DATA MINING LECTURE 4

Frequent Itemsets and Association Rules

This is how it all started…

- Rakesh Agrawal, Tomasz Imielinski, Arun N. Swami: Mining Association Rules between Sets of Items in Large Databases. SIGMOD Conference 1993: 207-216
- Rakesh Agrawal, Ramakrishnan Srikant: Fast Algorithms for Mining Association Rules in Large Databases. VLDB 1994: 487-499

- These two papers are credited with the birth of Data Mining
- For a long time people were fascinated with Association Rules and Frequent Itemsets
  - Some people (in industry and academia) still are.

Market-Basket Data

- A large set of items, e.g., things sold in a supermarket.
- A large set of baskets, each of which is a small subset of the items, e.g., the things one customer buys on one day.

Items: {Bread, Milk, Diaper, Beer, Eggs, Coke}

Baskets: 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

Frequent itemsets

- Goal: find combinations of items (itemsets) that occur frequently
  - Called Frequent Itemsets

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

Support $s(I)$: number of transactions that contain itemset I

Examples of frequent itemsets $s(I) \geq 3$

{Bread}: 4
{Milk} : 4
{Diaper} : 4
{Beer}: 3
{Diaper, Beer} : 3
{Milk, Bread} : 3

Market-Baskets – (2)

- Really, a general many-to-many mapping (association) between two kinds of things, where the one (the baskets) is a set of the other (the items)
  - But we ask about connections among "items," not "baskets."

- The technology focuses on common/frequent events, not rare events ("long tail").

Applications – (1)

- Items = products; baskets = sets of products someone bought in one trip to the store.

- Example application: given that many people buy beer and diapers together:
  - Run a sale on diapers; raise price of beer.
- Only useful if many buy diapers & beer.

Applications – (2)

- Baskets = Web pages; items = words.

- Example application: Unusual words appearing together in a large number of documents, e.g., “Brad” and “Angelina,” may indicate an interesting relationship.

Applications – (3)

- Baskets = sentences; items = documents containing those sentences.

- Example application: Items that appear together too often could represent plagiarism.
- Notice items do not have to be “in” baskets.

Definitions

- Itemset
  - A collection of one or more items
    - Example: {Milk, Bread, Diaper}
  - k-itemset
    - An itemset that contains k items

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

- Support (s)
  - **Count**: Frequency of occurrence of an itemset
    - E.g. s({Milk, Bread,Diaper}) = 2
  - **Fraction**: Fraction of transactions that contain an itemset
    - E.g. s({Milk, Bread, Diaper}) = 40%
- Frequent Itemset
  - An itemset I whose support is greater than or equal to a *minsup* threshold, $s(I) \geq$ minsup

Mining Frequent Itemsets task

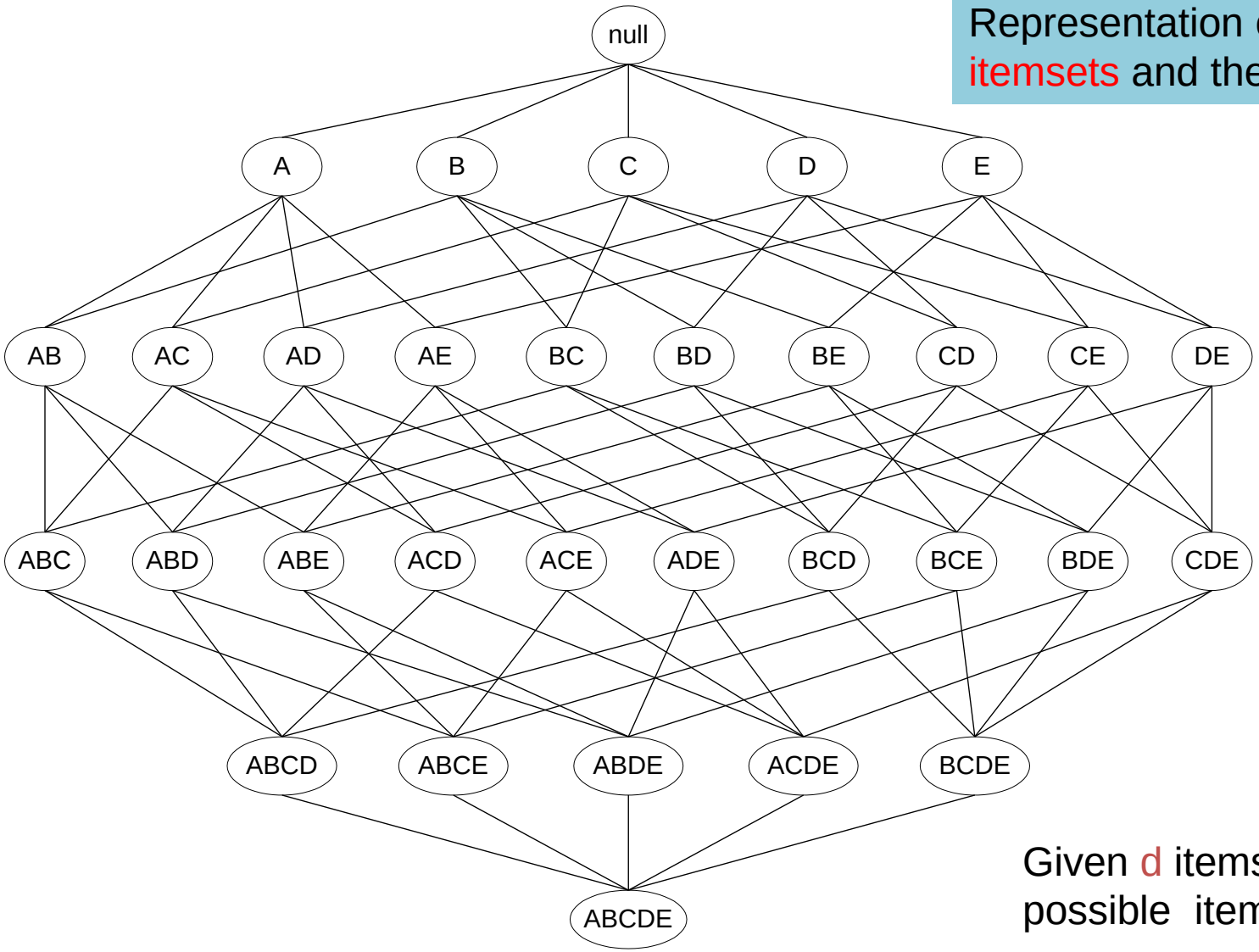
- Input: Market basket data, threshold *minsup*
- Output: All frequent itemsets with support ≥ *minsup*

- Problem parameters:
  - N (size): number of transactions
    - Wallmart: billions of baskets per year
    - Web: billions of pages
  - d (dimension): number of (distinct) items
    - Wallmart sells more than 100,000 items
    - Web: billions of words
  - w: max size of a basket
  - M: Number of possible itemsets.

$$\mathrm{M} = 2^d$$

The itemset lattice

Representation of all possible itemsets and their relationships

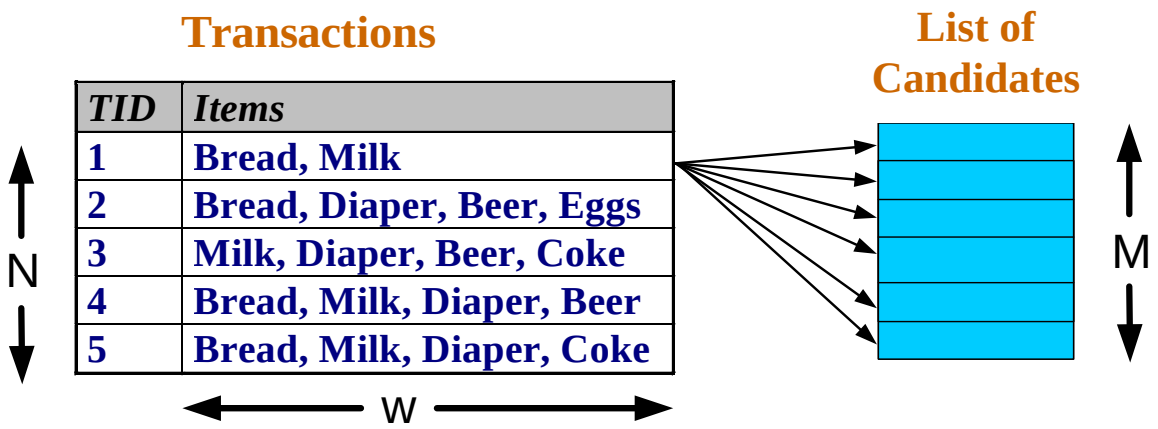


Given d items, there are 2^d possible itemsets

A Naïve Algorithm

- Brute-force approach: Every itemset is a candidate :
  - Consider all itemsets in the lattice, and scan the data for each candidate to compute the support
  - Time Complexity ~ O(NMw) , Space Complexity ~ O(d)
- OR
  - Scan the data, and for each transaction generate all possible itemsets. Keep a count for each itemset in the data.
  - Time Complexity ~ O($N2^w$) , Space Complexity ~ O(M)

- Expensive since M = 2^d !!!
  - No solution that considers all candidates is acceptable!



Computation Model

- Typically, data is kept in flat files rather than in a database system.
  - Stored on disk.
  - Stored basket-by-basket.
  - We can expand a baskets into pairs, triples, etc. as we read the data.
    - Use k nested loops, or recursion to generate all itemsets of size k.
- Data is too large to be loaded in memory.

Example file: retail

```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29
30 31 32
33 34 35
36 37 38 39 40 41 42 43 44 45 46
38 39 47 48
38 39 48 49 50 51 52 53 54 55 56 57 58
32 41 59 60 61 62
3 39 48
63 64 65 66 67 68
32 69
48 70 71 72
39 73 74 75 76 77 78 79
36 38 39 41 48 79 80 81
82 83 84
41 85 86 87 88
39 48 89 90 91 92 93 94 95 96 97 98 99 100 101
36 38 39 48 89
39 41 102 103 104 105 106 107 108
38 39 41 109 110
39 111 112 113 114 115 116 117 118
119 120 121 122 123 124 125 126 127 128 129 130 131 132 133
48 134 135 136
39 48 137 138 139 140 141 142 143 144 145 146 147 148 149
39 150 151 152
38 39 56 153 154 155
```

Example: items are positive integers, and each basket corresponds to a line in the file of space-separated integers

Computation Model – (2)

- The true cost of mining disk-resident data is usually the number of disk I/O's.
- In practice, association-rule algorithms read the data in passes – all baskets read in turn.

- Thus, we measure the cost by the number of passes an algorithm takes.

Main-Memory Bottleneck

- For many frequent-itemset algorithms, main memory is the critical resource.
  - As we read baskets, we need to count something, e.g., occurrences of pairs.
  - The number of different things we can count is limited by main memory.
  - Swapping counts in/out is too slow

The Apriori Principle

- Apriori principle (Main observation):
  - If an itemset is frequent, then all of its subsets must also be frequent
  - If an itemset is not frequent, then all of its supersets cannot be frequent
  - The support of an itemset never exceeds the support of its subsets

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

  - This is known as the anti-monotone property of support

Illustration of the Apriori principle

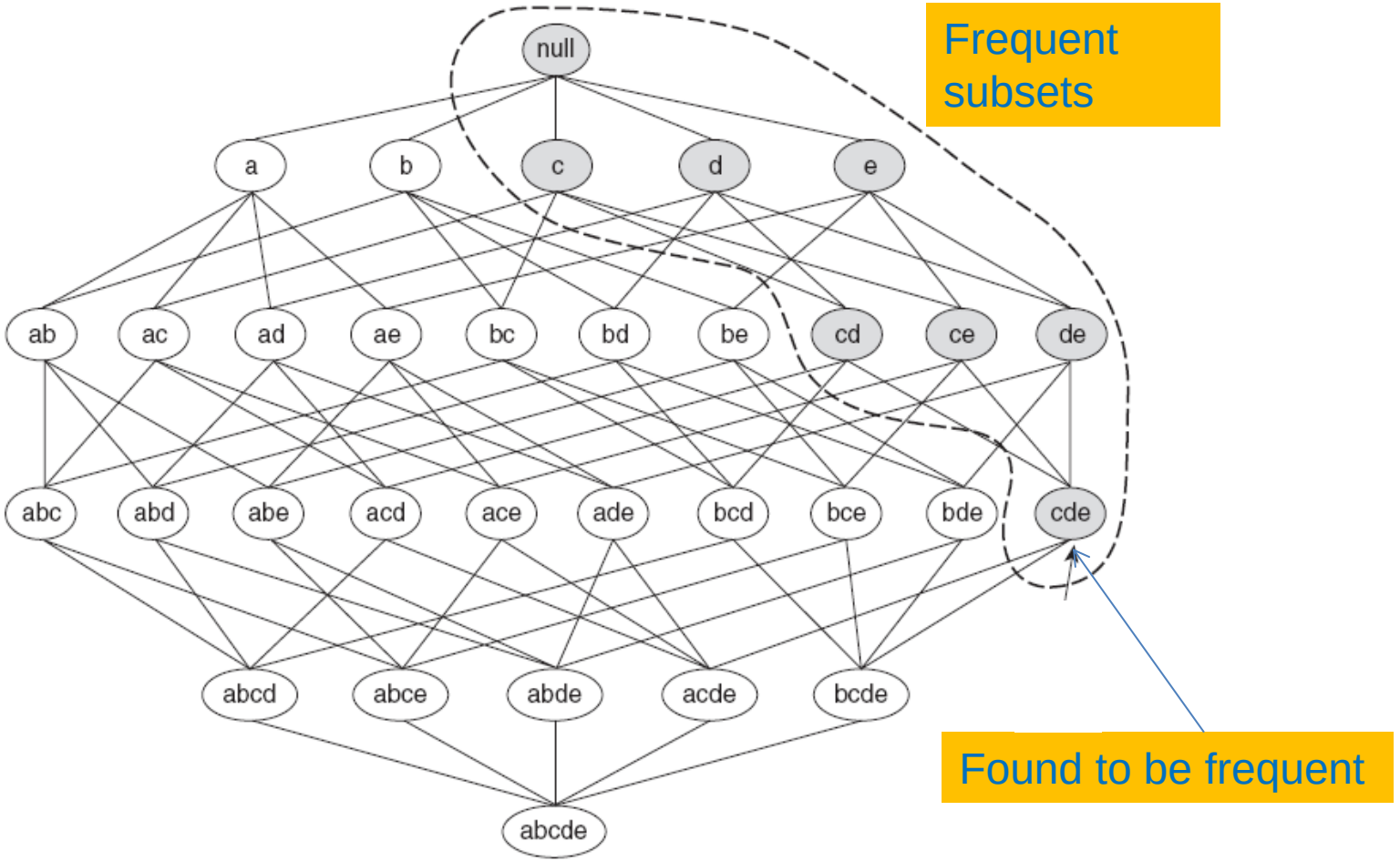
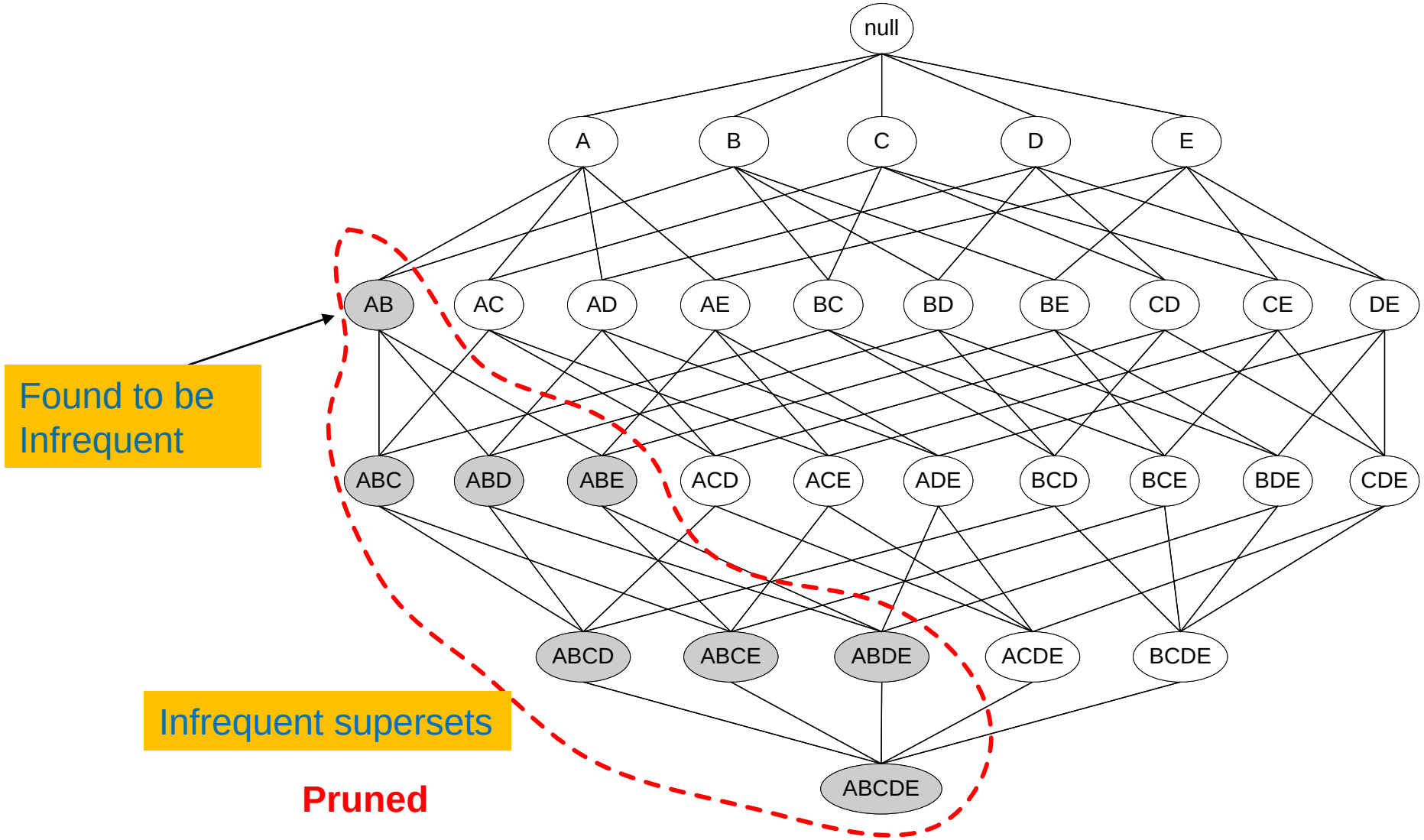


Figure 6.3. An illustration of the *Apriori* principle. If $\{c, d, e\}$ is frequent, then all subsets of this itemset are frequent.

Illustration of the Apriori principle



The Apriori algorithm

Level-wise approach

C_k = candidate itemsets of size k
L_k = frequent itemsets of size k

1. k = 1, C_1 = all items
2. While C_k not empty

Frequent itemset generation

3. **Scan the database** to find which itemsets in C_k are frequent and put them into L_k

Candidate generation

4. **Generate the candidate itemsets C_{k+1}** of size k+1 using L_k

5. k = k+1

R. Agrawal, R. Srikant: "Fast Algorithms for Mining Association Rules",
Proc. of the 20th Int'l Conference on Very Large Databases, 1994.

Candidate Generation

- Apriori principle:
  - An itemset of size k+1 is candidate to be frequent only if all of its subsets of size k are known to be frequent

Candidate generation:

- Construct a candidate of size k+1 by combining frequent itemsets of size k
  - If k = 1, take the all pairs of frequent items
  - If k > 1, join pairs of itemsets that differ by just one item
  - For each generated candidate itemset ensure that all subsets of size k are frequent.

Generate Candidates C_{k+1}

- Assumption: The items in an itemset are ordered
  - Integers ordered in increasing order, strings ordered in lexicographicly
  - The order ensures that if item $y > x$ appears before x, then x is not in the itemset
- The itemsets in L_k are also ordered

Create a candidate itemset of size k+1, by joining two itemsets of size k, that share the first k-1 items

Item 1	Item 2	Item 3
1	2	3
1	2	5
1	4	5

Generate Candidates C_{k+1}

- Assumption: The items in an itemset are ordered
  - Integers ordered in increasing order, strings ordered in lexicographicly
  - The order ensures that if item $y > x$ appears before x, then x is not in the itemset
- The itemsets in L_k are also ordered

Create a candidate itemset of size k+1, by joining two itemsets of size k, that share the first k-1 items

Item 1	Item 2	Item 3
1	2	3
1	2	5
1	4	5

Generate Candidates C_{k+1}

- Assumption: The items in an itemset are ordered
  - Integers ordered in increasing order, strings ordered in lexicographicly
  - The order ensures that if item $y > x$ appears before x, then x is not in the itemset
- The itemsets in L_k are also ordered

Create a candidate itemset of size k+1, by joining two itemsets of size k, that share the first k-1 items

Item 1	Item 2	Item 3
1	2	3
1	2	5
1	4	5

Are we missing something?
What about this candidate?

Generating Candidates C_{k+1} in SQL

- ***self-join*** L_k

  insert into C_{k+1}

  select $p.item_1, p.item_2, \ldots, p.item_k, q.item_k$

  from $L_k\ p, L_k\ q$

  where $p.item_1=q.item_1, \ldots, p.item_{k-1}=q.item_{k-1}, p.item_k < q.item_k$

Example

- L_3={abc, abd, acd, ace, bcd}
- Generating candidate set C_4
  - Self-join: L_3*L_3

item1	item2	item3
a	b	c
a	b	d
a	c	d
a	c	e
b	c	d

item1	item2	item3
a	b	c
a	b	d
a	c	d
a	c	e
b	c	d

p.item$_1$=q.item$_1$,p.item$_2$=q.item$_2$, p.item$_3$< q.item$_3$

Example

- L_3={abc, abd, acd, ace, bcd}
- Generating candidate set C_4
  - Self-join: L_3*L_3

item1	item2	item3
a	b	c
a	b	d
a	c	d
a	c	e
b	c	d

item1	item2	item3
a	b	c
a	b	d
a	c	d
a	c	e
b	c	d

p.item$_1$=q.item$_1$,p.item$_2$=q.item$_2$, p.item$_3$< q.item$_3$

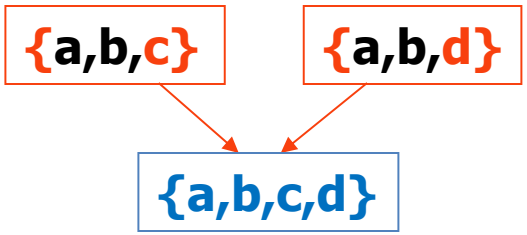
Example

- L_3={abc, abd, acd, ace, bcd}
- Generating candidate set C_4
  - Self-join: L_3*L_3

C_4 ={abcd}

item1	item2	item3
a	b	c
a	b	d
a	c	d
a	c	e
b	c	d

item1	item2	item3
a	b	c
a	b	d
a	c	d
a	c	e
b	c	d



$p.item_1=q.item_1, p.item_2=q.item_2, p.item_3< q.item_3$

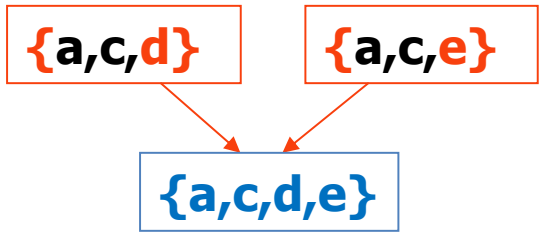
Example

- L_3={abc, abd, acd, ace, bcd}
- Generating candidate set C_4
  - Self-join: L_3*L_3

C_4 ={abcd
acde}

item1	item2	item3
a	b	c
a	b	d
a	c	d
a	c	e
b	c	d

item1	item2	item3
a	b	c
a	b	d
a	c	d
a	c	e
b	c	d



p.item$_1$=q.item$_1$,p.item$_2$=q.item$_2$, p.item$_3$< q.item$_3$

Illustration of the Apriori principle

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

minsup = 3

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)

Triplets (3-itemsets)

Itemset	Count
{Bread,Milk,Diaper}	2

If every subset is considered,

$\binom{6}{1} + \binom{6}{2} + \binom{6}{3} = 6 + 15 + 20 = 41$

With support-based pruning,

$\binom{6}{1} + \binom{4}{2} + 1 = 6 + 6 + 1 = 13$

Only this triplet has all subsets to be frequent
But it is below the minsup threshold

Generate Candidates C_{k+1}

- Are we done? Are all the candidates valid?

Item 1	Item 2	Item 3
1	2	3
1	2	5
1	4	5

Is this a valid candidate?

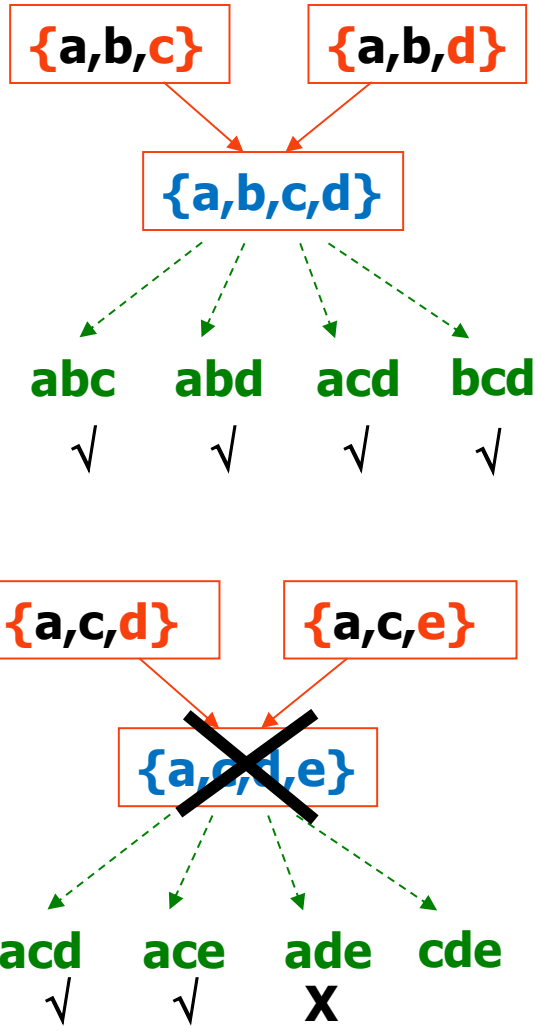
No. Subsets (1,3,5) and (2,3,5) should also be frequent

Apriori principle

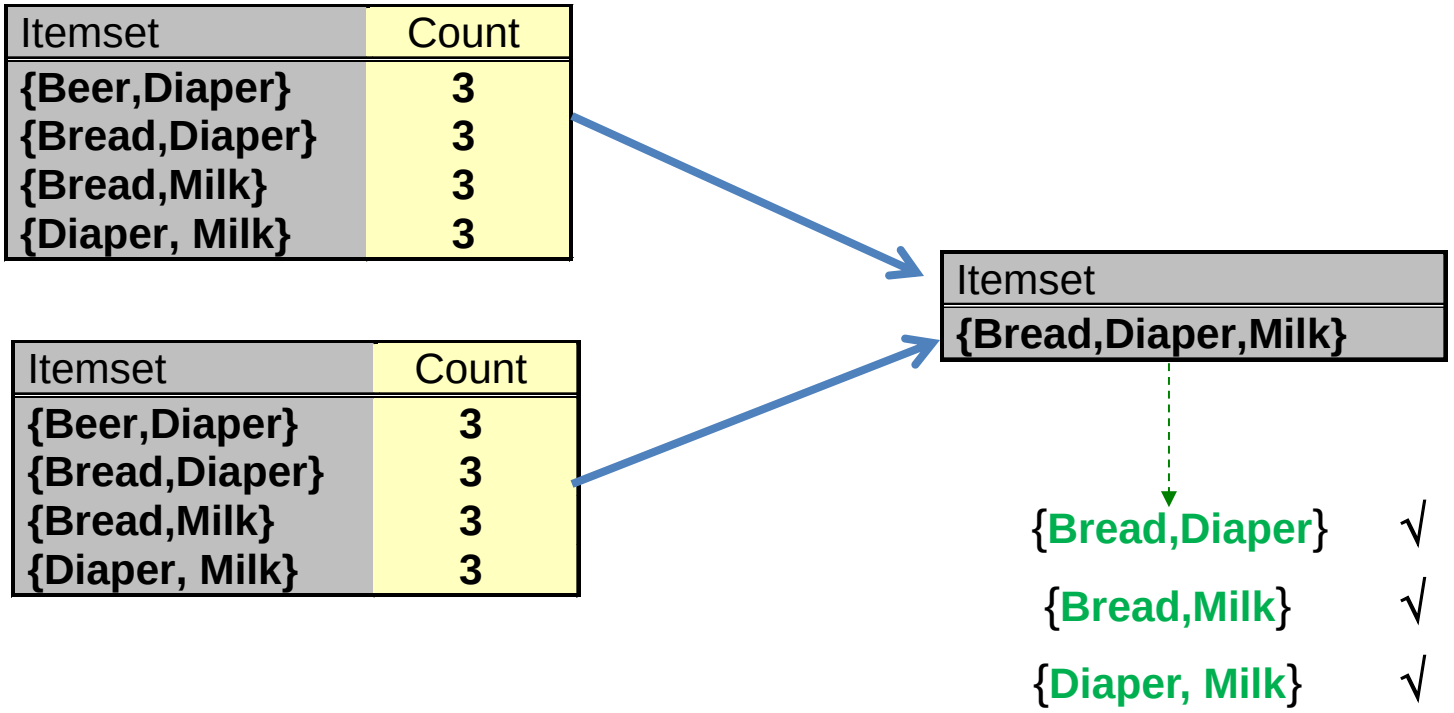
- Pruning step:
  - For each candidate (k+1)-itemset create all subset k-itemsets
  - Remove a candidate if it contains a subset k-itemset that is not frequent

Example

- L_3={abc, abd, acd, ace, bcd}
- ***Self-joining***: L_3*L_3
  - abcd from abc and abd
  - acde from acd and ace
- C_4={abcd, acde}
- ***Pruning:***
  - abcd is kept since all subset itemsets are in L_3
  - acde is removed because ade is not in L_3
- C_4={abcd}



Example II

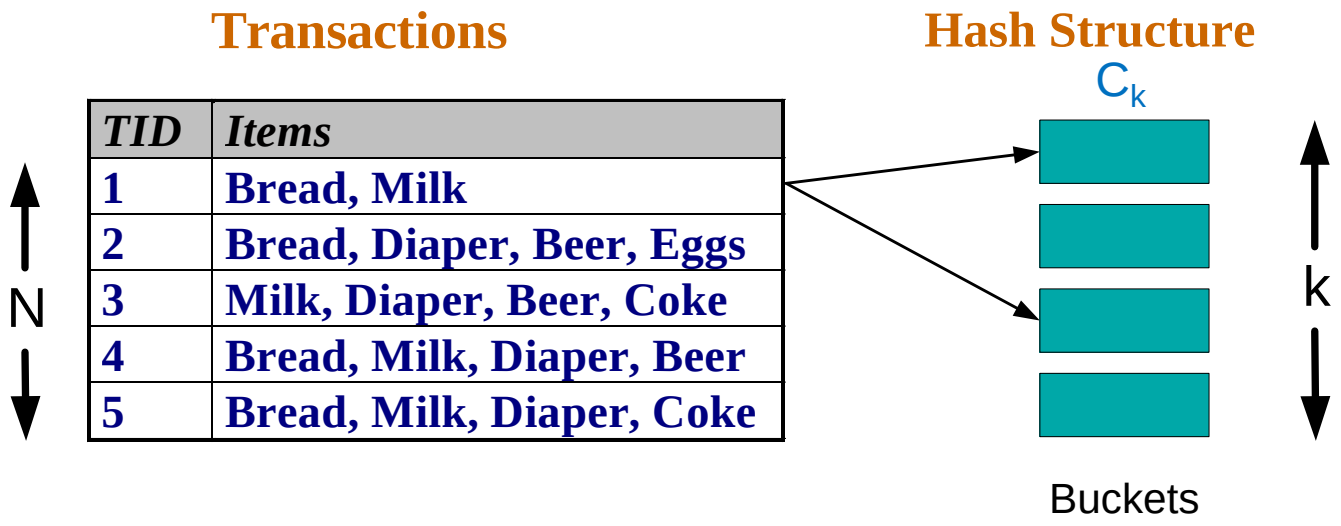

Itemset
Count
{Beer,Diaper}
3
{Bread,Diaper}
3
{Bread,Milk}
3
{Diaper, Milk}
3
Itemset
Count
{Beer,Diaper}
3
{Bread,Diaper}
3
{Bread,Milk}
3
{Diaper, Milk}
3
Itemset
{Bread,Diaper,Milk}
{Bread,Diaper} √
{Bread,Milk} √
{Diaper, Milk} √

Generate Candidates C_{k+1}

- We have all frequent k-itemsets L_k
- **Step 1: self-join** L_k
  - Create set C_{k+1} by joining frequent k-itemsets that share the first k-1 items
- **Step 2: prune**
  - Remove from C_{k+1} the itemsets that contain a subset k-itemset that is not frequent

Computing Frequent Itemsets

- Given the set of candidate itemsets C_k, we need to compute the support and find the frequent itemsets L_k.
- Scan the data, and use a hash structure to keep a counter for each candidate itemset that appears in the data



A simple hash structure

- Create a dictionary (hash table) that stores the candidate itemsets as keys, and the number of appearances as the value.
  - Initialize with zero
- Increment the counter for each itemset that you see in the data

Example

Suppose you have 15 candidate itemsets of length 3:

C_3 = {

{1 4 5}, {1 2 4}, {4 5 7}, {1 2 5}, {4 5 8},

{1 5 9}, {1 3 6}, {2 3 4}, {5 6 7}, {3 4 5},

{3 5 6}, {3 5 7}, {6 8 9}, {3 6 7}, {3 6 8}

}

Hash table stores the counts of the candidate itemsets as they have been computed so far

Key	Value
{3 6 7}	0
{3 4 5}	1
{1 3 6}	3
{1 4 5}	5
{2 3 4}	2
{1 5 9}	1
{3 6 8}	0
{4 5 7}	2
{6 8 9}	0
{5 6 7}	3
{1 2 4}	8
{3 5 7}	1
{1 2 5}	0
{3 5 6}	1
{4 5 8}	0

Example

A new tuple {1,2,3,5,6} generates the following itemsets of length 3:

{1 2 3}, {1 2 5}, {1 2 6}, {1 3 5}, {1 3 6},

{1 5 6}, {2 3 5}, {2 3 6}, {3 5 6},

Increment the counters for the itemsets in the dictionary

Key	Value
{3 6 7}	0
{3 4 5}	1
{1 3 6}	3
{1 4 5}	5
{2 3 4}	2
{1 5 9}	1
{3 6 8}	0
{4 5 7}	2
{6 8 9}	0
{5 6 7}	3
{1 2 4}	8
{3 5 7}	1
{1 2 5}	0
{3 5 6}	1
{4 5 8}	0

Example

A new tuple {1,2,3,5,6} generates the following itemsets of length 3:

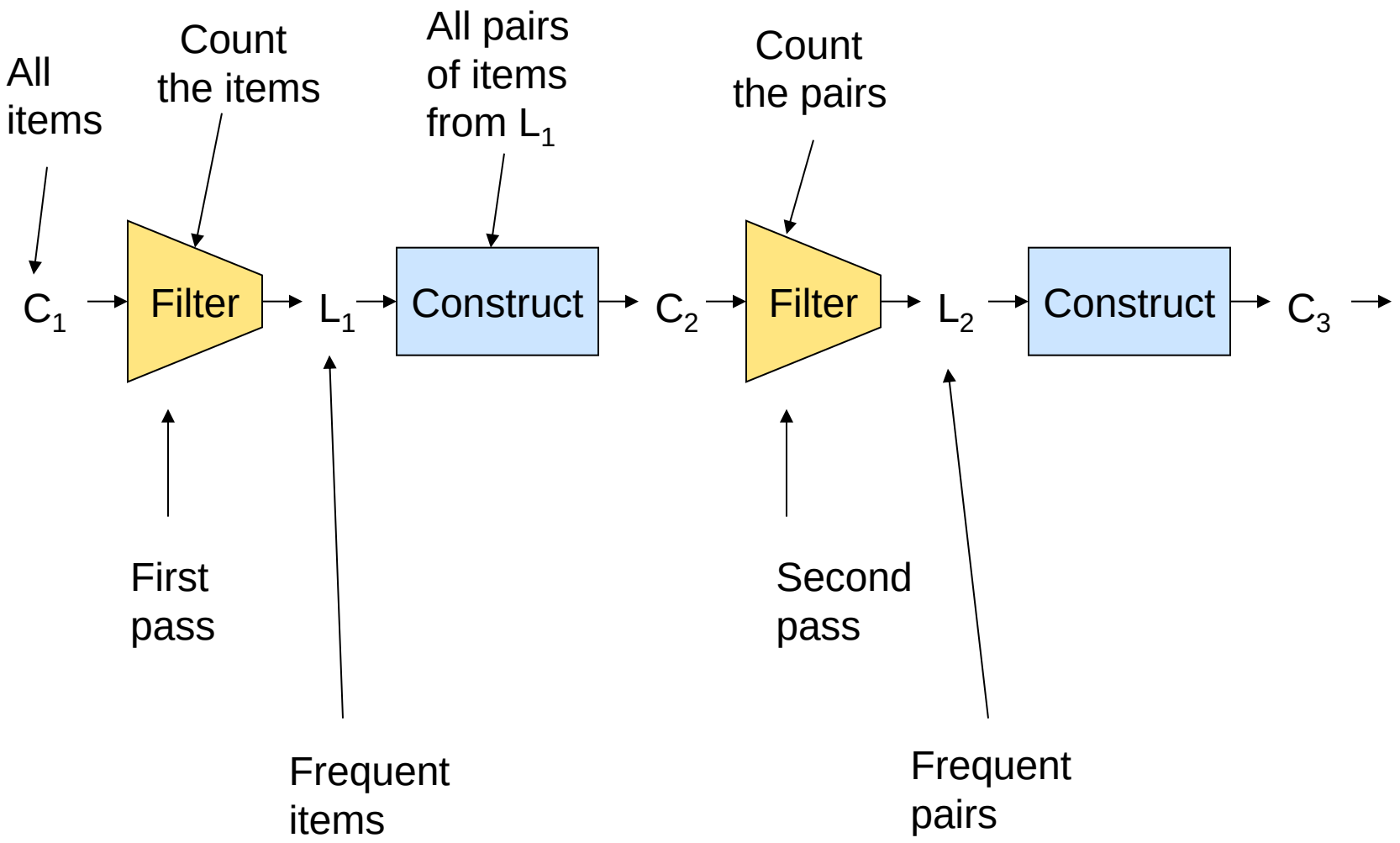
{1 2 3}, {1 2 5}, {1 2 6}, {1 3 5}, {1 3 6},

{1 5 6}, {2 3 5}, {2 3 6}, {3 5 6},

Increment the counters for the itemsets in the dictionary

Key	Value
{3 6 7}	0
{3 4 5}	1
{1 3 6}	4
{1 4 5}	5
{2 3 4}	2
{1 5 9}	1
{3 6 8}	0
{4 5 7}	2
{6 8 9}	0
{5 6 7}	3
{1 2 4}	8
{3 5 7}	1
{1 2 5}	1
{3 5 6}	2
{4 5 8}	0

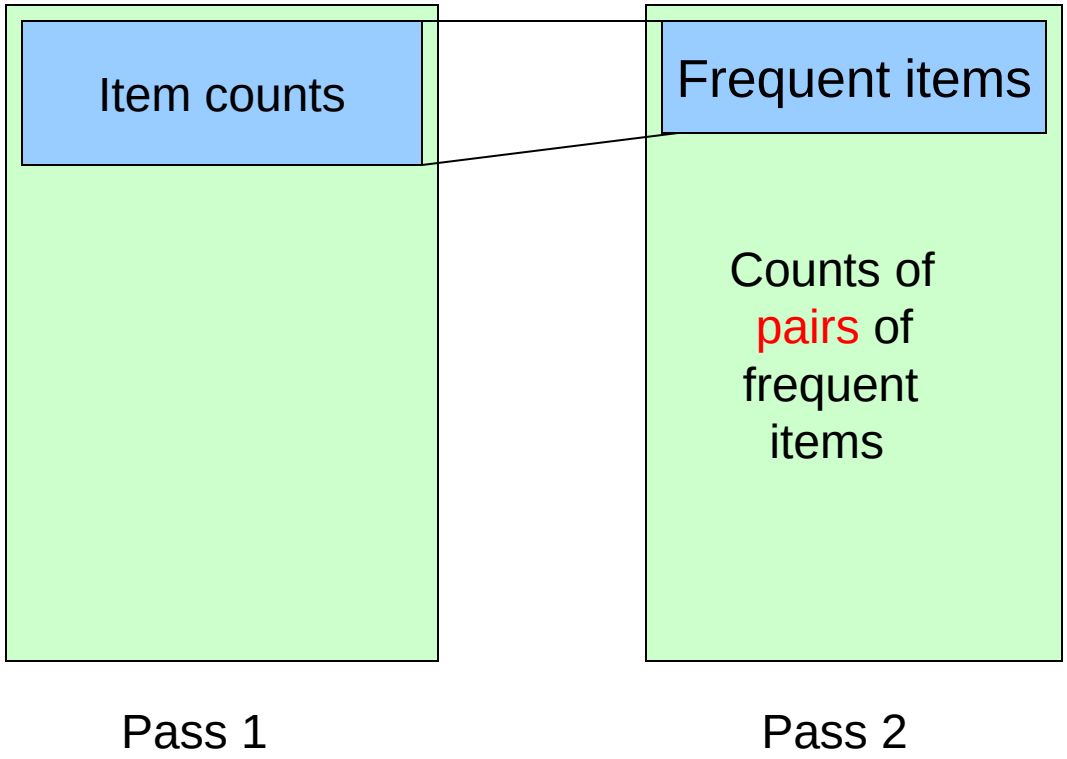
The frequent itemset algorithm



A-Priori for All Frequent Itemsets

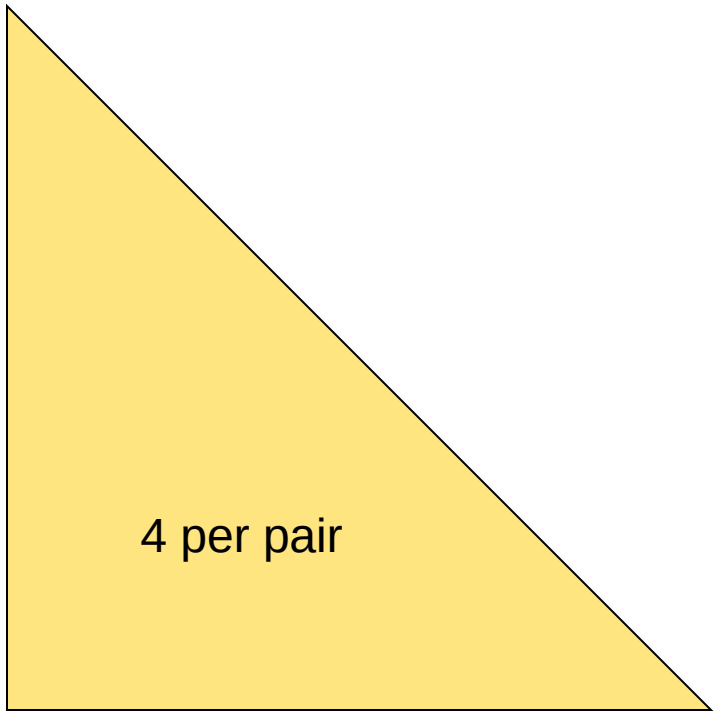
- One pass for each *k*.
- Needs room in main memory to count each candidate *k* -set.
- For typical market-basket data and reasonable support (e.g., 1%), *k* = 2 requires the most memory.

Picture of A-Priori

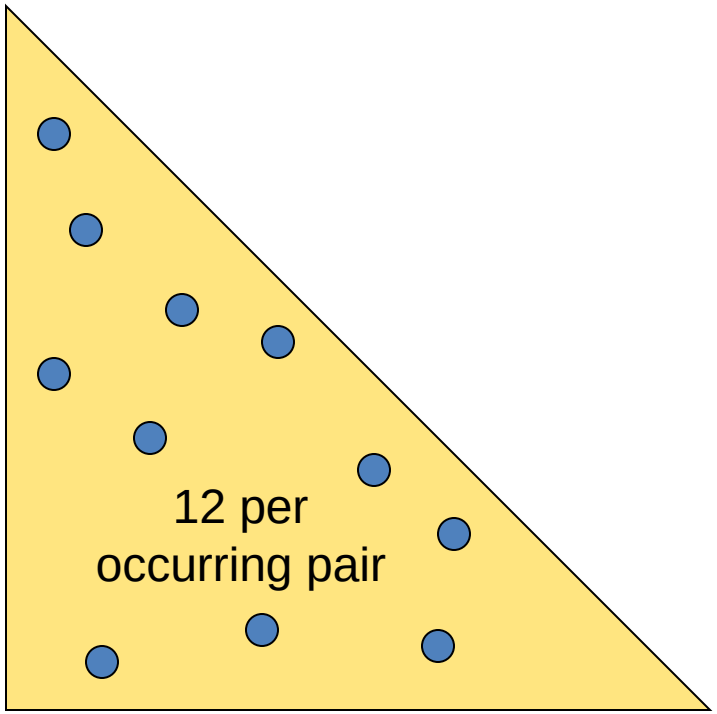


Details of Main-Memory Counting

- Two approaches:
  1. Count all pairs, using a "triangular matrix" = one dimensional array that stores the lower diagonal.
  2. Keep a table of triples [*i*, *j*, *c*] = "the count of the pair of items {*i*, *j* } is *c*."
- (1) requires only 4 bytes/pair.
  - Note: always assume integers are 4 bytes.
- (2) requires 12 bytes/pair, but only for those pairs with count > 0.



Method (1)



Method (2)

Triangular-Matrix Approach

- Number items 1, 2,…
  - Requires table of size O(n) to convert item names to consecutive integers.
- Count {i, j } only if $i < j$.
- Keep pairs in the order {1,2}, {1,3},…, {1,n }, {2,3}, {2,4},…,{2,n }, {3,4},…, {3,n },…{n -1,n }.
- Find pair {i, j } at the position

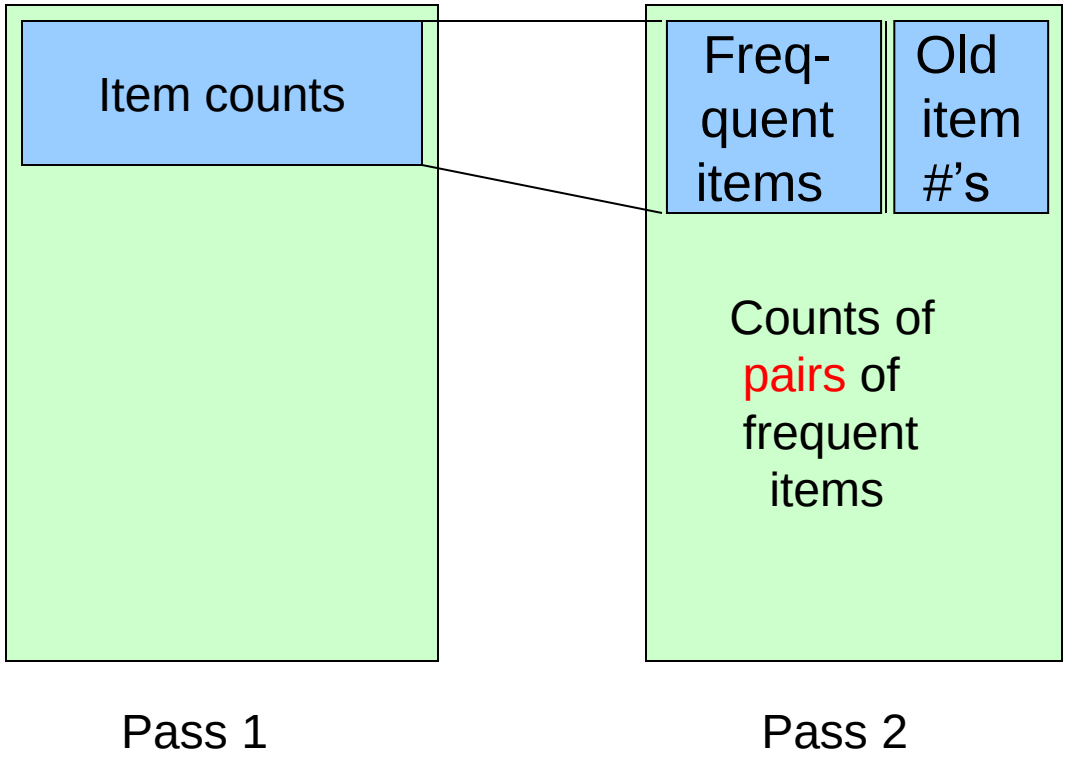
  $(i - 1)(n - i/2) + j - i$.

- Total number of pairs $n(n-1)/2$; total bytes about $2n^2$.

Details of Approach #2

- Total bytes used is about 12*p*, where *p* is the number of pairs that actually occur.
  - Beats triangular matrix if no more than1/3 of possible pairs actually occur.
- May require extra space for retrieval structure, e.g., a hash table.

A-Priori Using Triangular Matrix for Counts



ASSOCIATION RULES

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},

Implication means co-occurrence, not causality!

Mining Association Rules

- **Association Rule**
  - An implication expression of the form $X \rightarrow Y$, where X and Y are itemsets
    - {Milk, Diaper} → {Beer}
- **Rule Evaluation Metrics**
  - Support (s)
    - Fraction of transactions that contain both X and Y = the probability P(X,Y) that X and Y occur together
  - Confidence (c)
    - How often Y appears in transactions that contain X = the conditional probability P(Y|X) that Y occurs given that X has occurred.
- **Problem Definition**
  - Input: Market-basket data, minsup, minconf values
  - Output: All rules with items in I having s ≥ minsup and c≥ minconf

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:

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

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

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

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 partitioning of a frequent itemset into Left-Hand-Side (LHS) and Right-Hand-Side (RHS)

Frequent itemset: {A,B,C,D}
E.g., Rule: $AB \rightarrow CD$

All Candidate rules:

$BCD \rightarrow A$, $ACD \rightarrow B$, $ABD \rightarrow C$, $ABC \rightarrow D$,
$CD \rightarrow AB$, $BD \rightarrow AC$, $BC \rightarrow AD$, $AD \rightarrow BC$, $AB \rightarrow CD$, $AC \rightarrow BD$,
$D \rightarrow ABC$, $C \rightarrow ABD$, $B \rightarrow ACD$, $A \rightarrow BCD$

Association Rule anti-monotonicity

- In general, confidence does not have an anti-monotone property with respect to the size of the itemset:

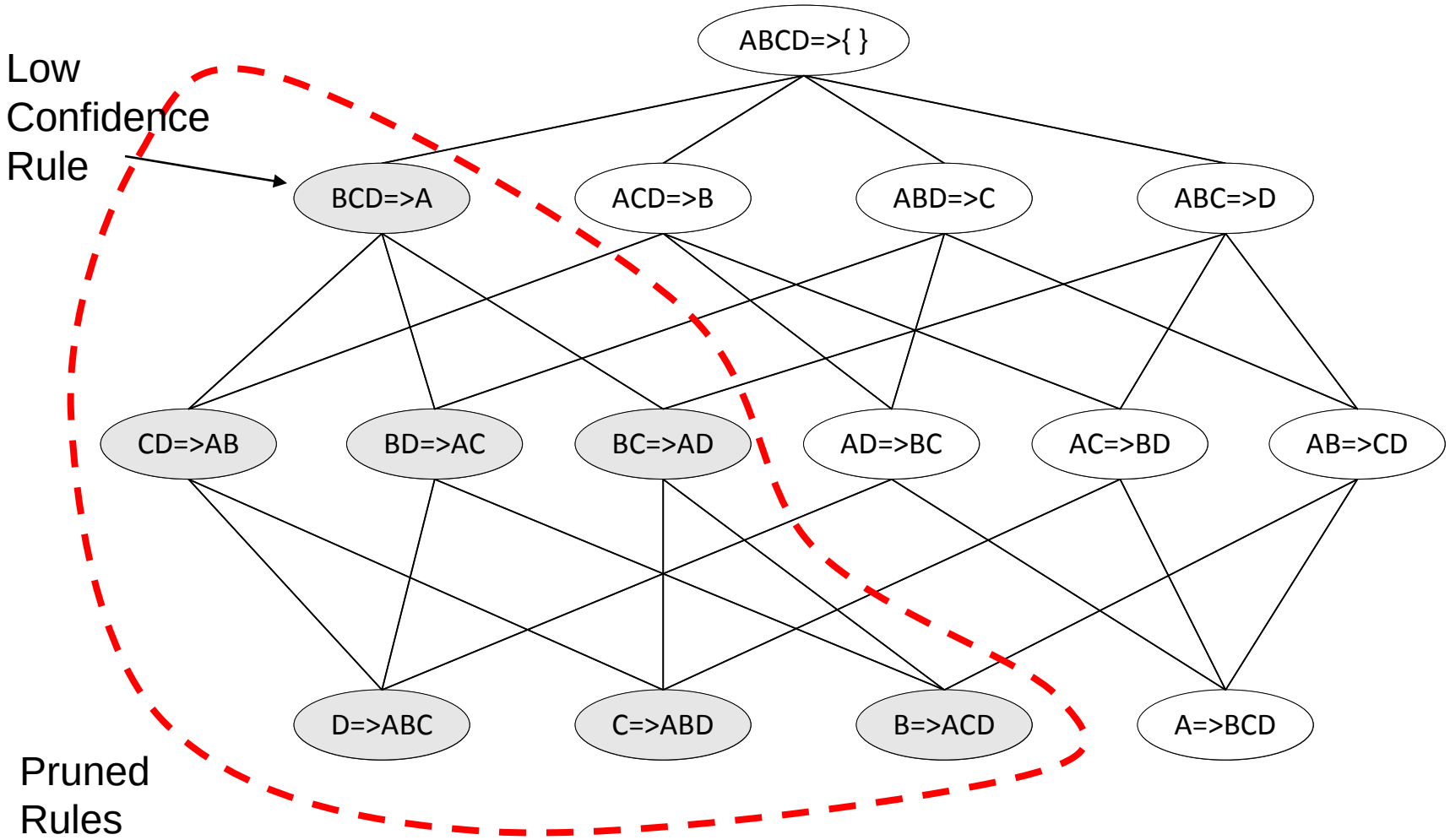
  c(ABC $\rightarrow$D) can be larger or smaller than c(AB $\rightarrow$D)

- But confidence is anti-monotone w.r.t. number of items on the RHS of the rule (or monotone with respect to the LHS of the rule)

- e.g., L = {A,B,C,D}:

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

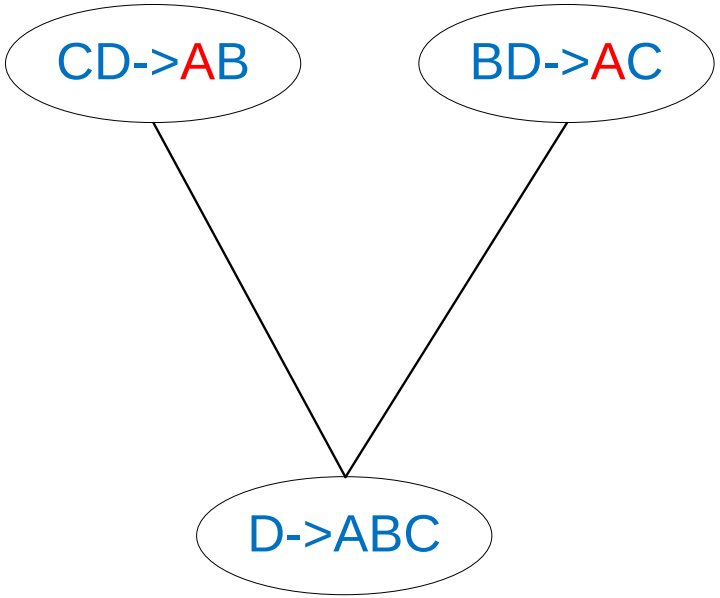
Rule Generation for Apriori Algorithm



Lattice of rules created by the RHS

Rule Generation for APriori Algorithm

- Candidate rule is generated by merging two rules that share the same prefix in the RHS



- 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
- Essentially we are doing APriori on the RHS

RESULT POST-PROCESSING

Compact Representation of Frequent Itemsets

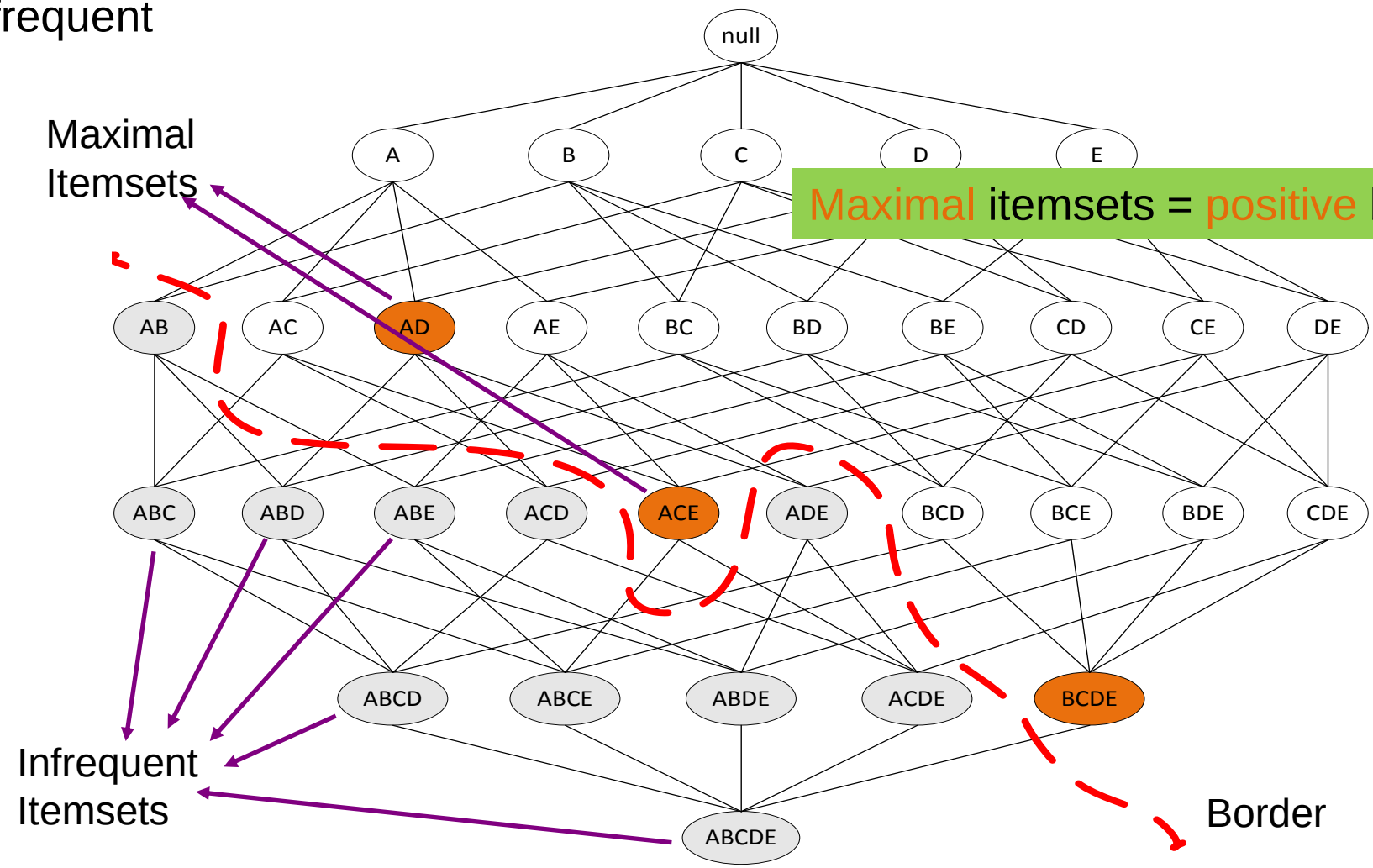
- Some itemsets are redundant because they have identical support as their supersets

TID	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1

- Number of frequent itemsets $= 3 \times \sum_{k=1}^{10} \binom{10}{k}$
- Need a compact representation

Maximal Frequent Itemsets

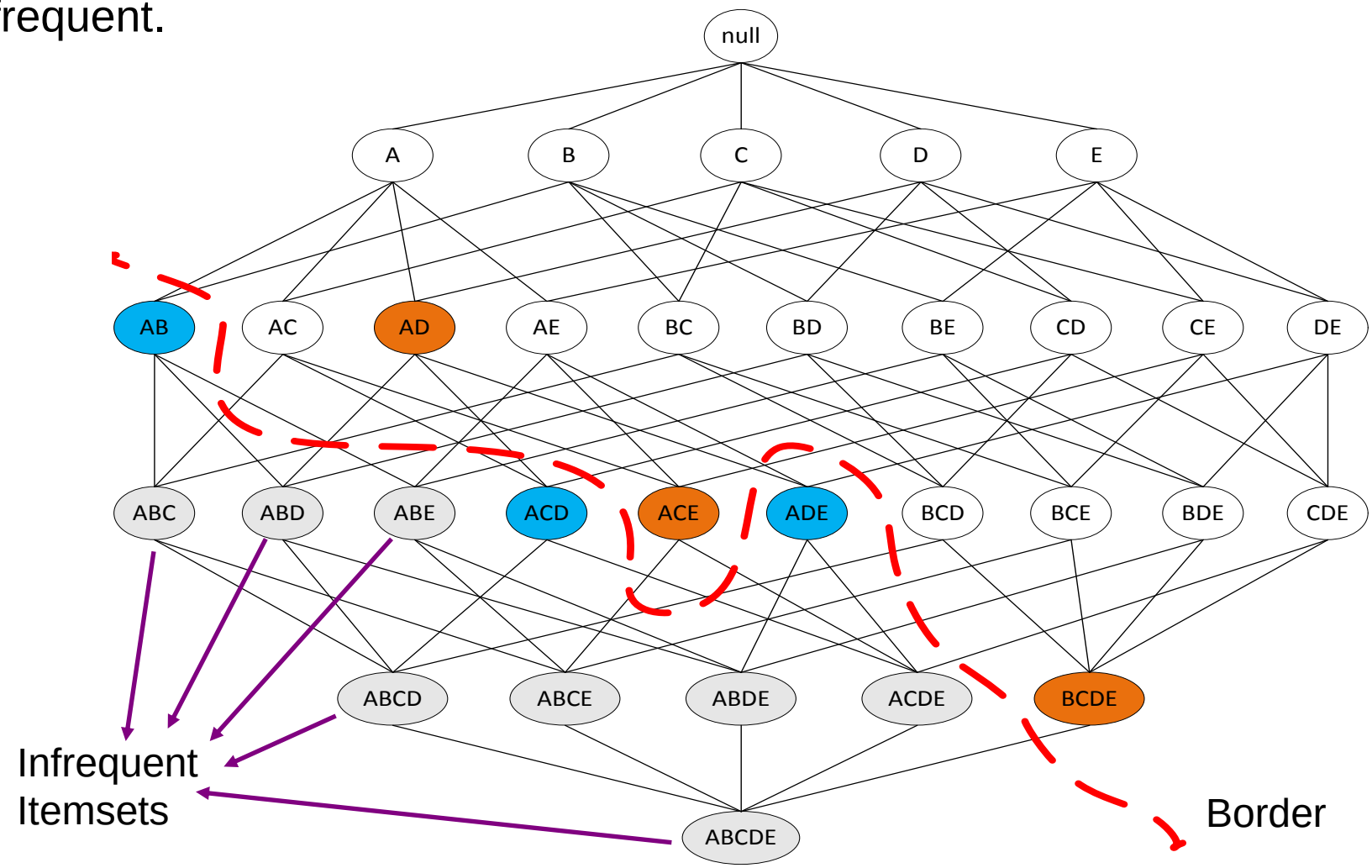
An itemset is maximal frequent if none of its immediate supersets is frequent



Maximal: no superset has this property

Negative Border

Itemsets that are not frequent, but all their immediate subsets are frequent.



Minimal: no subset has this property

Border

- Border = Positive Border + Negative Border
  - Itemsets such that all their immediate subsets are frequent and all their immediate supersets are infrequent.

- Either the positive, or the negative border is sufficient to summarize all frequent itemsets.

Closed Itemsets

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

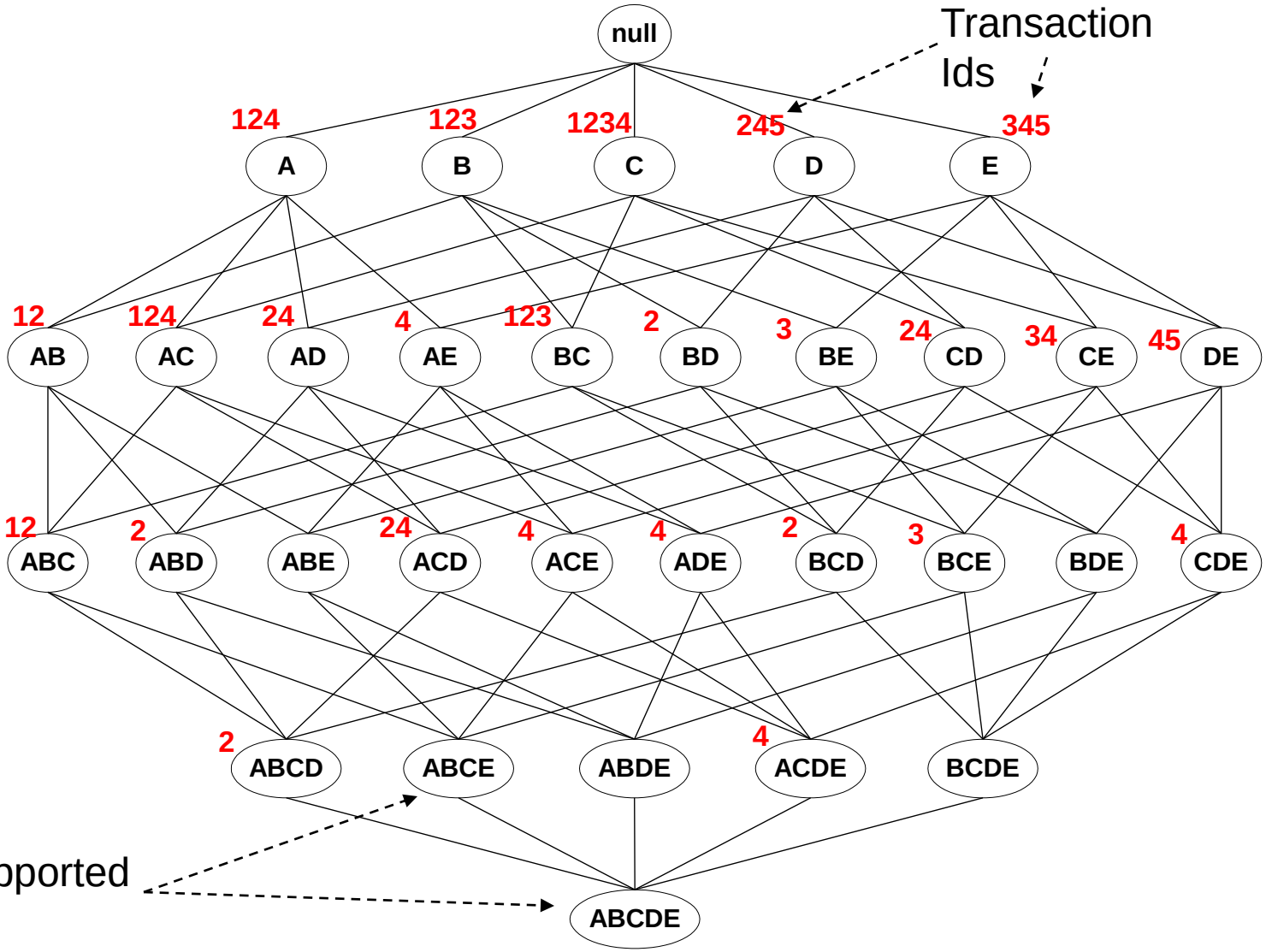
TID	Items
1	{A,B}
2	{B,C,D}
3	{A,B,C,D}
4	{A,B,D}
5	{A,B,C,D}

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

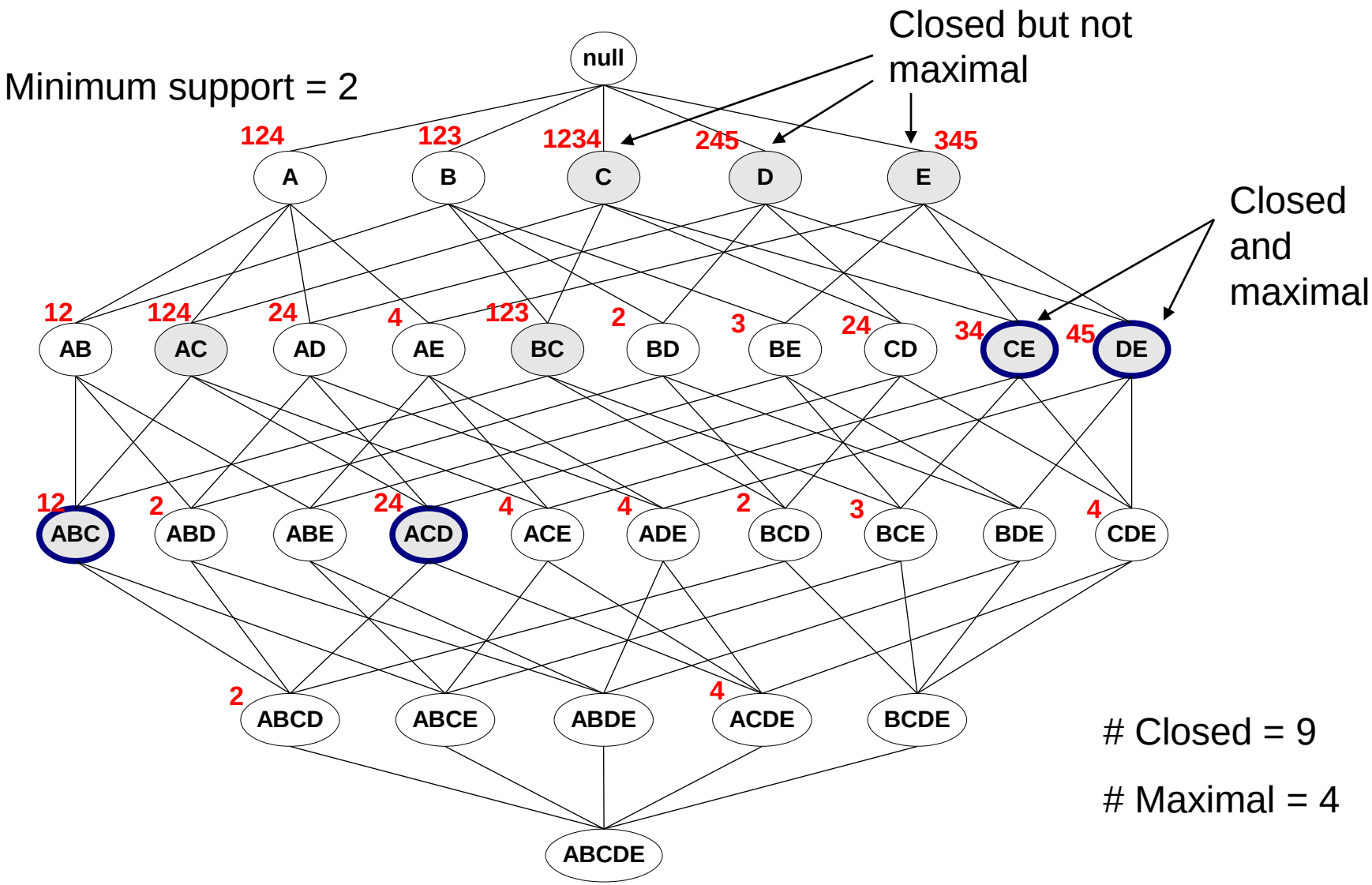
Itemset	Support
{A,B,C}	2
{A,B,D}	3
{A,C,D}	2
{B,C,D}	3
{A,B,C,D}	2

Maximal vs Closed Itemsets

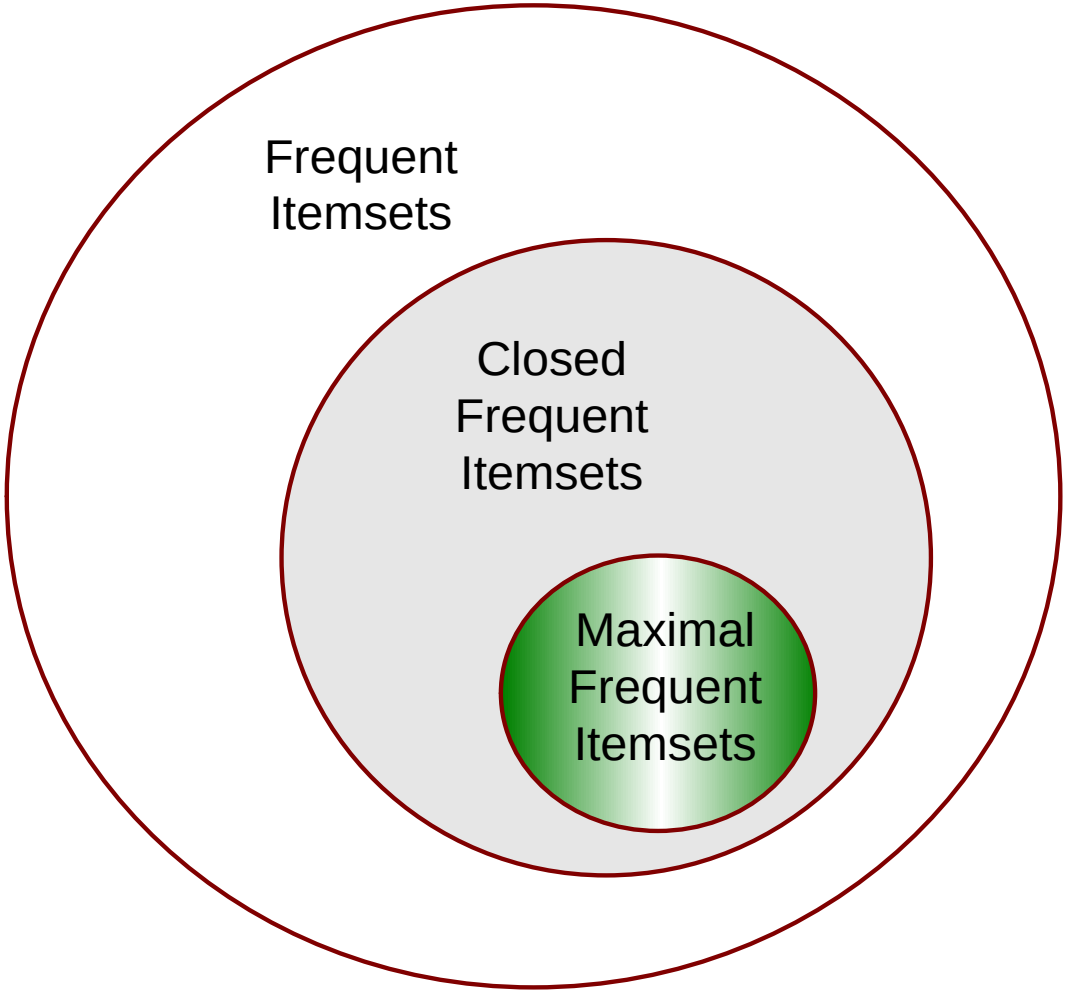
TID	Items
1	ABC
2	ABCD
3	BCE
4	ACDE
5	DE



Maximal vs Closed Frequent Itemsets



Maximal vs Closed Itemsets



Pattern Evaluation

- Association rule algorithms tend to produce too many rules but many of them are uninteresting or redundant
  - Redundant if {A,B,C} → {D} and {A,B} → {D} have same support & confidence
    - Summarization techniques
  - Uninteresting, if the pattern that is revealed does not offer useful information.
    - Interestingness measures: a hard problem to define
- Interestingness measures can be used to prune/rank the derived patterns
  - Subjective measures: require human analyst
  - Objective measures: rely on the data.
- 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	$\bar{Y}$	
X	f_{11}	f_{10}	f_{1+}
$\bar{X}$	f_{01}	f_{00}	f_{0+}
	f_{+1}	f_{+0}	N

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 $\bar{Y}$

X: itemset X appears in tuple
Y: itemset Y appears in tuple
$\bar{X}$: itemset X does not appear in tuple
$\bar{Y}$: itemset Y does not appear in tuple

Used to define various measures

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

Drawback of Confidence

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

Number of people that drink tea

Number of people that drink coffee and tea

Number of people that drink coffee but not tea

Number of people that drink coffee

Association Rule: Tea $\rightarrow$ Coffee

Confidence= $P(\text{Coffee}|\text{Tea}) = \frac{15}{20} = 0.75$

but $P(\text{Coffee}) = \frac{90}{100} = 0.9$

- Although confidence is high, rule is misleading
- $P(\text{Coffee}|\overline{\text{Tea}}) = 0.9375$

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,B) = 420/1000 = 0.42
  - P(S) × P(B) = 0.6 × 0.7 = 0.42

  - P(S,B) **=** P(S) × P(B) => Statistical independence

Statistical Independence

- Population of 1000 students
  - 600 students know how to swim (S)
  - 700 students know how to bike (B)
  - 500 students know how to swim and bike (S,B)

  - P(S,B) = 500/1000 = 0.5
  - P(S) × P(B) = 0.6 × 0.7 = 0.42

  - P(S,B) **>** P(S) × P(B) => Positively correlated

Statistical Independence

- Population of 1000 students
  - 600 students know how to swim (S)
  - 700 students know how to bike (B)
  - 300 students know how to swim and bike (S,B)

  - P(S,B) = 300/1000 = 0.3
  - P(S) × P(B) = 0.6 × 0.7 = 0.42

  - P(S,B) **<** P(S) × P(B) => Negatively correlated

Statistical-based Measures

- Measures that take into account statistical dependence
  - Lift/Interest/PMI

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

In text mining it is called: Pointwise Mutual Information

  - Piatesky-Shapiro

$$\text{PS} = P(X,Y) - P(X)P(Y)$$

- All these measures measure deviation from independence
  - The higher, the better (why?)

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)

= 0.15/(0.9*0.2)

Another Example

	of	the	of, the
Fraction of documents	0.9	0.9	0.8

$$P(\text{of}, \text{the}) \approx P(\text{of})P(\text{the})$$

If I was creating a document by picking words randomly, (of, the) have more or less the same probability of appearing together by chance

No correlation

	hong	kong	hong, kong
Fraction of documents	0.2	0.2	0.19

$$P(\text{hong}, \text{kong}) \gg P(\text{hong})P(\text{kong})$$

(hong, kong) have much lower probability to appear together by chance.
The two words appear almost always only together

Positive correlation

	obama	karagounis	obama, karagounis
Fraction of documents	0.2	0.2	0.001

$$P(\text{obama}, \text{karagounis}) \ll P(\text{obama})P(\text{karagounis})$$

(obama, karagounis) have much higher probability to appear together by chance.
The two words appear almost never together

Negative correlation

Drawbacks of Lift/Interest/Mutual Information

	honk	konk	honk, konk
Fraction of documents	0.0001	0.0001	0.0001

$$MI(honk, konk) = \frac{0.0001}{0.0001 * 0.0001} = 10000$$

	hong	kong	hong, kong
Fraction of documents	0.2	0.2	0.19

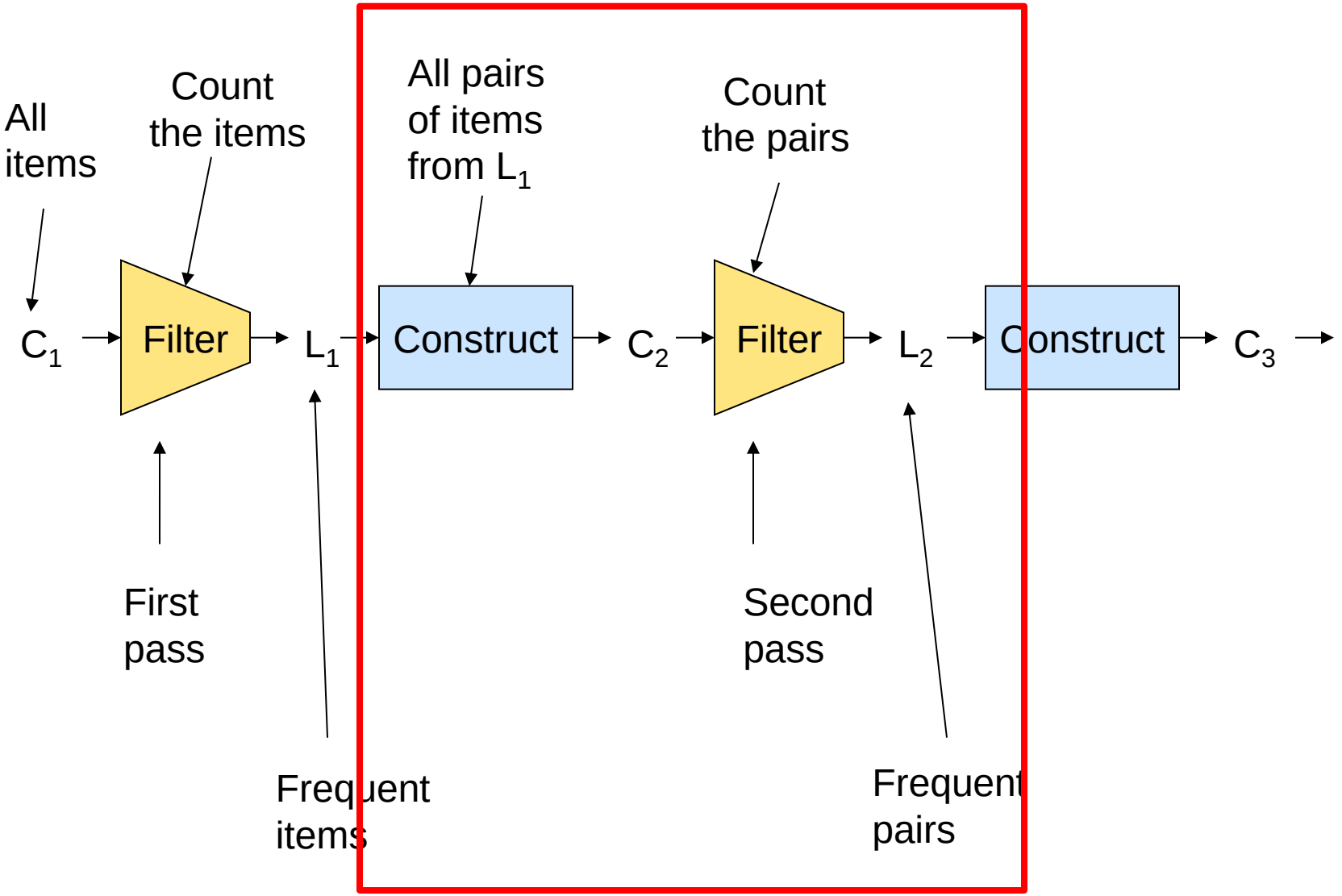
$$MI(hong, kong) = \frac{0.19}{0.2 * 0.2} = 4.75$$

Rare co-occurrences are deemed more interesting.
But this is not always what we want

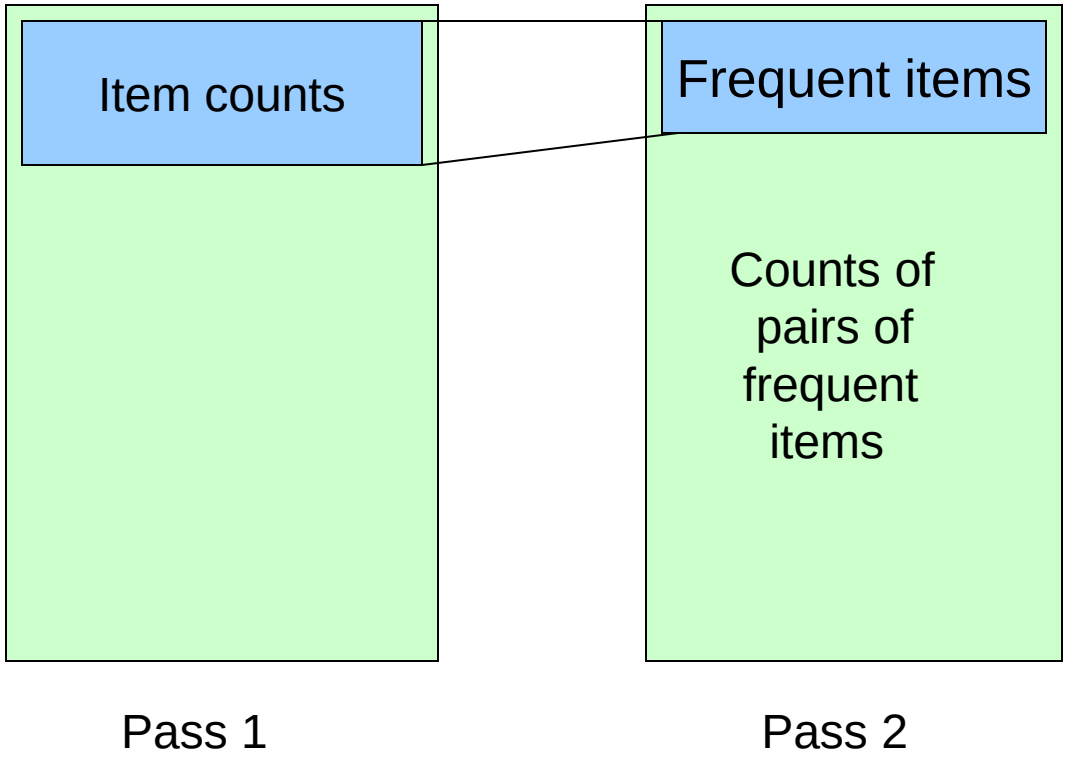
ALTERNATIVE FREQUENT ITEMSET COMPUTATION

Slides taken from Mining Massive Datasets course by Anand Rajaraman and Jeff Ullman.

Finding the frequent pairs is usually the most expensive operation

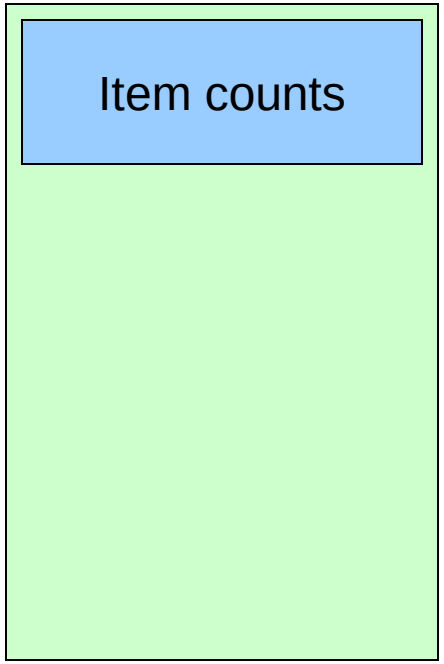


Picture of A-Priori



PCY Algorithm

- During Pass 1 (computing frequent items) of Apriori, most memory is idle.
- Use that memory to keep a hash table where pairs of items are hashed.
- The hash table keeps just counts of the number of pairs hashed in each bucket, not the pairs themselves.

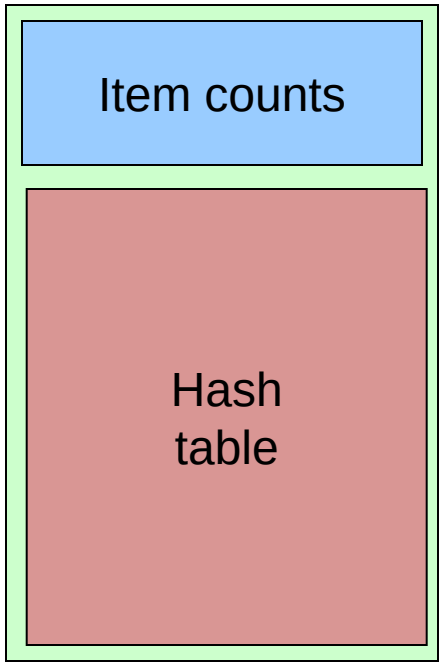


Pass 1

Needed Extensions

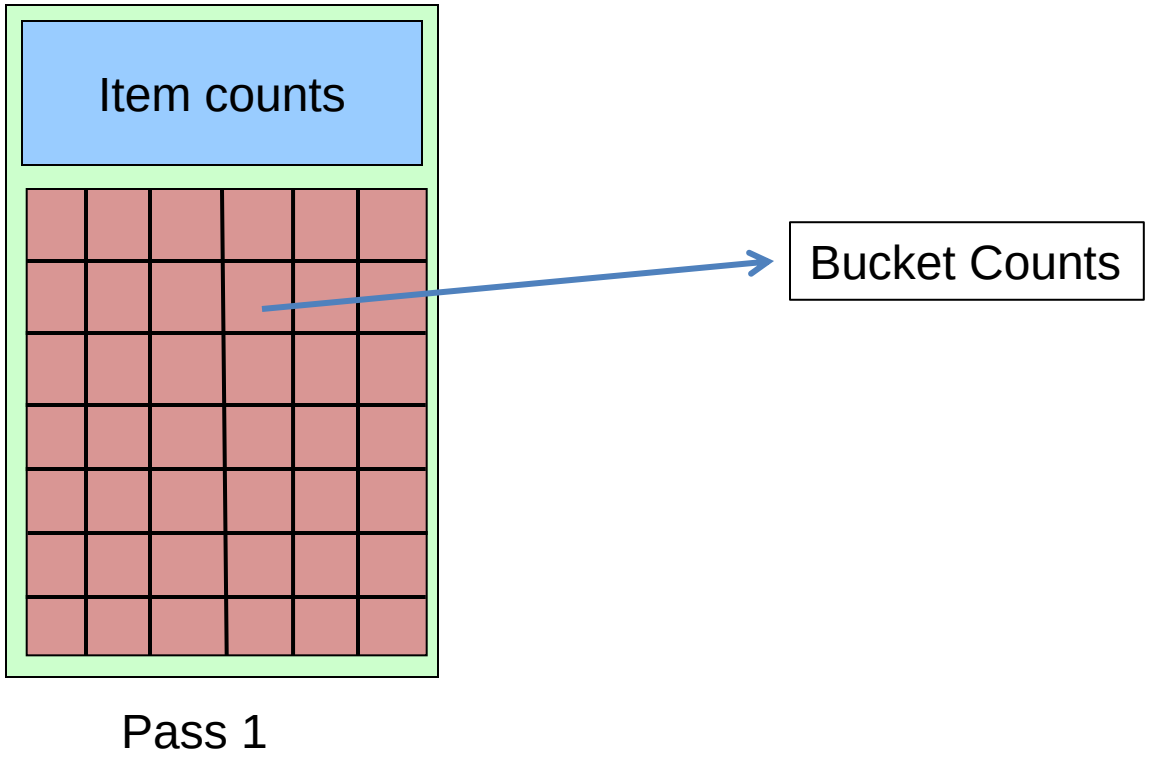
1. Pairs of items need to be generated from the input file; they are not present in the file.
2. Memory organization:
   - Space to count each item.
     - One (typically) 4-byte integer per item.
   - Use the rest of the space for as many integers, representing buckets, as we can.

Picture of PCY



Pass 1

Picture of PCY



PCY Algorithm – Pass 1

```
FOR (each basket) {
    FOR (each item in the basket)
      add 1 to item's count;
    FOR (each pair of items in the basket)
    {
      hash the pair to a bucket;
      add 1 to the count for that bucket
    }
}
```

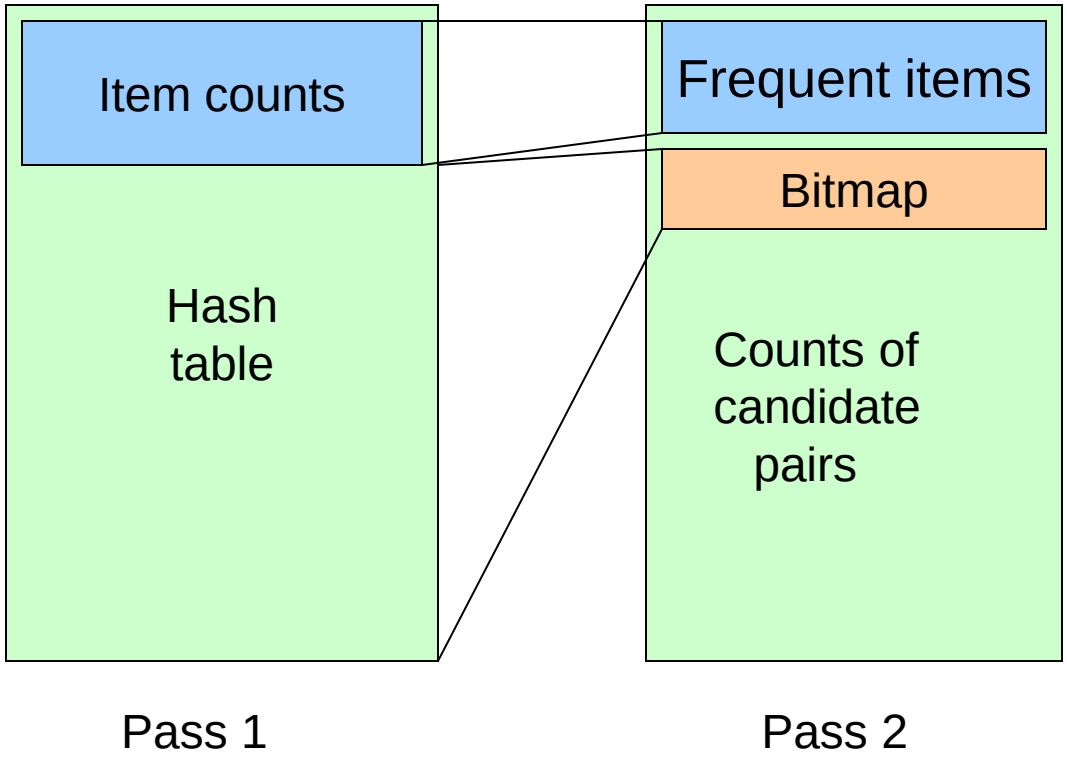
Observations About Buckets

- A bucket is frequent if its count is at least the support threshold.
- A bucket that a frequent pair hashes to is surely frequent.
  - We cannot use the hash table to eliminate any member of this bucket.
- Even without any frequent pair, a bucket can be frequent.
  - Again, nothing in the bucket can be eliminated.
- But in the best case, the count for a bucket is less than the support s.
  - Now, all pairs that hash to this bucket can be eliminated as candidates, even if the pair consists of two frequent items.
- On Pass 2 (frequent pairs), we only count pairs that hash to frequent buckets.

PCY Algorithm – Between Passes

- Replace the buckets by a bit-vector:
  - 1 means the bucket is frequent; 0 means it is not.
- 4-byte integers are replaced by bits, so the bit-vector requires 1/32 of memory.
- Also, find which items are frequent and list them for the second pass.
  - Same as with Apriori

Picture of PCY



PCY Algorithm – Pass 2

- Count all pairs {*i*, *j* } that meet the conditions for being a candidate pair:
  1. Both *i* and *j* are frequent items.
  2. The pair {*i*, *j* }, hashes to a bucket number whose bit in the bit vector is 1.

- Notice both these conditions are necessary for the pair to have a chance of being frequent.

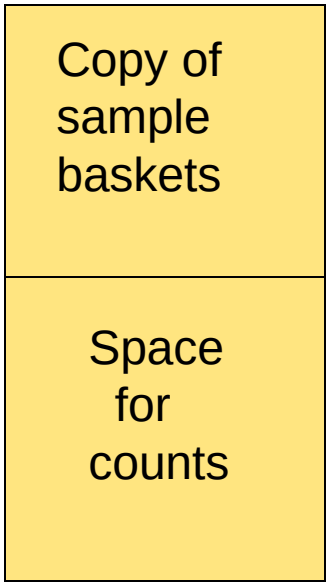
All (Or Most) Frequent Itemsets in less than 2 Passes

- A-Priori, PCY, etc., take *k* passes to find frequent itemsets of size *k*.
- Other techniques use 2 or fewer passes for all sizes:
  - Simple sampling algorithm.
  - SON (Savasere, Omiecinski, and Navathe).
  - Toivonen.

Simple Sampling Algorithm – (1)

- Take a random sample of the market baskets.
- Run Apriori or one of its improvements (for sets of all sizes, not just pairs) in main memory, so you don't pay for disk I/O each time you increase the size of itemsets.
  - Make sure the sample is such that there is enough space for counts.

Main-Memory Picture



Simple Algorithm – (2)

- Use as your support threshold a suitable, scaled-back number.
  - E.g., if your sample is 1/100 of the baskets, use *s* /100 as your support threshold instead of *s*.
- You could stop here (single pass)
  - What could be the problem?

Simple Algorithm – Option

- Optionally, verify that your guesses are truly frequent in the entire data set by a second pass (eliminate false positives)

- But you don't catch sets frequent in the whole but not in the sample. (false negatives)
  - Smaller threshold, e.g., *s* /125, helps catch more truly frequent itemsets.
    - But requires more space.

SON Algorithm – (1)

- First pass: Break the data into chunks that can be processed in main memory.
- Read one chunk at the time
  - Find all frequent itemsets for each chunk.
  - Threshold = s/number of chunks

- An itemset becomes a candidate if it is found to be frequent in any one or more chunks of the baskets.

SON Algorithm – (2)

- Second pass: count all the candidate itemsets and determine which are frequent in the entire set.

- Key "monotonicity" idea: an itemset cannot be frequent in the entire set of baskets unless it is frequent in at least one subset.
  - Why?

SON Algorithm – Distributed Version

- This idea lends itself to distributed data mining.
- If baskets are distributed among many nodes, compute frequent itemsets at each node, then distribute the candidates from each node.
- Finally, accumulate the counts of all candidates.

Toivonen's Algorithm – (1)

- Start as in the simple sampling algorithm, but lower the threshold slightly for the sample.
  - Example: if the sample is 1% of the baskets, use s /125 as the support threshold rather than s /100.
  - Goal is to avoid missing any itemset that is frequent in the full set of baskets.

Toivonen's Algorithm – (2)

- Add to the itemsets that are frequent in the sample the negative border of these itemsets.
- An itemset is in the negative border if it is not deemed frequent in the sample, but all its immediate subsets are.

Reminder: Negative Border

- *ABCD* is in the negative border if and only if:
  1. It is not frequent in the sample, but
  2. All of *ABC*, *BCD*, *ACD*, and *ABD* are.
- *A* is in the negative border if and only if it is not frequent in the sample.
  - Because the empty set is always frequent.
    - Unless there are fewer baskets than the support threshold (silly case).

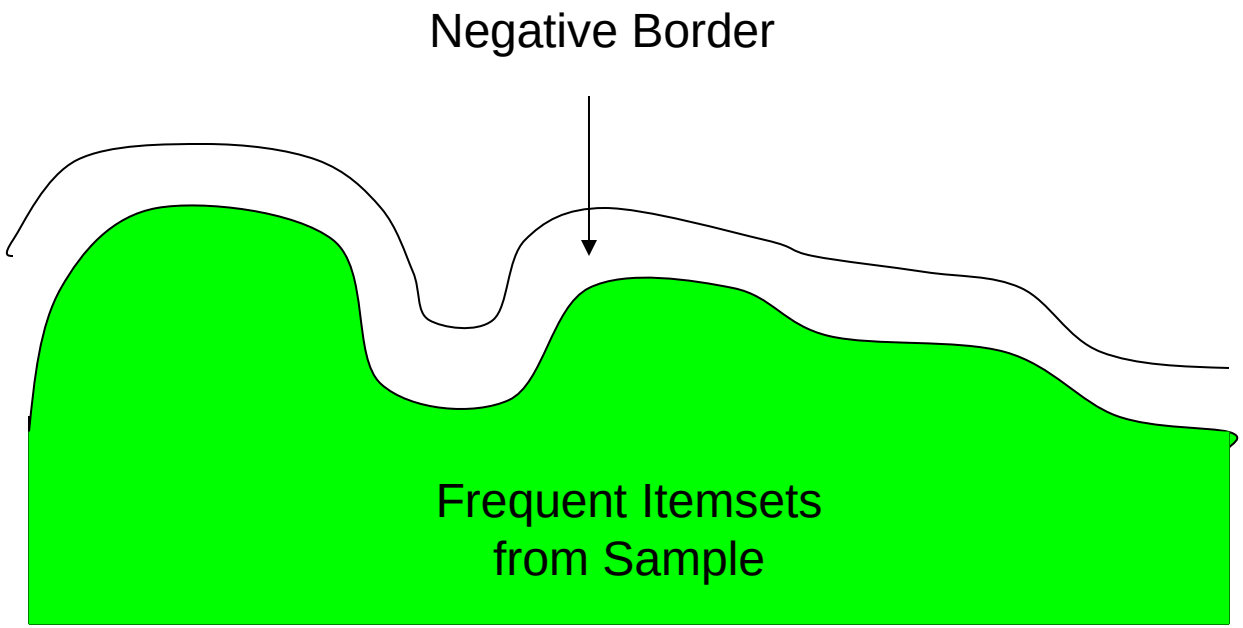
Picture of Negative Border

…

triples

pairs

singletons



Toivonen's Algorithm – (3)

- In a second pass, count all candidate frequent itemsets from the first pass, and also count their negative border.
- If no itemset from the negative border turns out to be frequent, then the candidates found to be frequent in the whole data are exactly the frequent itemsets.

Toivonen's Algorithm – (4)

- What if we find that something in the negative border is actually frequent?
  - We must start over again!
- Try to choose the support threshold so the probability of failure is low, while the number of itemsets checked on the second pass fits in main-memory.

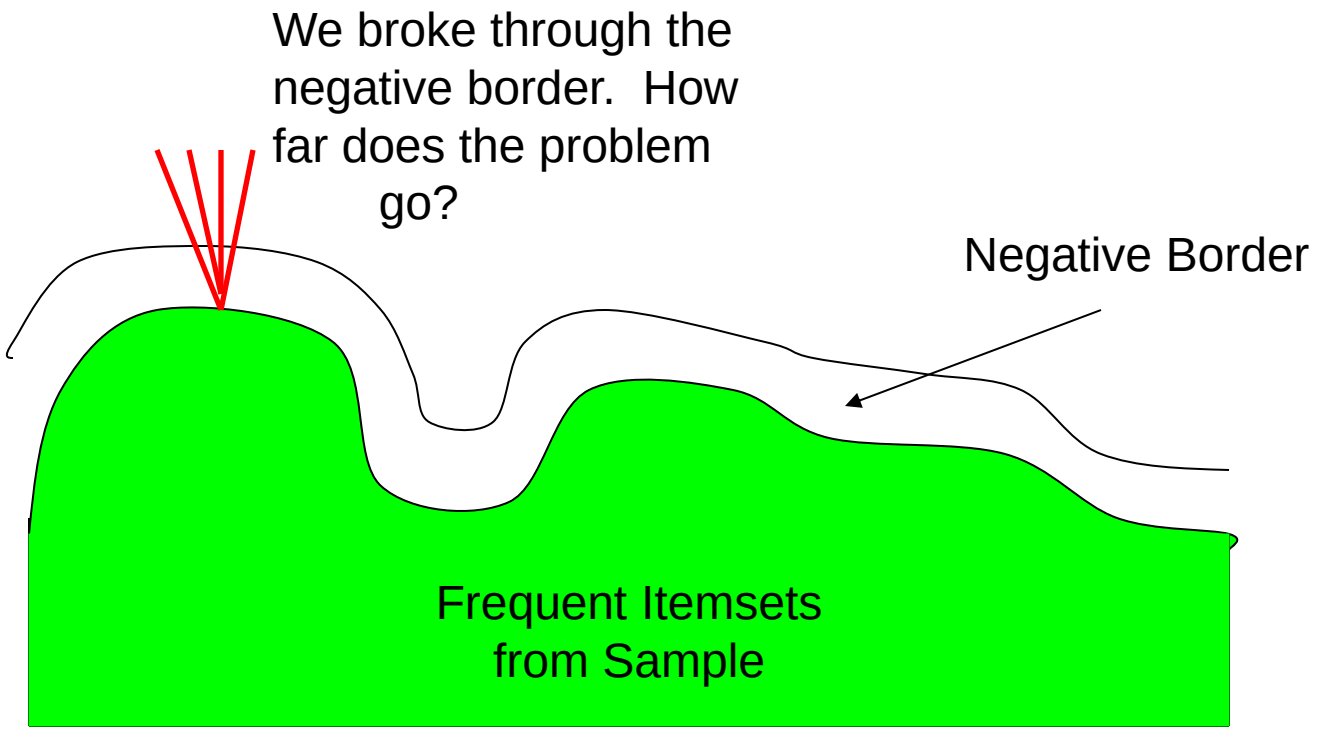
If Something in the Negative Border is Frequent . . .

…

tripletons

doubletons

singletons



Theorem:

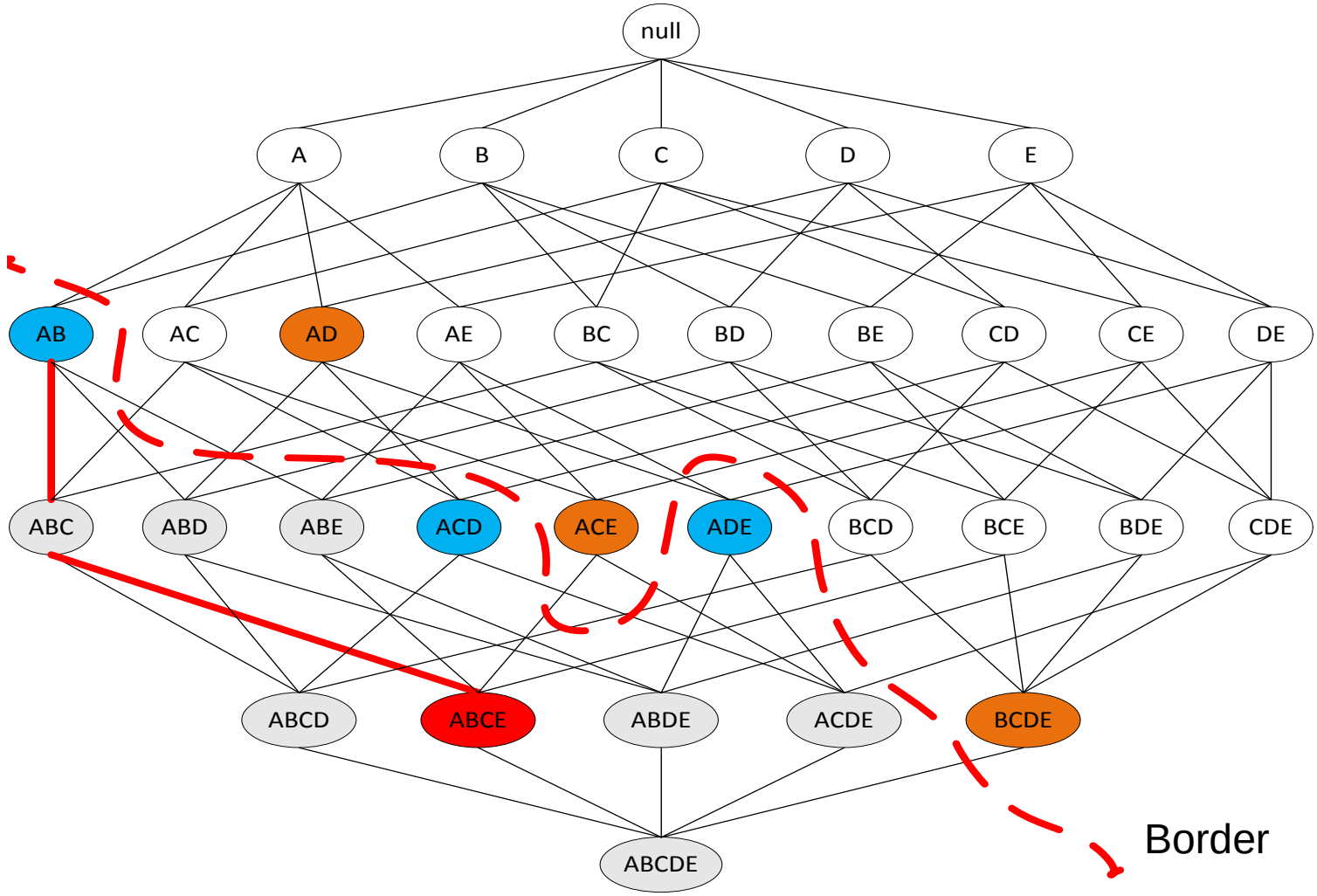
- If there is an itemset that is frequent in the whole, but not frequent in the sample, then there is a member of the negative border for the sample that is frequent in the whole.

Proof: Suppose not; i.e.;

1. There is an itemset *S* frequent in the whole but not frequent in the sample, and
2. Nothing in the negative border is frequent in the whole.

- Let *T* be a smallest subset of *S* that is not frequent in the sample.
- *T* is frequent in the whole (*S* is frequent + monotonicity).
- *T* is in the negative border (else not "smallest").

Example


null
A
B
C
D
E
AB
AC
AD
AE
BC
BD
BE
CD
CE
DE
ABC
ABD
ABE
ACD
ACE
ADE
BCD
BCE
BDE
CDE
ABCD
ABCE
ABDE
ACDE
BCDE
ABCDE
Border

FREQUENT ITEMSET RESEARCH

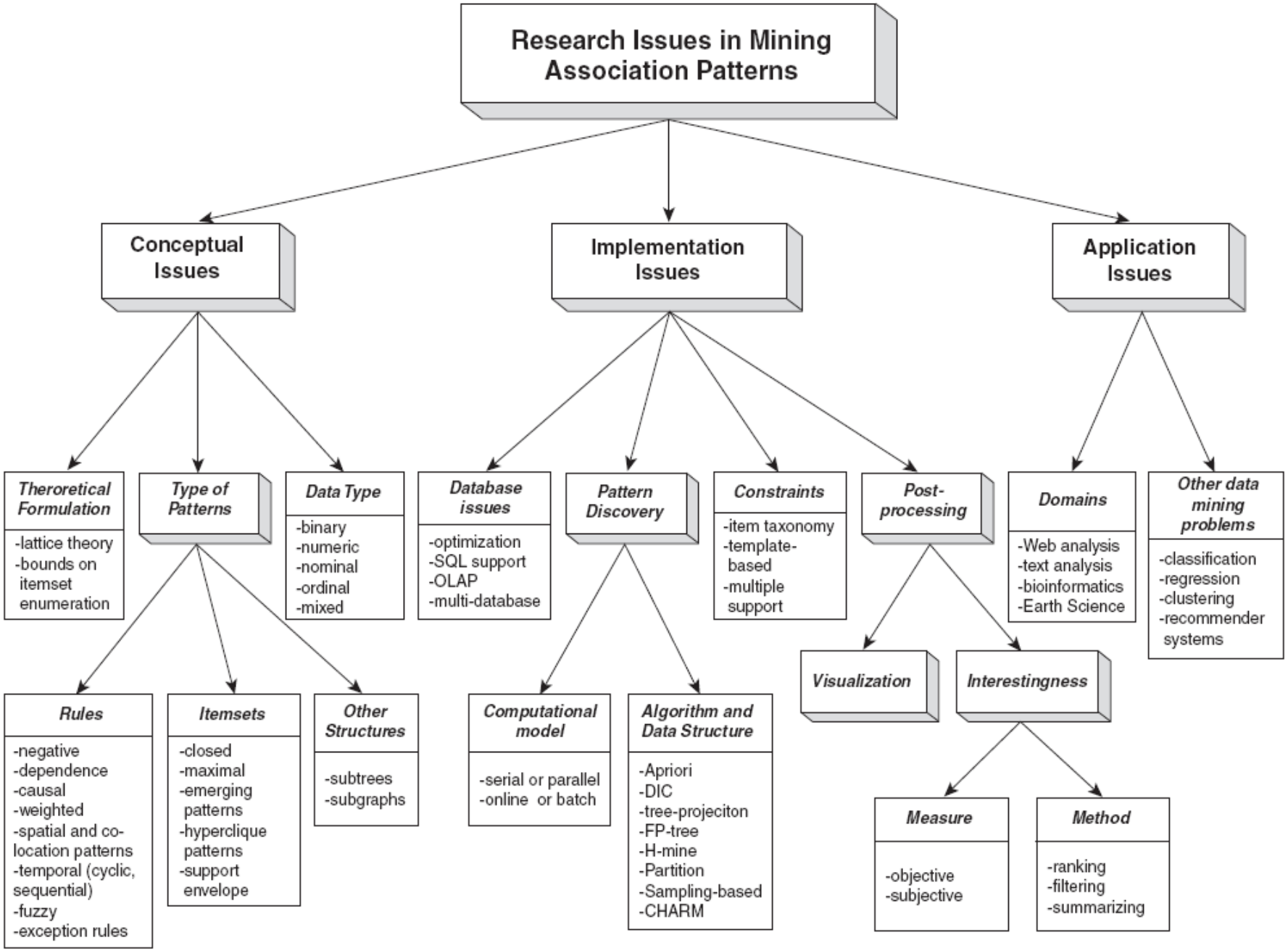


Figure 6.31. A summary of the various research activities in association analysis.