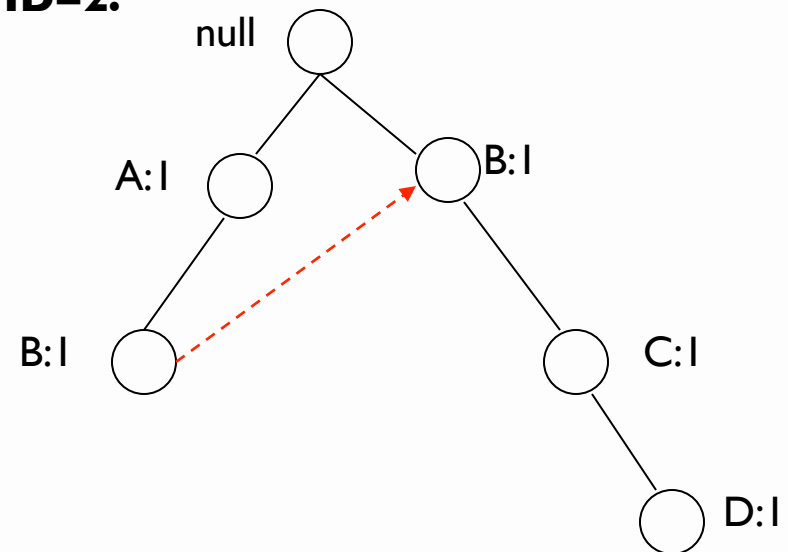
TID	Items
1	{A,B}
2	{B,C,D}
3	{A,C,D,E}
4	{A,D,E}
5	{A,B,C}
6	{A,B,C,D}
7	{B,C}
8	{A,B,C}
9	{A,B,D}
10	{B,C,E}

- Items are sorted in decreasing support counts

After reading TID=1:

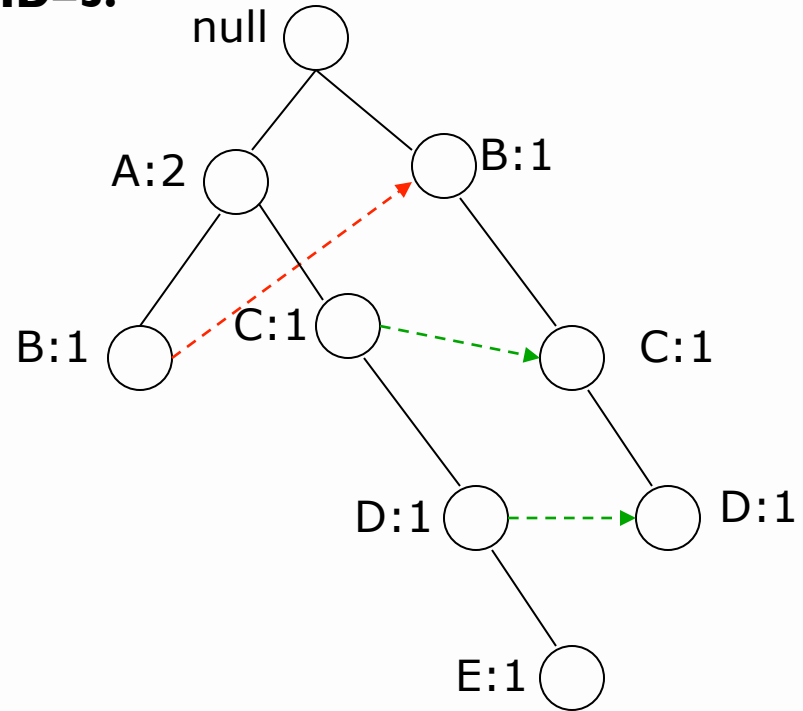


After reading TID=2:



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	{B,C}
8	{A,B,C}
9	{A,B,D}
10	{B,C,E}

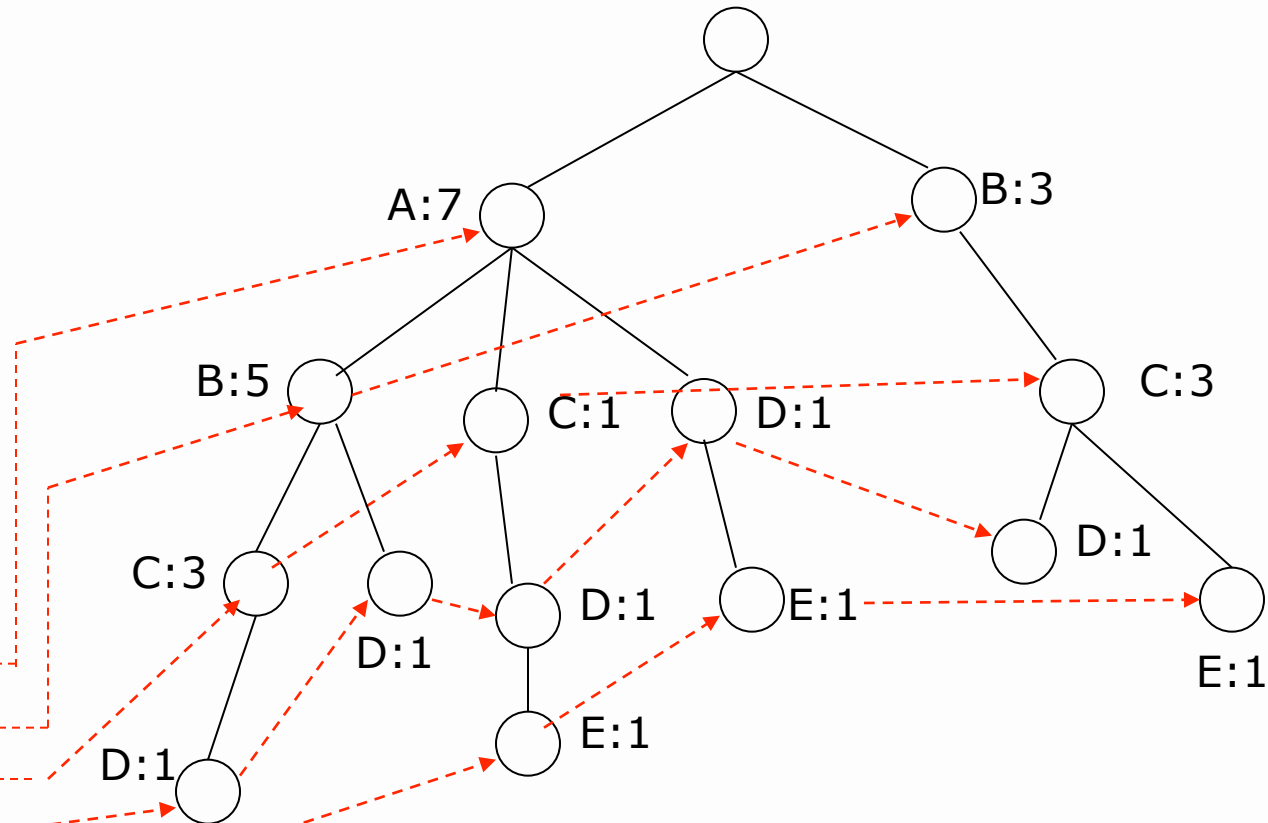
After reading TID=3:



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	{B,C}
8	{A,B,C}
9	{A,B,D}
10	{B,C,E}

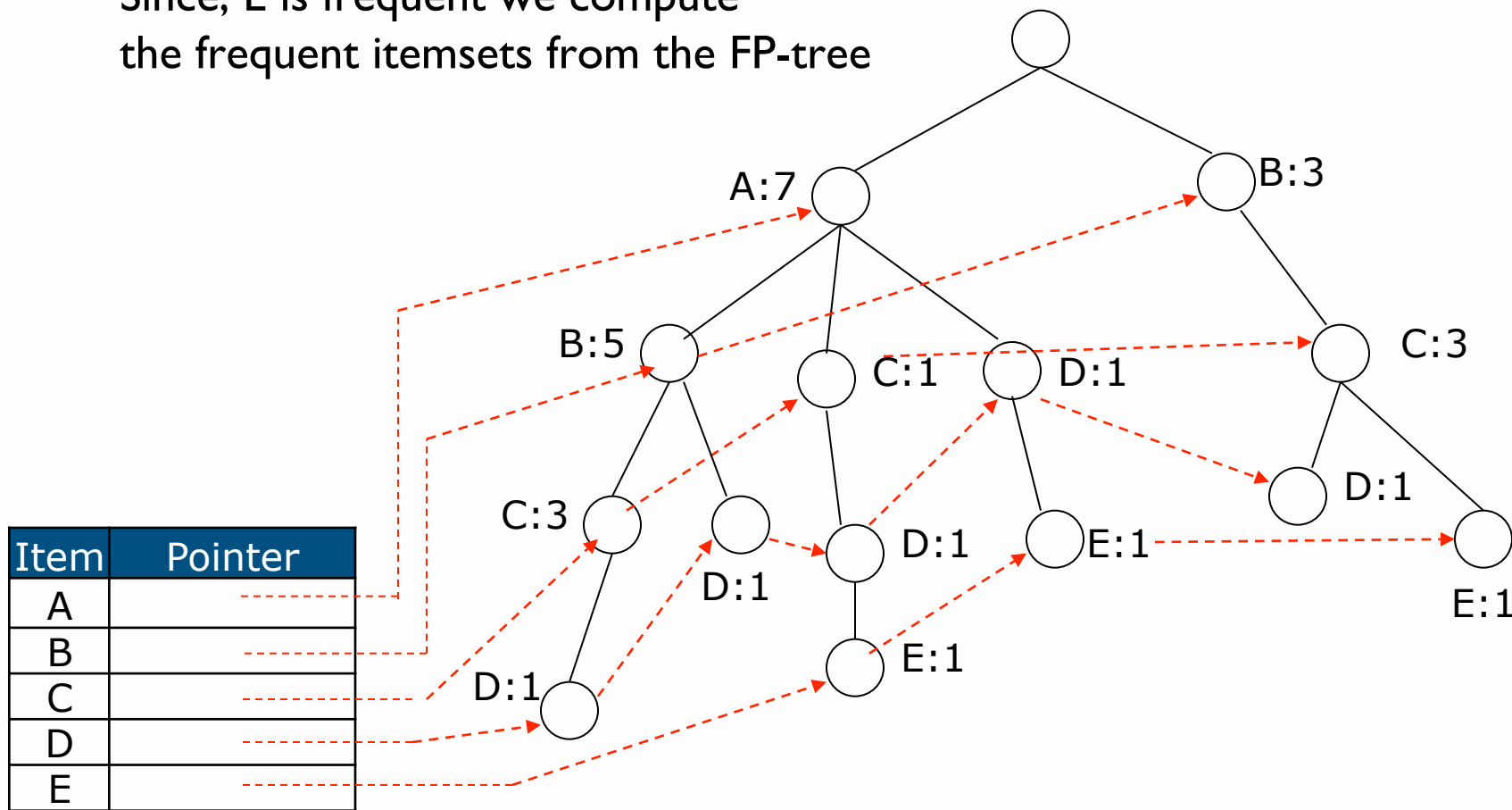
- Pointers are used to assist frequent itemset generation
- The FP-tree is typically smaller than the data

Item	Pointer
A	
B	
C	
D	
E	



minsup=2

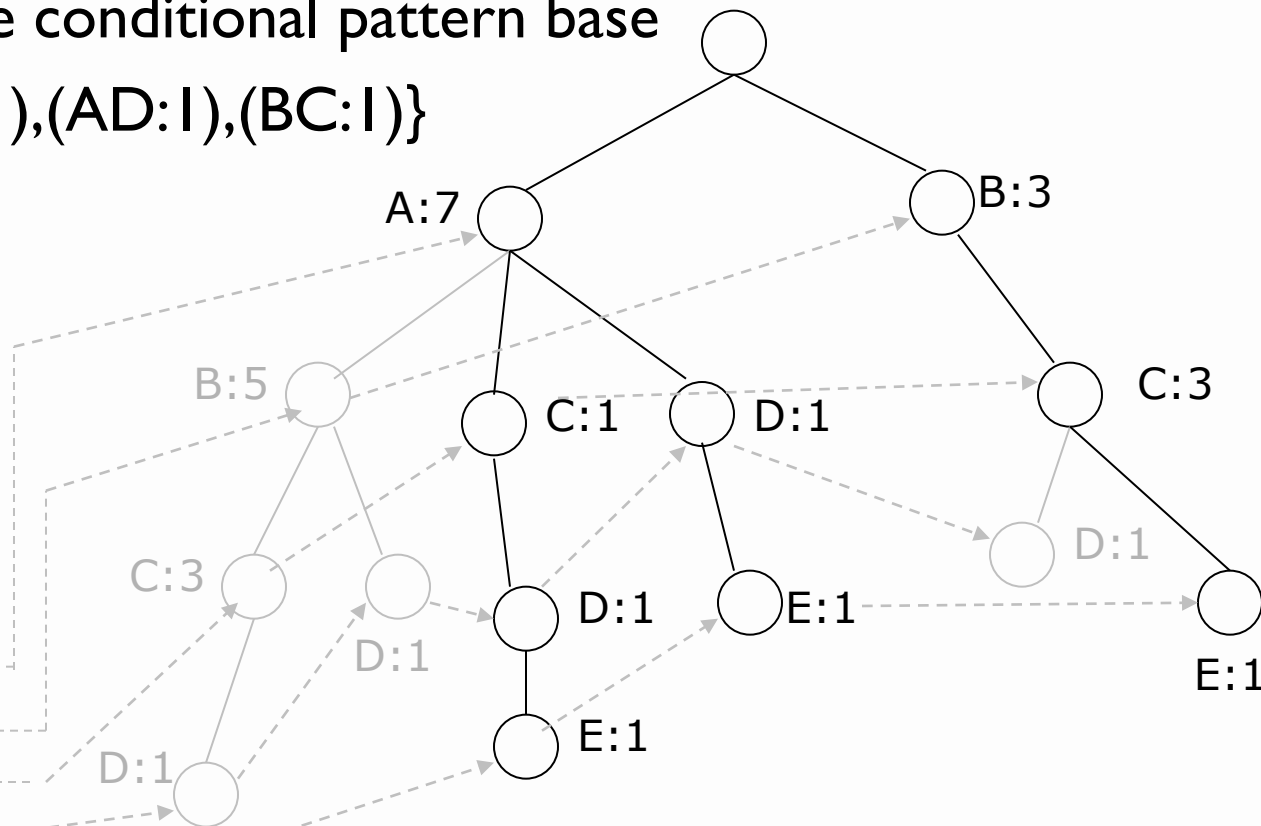
- Start from the bottom, from E
- Compute the support count, by adding the counts associated to E
- Since, E is frequent we compute the frequent itemsets from the FP-tree



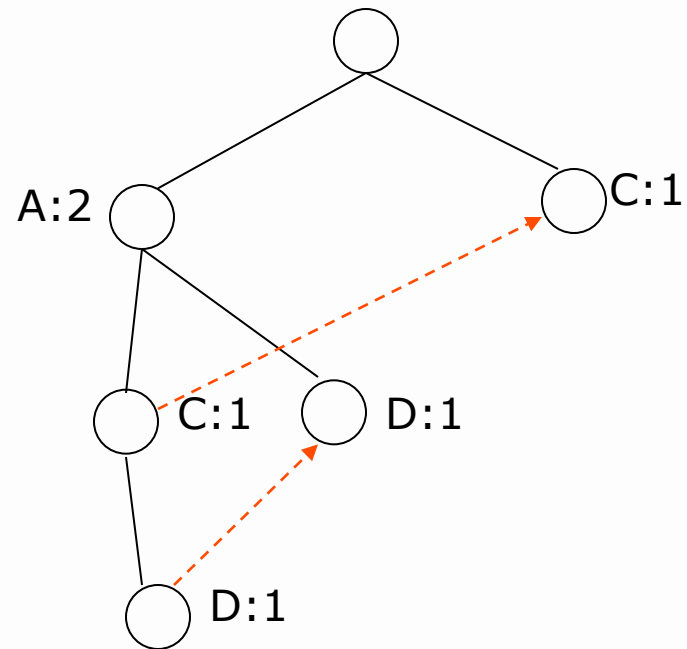
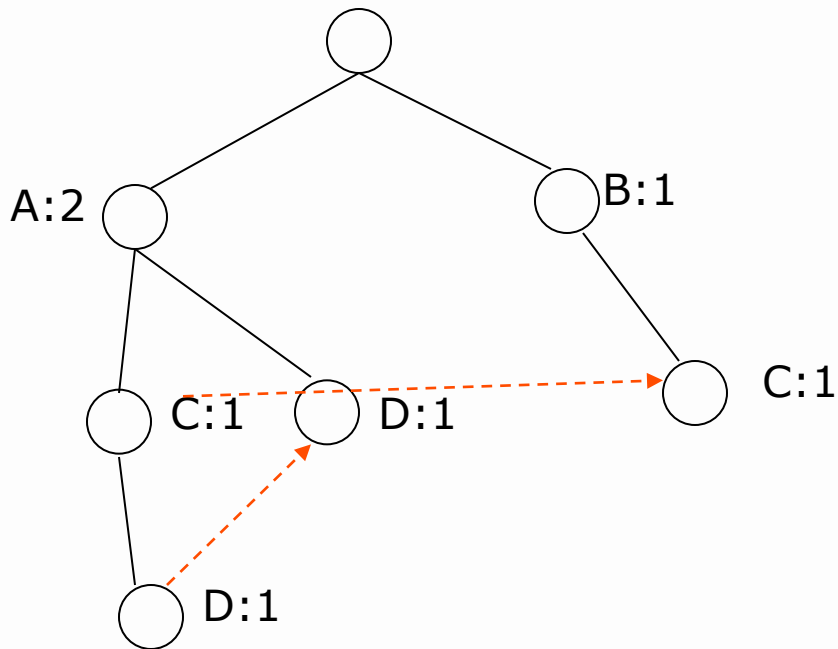
minsup=2

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

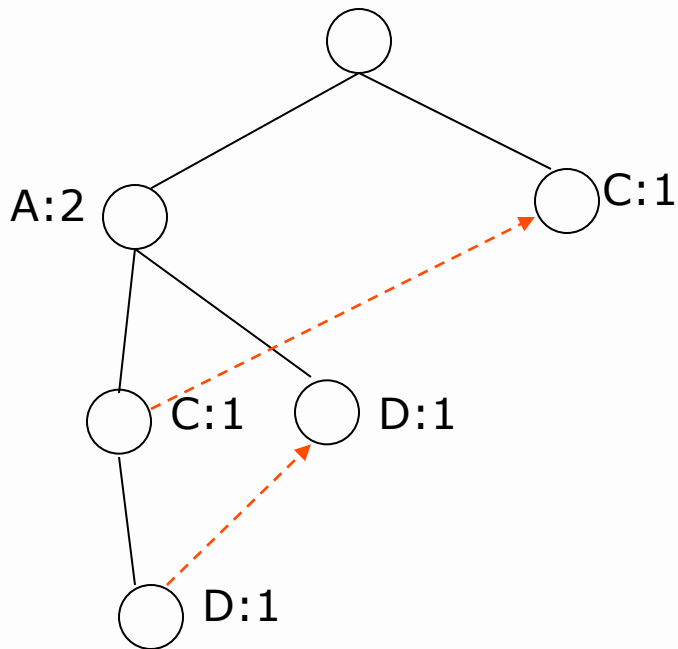
Item	Pointer
A	-----
B	-----
C	-----
D	-----
E	-----



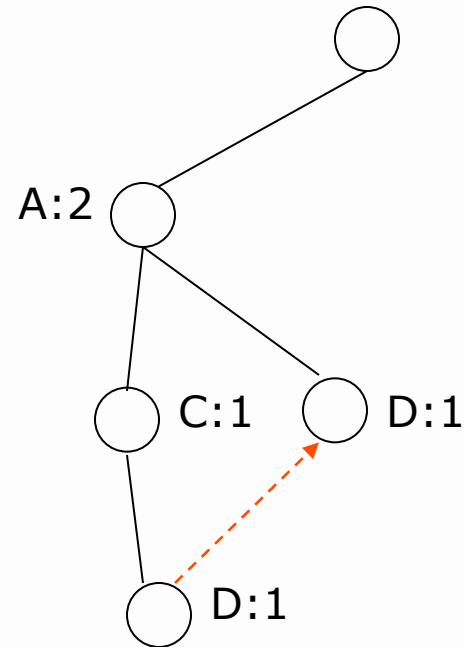
- From the conditional pattern base $\{(ACD:1),(AD:1),(BC:1)\}$ build the conditional FP-tree
- Delete unfrequent items



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

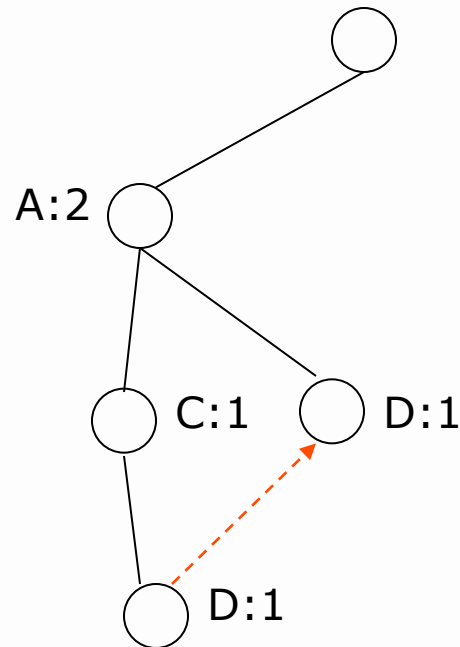


Conditional FP-tree for E

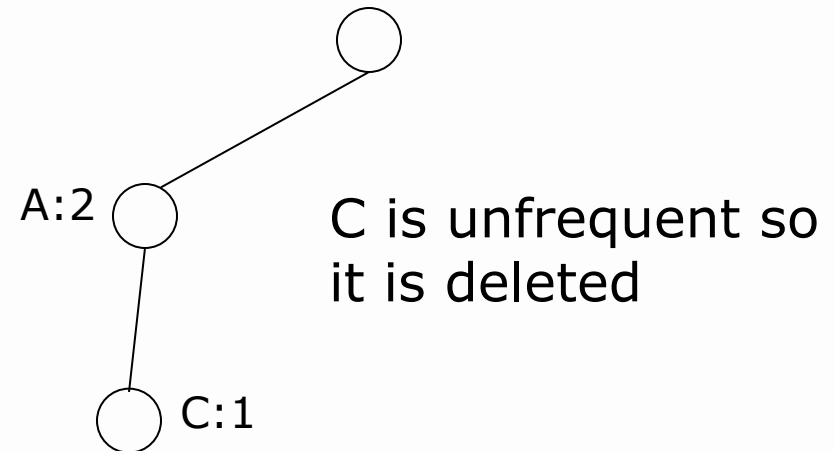


Prefix paths ending in DE

- 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}

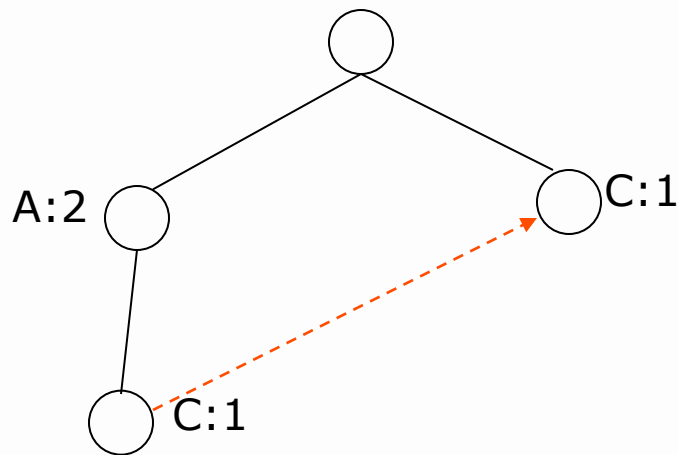


Prefix paths ending in de

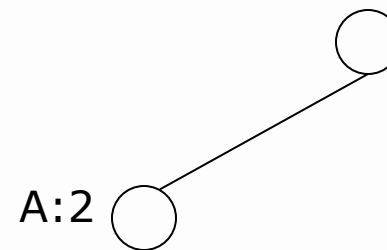


Conditional FP-tree for de

- 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}



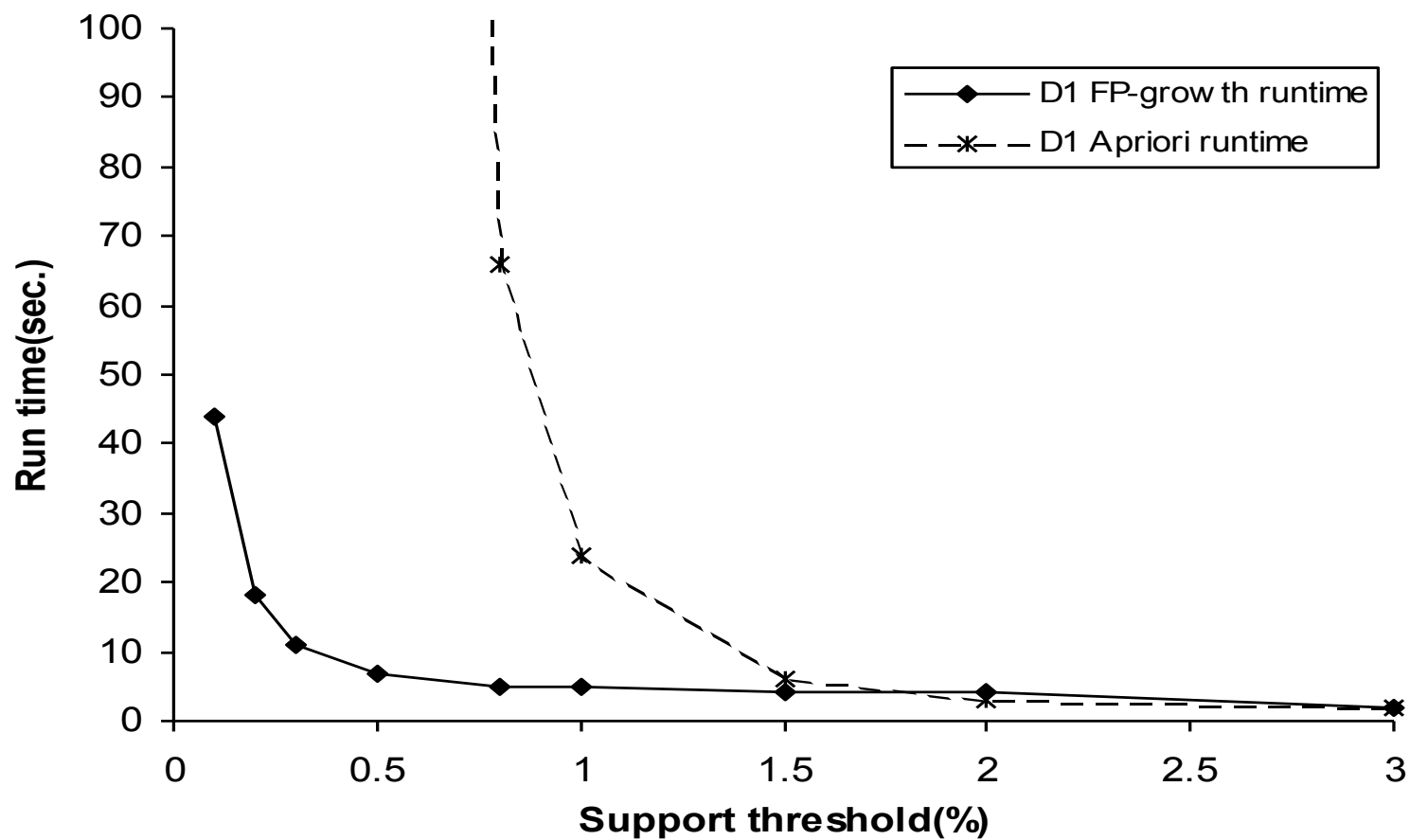
Prefix paths ending in CE



Prefix paths ending in AE

- 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



- 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

- 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

{2% milk} \Rightarrow {wheat bread}

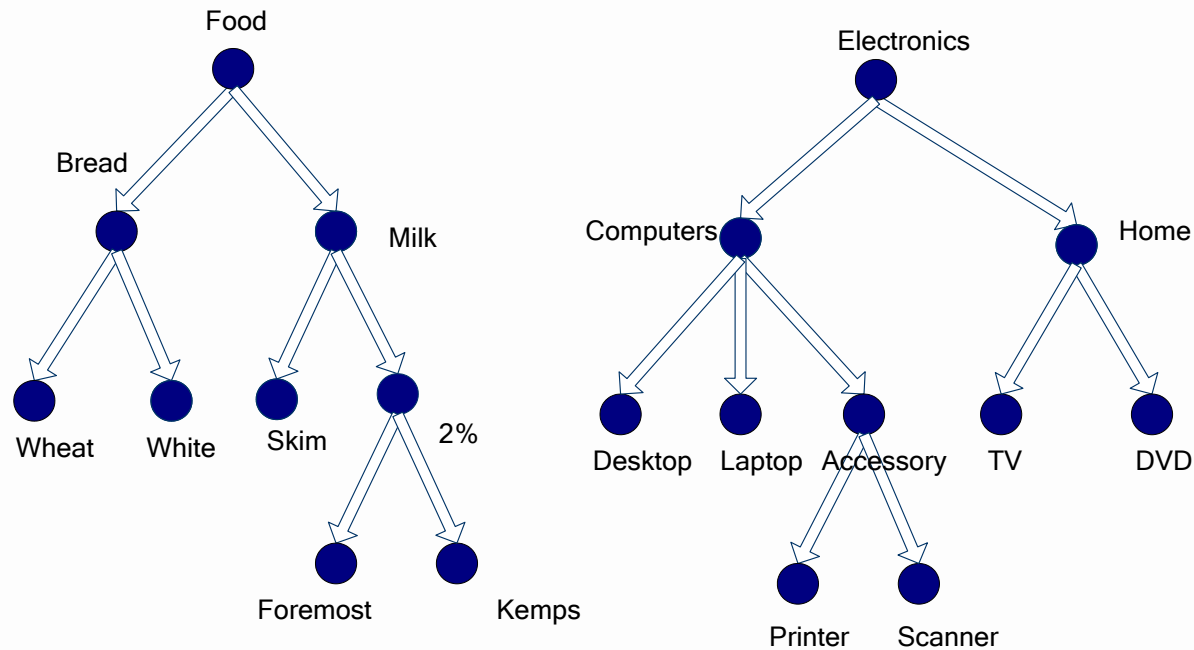
or

{milk} \Rightarrow {bread}

or

{Brand X 2% milk} \Rightarrow {Brand Y wheat bread}

- Items are often organized in hierarchies
- Items at the lower level are expected to have lower support
- Association rules should consider itemsets at appropriate levels



- 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
- For instance, an association between milk and bread may look like this

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

- How do support and confidence vary as we traverse the concept hierarchy?
- If X is the parent item for both X_1 and X_2 , then $\sigma(X) \leq \sigma(X_1) + \sigma(X_2)$
- If $\sigma(X_1 \cup Y_1) \geq \text{minsup}$, and X is parent of X_1 , Y is parent of Y_1 , then $\sigma(X \cup Y_1) \geq \text{minsup}$, $\sigma(X_1 \cup Y) \geq \text{minsup}$, $\sigma(X \cup Y) \geq \text{minsup}$
- If $\text{conf}(X_1 \Rightarrow Y_1) \geq \text{minconf}$, then $\text{conf}(X_1 \Rightarrow Y) \geq \text{minconf}$

- Approach I
 - 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

- 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

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

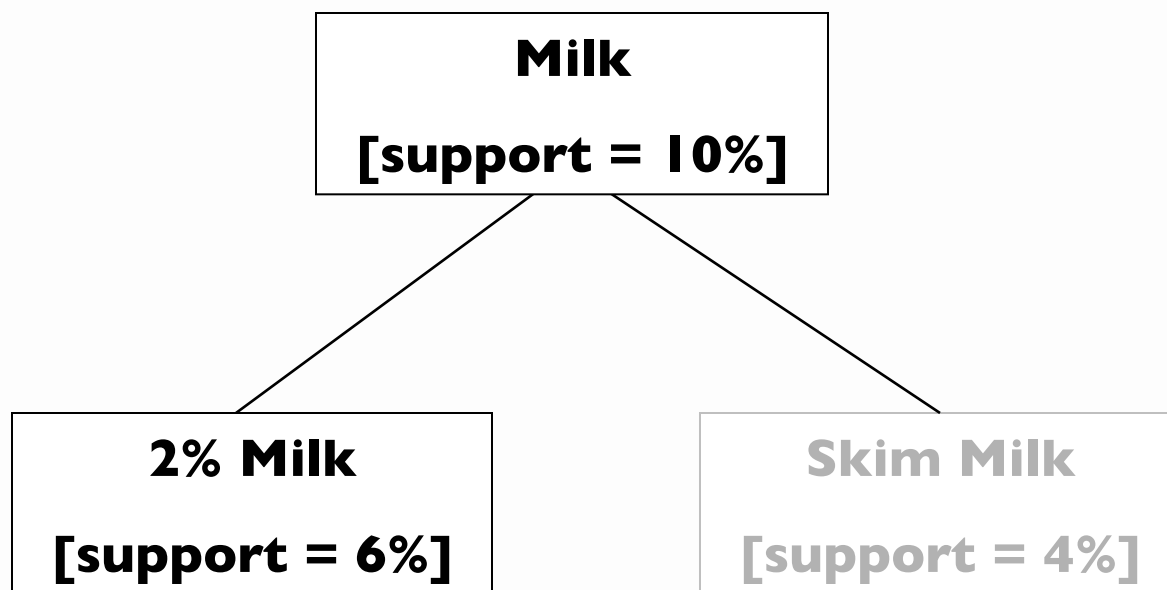
Multi-level Association: Redundancy Filtering

- Some rules may be redundant due to “ancestor” relationships between items
- Example
 - milk \Rightarrow wheat bread
[support = 8%, confidence = 70%]
 - 2% milk \Rightarrow wheat bread
[support = 2%, confidence = 72%]
- We say the first rule is an ancestor of the second rule.
- A rule is redundant if its support is close to the “expected” value, based on the rule’s ancestor.

Multi-level mining with uniform support

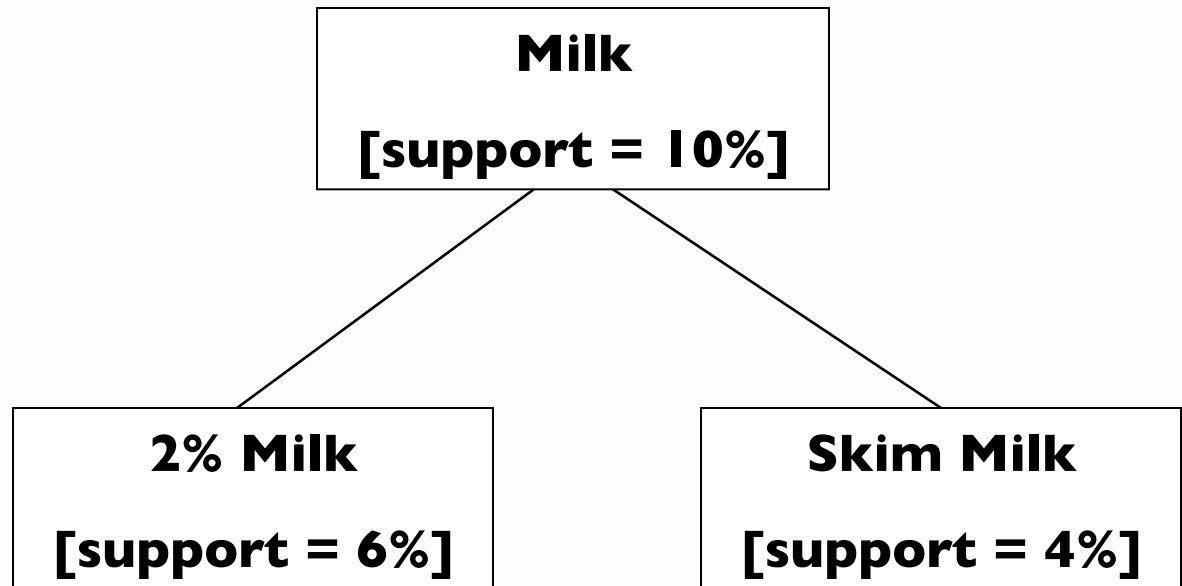
Level 1
minsup = 5%

Level 2
minsup = 5%



Level 1
minsup = 5%

Level 2
minsup = 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.

Sequential Pattern Mining

- Association rule mining does not consider the order of transactions
- However, in many applications order is important
- Examples
 - In market basket analysis, discover whether people buy products in sequence (bed then bed sheets)
 - In Web usage mining, discover navigational patterns of users in a Web site from sequences of page visits

- Web sequence
 - $\langle \{\text{Homepage}\} \{\text{Electronics}\} \{\text{Digital Cameras}\} \{\text{Canon Digital Camera}\} \{\text{Shopping Cart}\} \{\text{Order Confirmation}\} \{\text{Return to Shopping}\} \rangle$
- Sequence of initiating events causing the nuclear accident at 3-mile Island:
 - $\langle \{\text{clogged resin}\} \{\text{outlet valve closure}\} \{\text{loss of feedwater}\} \{\text{condenser polisher outlet valve shut}\} \{\text{booster pumps trip}\} \{\text{main waterpump trips}\} \{\text{main turbine trips}\} \{\text{reactor pressure increases}\} \rangle$
- Sequence of books checked out at a library:
 - $\langle \{\text{Fellowship of the Ring}\} \{\text{The Two Towers}\} \{\text{Return of the King}\} \rangle$

- Let $I = \{i_1, i_2, \dots, i_m\}$ be a set of items.
- A sequence is an ordered list of itemsets.
- We denote a sequence s by $\langle a_1, a_2, \dots, a_r \rangle$, where a_i is an itemset, also called an element of s .
- An element (or an itemset) of a sequence is denoted by $\{x_1, x_2, \dots, x_k\}$, 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 of a sequence
 - Number of elements (or itemsets) in the sequence
- Length
 - Number of items in the sequence
 - A sequence of length k is called k -sequence

- Given two sequences, $s_1 = \langle a_1 a_2 \dots a_r \rangle$ and $s_2 = \langle b_1 b_2 \dots b_v \rangle$
- s_1 is a subsequence of s_2 if there exist integers $1 \leq j_1 < j_2 < \dots < j_{r-1} < j_r \leq v$ such that $a_1 \subseteq b_{j_1}, a_2 \subseteq b_{j_2}, \dots, a_r \subseteq b_{j_r}$
- A sequence $s_1 = \langle a_1 a_2 \dots a_r \rangle$ is a subsequence of another sequence $s_2 = \langle b_1 b_2 \dots b_v \rangle$, or s_2 is a supersequence of s_1 , if there exist integers $1 \leq j_1 < j_2 < \dots < j_{r-1} < j_r \leq v$ such that $a_1 \subseteq b_{j_1}, a_2 \subseteq b_{j_2}, \dots, a_r \subseteq b_{j_r}$. We also say that s_2 contains s_1 .

- 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.

- 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 minsup

- 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

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