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

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 Association Rules

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

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. σ({Milk, Bread,Diaper}) = 2
- **Support**
  - Fraction of transactions that contain an itemset
  - E.g. s({Milk, Bread, Diaper}) = 2/5
- **Frequent Itemset**
  - An itemset whose support is greater than or equal to a *minsup* threshold

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

Definition: Association Rule

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

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:

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

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

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

Association Rule Mining Task

- Given a set of transactions T, the goal of association rule mining is to find all rules having
  - support ≥ *minsup* threshold
  - confidence ≥ *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:

{Milk,Diaper} → {Beer} (s=0.4, c=0.67)
{Milk,Beer} → {Diaper} (s=0.4, c=1.0)
{Diaper,Beer} → {Milk} (s=0.4, c=0.67)
{Beer} → {Milk,Diaper} (s=0.4, c=0.67)
{Diaper} → {Milk,Beer} (s=0.4, c=0.5)
{Milk} → {Diaper,Beer} (s=0.4, c=0.5)

Observations:

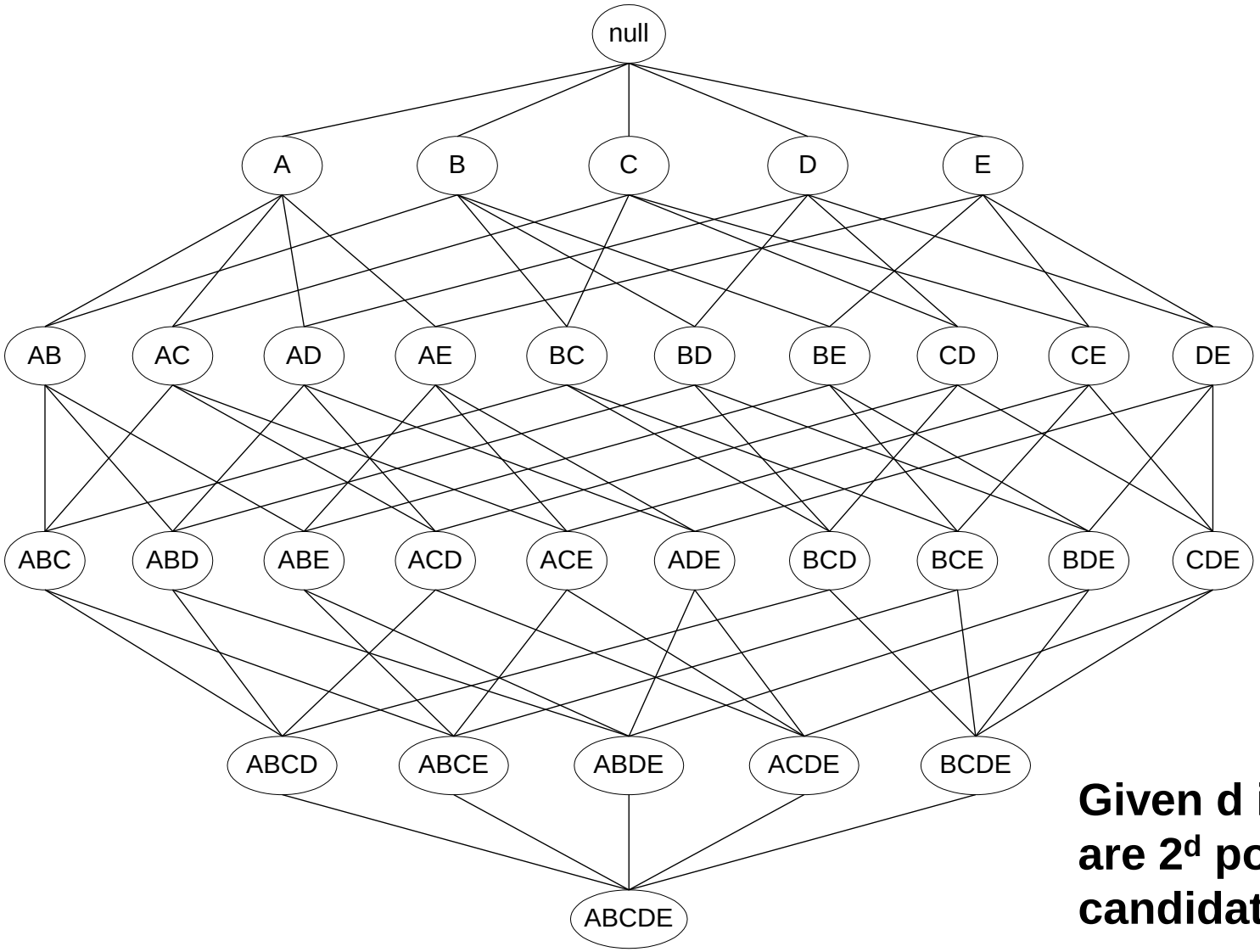
- All the above rules are binary partitions of the same itemset:
  {Milk, Diaper, Beer}
- Rules originating from the same itemset have identical support but can have different confidence
- Thus, we may decouple the support and confidence requirements

Mining Association Rules

- Two-step approach:
  1. Frequent Itemset Generation
     - Generate all itemsets whose support $\geq$ 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

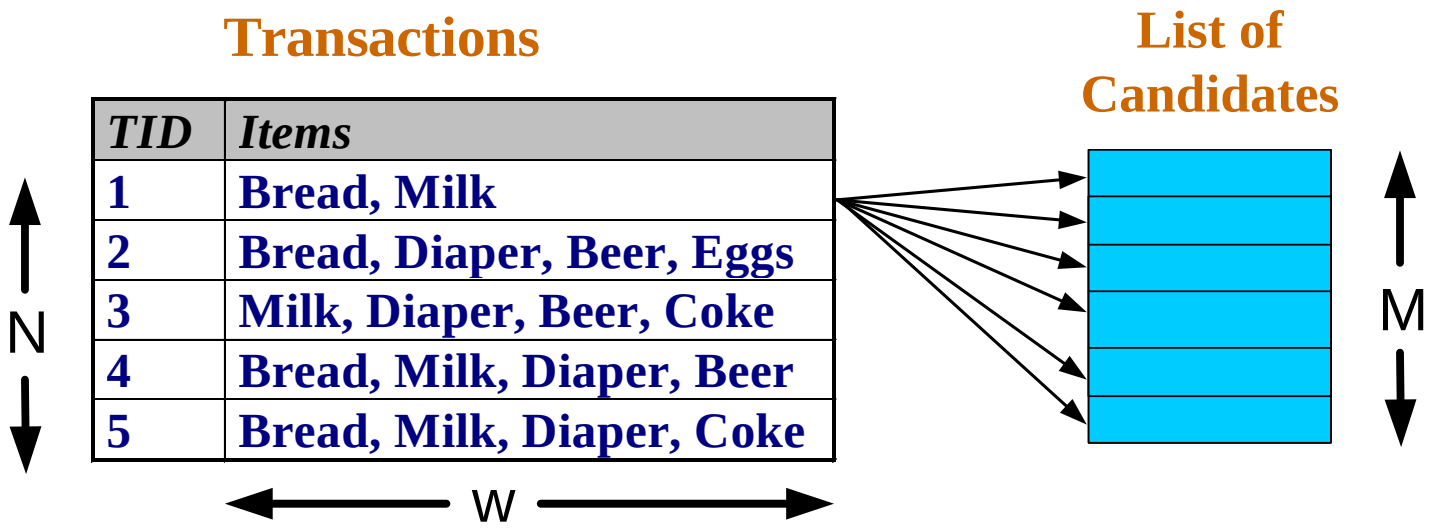
Frequent Itemset Generation



Given d items, there are 2^d possible candidate itemsets

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 ~ O(NMw) => Expensive since $M = 2^d$!!!

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:

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

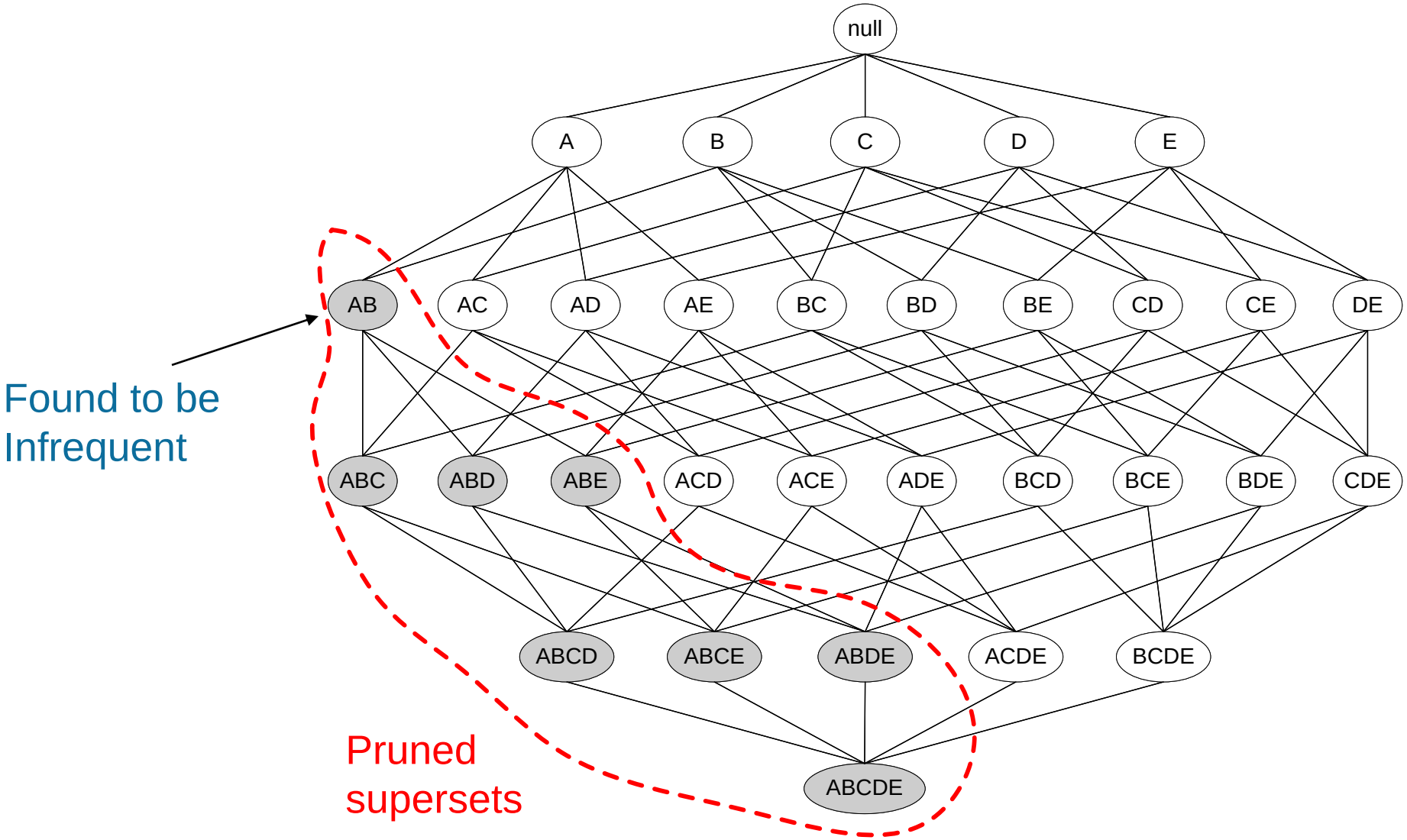
  - Support of an itemset never exceeds the support of its subsets
  - This is known as the anti-monotone property of support

Apriori: A Candidate Generation-and-test Approach

- Any subset of a frequent itemset must be frequent
  - if **{beer, diaper, nuts}** is frequent, so is **{beer, diaper}**
  - Every transaction having {beer, diaper, nuts} also contains {beer, diaper}

- Apriori pruning principle: If there is any itemset which is infrequent, its superset should not be generated/tested!

Illustrating Apriori Principle



Illustrating Apriori Principle

Items (1-itemsets)

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

Pairs (2-itemsets)

(No need to generate candidates involving Coke or Eggs)

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

Minimum Support = 3

Triplets (3-itemsets)

If every subset is considered,
$^6C_1 + {}^6C_2 + {}^6C_3 = 41$
With support-based pruning,
$6 + 6 + 1 = 13$

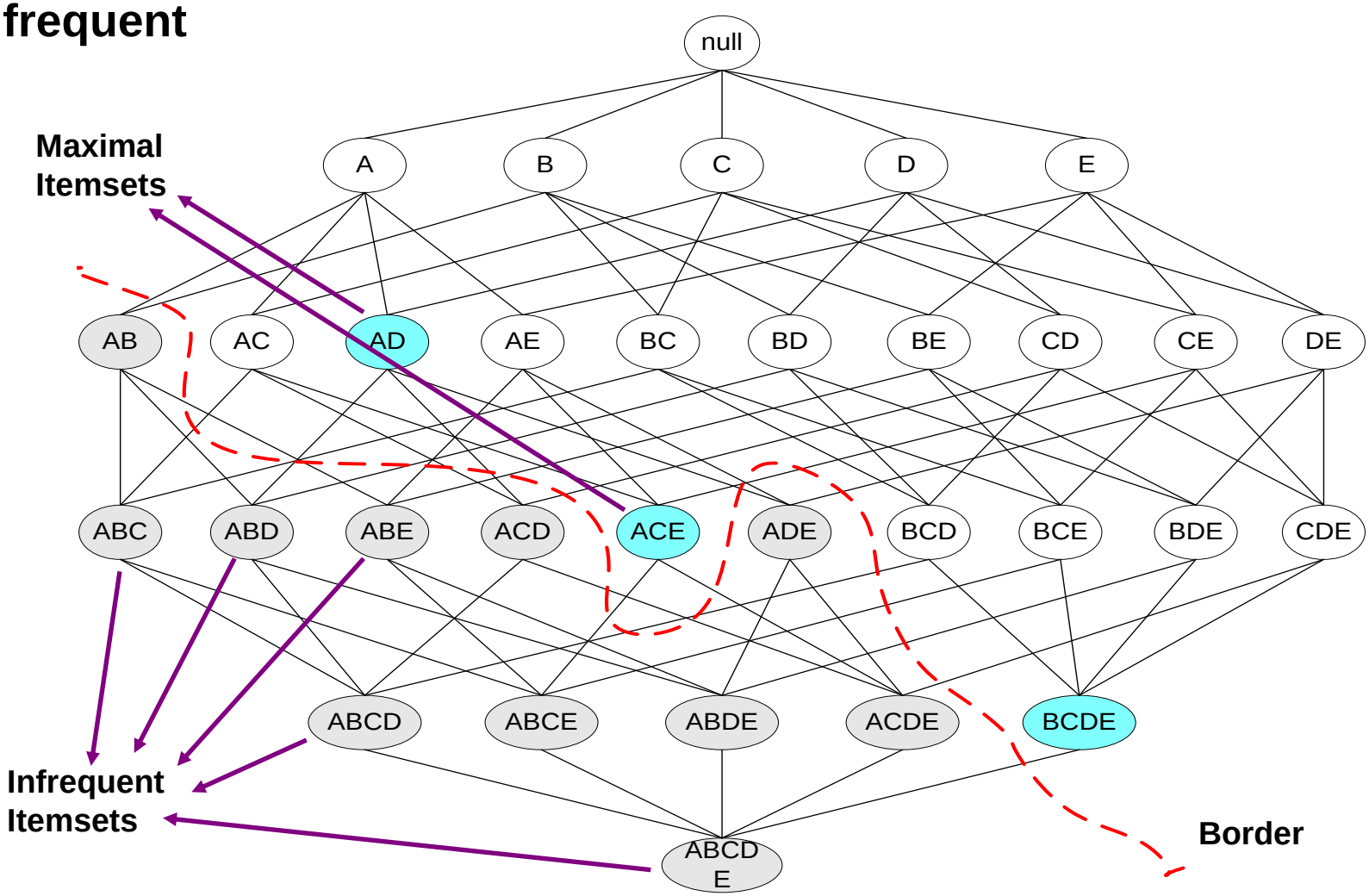
TID	*Items*
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

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

Maximal Frequent Itemset

An itemset is maximal frequent if none of its immediate supersets is frequent



Closed Itemset

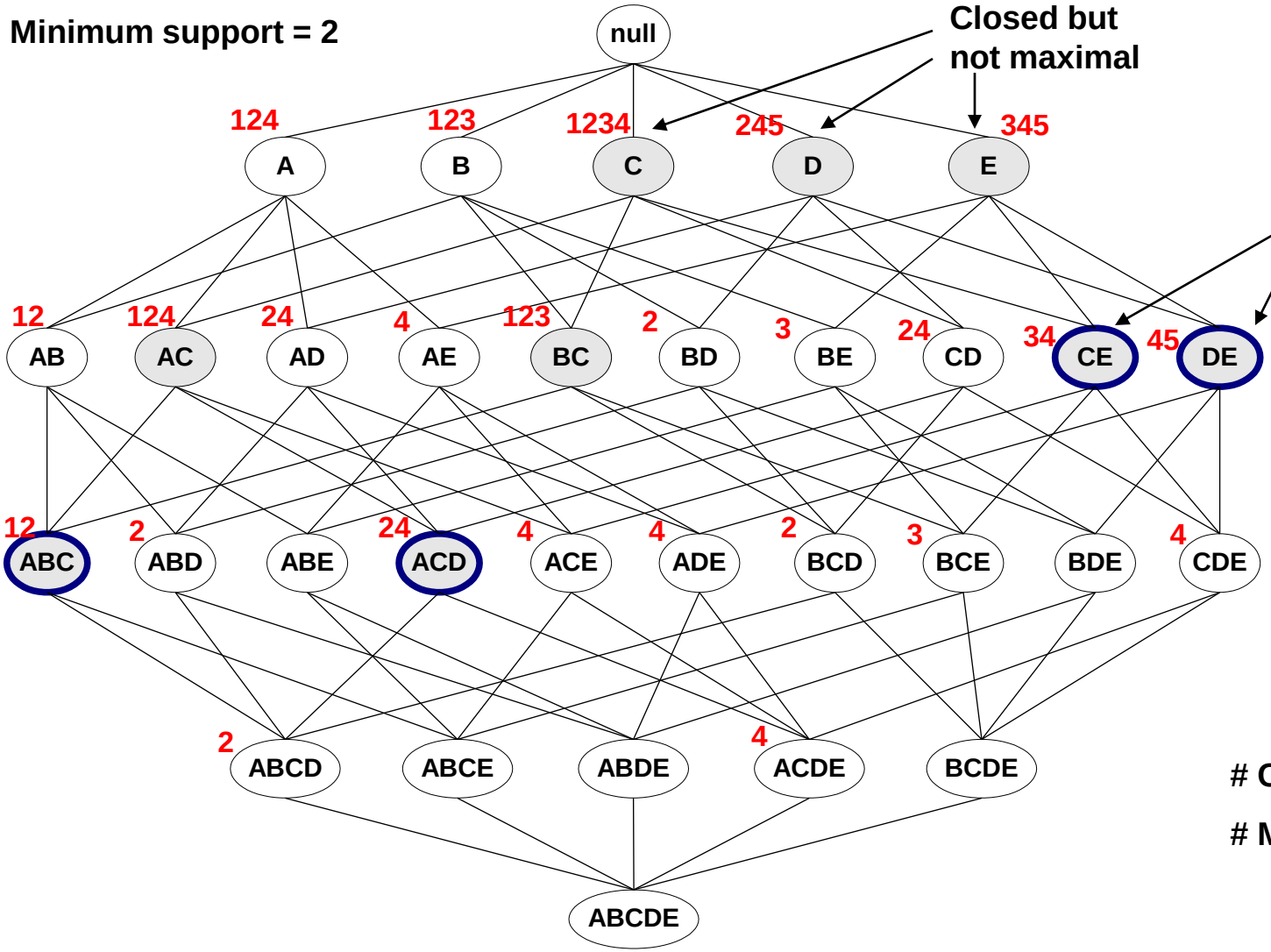
- An itemset is closed if none of its immediate supersets has the same support count as the itemset

TID	Items
1	{A,B}
2	{B,C,D}
3	{A,B,C,D}
4	{A,B,D}
5	{A,B,C,D}

Itemset	Support
{A}	4
{B}	5
{C}	3
{D}	4
{A,B}	4
{A,C}	2
{A,D}	3
{B,C}	3
{B,D}	4
{C,D}	3

Itemset	Support
{A,B,C}	2
{A,B,D}	3
{A,C,D}	2
{B,C,D}	3
{A,B,C,D}	2

Maximal vs Closed Frequent Itemsets

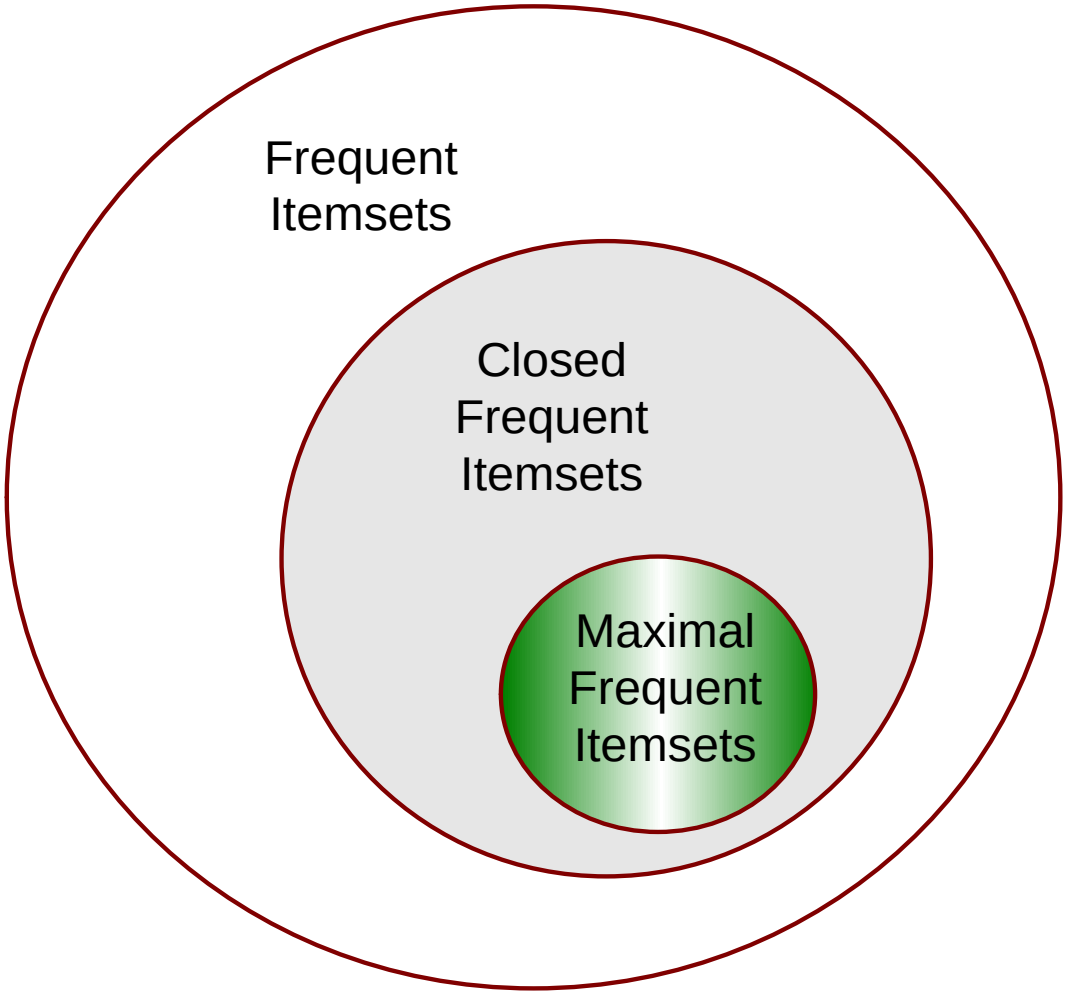


TID	Items
1	ABC
2	ABCD
3	BCE
4	ACDE
5	DE

 # Closed = 9

 # Maximal = 4

Maximal vs Closed Itemsets



Problems with A-priori Algorithms

- It is costly to handle a **huge number of candidate sets**.
- The candidate generation is the **inherent cost** of the Apriori Algorithms, no matter what implementation technique is applied.
- To mine a large data sets for long patterns – this algorithm is NOT a good idea.
- We need to mine Frequent Patterns **Without Candidate Generation**

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-Growth

Tid	Items
1	Beer, bread, butter, milk
2	Beer, milk, butter
3	Beer, milk, cheese
4	Beer, butter, diapers, cheese
5	Beer, cheese, bread

Transactions of Mario

Step 1: count items

Items	count
Beer	5
Bread	2
Butter	3
Milk	3
Cheese	3
diapers	1

FP-Growth

Step 2: Apply Threshold

Threshold = 30% so each item has to appear at least twice

Items	count
Beer	5
Bread	2
Butter	3
Milk	3
Cheese	3
diapers	1

Step 3: sort items in descending order

Items	count
Beer	5
Butter	3
Milk	3
Cheese	3
Bread	2

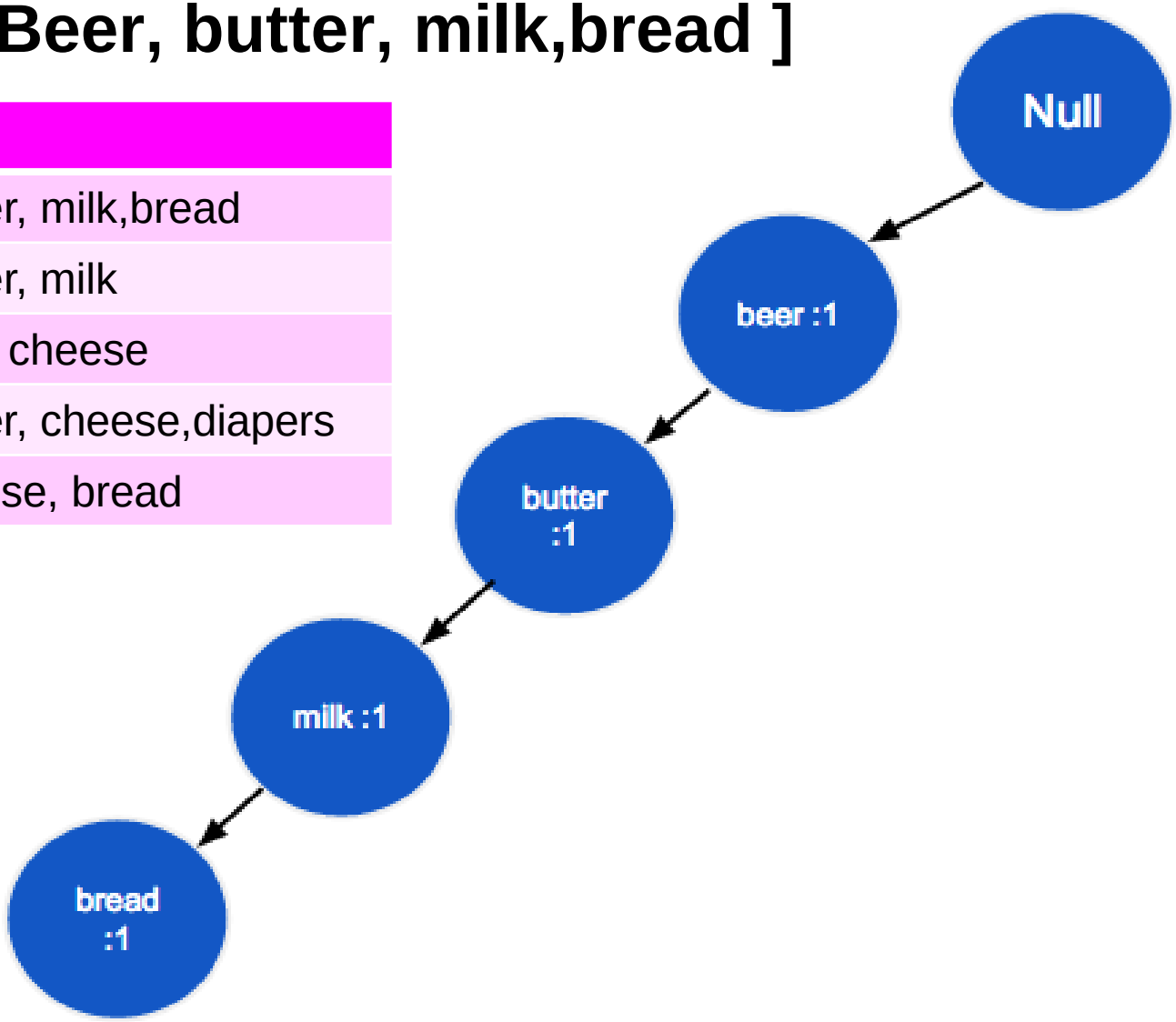
Step 4: Rearrange Transaction set as per sorted items

Tid	Items
1	Beer, butter, milk,bread
2	Beer, butter, milk
3	Beer, milk, cheese
4	Beer, butter, cheese,diapers
5	Beer, cheese, bread

FP-Growth

- Step 5.1: start building FP-tree
- **Tid =1[Beer, butter, milk,bread]**

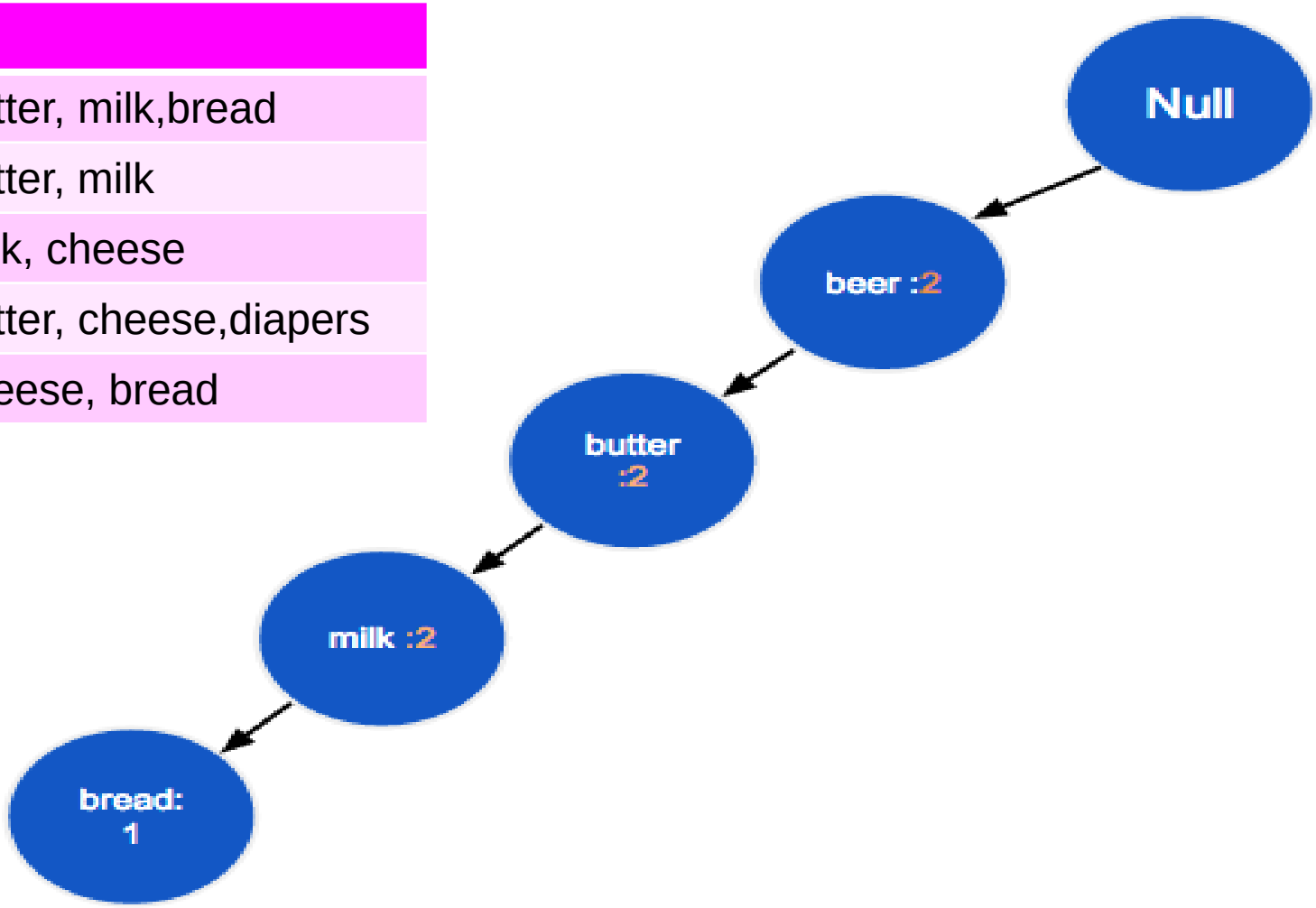
Tid	Items
1	Beer, butter, milk,bread
2	Beer, butter, milk
3	Beer, milk, cheese
4	Beer, butter, cheese,diapers
5	Beer, cheese, bread



FP-Growth

- Step 5.2: continue building FP-tree
- Tid = **2[Beer, butter, milk]**

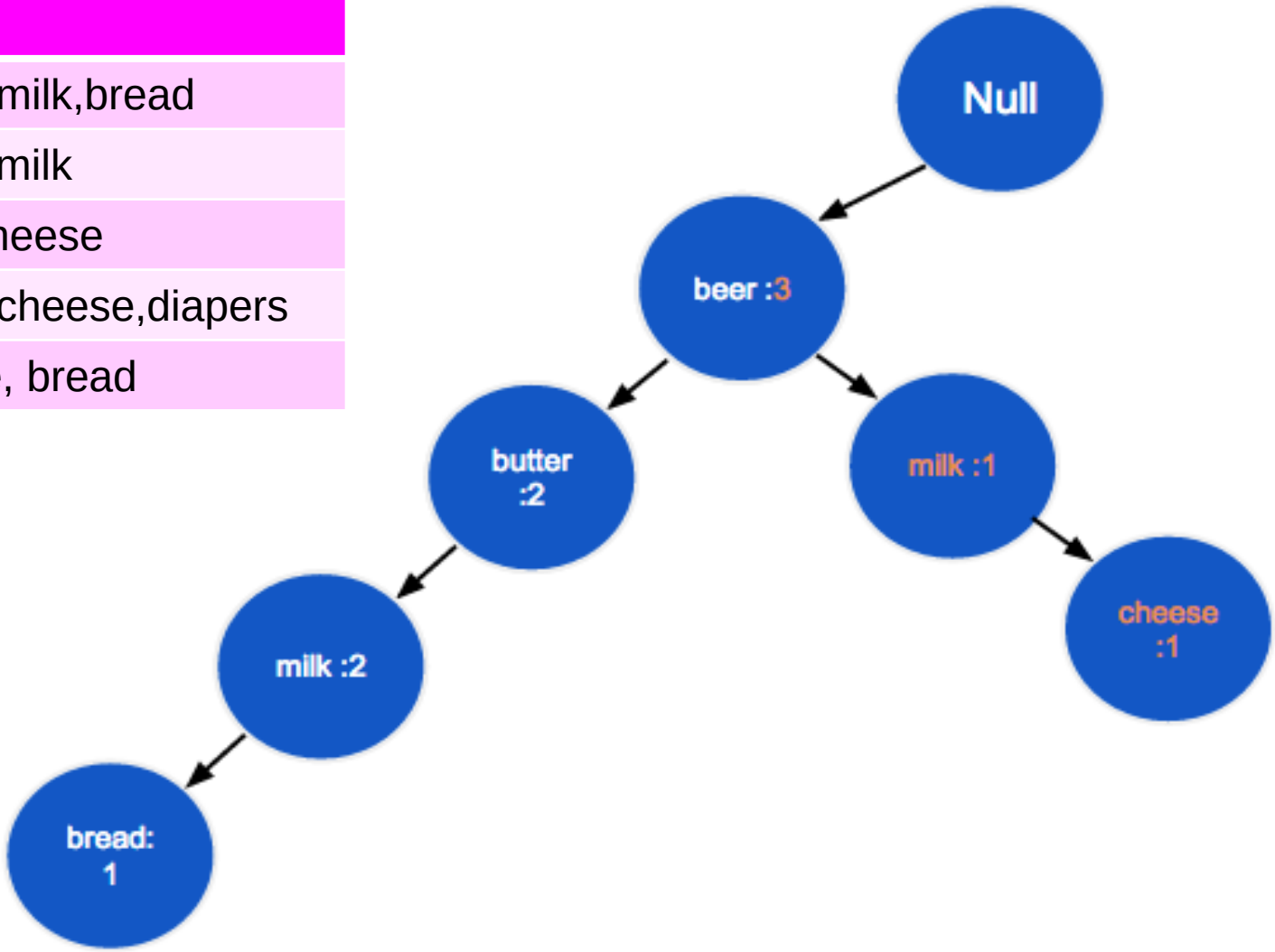
Tid	Items
1	Beer, butter, milk,bread
2	Beer, butter, milk
3	Beer, milk, cheese
4	Beer, butter, cheese,diapers
5	Beer, cheese, bread



FP-Growth

- Step 5.3: continue building FP-tree
- Tid = **3**[Beer, milk, cheese

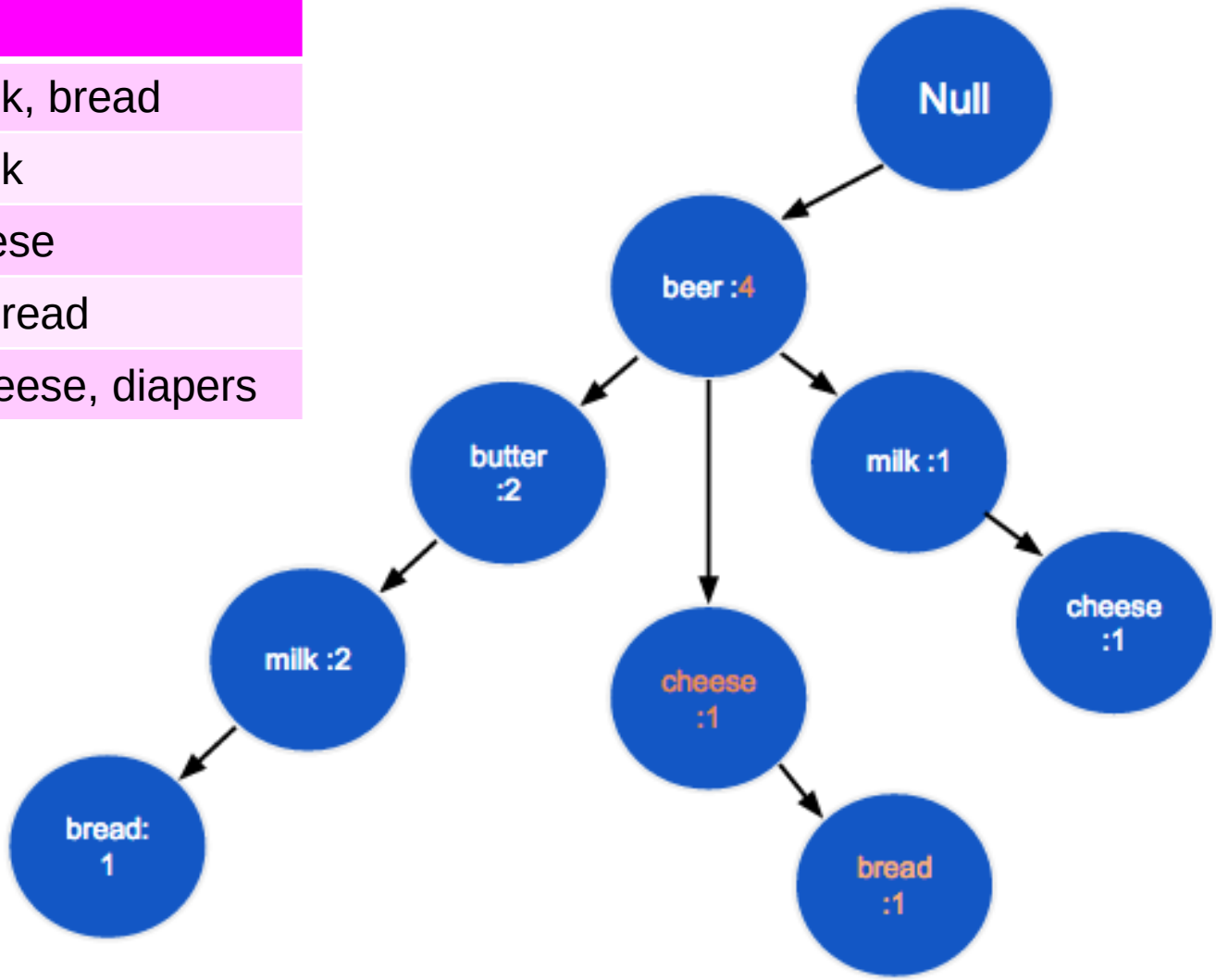
Tid	Items
1	Beer, butter, milk,bread
2	Beer, butter, milk
3	Beer, milk, cheese
4	Beer, butter, cheese,diapers
5	Beer, cheese, bread



FP-Growth

- Step 5.4: continue building FP-tree
- Tid = 4 [Beer, cheese, bread]

Tid	Items
1	Beer, butter, milk, bread
2	Beer, butter, milk
3	Beer, milk, cheese
4	Beer, cheese, bread
5	Beer, butter, cheese, diapers



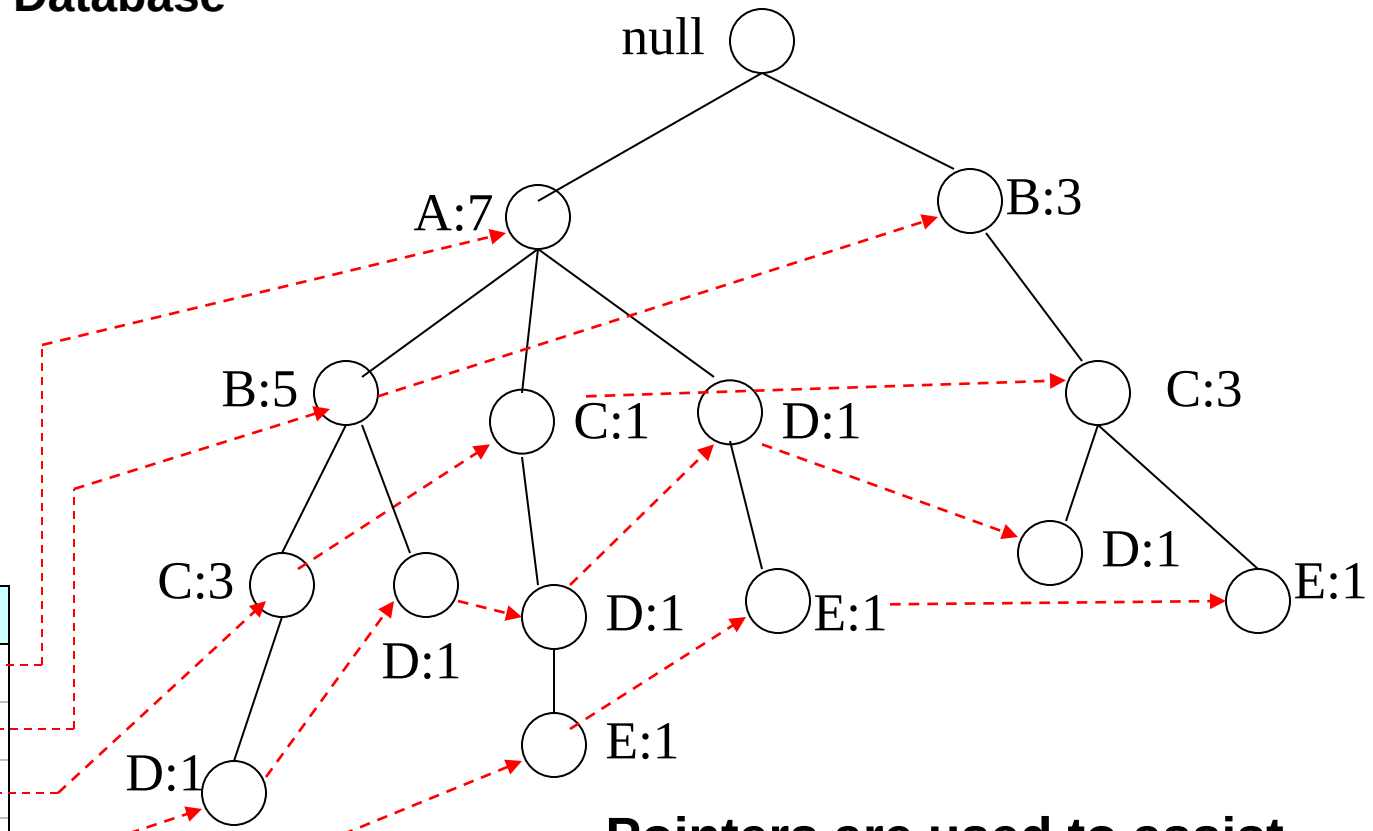
FP-Tree Construction

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

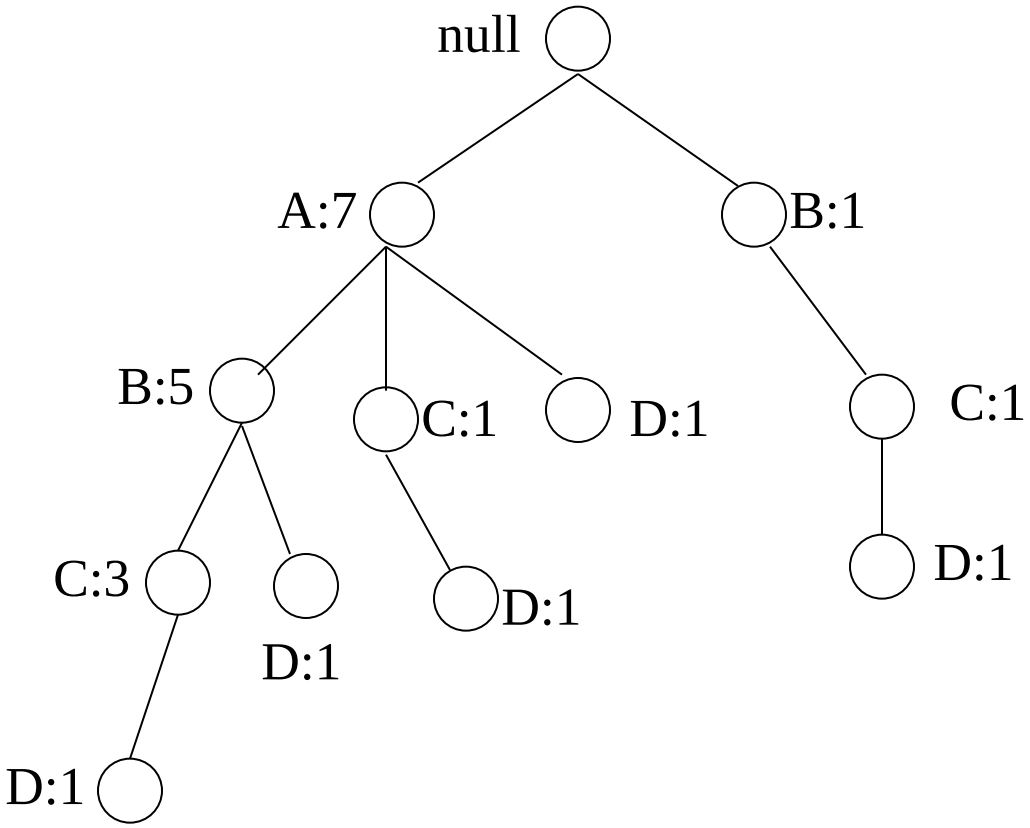
Header table

Item	Pointer
A	
B	
C	
D	
E	



Pointers are used to assist frequent itemset generation

FP-growth



Conditional Pattern base for D:

P = {(A:1,B:1,C:1),
(A:1,B:1),
(A:1,C:1),
(A:1),
(B:1,C:1)}

FP-tree for D with sup>1:

A, B, C, AB, AC,BC

Frequent Itemsets found (with sup > 1):
AD, BD, CD, ABD, ACD, BCD

Solve complete tree for E, C, B and A to get more frequent itemsets

Apriori vs FP-Growth

Algorithm	Technique	Runtime	Memory usage	Parallelizability
Apriori	Generate singletons, pairs, triplets, etc.	Candidate generation is extremely slow. Runtime increases exponentially depending on the number of different items.	Saves singletons, pairs, triplets, etc.	Candidate generation is very parallelizable
FP-Growth	Insert sorted items by frequency into a pattern tree	Runtime increases linearly, depending on the number of transactions and items	Stores a compact version of the database.	Data are very inter dependent, each node needs the root.

FP-Growth beats Apriori by far. It has **less memory usage** and **less runtime**. The **differences are huge**. **FP-Growth is more scalable because of its linear running time**.

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 $L \rightarrow \varnothing$ and $\varnothing \rightarrow L$)

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 $c(AB \rightarrow D)$

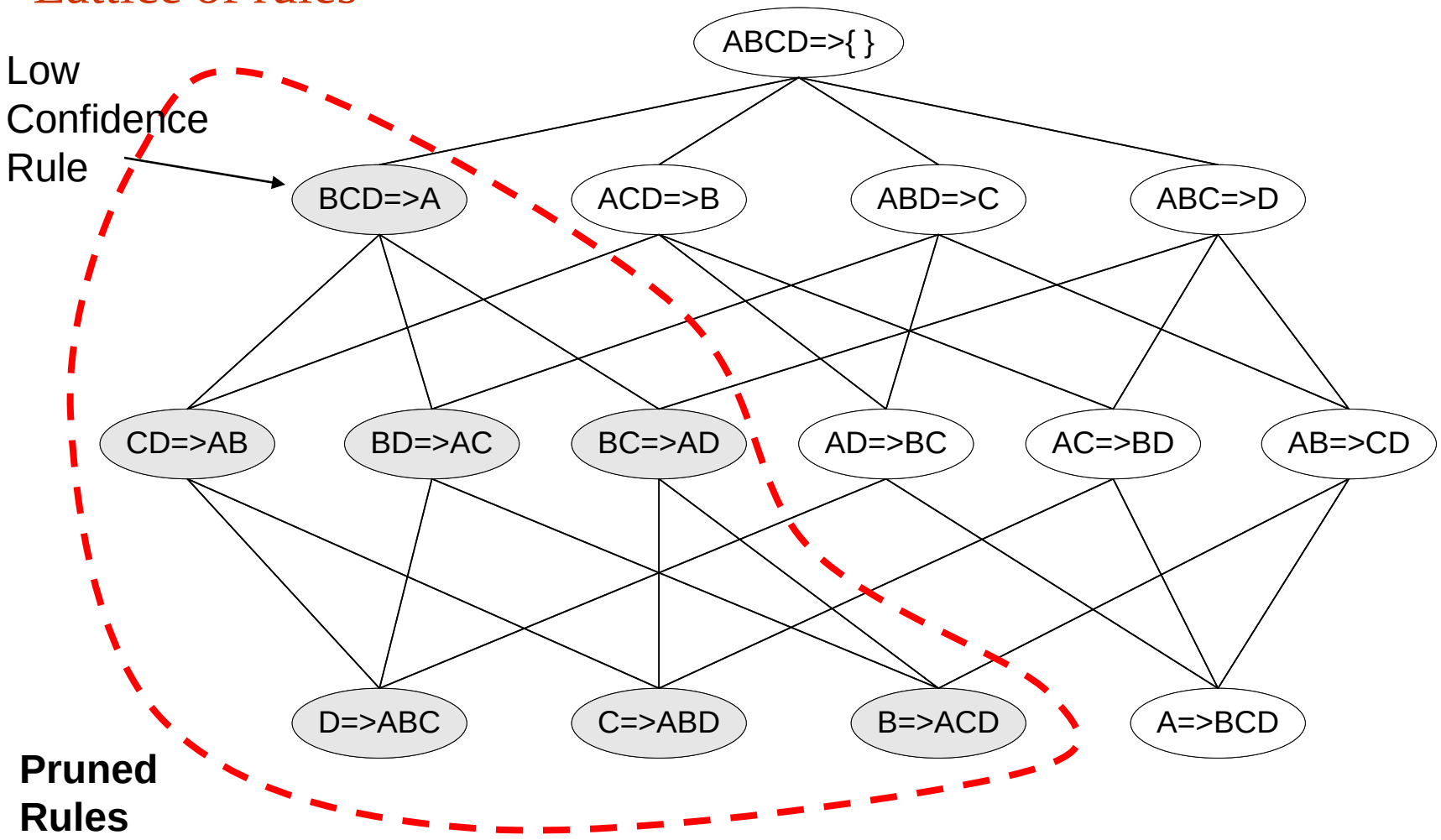
  - But confidence of rules generated from the same itemset has an anti-monotone property
  - e.g., L = {A,B,C,D}:

    $$c(ABC \rightarrow D) \geq c(AB \rightarrow CD) \geq c(A \rightarrow BCD)$$

    - Confidence is anti-monotone w.r.t. number of items on the RHS of the rule

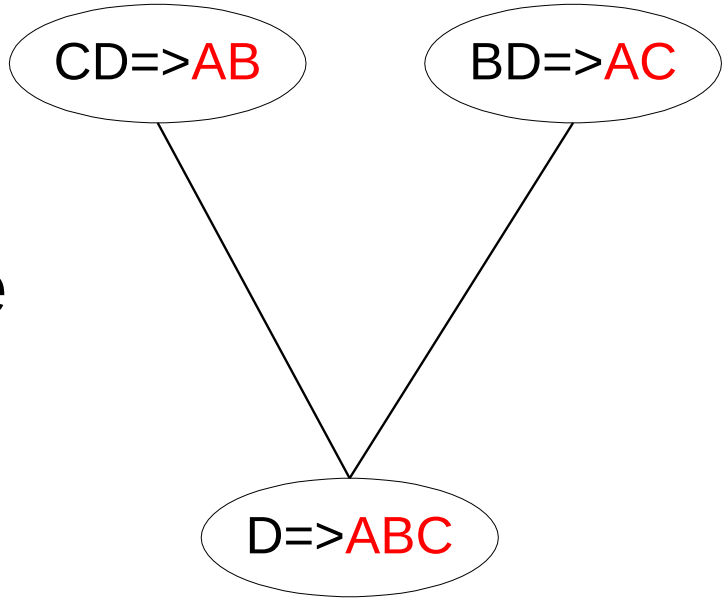
Rule Generation for Apriori Algorithm

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



- join(CD=>AB,BD=>AC) would produce the candidate rule D => ABC

- Prune rule D=>ABC if its subset AD=>BC does not have high confidence

Pattern Evaluation

- Association rule algorithms tend to produce **too many rules**
  - many of them are **uninteresting or redundant**
  - Redundant if **{A,B,C} → {D} and {A,B} → {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

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	$\overline{Y}$	
X	f_{11}	f_{10}	f_{1+}
$\overline{X}$	f_{01}	f_{00}	f_{o+}
	f_{+1}	f_{+0}	|T|

f_{11}: support of X and Y
f_{10}: support of $\overline{X}$ and $\overline{Y}$
f_{01}: support of $\overline{X}$ and Y
f_{00}: support of X and $\overline{Y}$

Used to define various measures

- support, confidence, lift

Drawback of Confidence

	Coffee	$\overline{\text{Coffee}}$	
Tea	15	5	20
$\overline{\text{Tea}}$	75	5	80
	90	10	100

Association Rule: Tea $\rightarrow$ Coffee

Confidence= P(Coffee|Tea) = P(coffee, tea)/P(tea) = 15/20= 0.75

P(Coffee) = 90/100 =0.9

$\Rightarrow$ Although confidence is high, rule is misleading (if a person is tea drinker is also a coffee drinker with 75% confidence But it is actually less than probability of coffee drinker 90%)

$\Rightarrow$P(Coffee|Tea) = P(coffee, tea)/P(tea) = 75/80 = 0.9375

Statistical Independence & Correlation

- 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)
  - $P(S \wedge B) = 420/1000 = 0.42$
  - $P(S) \times P(B) = 0.6 \times 0.7 = 0.42$

  - $P(S \wedge B) = P(S) \times P(B)$ => Statistical independence
    i.e. no correlation
  - $P(S \wedge B) > P(S) \times P(B)$ => Positively correlated
    i.e. occurrence of S increases occurrence of B
  - $P(S \wedge B) < P(S) \times P(B)$ => Negatively correlated
    i.e. occurrence of S decreases occurrence of B

Statistical-based Measures

- Measures that take into account statistical dependence
- Lift = P(Y|X) / P(Y) (= Interest for Binary)
- Lift handles misleading nature of High confidence by considering support of the consequent

Example: Lift/Interest

	Coffee	$\overline{\text{Coffee}}$	
Tea	15	5	20
$\overline{\text{Tea}}$	75	5	80
	90	10	100

Association Rule: Tea $\rightarrow$ Coffee

Confidence= P(Coffee|Tea) = 0.75

but P(Coffee) = 0.9

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

Infrequent Patterns

- An infrequent pattern is an itemset or a rule whose support is less than the minsup threshold
- majority of infrequent patterns are uninteresting, but some of them might be useful to the analysts, particularly those that correspond to negative correlations in the data.
- For example, the sale of DVDs and VCRs together is low because any customer who buys a DVD will most likely not buy a VCR, and vice versa.
- Such negative-correlated patterns are useful to help identify **competing items**, which are items that can be substituted for one another.
- Examples of competing items include tea versus coffee, regular versus diet soda, and desktop versus laptop computers

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
- To extract interesting categorical rules from existing association algorithms, wee need to transform categorical attribute into binary variables
- Example: Gender → male = 0, Female = 1
- Introduce a new “item” for each distinct attribute-value pair
  - Example: replace Browser Type attribute with
    - Browser Type = Internet Explorer
    - Browser Type = Mozilla

Categorical to Binary

- Categorical Data

Gender	Level of Education	State	Computer at Home
Female	Graduate	Illinois	Yes
Male	College	California	No
Male	Graduate	Michigan	Yes
Female	College	Virginia	No
Female	Graduate	California	Yes
Male	College	Minnesota	Yes
Male	College	Alaska	Yes
Male	High School	Oregon	Yes
Female	Graduate	Texas	No
...	...	...	...

Binarized Data

Male	Female
0	1
1	0
1	0
0	1
0	1
1	0
1	0
1	0
0	1

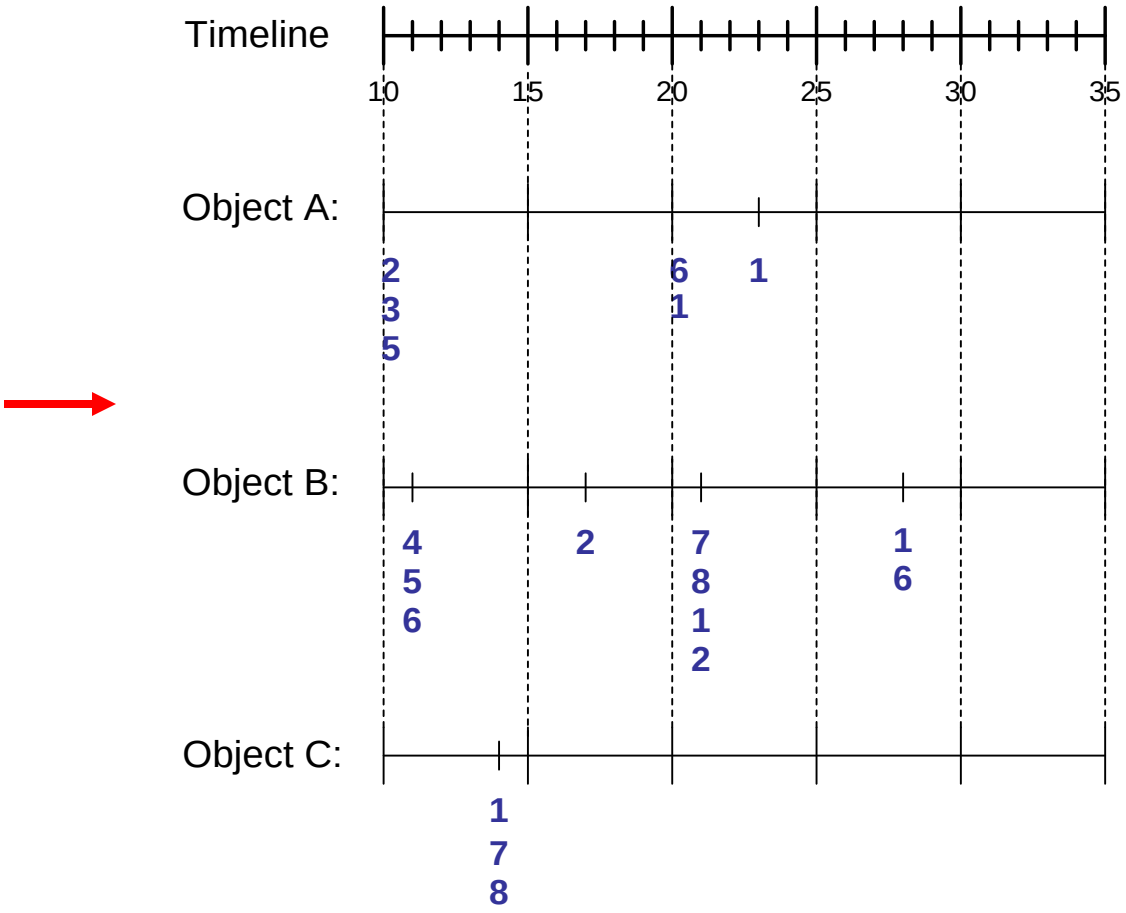
Issues while Handling Categorical Attributes

- 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
    - Reducing threshold?? Computationally expensive
      - Potential solution: Aggregate the low-support attribute 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
  - If newly created item become frequent, that increases computational time for generating candidate item sets
      - Potential solution: Avoid generating candidate item sets ¨such as {state = x, state = y,…} having support value 0

Sequence Data

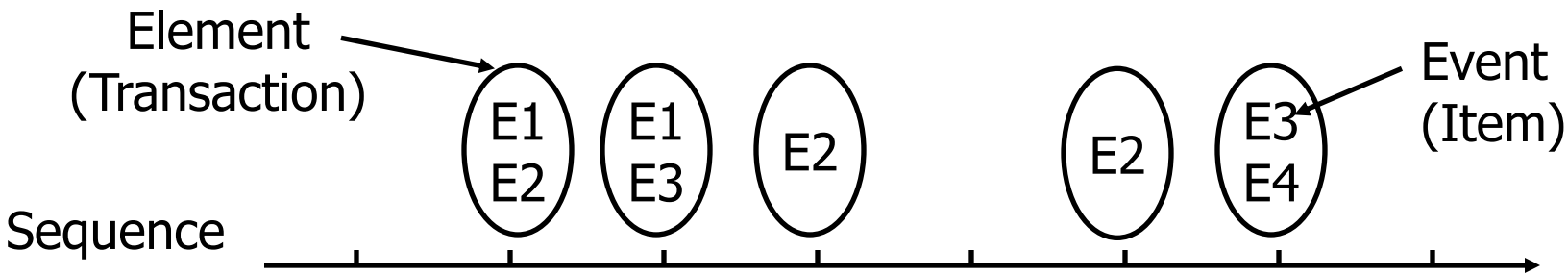
Sequence Database:

Object	Timestamp	Events
A	10	2, 3, 5
A	20	6, 1
A	23	1
B	11	4, 5, 6
B	17	2
B	21	7, 8, 1, 2
B	28	1, 6
C	14	1, 8, 7



Examples of Sequence Data

Sequence Database	Sequence	Element (Transaction)	Event (Item)
Customer	Purchase history of a given customer	A set of items bought by a customer at time t	Books, diary products, CDs, etc
Web Data	Browsing activity of a particular Web visitor	A collection of files viewed by a Web visitor after a single mouse click	Home page, index page, contact info, etc
Event data	History of events generated by a given sensor	Events triggered by a sensor at time t	Types of alarms generated by sensors
Genome sequences	DNA sequence of a particular species	An element of the DNA sequence	Bases A,T,G,C



Formal Definition of a Sequence

- A sequence is an ordered list of elements (transactions)

$$s = < e_1\, e_2\, e_3 \ldots >$$

  - Each element contains a collection of events (items)

$$e_i = \{i_1, i_2, \ldots, i_k\}$$

  - Each element is attributed to a specific time or location

- Length of a sequence, $|s|$, is given by the number of elements of the sequence

- A k-sequence is a sequence that contains k events (items)

Formal Definition of a Subsequence

- A sequence $\langle a_1\ a_2 \ldots a_n \rangle$ is contained in another sequence $\langle b_1\ b_2 \ldots b_m \rangle$ ($m \geq n$) if there exist integers $i_1 < i_2 < \ldots < i_n$ such that $a_1 \subseteq b_{i1}$, $a_2 \subseteq b_{i1}$, …, $a_n \subseteq b_{in}$

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

- The support of a subsequence w is defined as the fraction of data sequences that contain w
- A *sequential pattern* is a frequent subsequence (i.e., a subsequence whose support is ≥ *minsup*)

Sequential Pattern Mining: Definition

- Given:
  - a database of sequences
  - a user-specified minimum support threshold, *minsup*
- Task:
  - Find all subsequences with support ≥ *minsup*

Sequential Pattern Mining: Example

Object	Timestamp	Events
A	1	1,2,4
A	2	2,3
A	3	5
B	1	1,2
B	2	2,3,4
C	1	1, 2
C	2	2,3,4
C	3	2,4,5
D	1	2
D	2	3, 4
D	3	4, 5
E	1	1, 3
E	2	2, 4, 5

Minsup = 50%

Examples of Frequent Subsequences:

< {1,2} >	s=60%
< {2,3} >	s=60%
< {2,4}>	s=80%
< {3} {5}>	s=80%
< {1} {2} >	s=80%
< {2} {2} >	s=60%
< {1} {2,3} >	s=60%
< {2} {2,3} >	s=60%
< {1,2} {2,3} >	s=60%

Extracting Sequential Patterns

- Given n events: $i_1, i_2, i_3, \ldots, i_n$
- Candidate 1-subsequences:

  $\langle\{i_1\}\rangle, \langle\{i_2\}\rangle, \langle\{i_3\}\rangle, \ldots, \langle\{i_n\}\rangle$
- Candidate 2-subsequences:

  $\langle\{i_1, i_2\}\rangle, \langle\{i_1, i_3\}\rangle, \ldots, \langle\{i_1\}\ \{i_1\}\rangle, \langle\{i_1\}\ \{i_2\}\rangle, \ldots, \langle\{i_{n-1}\}\ \{i_n\}\rangle$
- Candidate 3-subsequences:

  $\langle\{i_1, i_2, i_3\}\rangle, \langle\{i_1, i_2, i_4\}\rangle, \ldots, \langle\{i_1, i_2\}\ \{i_1\}\rangle, \langle\{i_1, i_2\}\ \{i_2\}\rangle, \ldots,$

  $\langle\{i_1\}\ \{i_1, i_2\}\rangle, \langle\{i_1\}\ \{i_1, i_3\}\rangle, \ldots, \langle\{i_1\}\ \{i_1\}\ \{i_1\}\rangle, \langle\{i_1\}\ \{i_1\}\ \{i_2\}\rangle, \ldots$

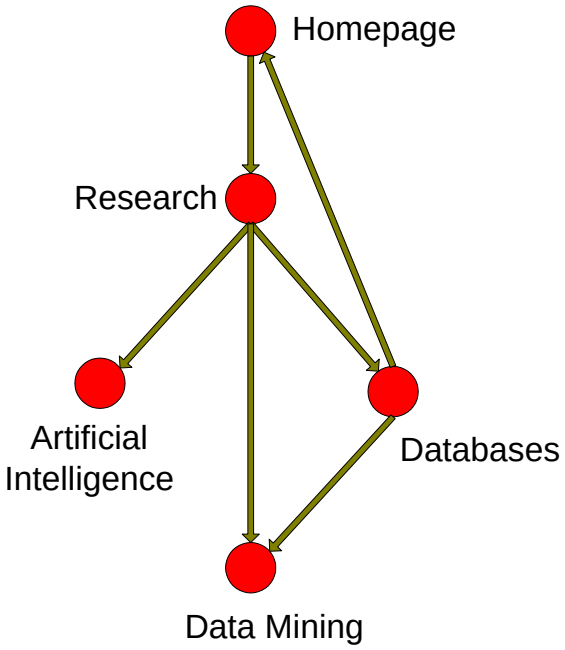
Generalized Sequential Pattern (GSP)

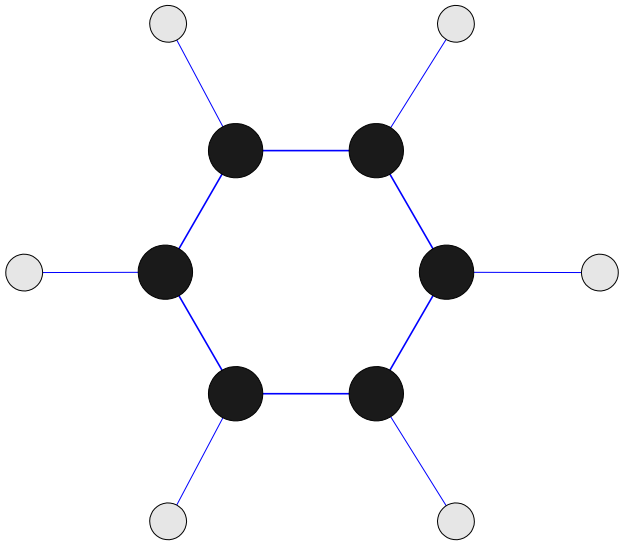
- **Step 1**:
  - Make the first pass over the sequence database D to yield all the 1-element frequent sequences
- **Step 2**:

  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
  - **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*

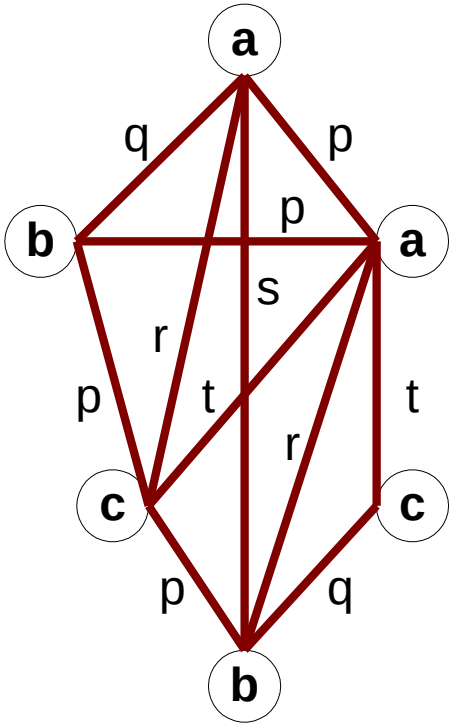
Frequent Subgraph Mining

- Extend association rule mining to finding frequent subgraphs
- Useful for Web Mining, computational chemistry, bioinformatics, spatial data sets, etc

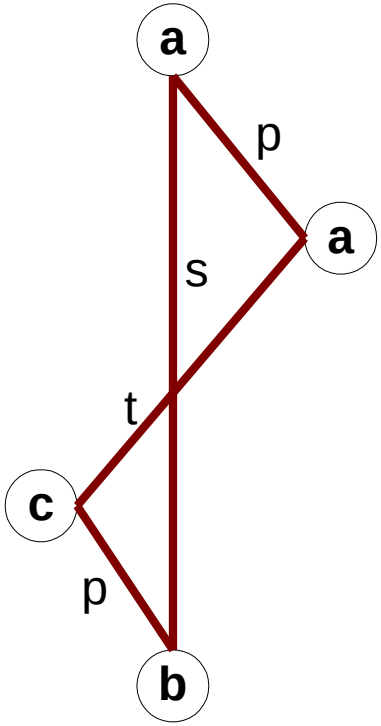




Graph Definitions



(a) Labeled Graph

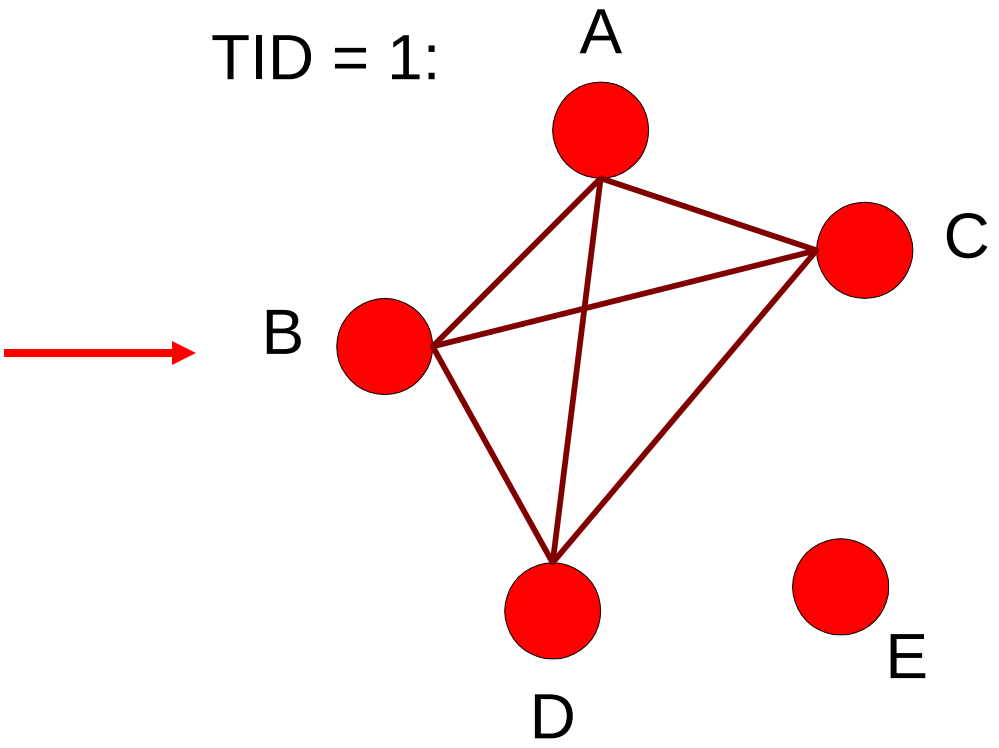


(b) Subgraph

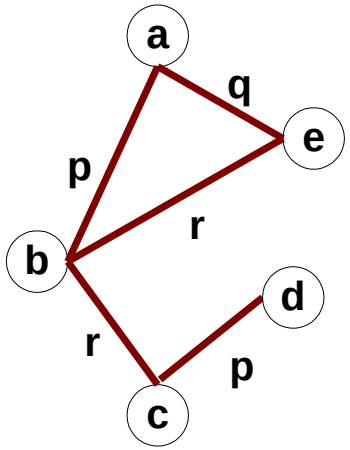
Representing Transactions as Graphs

- Each transaction is a clique of items

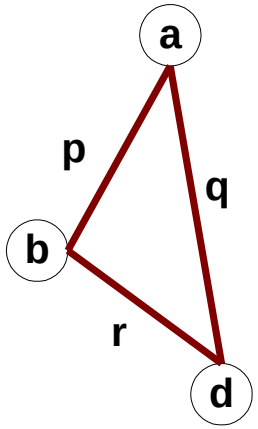
Transaction Id	Items
1	{A,B,C,D}
2	{A,B,E}
3	{B,C}
4	{A,B,D,E}
5	{B,C,D}



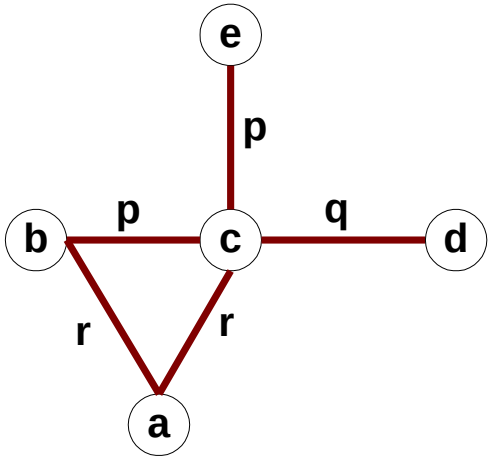
Representing Graphs as Transactions



G1



G2



G3

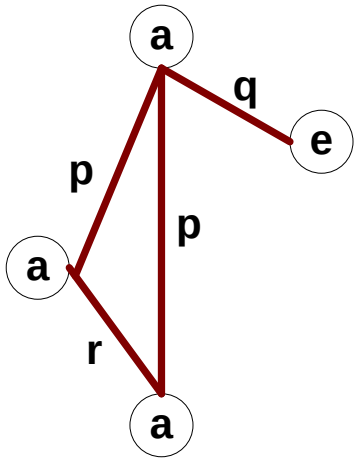
	(a,b,p)	(a,b,q)	(a,b,r)	(b,c,p)	(b,c,q)	(b,c,r)	…	(d,e,r)
G1	1	0	0	0	0	1	…	0
G2	1	0	0	0	0	0	…	0
G3	0	0	1	1	0	0	…	0
G3	…	…	…	…	…	…	…	…

Challenges

- Node may contain duplicate labels
- Support and confidence
  - How to define them?
- Additional constraints imposed by pattern structure
  - Support and confidence are not the only constraints
  - Assumption: frequent subgraphs must be connected
- Apriori-like approach:
  - Use frequent k-subgraphs to generate frequent (k+1) subgraphs
    - What is k?

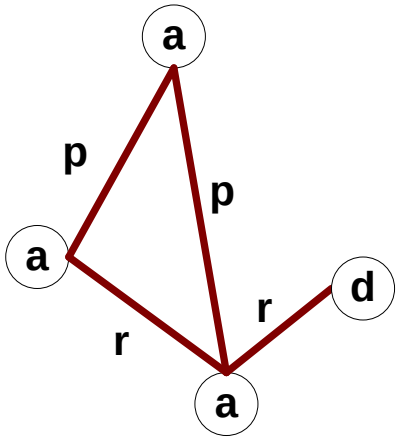
- Support:
  - number of graphs that contain a particular subgraph
- Apriori principle still holds
- Level-wise (Apriori-like) approach:
  - Vertex growing:
    - k is the number of vertices
  - Edge growing:
    - k is the number of edges

Vertex Growing

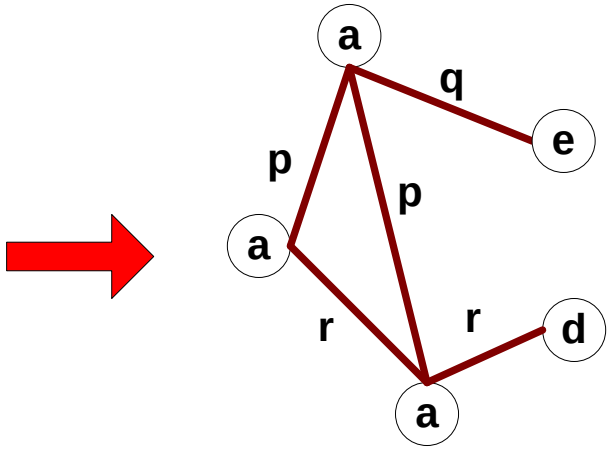


G1

+



G2



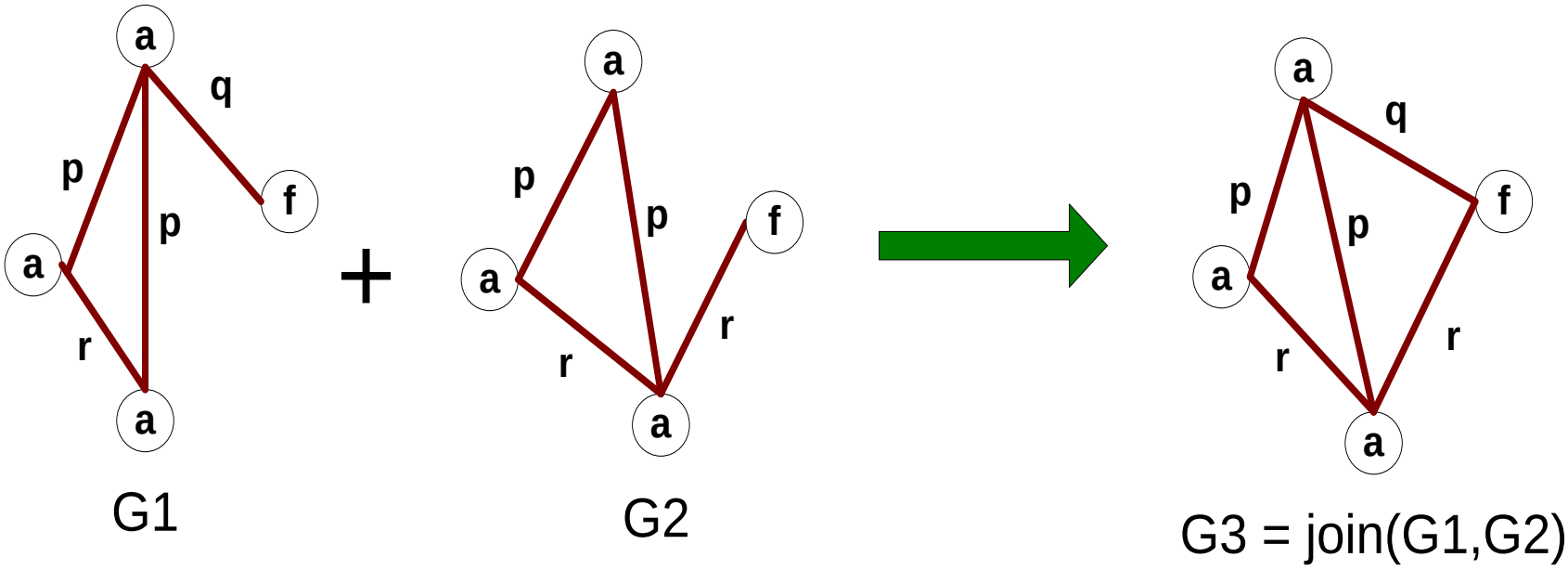
G3 = join(G1,G2)

$$M_{G1} = \begin{pmatrix} 0 & p & p & q \\ p & 0 & r & 0 \\ p & r & 0 & 0 \\ q & 0 & 0 & 0 \end{pmatrix}$$

$$M_{G2} = \begin{pmatrix} 0 & p & p & 0 \\ p & 0 & r & 0 \\ p & r & 0 & r \\ 0 & 0 & r & 0 \end{pmatrix}$$

$$M_{G3} = \begin{pmatrix} 0 & p & p & 0 & q \\ p & 0 & r & 0 & 0 \\ p & r & 0 & r & 0 \\ 0 & 0 & r & 0 & 0 \\ q & 0 & 0 & 0 & 0 \end{pmatrix}$$

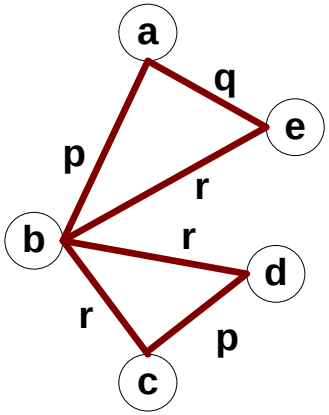
Edge Growing



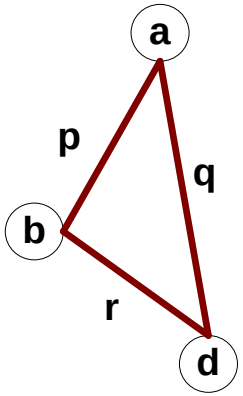
Apriori-like Algorithm

- Find frequent 1-subgraphs
- Repeat
  - Candidate generation
    - Use frequent (*k-1*)-subgraphs to generate candidate *k*-subgraph
  - Candidate pruning
    - Prune candidate subgraphs that contain infrequent (*k-1*)-subgraphs
  - Support counting
    - Count the support of each remaining candidate
  - Eliminate candidate *k*-subgraphs that are infrequent

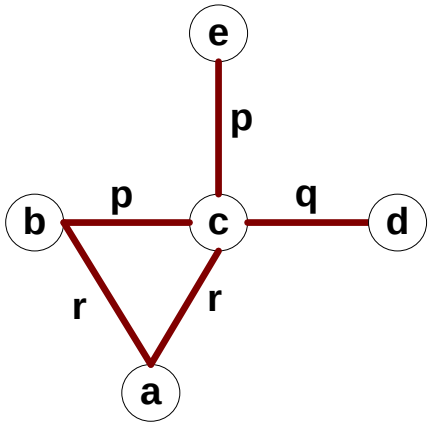
Example: Dataset



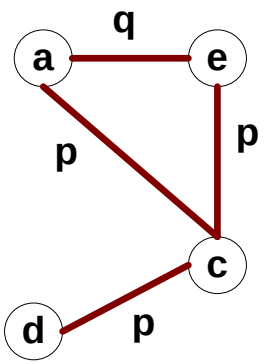
G1



G2



G3



G4

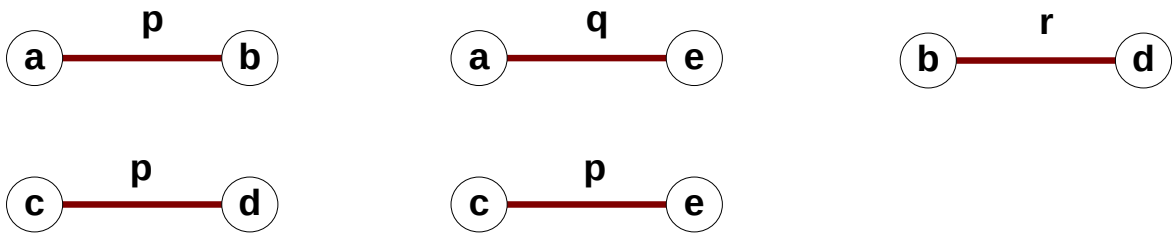
	(a,b,p)	(a,b,q)	(a,b,r)	(b,c,p)	(b,c,q)	(b,c,r)	…	(d,e,r)
G1	1	0	0	0	0	1	…	0
G2	1	0	0	0	0	0	…	0
G3	0	0	1	1	0	0	…	0
G4	0	0	0	0	0	0	…	0

Example

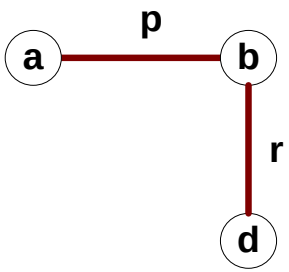
Minimum support count = 2

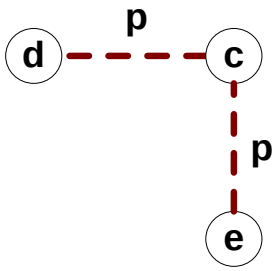
k=1
Frequent
Subgraphs

k=2
Frequent
Subgraphs



k=3
Candidate
Subgraphs





(Pruned candidate)