

Association Analysis

Chapter 4

Association Rule Mining

- Given a set of transactions, find rules that will predict the occurrence of an item based on the occurrences of other items in the transaction

Market-Basket transactions

<i>TID</i>	<i>Items</i>
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

Example of Association Rules

$\{\text{Diaper}\} \rightarrow \{\text{Beer}\}$,
 $\{\text{Milk, Bread}\} \rightarrow \{\text{Eggs, Coke}\}$,
 $\{\text{Beer, Bread}\} \rightarrow \{\text{Milk}\}$,

Implication means co-occurrence, not causality!

Definition: Frequent Itemset

- **Itemset**
 - A collection of one or more items
 - Example: {Milk, Bread, Diaper}
 - k-itemset
 - An itemset that contains k items
- **Support count (σ)**
 - Frequency of occurrence of an itemset
 - E.g. $\sigma(\{\text{Milk, Bread, Diaper}\}) = 2$
- **Support**
 - Fraction of transactions that contain an itemset
 - E.g. $s(\{\text{Milk, Bread, Diaper}\}) = 2/5$
- **Frequent Itemset**
 - An itemset whose support is greater than or equal to a *minsup* threshold

<i>TID</i>	<i>Items</i>
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

Definition: Association Rule

Association Rule

- An implication expression of the form $X \rightarrow Y$, where X and Y are itemsets
- Example:
 $\{\text{Milk, Diaper}\} \rightarrow \{\text{Beer}\}$

Rule Evaluation Metrics

- Support (s)
 - ◆ Fraction of transactions that contain both X and Y
- Confidence (c)
 - ◆ Measures how often items in Y appear in transactions that contain X

<i>TID</i>	<i>Items</i>
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

Example:

$$\{\text{Milk, Diaper}\} \Rightarrow \text{Beer}$$

$$s = \frac{\sigma(\text{Milk, Diaper, Beer})}{|T|} = \frac{2}{5} = 0.4$$

$$c = \frac{\sigma(\text{Milk, Diaper, Beer})}{\sigma(\text{Milk, Diaper})} = \frac{2}{3} = 0.67$$

Association Rule Mining Task

- Given a set of transactions T, the goal of association rule mining is to find all rules having
 - support $\geq \text{minsup}$ threshold
 - confidence $\geq \text{minconf}$ threshold
- Brute-force approach:
 - List all possible association rules
 - Compute the support and confidence for each rule
 - Prune rules that fail the minsup and minconf thresholds

⇒ Computationally prohibitive!

Mining Association Rules

TID	Items
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

Example of Rules:

$\{\text{Milk}, \text{Diaper}\} \rightarrow \{\text{Beer}\}$ (s=0.4, c=0.67)
 $\{\text{Milk}, \text{Beer}\} \rightarrow \{\text{Diaper}\}$ (s=0.4, c=1.0)
 $\{\text{Diaper}, \text{Beer}\} \rightarrow \{\text{Milk}\}$ (s=0.4, c=0.67)
 $\{\text{Beer}\} \rightarrow \{\text{Milk}, \text{Diaper}\}$ (s=0.4, c=0.67)
 $\{\text{Diaper}\} \rightarrow \{\text{Milk}, \text{Beer}\}$ (s=0.4, c=0.5)
 $\{\text{Milk}\} \rightarrow \{\text{Diaper}, \text{Beer}\}$ (s=0.4, c=0.5)

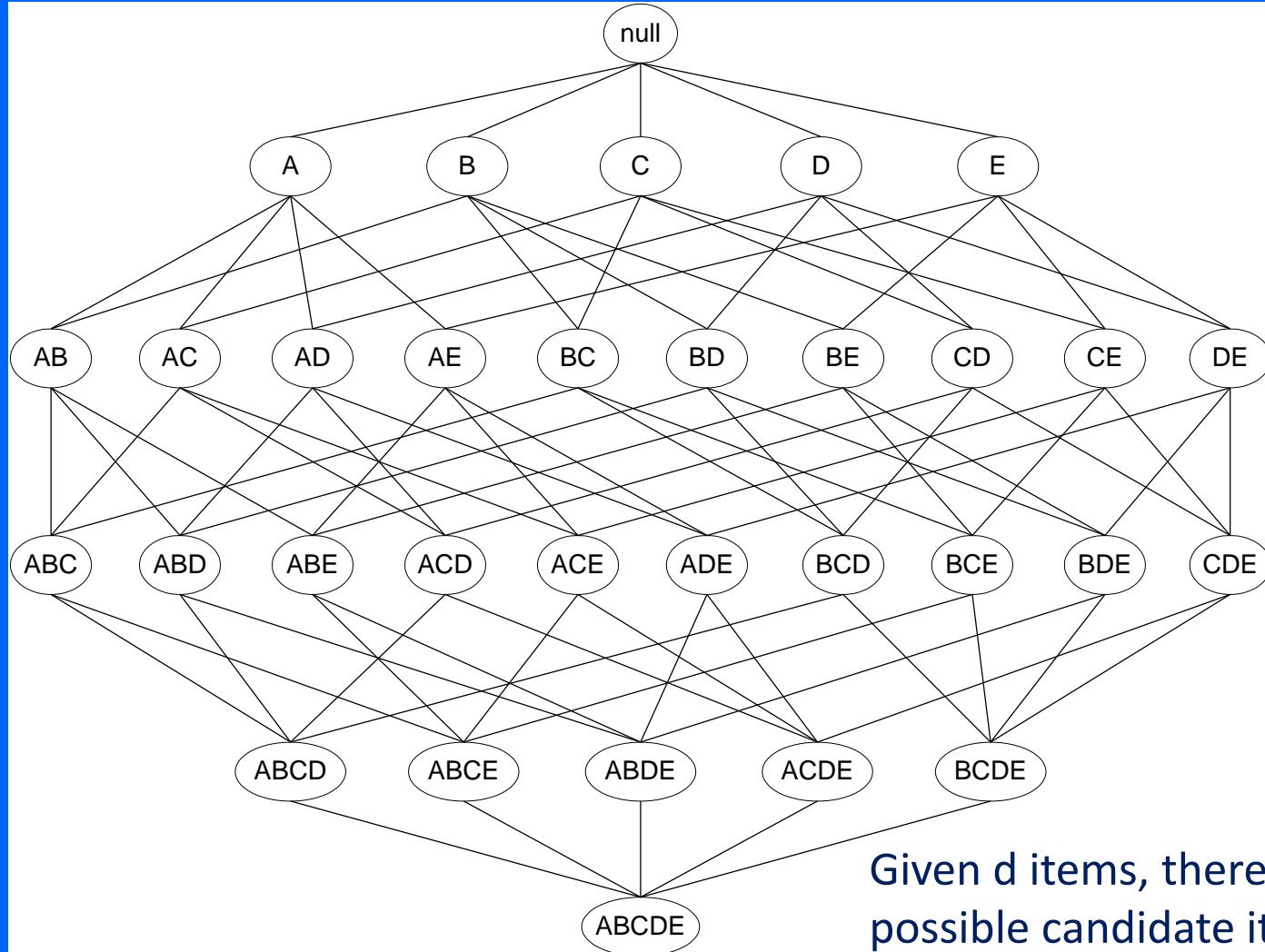
Observations:

- All the above rules are binary partitions of the same itemset:
 $\{\text{Milk}, \text{Diaper}, \text{Beer}\}$
- Rules originating from the same itemset have identical support but can have different confidence
- Thus, we may decouple the support and confidence requirements

Mining Association Rules

- Two-step approach:
 1. Frequent Itemset Generation
 - Generate all itemsets whose support $\geq \text{minsup}$
 2. Rule Generation
 - Generate high confidence rules from each frequent itemset, where each rule is a binary partitioning of a frequent itemset
- Frequent itemset generation is still computationally expensive

Frequent Itemset Generation



Frequent Itemset Generation

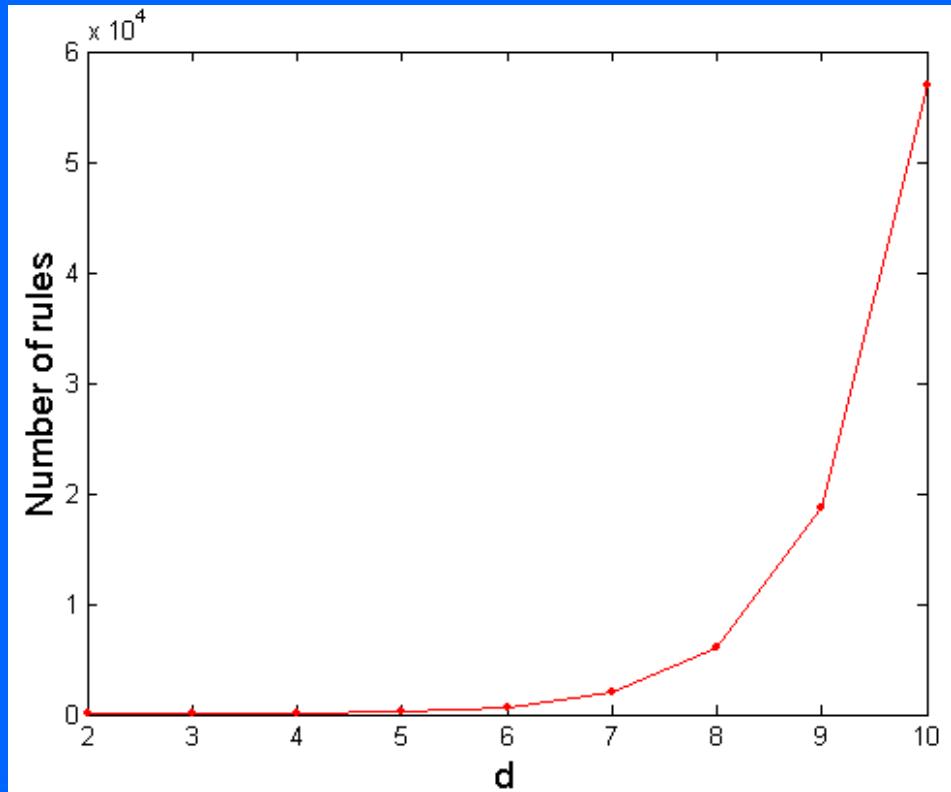
- Brute-force approach:
 - Each itemset in the lattice is a **candidate** frequent itemset
 - Count the support of each candidate by scanning the database



- Match each transaction against every candidate
- Complexity $\sim O(NMw) \Rightarrow$ Expensive since $M = 2^d !!!$

Computational Complexity

- Given d unique items:
 - Total number of itemsets = 2^d
 - Total number of possible association rules:



$$\begin{aligned} R &= \sum_{k=1}^{d-1} \binom{d}{k} \times \sum_{j=1}^{d-k} \binom{d-k}{j} \\ &= 3^d - 2^{d+1} + 1 \end{aligned}$$

If $d=6$, $R = 602$ rules

Frequent Itemset Generation Strategies

- Reduce the number of candidates (M)
 - Complete search: $M=2^d$
 - Use pruning techniques to reduce M
- Reduce the number of transactions (N)
 - Reduce size of N as the size of itemset increases
 - Used by DHP and vertical-based mining algorithms
- Reduce the number of comparisons (NM)
 - Use efficient data structures to store the candidates or transactions
 - No need to match every candidate against every transaction

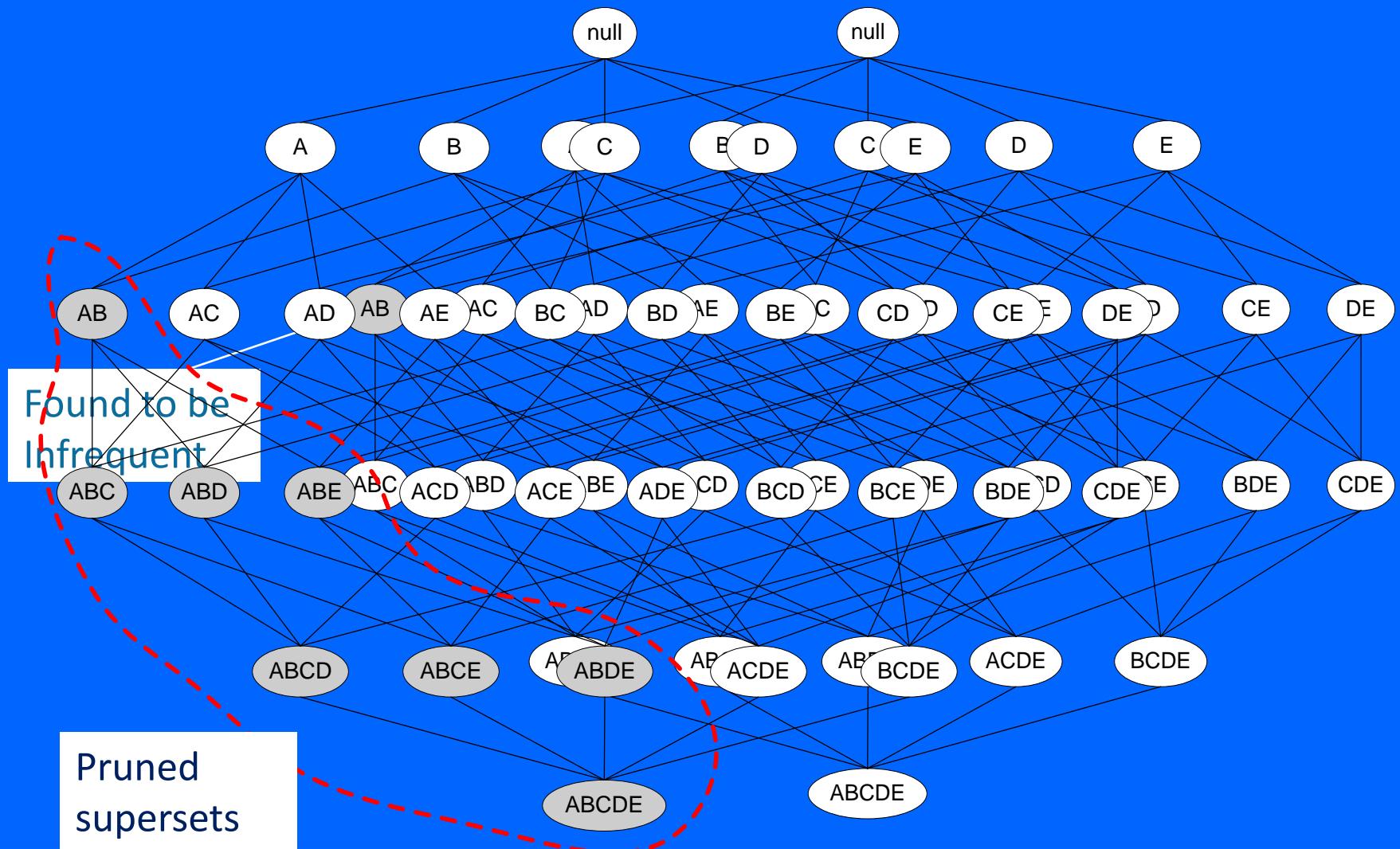
Reducing Number of Candidates

- Apriori principle:
 - If an itemset is frequent, then all of its subsets must also be frequent
- Apriori principle holds due to the following property of the support measure:

$$\forall X, Y : (X \subseteq Y) \Rightarrow s(X) \geq s(Y)$$

- Support of an itemset never exceeds the support of its subsets
- This is known as the **anti-monotone** property of support

Illustrating Apriori Principle



Illustrating Apriori Principle

Item	Count
Bread	4
Coke	2
Milk	4
Beer	3
Diaper	4
Eggs	1

Items (1-itemsets)



Itemset	Count
{Bread,Milk}	3
{Bread,Beer}	2
{Bread,Diaper}	3
{Milk,Beer}	2
{Milk,Diaper}	3
{Beer,Diaper}	3

Pairs (2-itemsets)

(No need to generate candidates involving Coke or Eggs)

Minimum Support = 3

If every subset is considered,

$${}^6C_1 + {}^6C_2 + {}^6C_3 = 41$$

With support-based pruning,

$$6 + 6 + 1 = 13$$



Triplets (3-itemsets)

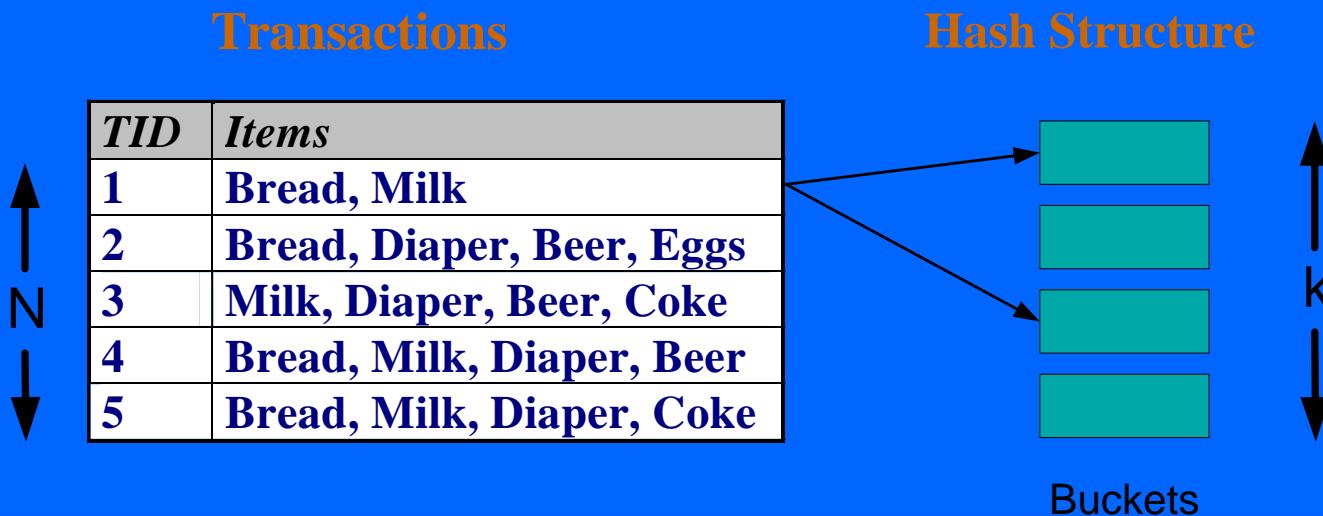
Itemset	Count
{Bread,Milk,Diaper}	3

Apriori Algorithm

- Method:
 - Let $k=1$
 - Generate frequent itemsets of length 1
 - Repeat until no new frequent itemsets are identified
 - Generate length $(k+1)$ candidate itemsets from length k frequent itemsets
 - Prune candidate itemsets containing subsets of length k that are infrequent
 - Count the support of each candidate by scanning the DB
 - Eliminate candidates that are infrequent, leaving only those that are frequent

Reducing Number of Comparisons

- Candidate counting:
 - Scan the database of transactions to determine the support of each candidate itemset
 - To reduce the number of comparisons, store the candidates in a hash structure
 - Instead of matching each transaction against every candidate, match it against candidates contained in the hashed buckets



Example of Apriori algorithm

- Consider the following transaction table

Transaction No.	Items
T1	1, 2, 3, 4, 5, 6
T2	7, 2, 3, 4, 5, 6
T3	1, 8, 4, 5
T4	1, 9, 0, 4, 6
T5	0, 2, 2, 4, 5

Example of Apriori algorithm

- Step 1: Count the occurrence of each item.

Item	Occurrence / Frequency
1	3
2	3
3	2
4	5
5	4
6	3
7	1
8	1
9	2
0	2

Example of Apriori algorithm

- Step 2: Remember, the algorithm says, an item is considered to be frequent if it's bought more than the Support/Threshold i.e. 3. Therefore, below is the list of Frequent Singletons.

Item	Occurrence / Frequency
1	3
2	3
4	5
5	4
6	3

Example of Apriori algorithm

- Step 3: We start making pairs out of the frequent itemsets we got in the above step.

ItemPairs
12
14
15
16
24
25
26
45
46
56

Example of Apriori algorithm

- Step 4: After getting the frequent Item Pairs, we start counting the occurrence of these pairs in the Transaction Set.

ItemPairs	Occurrence / Frequency
12	1
14	2
15	2
16	1
24	3
25	3
26	2
45	4
46	3
56	2

Example of Apriori algorithm

- Step 5: Now again, follow the Golden Rule, and discard non-frequent pairs.

ItemPairs	Occurrence / Frequency
14	3
24	3
25	3
45	4
46	3

Example of Apriori algorithm

- Now we have a table with pair of frequent items. Suppose we want to find frequent triplets. We the above table and make all the possible combinations.
- Step 6: Make combinations of triples using the frequent Item pairs.**
- To make triples, the rule is: IF 12 and 13 are frequent, then the triple would be 123. Similarly, if 24 and 26 then triple would be 246.
- So, using the above logic and our Frequent ItemPairs table, we get the below triples:

ItemTriples
245
456

Example of Apriori algorithm

- Step 7: Get the count of the above triples (Candidates).

ItemTriples	Occurrence / Frequency
245	3
456	2

- After, this, if we can find quartets, then we find those and count their occurrence/frequency.
- If we had 123, 124, 134, 135, 234 and we wanted to generate a quartet then it would be 1234 and 1345. And after finding quartet we would have again got their count of occurrence /frequency and repeated the same also, until the Frequent ItemSet is null.

Example of Apriori algorithm

- Thus, the frequent ItemSets are:
- Frequent Itemsets of Size 1: 1, 2, 4, 5, 6
- Frequent Itemsets of Size 2: 14, 24, 25, 45, 46
- Frequent Itemsets of Size 3: 245

Example of Apriori algorithm

- Now if we want to check the association rule for $\{2, 4\} \rightarrow 5$.
The confidence is: Ratio of $\{2, 4\} \cup \{5\}$ with support of $\{2, 4\}$. Therefore,
- Confidence = $3 / 3 \Rightarrow 1$
- We can say that, $\{2, 4\} \rightarrow \{5\}$ has a confidence of 1. But, we want to know how interesting the rule is. For this, we have a new parameter called Interest.
- Interest of an association rule is the difference of its confidence and the fraction of baskets which contain item j.
$$\begin{aligned}\{\{2, 4\} \rightarrow 5\} &= \text{conf}(\{2, 4\} \rightarrow 5) - Fr(5) \\ &= 1 - (3/4) \\ &= 1 - .75 \\ &= .25\end{aligned}$$

Therefore, the Interest is just 25 %. It's not an interesting rule.

Frequent Pattern (FP) Growth Method

- Mining frequent itemsets without candidate generation.
- It is a divide and conquers strategy.
- It compresses the database representing frequent items into a frequent-pattern tree (FP-Tree), which retains the itemset association information.
- Divides the compressed database into a set of conditional databases, each associated with one frequent item or pattern fragment and then mines each such database separately.

Frequent Pattern (FP) Growth Method

- Choice of minimum support threshold: Lowering support threshold results in more frequent itemsets. This may increase number of candidates and max length of frequent itemsets.
- Dimensionality (number of items) of the data set: More space is needed to store support count of each item. If number of frequent items increases, both computation and I/O costs may also increase.
- Size of database: Since Apriori makes multiple passes, run time of algorithm may increase with number of transactions.
- Average transaction width: Transaction width increases with denser data sets. This may increase max length of frequent itemsets and traversals of hash tree (number of subsets in a transaction increases with its width)

Apriori and Fp-Growth difference

- Apriori: uses a generate-and-test approach – generates candidate itemsets and tests if they are frequent
 - Generation of candidate itemsets is expensive(in both space and time)
 - Support counting is expensive
 - Subset checking (computationally expensive)
 - Multiple Database scans (I/O)
- FP-Growth: allows frequent itemset discovery without candidate itemset generation. Two step approach:
 - Step 1: Build a compact data structure called the FP-tree
 - Built using 2 passes over the data-set.
 - Step 2: Extracts frequent itemsets directly from the FP-tree

Frequent Pattern (FP) Growth Method

- FP-Growth method transforms the problem of finding long frequent patterns to searching for shorter ones recursively and then concatenating the suffix.
- It uses least frequent items as suffix .
- Advantage: Reduce search cost, has good selectivity, faster than apriori.
- Disadvantage: When the database is large, it is sometimes unrealistic to construct a man memory based FP-tree.

Step 1: FP-Tree Construction

- FP-Tree is constructed using 2 passes over the data-set:

Pass 1:

- Scan data and find support for each item.
- Discard infrequent items.
- Sort frequent items in decreasing order based on their support.

Use this order when building the FP-Tree, so common prefixes can be shared.

Step 1: FP-Tree Construction

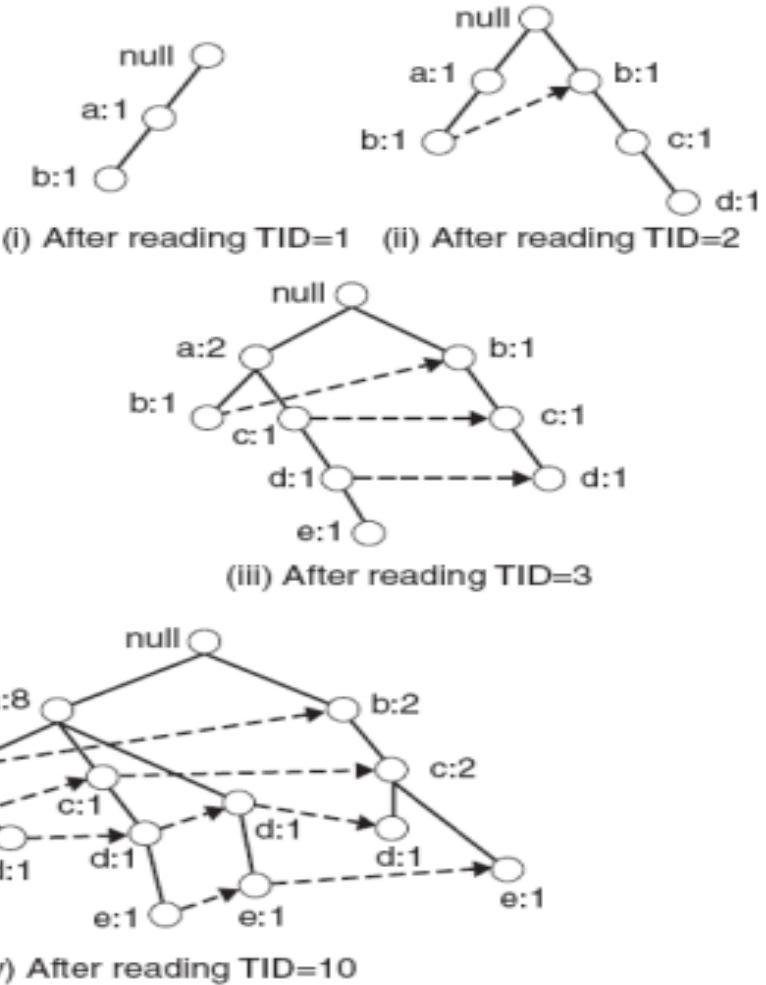
Pass 2:

Nodes correspond to items and have a counter

1. FP-Growth reads 1 transaction at a time and maps it to a path
2. Fixed order is used, so paths can overlap when transactions share items (when they have the same prefix).
 - In this case, counters are incremented
3. Pointers are maintained between nodes containing the same item, creating singly linked lists (dotted lines)
 - The more paths that overlap, the higher the compression. FP-tree may fit in memory.
4. Frequent itemsets extracted from the FP-Tree.

Step 1: FP-Tree Construction (Example)

Transaction Data Set	
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}



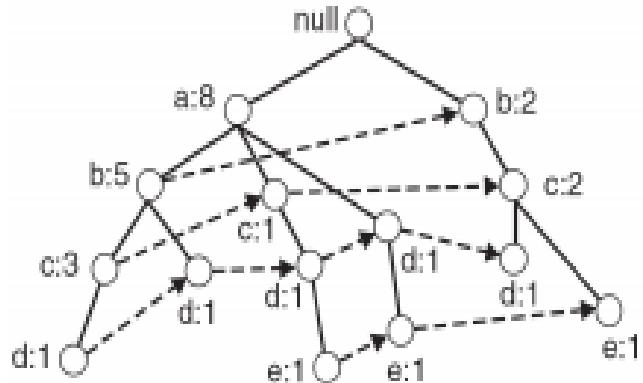
FP-Tree size

- The FP-Tree usually has a smaller size than the uncompressed data - typically many transactions share items (and hence prefixes).
 - Best case scenario: all transactions contain the same set of items.
 - 1 path in the FP-tree
 - Worst case scenario: every transaction has a unique set of items (no items in common)
 - Size of the FP-tree is at least as large as the original data.
 - Storage requirements for the FP-tree are higher - need to store the pointers between the nodes and the counters.
- The size of the FP-tree depends on how the items are ordered
- Ordering by decreasing support is typically used but it does not always lead to the smallest tree (it's a heuristic).

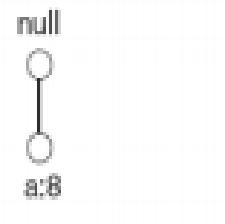
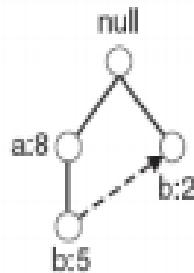
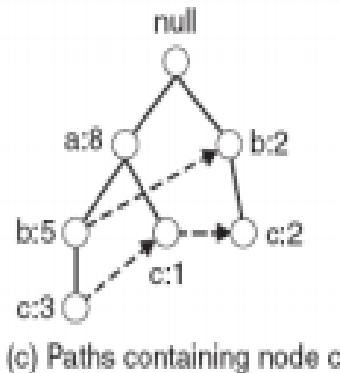
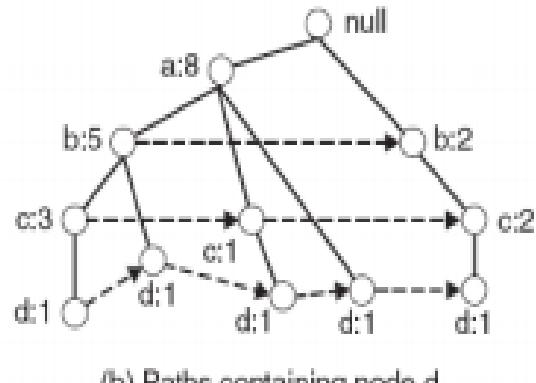
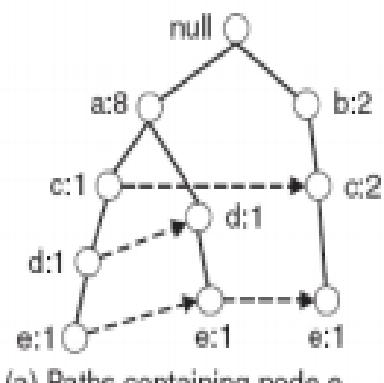
Step 2: Frequent Itemset Generation

- FP-Growth extracts frequent itemsets from the FP-tree.
- Bottom-up algorithm - from the leaves towards the root
- Divide and conquer: first look for frequent itemsets ending in e, then de, etc. . . then d, then cd, etc. . .
- First, extract prefix path sub-trees ending in an item(set). (hint: use the linked lists)

Prefix path sub-trees (Example)

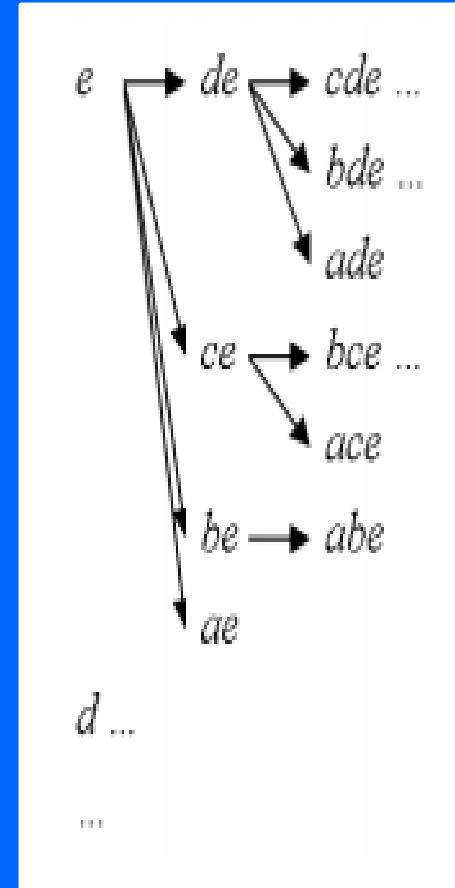


↑ Complete FP-tree



Step 2: Frequent Itemset Generation

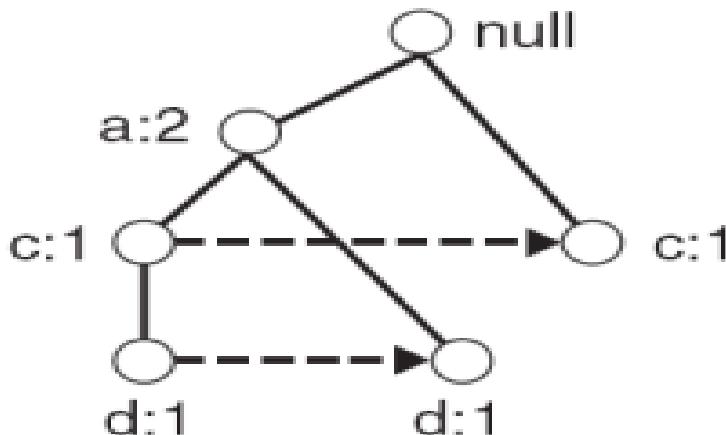
- Each prefix path sub-tree is processed recursively to extract the frequent itemsets. Solutions are then merged.
 - E.g. the prefix path sub-tree for e will be used to extract frequent itemsets ending in e , then in de , ce , be and ae , then in cde , bde , ade , etc.
 - Divide and conquer approach



Conditional FP-Tree

- The FP-Tree that would be built if we only consider transactions containing a particular itemset (and then removing that itemset from all transactions).
- Example: FP-Tree conditional one.

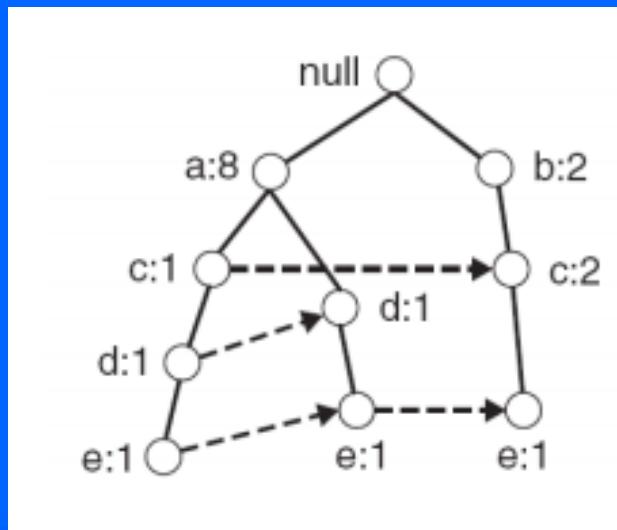
TID	Items
1	{a,b}
2	{b,c,d}
3	{a,c,d,✓}
4	{a,d,✓}
5	{a,b,c}
6	{a,b,c,d}
7	{a}
8	{a,b,c}
9	{a,b,d}
10	{b,c,✓}



Example

Let minSup = 2 and extract all frequent itemsets containing e.

- 1. Obtain the prefix path sub-tree for e:



Example

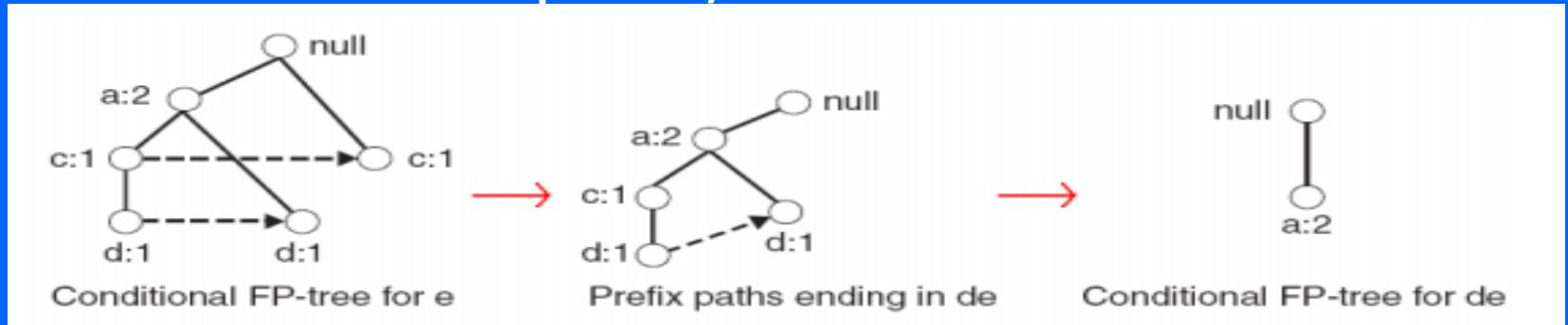
- 2. Check if e is a frequent item by adding the counts along the linked list (dotted line). If so, extract it.
 - Yes, count =3 so {e} is extracted as a frequent itemset.
- 3. As e is frequent, find frequent itemsets ending in e. i.e. de, ce, be and ae.

Example

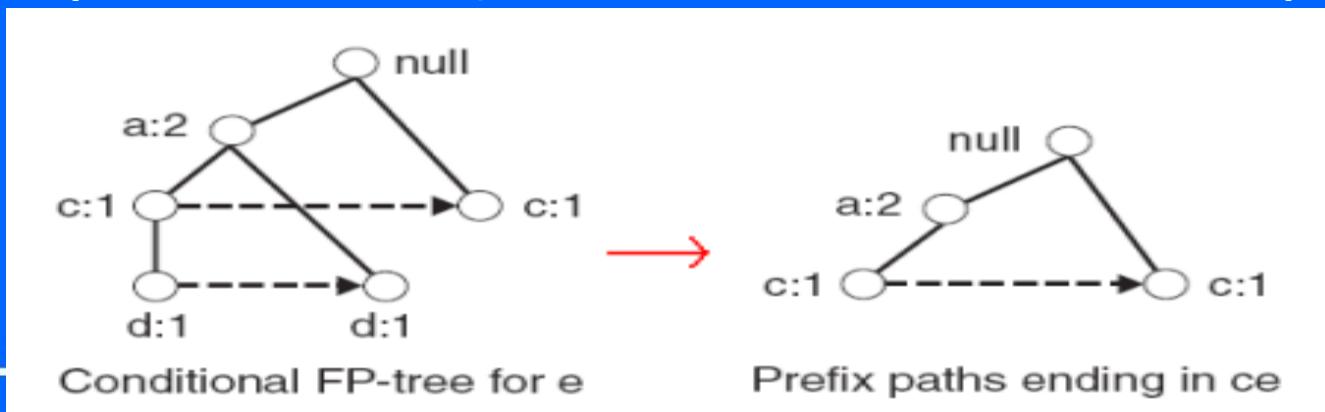
- 4. Use the the conditional FP-tree for e to find frequent itemsets ending in de, ce and ae
 - Note that be is not considered as b is not in the conditional FP-tree for e.
- I For each of them (e.g. de), find the prefix paths from the conditional tree for e, extract frequent itemsets, generate conditional FP-tree, etc... (recursive)

Example

- Example: $e \rightarrow de \rightarrow ade$ ($\{d,e\}$, $\{a,d,e\}$ are found to be frequent)



- Example: $e \rightarrow ce$ ($\{c,e\}$ is found to be frequent)



Result

Frequent itemsets found (ordered by suffix and order in which they are found):

Transaction Data Set	
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}

Suffix	Frequent Itemsets
e	{e}, {d,e}, {a,d,e}, {c,e},{a,e}
d	{d}, {c,d}, {b,c,d}, {a,c,d}, {b,d}, {a,b,d}, {a,d}
c	{c}, {b,c}, {a,b,c}, {a,c}
b	{b}, {a,b}
a	{a}

Advantages and disadvantages

➤ Advantages of FP-Growth

- only 2 passes over data-set
- “compresses” data-set
- no candidate generation
- much faster than Apriori

➤ Disadvantages of FP-Growth

- FP-Tree may not fit in memory!!
- FP-Tree is expensive to build

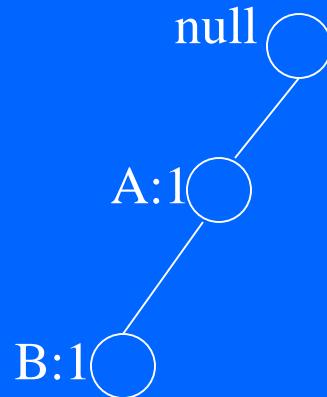
FP-growth Algorithm

- Use a compressed representation of the database using an **FP-tree**
- Once an FP-tree has been constructed, it uses a recursive divide-and-conquer approach to mine the frequent itemsets

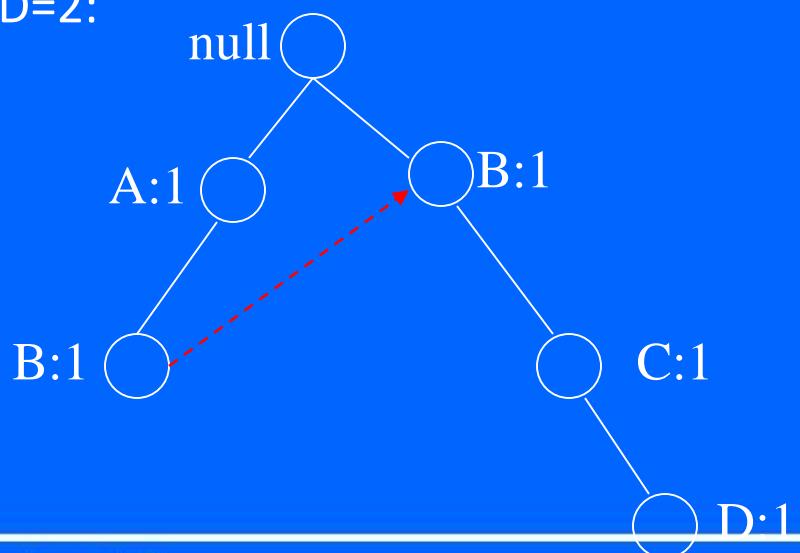
FP-tree construction

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=1:



After reading TID=2:



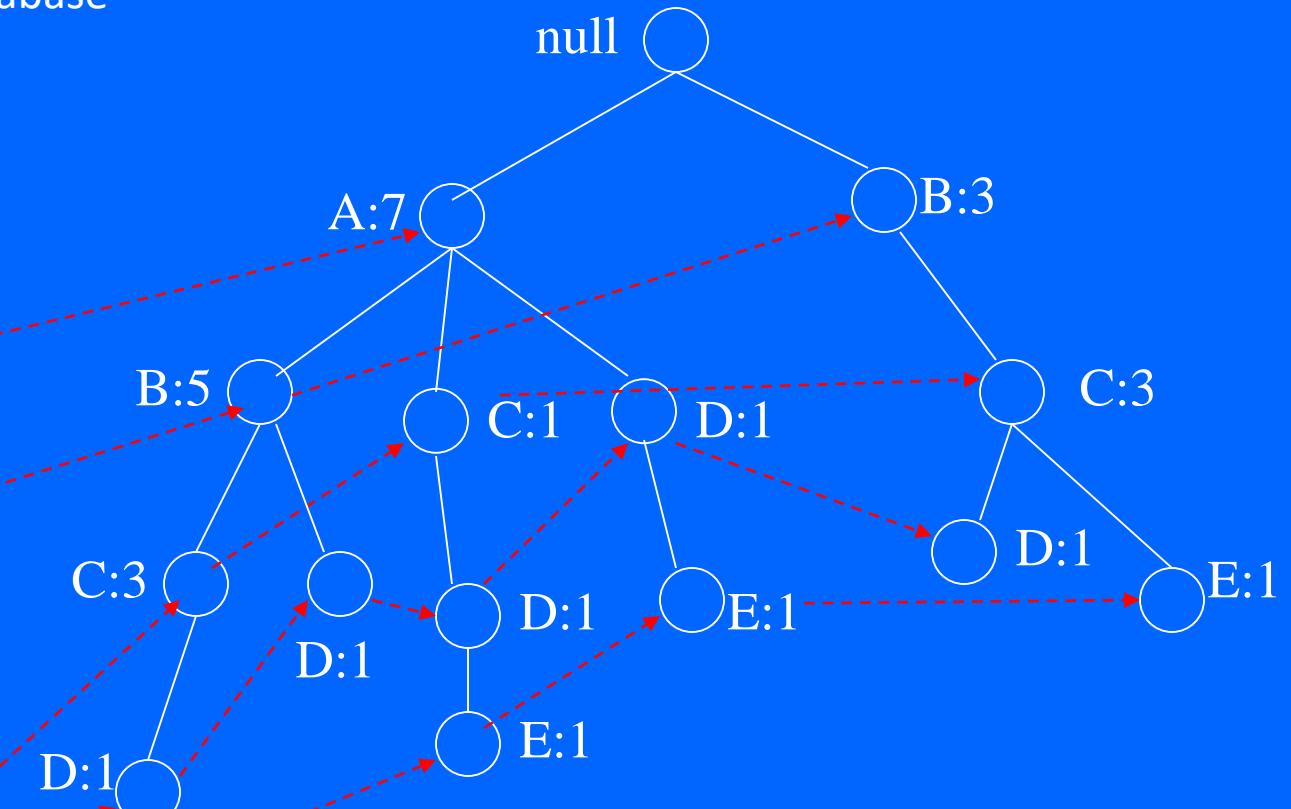
FP-Tree Construction

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}

Transaction Database

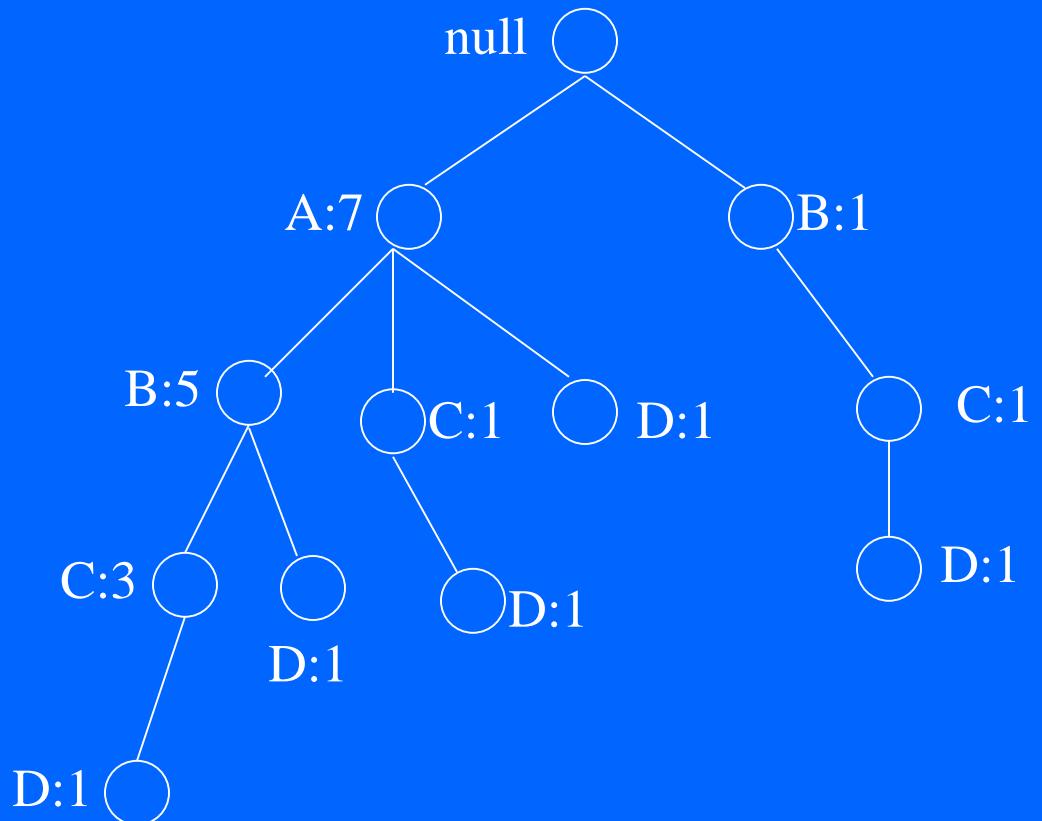
Header table

Item	Pointer
A	
B	
C	
D	
E	



Pointers are used to assist frequent itemset generation

FP-growth



Conditional Pattern base for D:

$$\begin{aligned} P = \{ &(A:1, B:1, C:1), \\ &(A:1, B:1), \\ &(A:1, C:1), \\ &(A:1), \\ &(B:1, C:1) \} \end{aligned}$$

Recursively apply FP-growth on P

Frequent Itemsets found (with sup > 1):

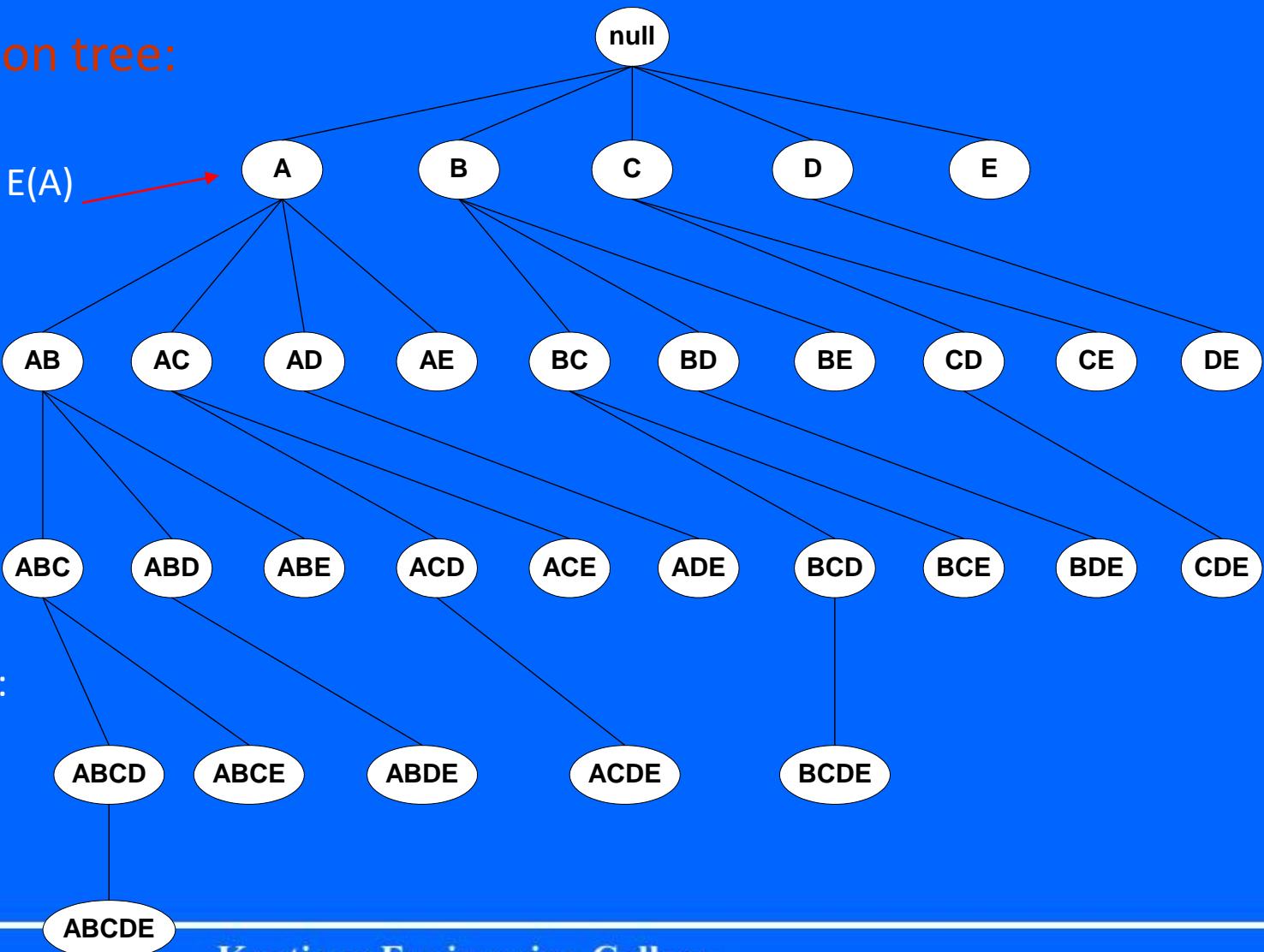
AD, BD, CD, ACD, BCD

Tree Projection

Set enumeration tree:

Possible Extension: $E(A) = \{B, C, D, E\}$

Possible Extension:
 $E(ABC) = \{D, E\}$



Tree Projection

- Items are listed in lexicographic order
- Each node P stores the following information:
 - Itemset for node P
 - List of possible lexicographic extensions of P: E(P)
 - Pointer to projected database of its ancestor node
 - Bitvector containing information about which transactions in the projected database contain the itemset

Projected Database

Original Database:

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}

Projected Database for node A:

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

For each transaction T, projected transaction at node A is $T \cap E(A)$

ECLAT

- For each item, store a list of transaction ids (tids)

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 Data Layout

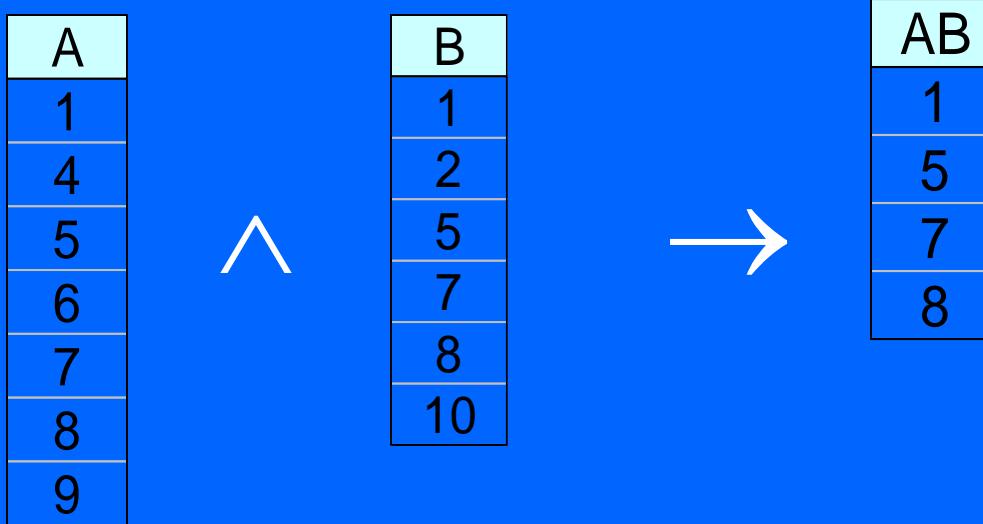
A	B	C	D	E
1	1	2	2	1
4	2	3	4	3
5	5	4	5	6
6	7	8	9	
7	8	9		
8	10			
9				

TID-list



ECLAT

- Determine support of any k-itemset by intersecting tid-lists of two of its (k-1) subsets.



- 3 traversal approaches:
 - top-down, bottom-up and hybrid
- Advantage: very fast support counting
- Disadvantage: intermediate tid-lists may become too large for memory

Rule Generation

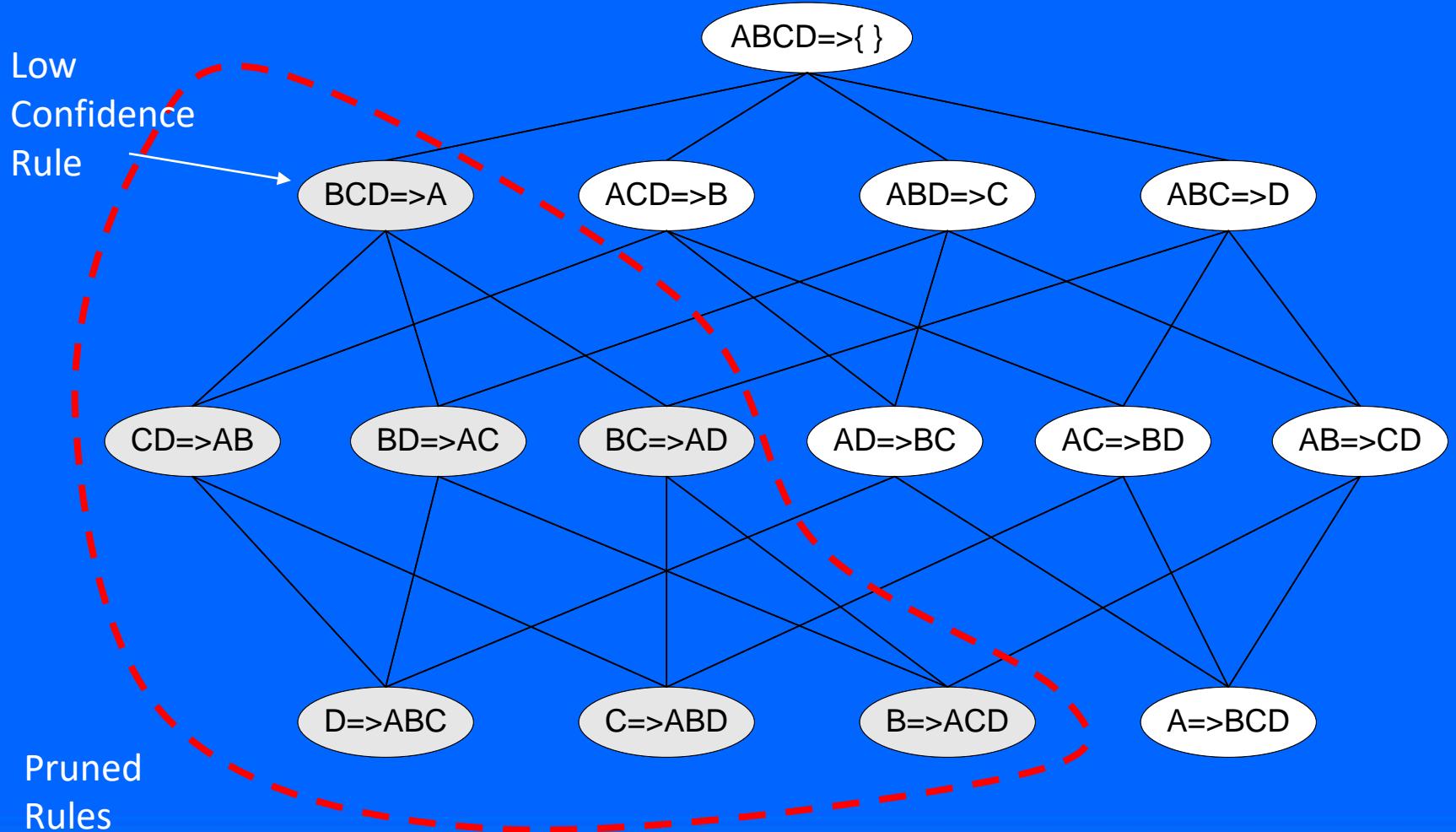
- Given a frequent itemset L, find all non-empty subsets $f \subset L$ such that $f \rightarrow L - f$ satisfies the minimum confidence requirement
 - If $\{A, B, C, D\}$ is a frequent itemset, candidate rules:
 $ABC \rightarrow D, ABD \rightarrow C, ACD \rightarrow B, BCD \rightarrow A,$
 $A \rightarrow BCD, B \rightarrow ACD, C \rightarrow ABD, D \rightarrow ABC$
 $AB \rightarrow CD, AC \rightarrow BD, AD \rightarrow BC, BC \rightarrow AD,$
 $BD \rightarrow AC, CD \rightarrow AB,$
- If $|L| = k$, then there are $2^k - 2$ candidate association rules (ignoring $L \rightarrow \emptyset$ and $\emptyset \rightarrow L$)

Rule Generation

- How to efficiently generate rules from frequent itemsets?
 - In general, confidence does not have an anti-monotone property $c(ABC \rightarrow D)$ can be larger or smaller than $c(AB \rightarrow D)$
 - But confidence of rules generated from the same itemset has an anti-monotone property
 - e.g., $L = \{A, B, C, D\}$:
$$c(ABC \rightarrow D) \geq c(AB \rightarrow CD) \geq c(A \rightarrow BCD)$$
- Confidence is anti-monotone w.r.t. number of items on the RHS of the rule

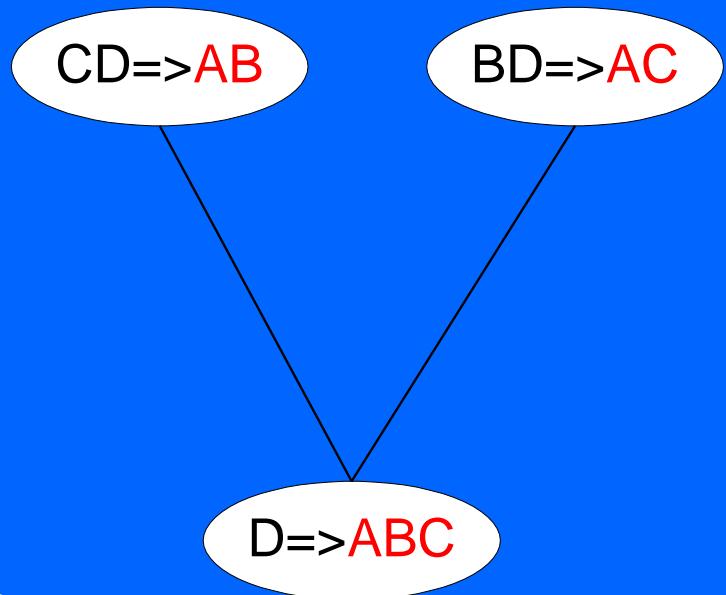
Rule Generation for Apriori Algorithm

Lattice of rules



Rule Generation for Apriori Algorithm

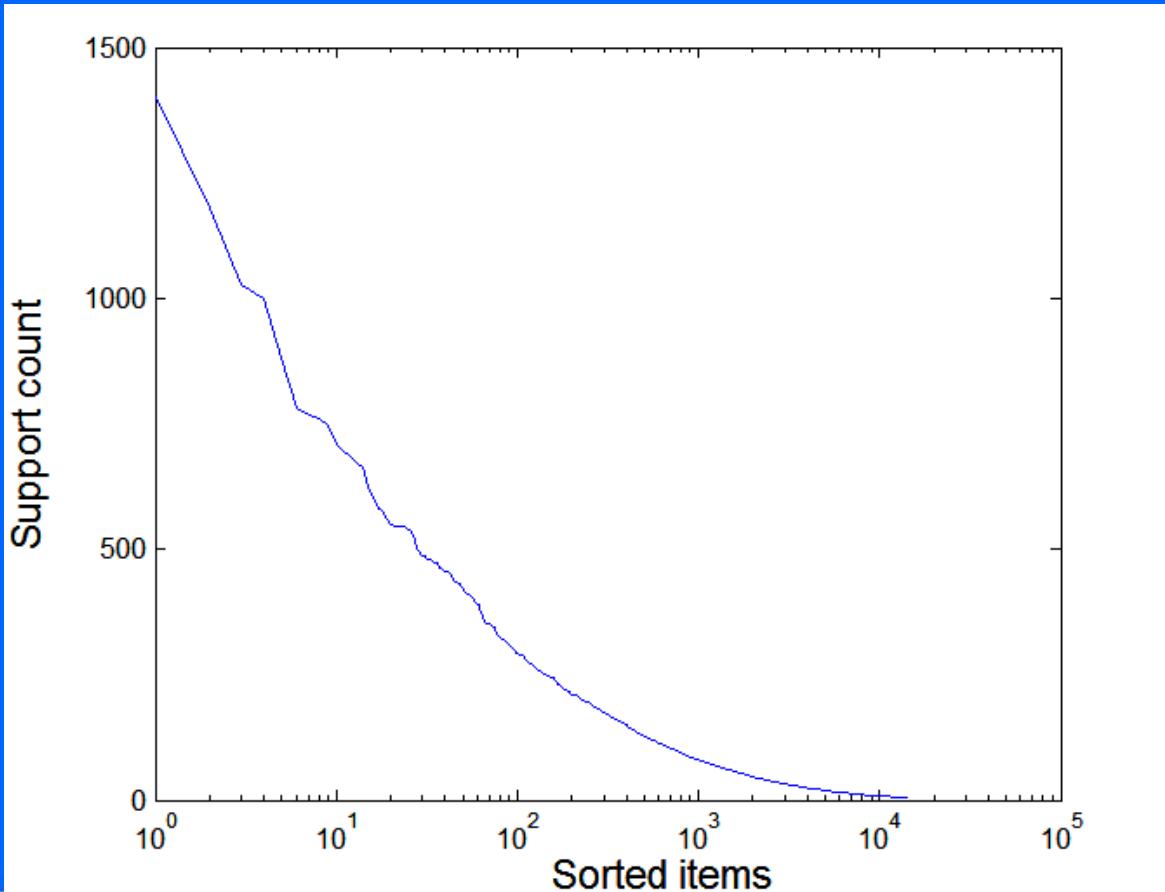
- Candidate rule is generated by merging two rules that share the same prefix in the rule consequent
- $\text{join}(\text{CD} \Rightarrow \text{AB}, \text{BD} \Rightarrow \text{AC})$ would produce the candidate rule $\text{D} \Rightarrow \text{ABC}$
- Prune rule $\text{D} \Rightarrow \text{ABC}$ if its subset $\text{AD} \Rightarrow \text{BC}$ does not have high confidence



Effect of Support Distribution

- Many real data sets have skewed support distribution

Support distribution of a retail data set



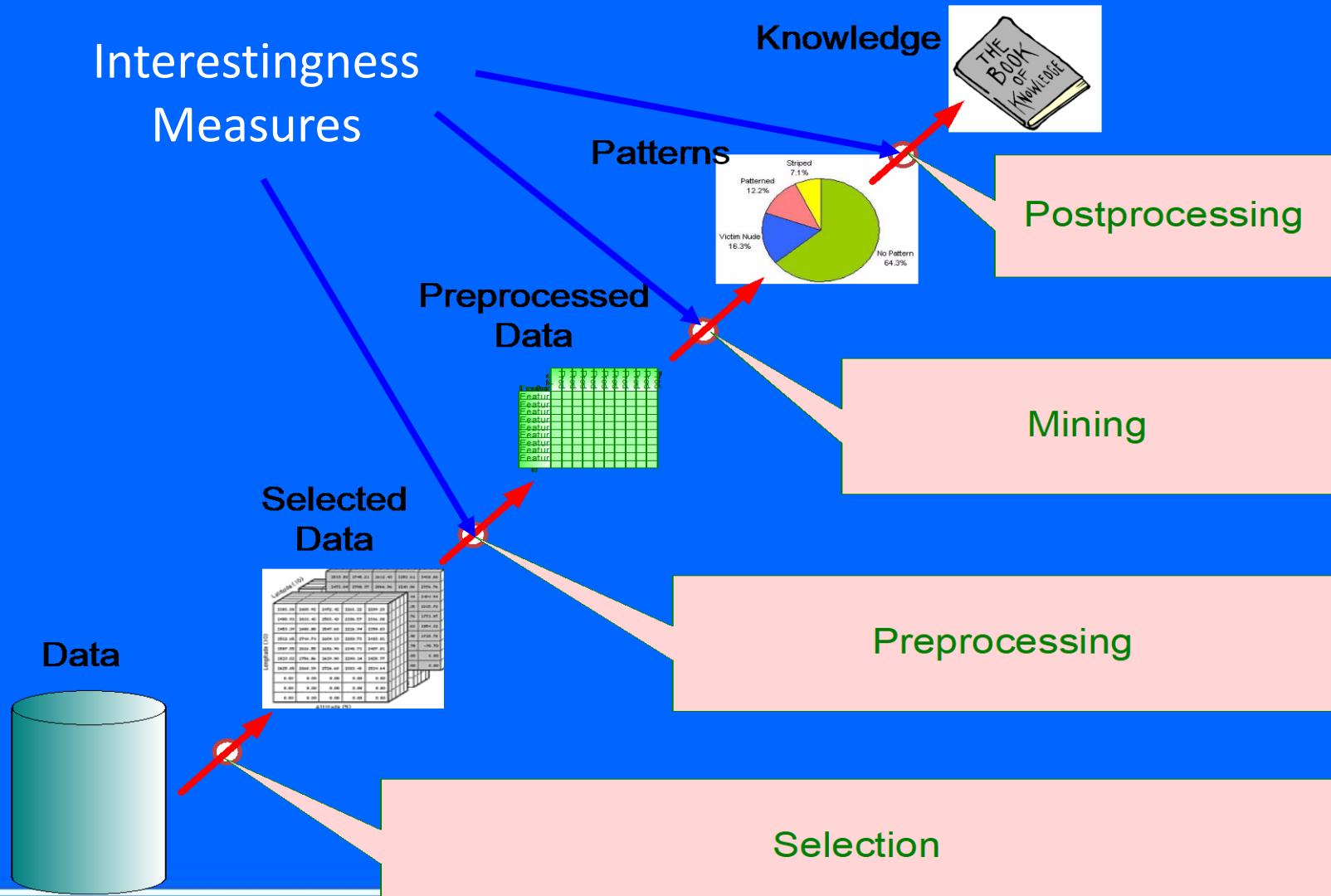
Effect of Support Distribution

- How to set the appropriate *minsup* threshold?
 - If *minsup* is set too high, we could miss itemsets involving interesting rare items (e.g., expensive products)
 - If *minsup* is set too low, it is computationally expensive and the number of itemsets is very large
- Using a single minimum support threshold may not be effective

Pattern Evaluation

- Association rule algorithms tend to produce too many rules
 - many of them are uninteresting or redundant
 - Redundant if $\{A,B,C\} \rightarrow \{D\}$ and $\{A,B\} \rightarrow \{D\}$ have same support & confidence
- Interestingness measures can be used to prune/rank the derived patterns
- In the original formulation of association rules, support & confidence are the only measures used

Application of Interestingness Measure



Computing Interestingness Measure

- Given a rule $X \rightarrow Y$, information needed to compute rule interestingness can be obtained from a contingency table

Contingency table for $X \rightarrow Y$

	Y	\bar{Y}	
X	f_{11}	f_{10}	f_{1+}
\bar{X}	f_{01}	f_{00}	f_{0+}
	f_{+1}	f_{+0}	$ T $

f_{11} : support of X and Y

f_{10} : support of X and \bar{Y}

f_{01} : support of \bar{X} and Y

f_{00} : support of \bar{X} and \bar{Y}

Used to define various measures

- support, confidence, lift, Gini, J-measure, etc.

Drawback of Confidence

	Coffee	<u>Coffee</u>	
Tea	15	5	20
<u>Tea</u>	75	5	80
	90	10	100

Association Rule: Tea → Coffee

$$\text{Confidence} = P(\text{Coffee} | \text{Tea}) = 0.75$$

$$\text{but } P(\text{Coffee}) = 0.9$$

⇒ Although confidence is high, rule is misleading

$$\Rightarrow P(\text{Coffee} | \overline{\text{Tea}}) = 0.9375$$

Statistical Independence

- Population of 1000 students
 - 600 students know how to swim (S)
 - 700 students know how to bike (B)
 - 420 students know how to swim and bike (S,B)
 - $P(S \wedge B) = 420/1000 = 0.42$
 - $P(S) \times P(B) = 0.6 \times 0.7 = 0.42$
 - $P(S \wedge B) = P(S) \times P(B) \Rightarrow$ Statistical independence
 - $P(S \wedge B) > P(S) \times P(B) \Rightarrow$ Positively correlated
 - $P(S \wedge B) < P(S) \times P(B) \Rightarrow$ Negatively correlated

Statistical-based Measures

- Measures that take into account statistical dependence

$$Lift = \frac{P(Y | X)}{P(Y)}$$

$$Interest = \frac{P(X, Y)}{P(X)P(Y)}$$

$$PS = P(X, Y) - P(X)P(Y)$$

$$\phi-coefficient = \frac{P(X, Y) - P(X)P(Y)}{\sqrt{P(X)[1 - P(X)]P(Y)[1 - P(Y)]}}$$

Example: Lift/Interest

	Coffee	<u>Coffee</u>	
Tea	15	5	20
<u>Tea</u>	75	5	80
	90	10	100

Association Rule: Tea → Coffee

$$\text{Confidence} = P(\text{Coffee}|\text{Tea}) = 0.75$$

$$\text{but } P(\text{Coffee}) = 0.9$$

$$\Rightarrow \text{Lift} = 0.75/0.9 = 0.8333 (< 1, \text{ therefore is negatively associated})$$

Drawback of Lift & Interest

	Y	\bar{Y}	
X	10	0	10
\bar{X}	0	90	90
	10	90	100

	Y	\bar{Y}	
X	90	0	90
\bar{X}	0	10	10
	90	10	100

$$Lift = \frac{0.1}{(0.1)(0.1)} = 10$$

$$Lift = \frac{0.9}{(0.9)(0.9)} = 1.11$$

Statistical independence:

If $P(X,Y) = P(X)P(Y)$ \Rightarrow Lift = 1

There are lots of measures proposed in the literature

Some measures are good for certain applications, but not for others

What criteria should we use to determine whether a measure is good or bad?

What about Apriori-style support based pruning? How does it affect these measures?

#	Measure	Formula
1	ϕ -coefficient	$\frac{P(A,B) - P(A)P(B)}{\sqrt{P(A)P(B)(1-P(A))(1-P(B))}}$
2	Goodman-Kruskal's (λ)	$\frac{\sum_j \max_i P(A_j, B_k) + \sum_k \max_j P(A_j, B_k) - \max_j P(A_j) - \max_k P(B_k)}{2 - \max_j P(A_j) - \max_k P(B_k)}$
3	Odds ratio (α)	$\frac{P(A, B)P(\bar{A}, \bar{B})}{P(\bar{A}, \bar{B})P(\bar{A}, B)}$
4	Yule's Q	$\frac{P(A, B)P(\bar{A}B) - P(A, \bar{B})P(\bar{A}, B)}{P(A, B)P(\bar{A}B) + P(A, \bar{B})P(\bar{A}, B)} = \frac{\alpha - 1}{\alpha + 1}$
5	Yule's Y	$\frac{\sqrt{P(A, B)P(\bar{A}B)} - \sqrt{P(A, \bar{B})P(\bar{A}, B)}}{\sqrt{P(A, B)P(\bar{A}B)} + \sqrt{P(A, \bar{B})P(\bar{A}, B)}} = \frac{\sqrt{\alpha} - 1}{\sqrt{\alpha} + 1}$
6	Kappa (κ)	$\frac{P(A, B) + P(\bar{A}, \bar{B}) - P(A)P(B) - P(\bar{A})P(\bar{B})}{1 - P(A)P(B) - P(\bar{A})P(\bar{B})}$
7	Mutual Information (M)	$\frac{\sum_i \sum_j P(A_i, B_j) \log \frac{P(A_i, B_j)}{P(A_i)P(B_j)}}{\min(-\sum_i P(A_i) \log P(A_i), -\sum_j P(B_j) \log P(B_j))}$
8	J-Measure (J)	$\max \left(P(A, B) \log \left(\frac{P(B A)}{P(B)} \right) + P(\bar{A}B) \log \left(\frac{P(\bar{B} A)}{P(\bar{B})} \right), P(A, \bar{B}) \log \left(\frac{P(A \bar{B})}{P(A)} \right) + P(\bar{A}\bar{B}) \log \left(\frac{P(\bar{A} \bar{B})}{P(\bar{A})} \right) \right)$
9	Gini index (G)	$\max \left(P(A)[P(B A)^2 + P(\bar{B} A)^2] + P(\bar{A})[P(B \bar{A})^2 + P(\bar{B} \bar{A})^2] - P(B)^2 - P(\bar{B})^2, P(B)[P(A B)^2 + P(\bar{A} B)^2] + P(\bar{B})[P(A \bar{B})^2 + P(\bar{A} \bar{B})^2] - P(A)^2 - P(\bar{A})^2 \right)$
10	Support (s)	$P(A, B)$
11	Confidence (c)	$\max(P(B A), P(A B))$
12	Laplace (L)	$\max \left(\frac{NP(A, B) + 1}{NP(A) + 2}, \frac{NP(A, B) + 1}{NP(B) + 2} \right)$
13	Conviction (V)	$\max \left(\frac{P(A)P(\bar{B})}{P(A\bar{B})}, \frac{P(B)P(\bar{A})}{P(B\bar{A})} \right)$
14	Interest (I)	$\frac{P(A, B)}{P(A)P(B)}$
15	cosine (IS)	$\frac{P(A, B)}{\sqrt{P(A)P(B)}}$
16	Piatetsky-Shapiro's (PS)	$P(A, B) - P(A)P(B)$
17	Certainty factor (F)	$\max \left(\frac{P(B A) - P(B)}{1 - P(B)}, \frac{P(A B) - P(A)}{1 - P(A)} \right)$
18	Added Value (AV)	$\max(P(B A) - P(B), P(A B) - P(A))$
19	Collective strength (S)	$\frac{P(A, B) + P(\bar{A}B)}{P(A)P(B) + P(\bar{A})P(B)} \times \frac{1 - P(A)P(B) - P(\bar{A})P(\bar{B})}{1 - P(A, B) - P(\bar{A}\bar{B})}$
20	Jaccard (ζ)	$\frac{P(A \cap B)}{P(A) + P(B) - P(A \cap B)}$
21	Klosgen (K)	$\sqrt{P(A, B)} \max(P(B A) - P(B), P(A B) - P(A))$

Properties of A Good Measure

- Piatetsky-Shapiro:
3 properties a good measure M must satisfy:
 - $M(A,B) = 0$ if A and B are statistically independent
 - $M(A,B)$ increase monotonically with $P(A,B)$ when $P(A)$ and $P(B)$ remain unchanged
 - $M(A,B)$ decreases monotonically with $P(A)$ [or $P(B)$] when $P(A,B)$ and $P(B)$ [or $P(A)$] remain unchanged

Factors Affecting Complexity

- ***Choice of minimum support threshold:*** Lowering support threshold results in more frequent itemsets. This may increase number of candidates and max length of frequent itemsets.
- ***Dimensionality (number of items) of the data set:*** More space is needed to store support count of each item. If number of frequent items increases, both computation and I/O costs may also increase.
- ***Size of database:*** Since Apriori makes multiple passes, run time of algorithm may increase with number of transactions.
- ***Average transaction width:*** Transaction width increases with denser data sets. This may increase max length of frequent itemsets and traversals of hash tree (number of subsets in a transaction increases with its width)

Frequent Pattern (FP) Growth Method

- Mining frequent itemsets without candidate generation.
- It is a divide and conquers strategy.
- It compresses the database representing frequent items into a frequent-pattern tree (FP-Tree), which retains the itemset association information.
- Divides the compressed database into a set of conditional databases, each associated with one frequent item or pattern fragment and then mines each such database separately.
- FP-Growth method transforms the problem of finding long frequent patterns to searching for shorter ones recursively and then concatenating the suffix.
- It uses least frequent items as suffix .

Advantage and Dis-advantage

- Advantage:
 - Reduce search cost, has good selectivity, faster than apriori.
- Disadvantes:
 - When the database is large, it is sometimes unrealistic to construct a man memory based FP-tree.

FP-Tree algorithm

- Create root node of tree, labeled with null.
- Scan the transactional database.
- The items in each transaction are processed in sorted order (Descending) and branch is created for each transaction.

FP-Tree algorithm

- Start from each frequent length pattern as an initial suffix pattern.
- Construct conditional pattern base. (Pattern base is a sub database which consists of the set of prefix paths in the FP-tree co-occurring with suffix pattern.
- Construct its FP-tree and perform mining recursively on such a tree

Categorical data

- Categorical data is a statistical data type consisting of categorical variables, used for observed data whose value is one of a fixed number of nominal categories.
- More specifically, categorical data may derive from either or both of observations made of qualitative data, where the observations are summarized as counts or cross tabulations, or of quantitative data.
- Observations might be directly observed counts of events happening or they might counts of values that occur within given intervals.
- Often, purely categorical data are summarized in the form of a contingency table.
- However, particularly when considering data analysis, it is common to use the term "categorical data" to apply to data sets that, while containing some categorical variables, may also contain non-categorical variables.

Potential Issues

- ***What if attribute has many possible values:*** Example: attribute country has more than 200 possible values. Many of the attribute values may have very low support.
 - Potential solution: Aggregate the low-support attributes values.
- ***What if distribution of attribute values is highly skewed:*** Example: 95% of the visitors have Buy = No. Most of the items will be associated with (Buy=No) item
 - Potential solution: drop the highly frequent items

Handling Categorical Attributes

- Transform categorical attribute into asymmetric binary variables. i.e If the outcomes of a binary variable are not equally important.
- Introduce a new “item” for each distinct attribute- value pair.

Sequential Pattern

- Mining of frequently occurring ordered events or subsequences as patterns. Eg: web sequence, book issued in library etc.
- Used mostly in marketing, customer analysis, prediction modeling.
- A sequence is an ordered list of events where an item can occur at most in an event of a sequence but can occur multiple times in different events of a sequence.
- Given a set of sequences, where each sequence consists of a list of events or elements and each event consists of set of items, given a minimum support threshold, sequential pattern mining finds all frequent subsequences.
- Sequence with minimum support is called frequent sequence or sequential pattern.
- A sequential pattern with length ‘l’ is called an l-pattern sequential pattern.

Sequential Pattern

- Sequential pattern is computationally challenging because such mining may generate combinationally explosive number of intermediate subsequences.
- For efficient and scalable sequential pattern mining two common approaches are:
 - Mining the full set of sequential patterns
 - Mining only the set of closed sequential pattern
- A sequence database is a set of tuples with sequence_ID and sequences. Eg

Sequence_ID	Sequence
1	{(a, (a,b,c), (a,c), (b,c)}
2	{(a,b,c), (a,d),e,(d,e)}
3	{(c,d), (a,d,e),e}
4	{ (e,f),d,(a,b,c),f]

Sub-graph Patterns

- It finds characteristics sub-graphs within the network.
- It is a form of graph search.
- Given a labeled graph data set, $D = \{G_1, G_2, \dots, G_n\}$, a frequent graph has minimum support not less than minimum threshold support.
- Frequent sub-graph pattern can be discovered by generating frequent substructures candidate and hence check the frequency of each candidate.
- Apriori methos and frequent –growth are tow common basic methods for finding frequent sub-graph
- Extend association rule mining to finding frequent subgraphs

What is frequent-pattern mining in Data Streams?

- Frequent-pattern mining finds a set of patterns that occur frequently in a data set, where a pattern can be a set of items (called an itemset), a subsequence, or a substructure.
- A pattern is considered frequent if its count satisfies a minimum support. Scalable methods for mining frequent patterns have been extensively studied for static data sets.
- Challenges in mining data streams:
 - Many existing frequent-pattern mining algorithms require the system to scan the whole data set more than once, but this is unrealistic for infinite data streams.
 - A frequent itemset can become infrequent as well. The number of infrequent itemsets is exponential and so it is impossible to keep track of all of them.