Association mining

Salvatore Orlando

Association Rule Mining

- **Given a set of transactions, find rules that will predict the occurrence of an item (a set of items) 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} → {Beer}
{Milk, Bread} → {Eggs,Coke}
{Beer, Bread} → {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)
  - Number of transaction occurrences 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 (not less than) 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 → Y, where X and Y are itemsets
  - $X \cap Y = \emptyset$
  - Example: {Milk, Diaper} → {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

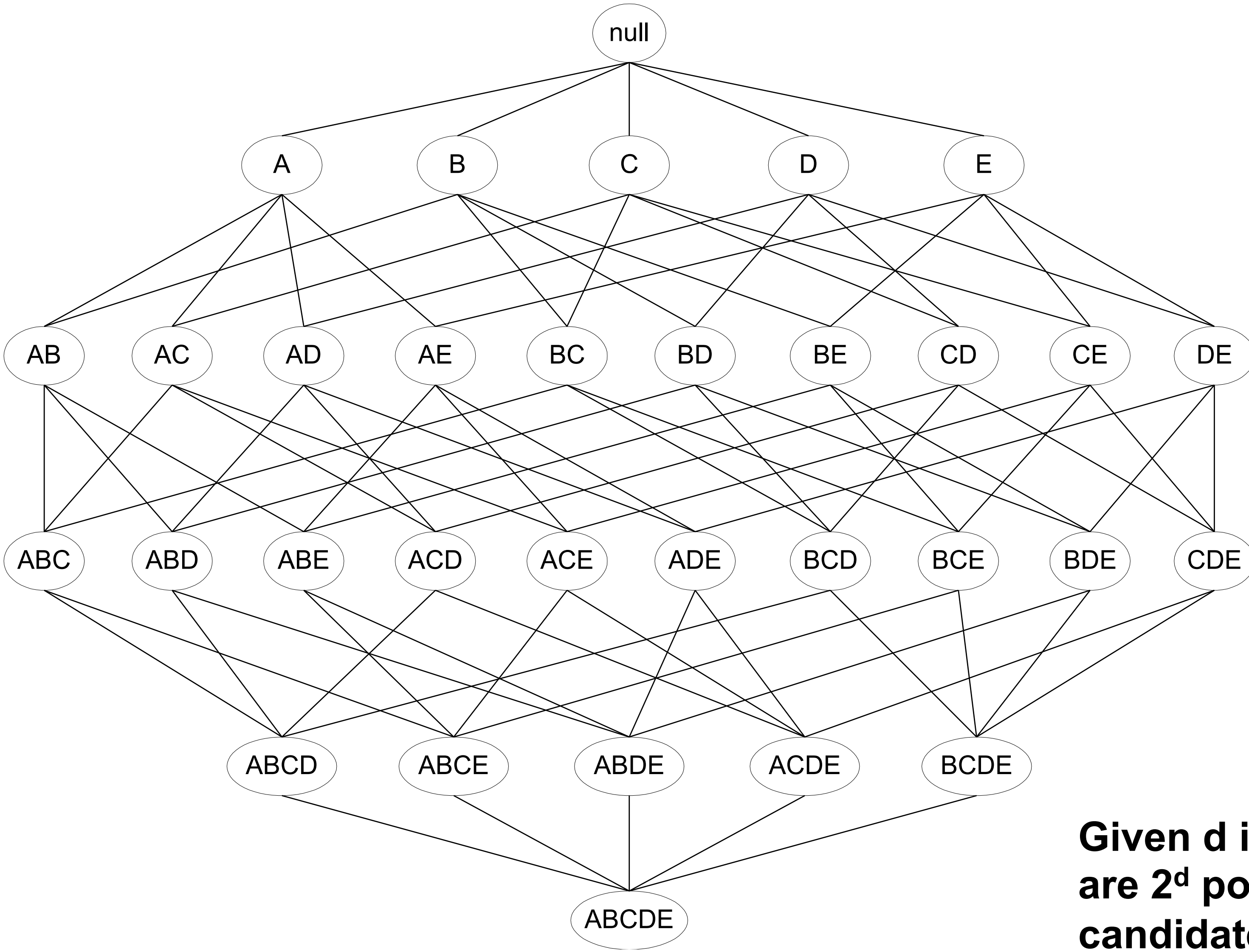
- Two-step approach:

  1. Frequent Itemset Generation
     – Generate all itemsets whose support ≥ 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

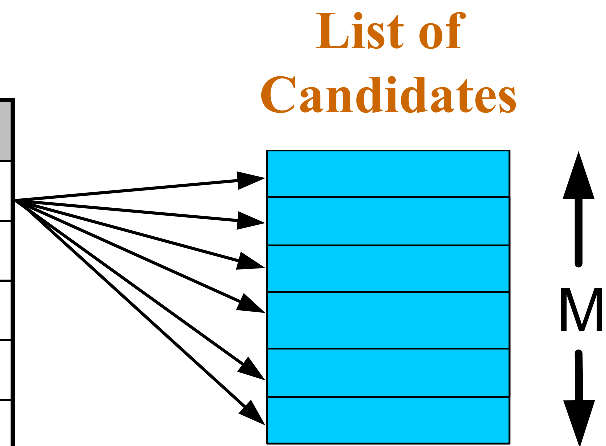
- **Brute-force approach:**
  - Each itemset in the lattice is a candidate frequent itemset
  - Count the support of each candidate by scanning the database

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

N

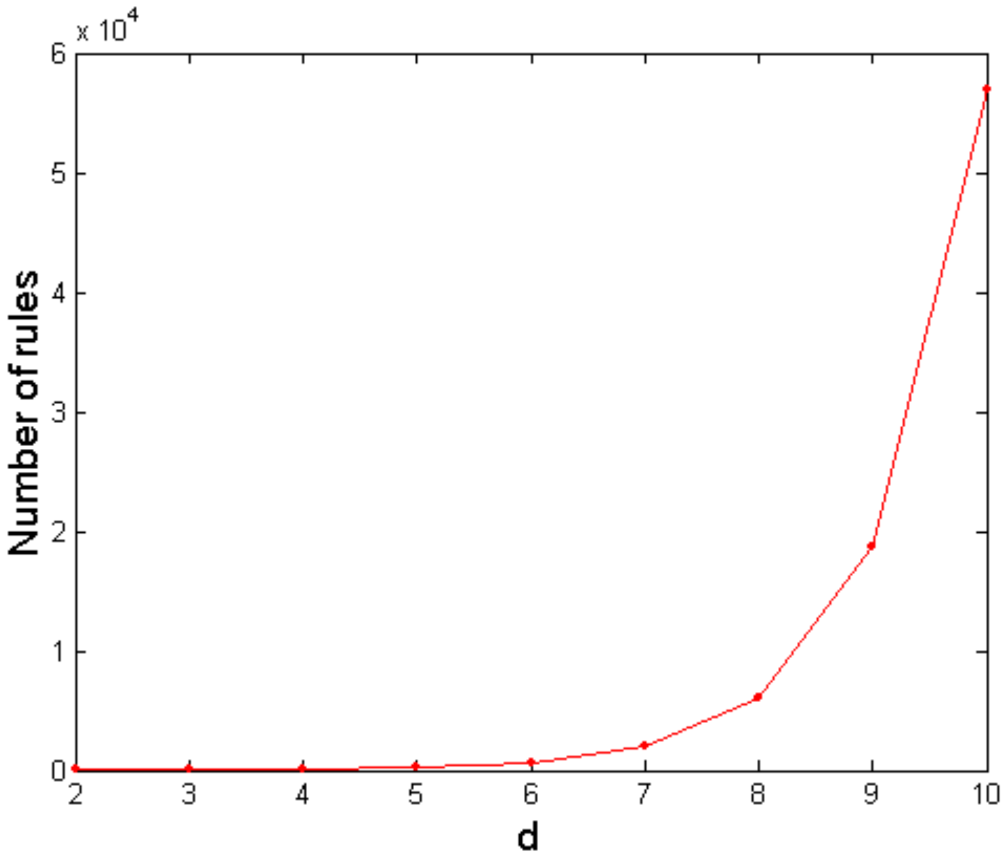
W



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

Computational Complexity

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



k is the number of items on the right hand of the rule

Select (in all the possible ways) the number j of elements occurring in the left hand of the rule

$$R = \sum_{k=1}^{d-1} \left[\binom{d}{k} \times \sum_{j=1}^{d-k} \binom{d-k}{j} \right] = 3^d - 2^{d+1} + 1$$

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

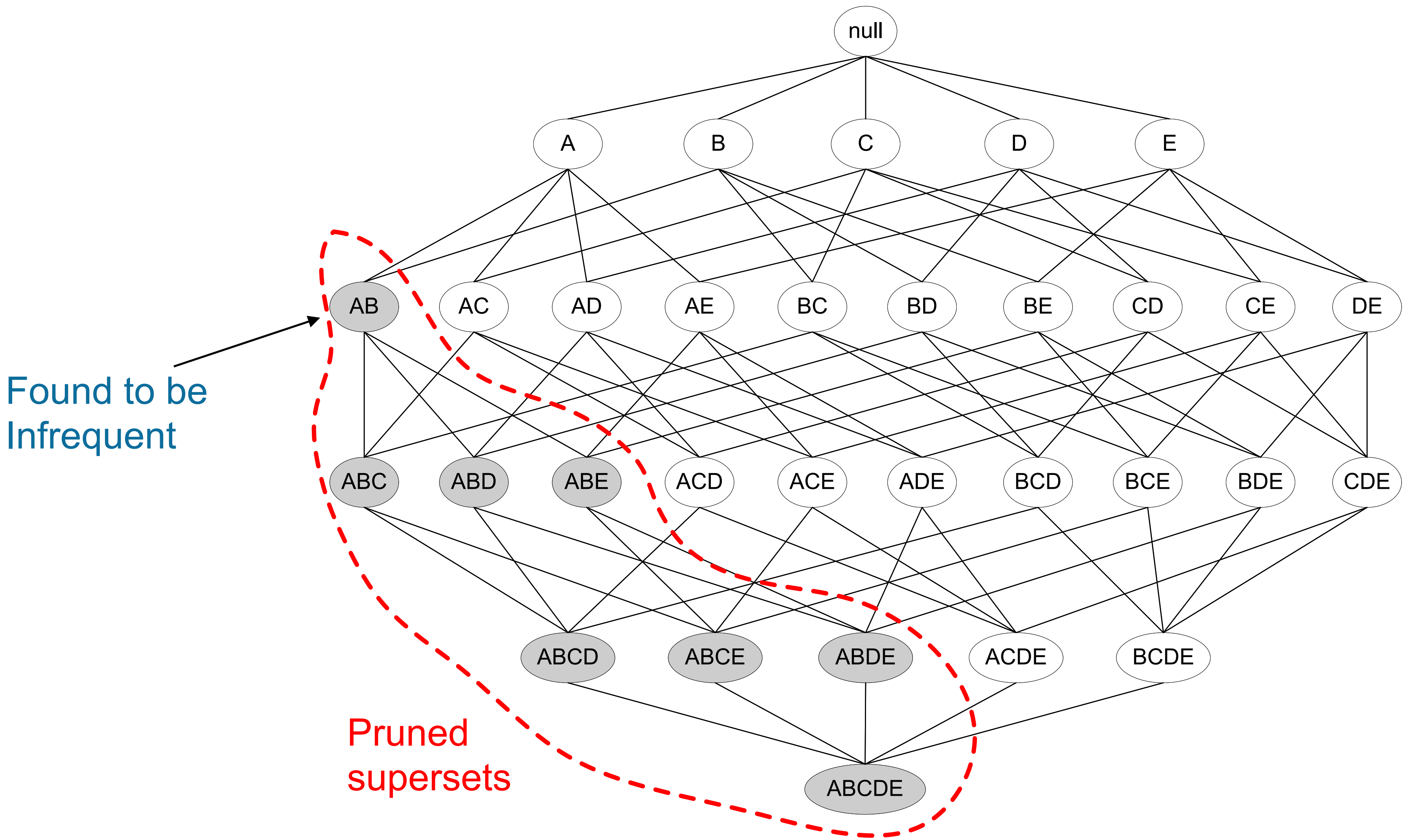
- **Apriori principle**:
  - If an itemset is frequent, then all of its subsets must also be frequent
- Apriori principle holds due to the following property of the support measure:

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

  - Support of an itemset never exceeds the support of its subsets
  - This is known as the anti-monotone property of support
- Apriori principle application for candidate pruning
  - Given a candidate itemset *Y*, if there exists *X*, where *X is a* subset of *Y*, and *X* è infrequent since *s(X) < minsup*, then also *Y* is infrequent due to the Apriori principle

minsup > s(X) ≥ s(Y)

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)

Itemset	Count
{Bread,Milk,Diaper}	**3**

If every subset is considered,

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

With support-based pruning,

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

Apriori algorithm

- C_k is the set of candidates (*k*-itemsets) at iteration *k*
  - The algorithm compute their supports
- L_k is the set of k-itemsets that result to be *frequent*
  - $L_k \subseteq C_k$
  - Along with L_k, also the associated supports are returned
  - <u>Note</u>: L stands for <u>*large*</u>. In the original paper, the frequent itemset were called “large itemset”

- **Gen Step**:
  - C_k is generated by self-joining L_{k-1}, by keeping only the itemsets of length k
  - **Pruning of C_k**: A *k-itemset* cannot be frequent, and thus cannot be a candidate of C_k, if it includes at least a subset that is not frequent. So, it is reasonable <u>start from</u> L_{k-1} to generate C_k

Gen Step

- **Suppose that**
  - **Each itemset is an ordered list of items**
  - **If the itemsets in L_{k-1} are sorted according to a lexicographic order, this simplify the self-join step**

- **Step 1: self-joining L_{k-1}**

  insert into C_k

  all pairs $(p, q) \in L_{k-1}$

  where $p.item_1 = q.item_1$, ……, $p.item_{k-2} = q.item_{k-2}$, $p.item_{k-1} < q.item_{k-1}$

  (p and q share a **common prefix of length k-2**)

  (*the condition* $p.item_{k-1} < q.item_{k-1}$ guarantees that no duplicates are generated)

- **Step 2: pruning**

  forall *itemsets c in* C_k **do**

  forall ***(k-1)-subsets*** *s of c* **do**

  if (*s is not in* L_{k-1}) **then delete** *c* **from** C_k

From this check, we can omit the pair of generators *(p, q)*, **which are surely included in L_{k-1}**

Example of candidate generation

- L_3={*abc, abd, acd, ace, bcd*}

- Self-joining: $L_3 * L_3$
  - *abcd* from p=*abc* and q=*abd*
  - *acde* from p=*acd* and q=*ace*
- Pruning:
  - *acde* is then pruned because *ade* is not included in L_3

- C_4={*abcd*}

Apriori algorithm

- **Pseudo-code:**

  C_k: **Candidate itemsets of size k**
  L_k : **frequent itemsets of size k**

  L_1 **= {frequent items};**
  for **(*k* = 1; L_k !=∅; *k*++)** do begin
  C_{k+1} **= candidates generated from L_k;**
  for each **transaction *t* in database *D* do**
  increment the count of all candidates in C_{k+1}
  that are contained in ***t***
  L_{k+1} **= candidates in C_{k+1} with are frquent**
  end
  return $\cup_k L_k$

Apriori: another example (*minsup* = 2)

Database D

TID	Items
100	1 3 4
200	2 3 5
300	1 2 3 5
400	2 5

Scan D → C_1

itemset	sup.
{1}	2
{2}	3
{3}	3
{4}	1
{5}	3

→ L_1

itemset	sup.
{1}	2
{2}	3
{3}	3
{5}	3

C_2

itemset
{1 2}
{1 3}
{1 5}
{2 3}
{2 5}
{3 5}

Scan D → C_2

itemset	sup
{1 2}	1
{1 3}	2
{1 5}	1
{2 3}	2
{2 5}	3
{3 5}	2

→ L_2

itemset	sup
{1 3}	2
{2 3}	2
{2 5}	3
{3 5}	2

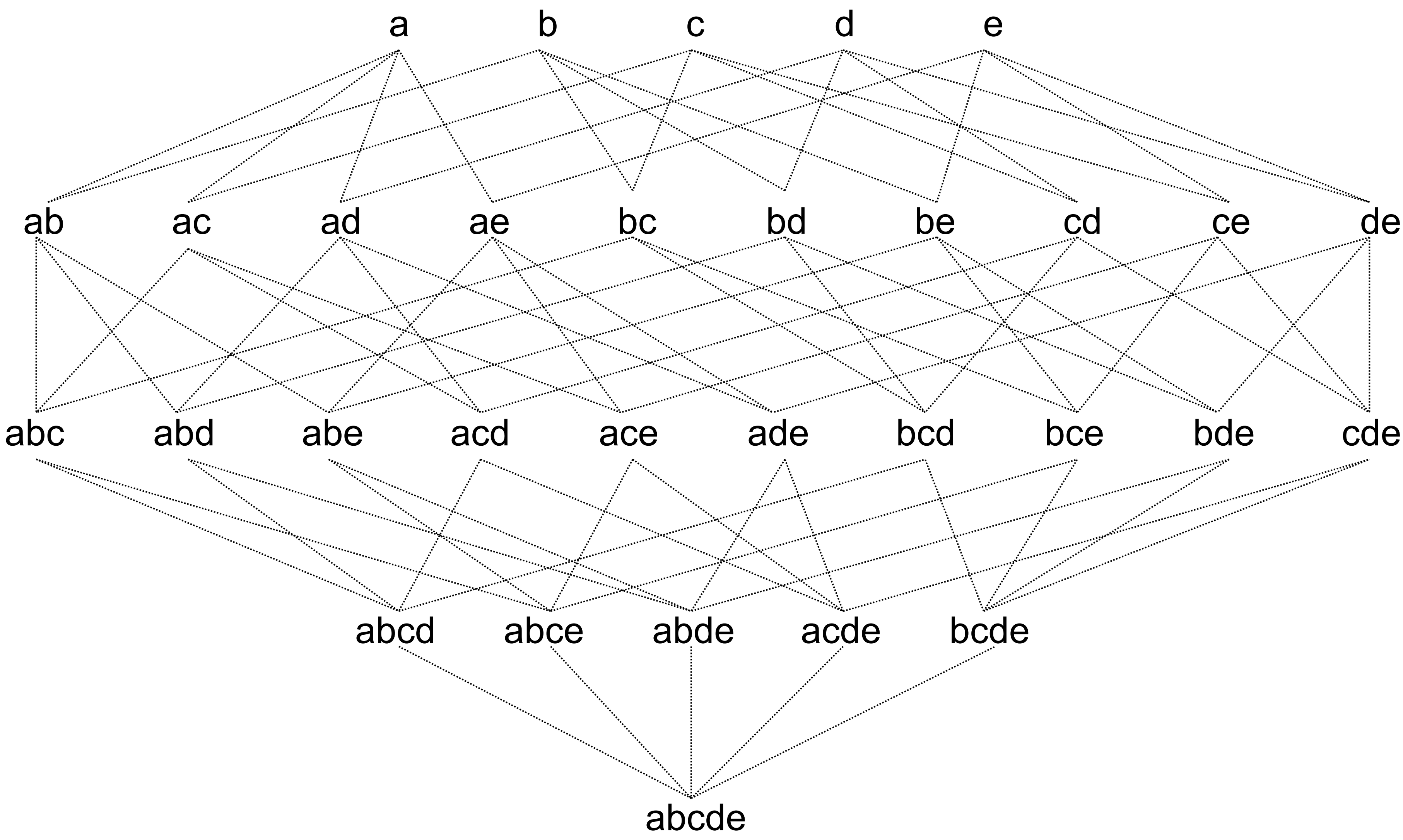
C_3

itemset
{2 3 5}

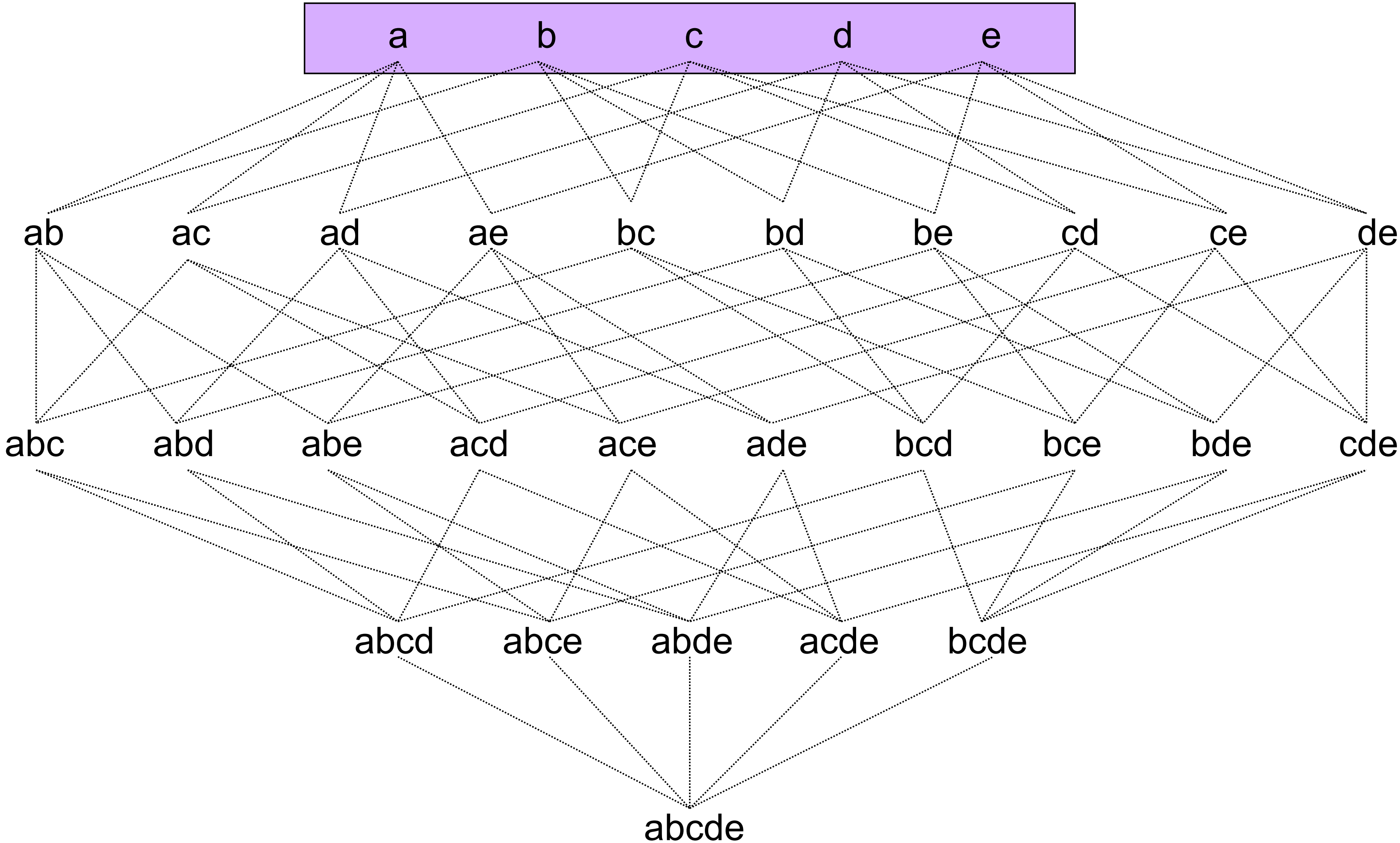
Scan D → L_3

itemset	sup
{2 3 5}	2

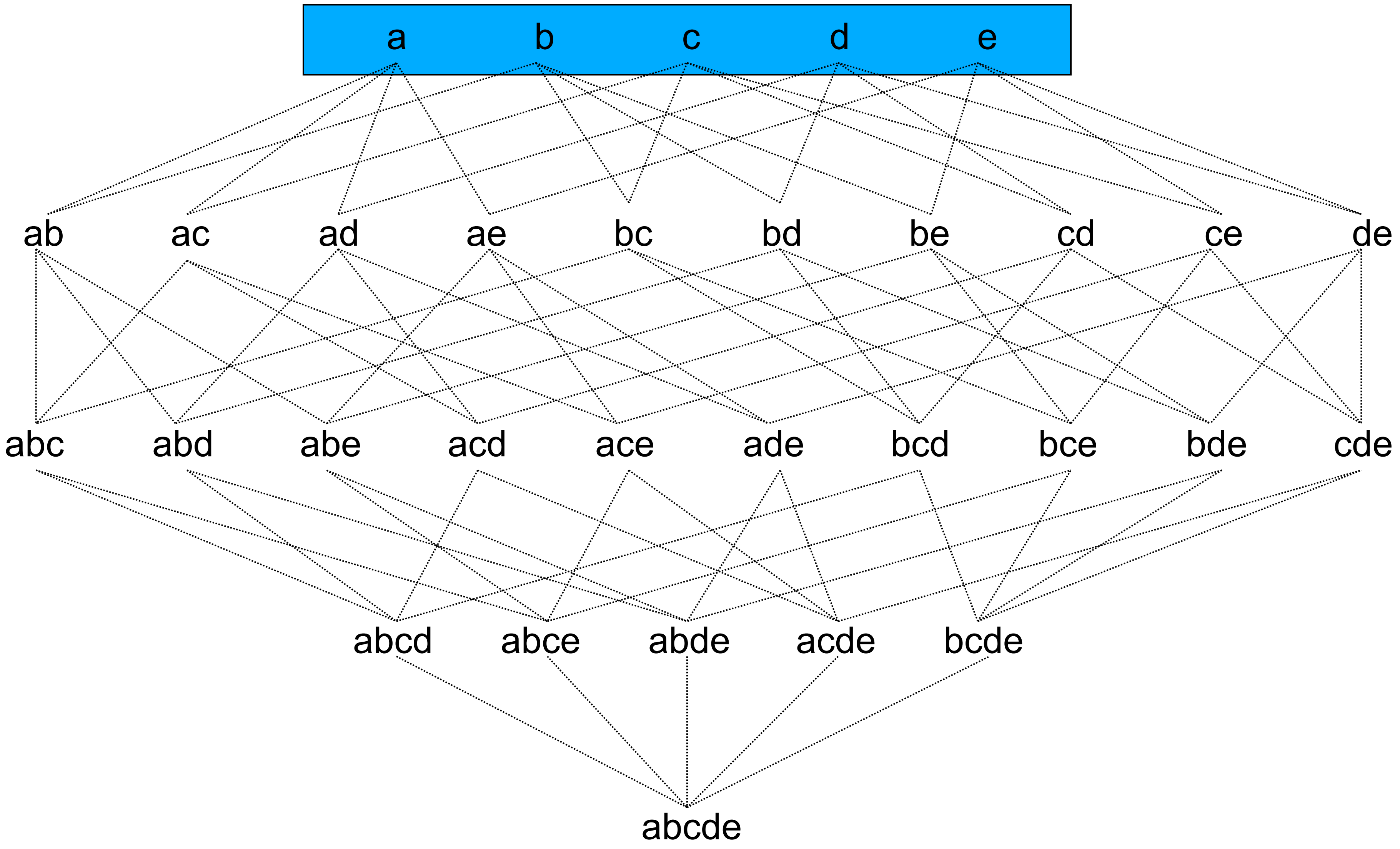
Apriori: *Breadth first* visit of the lattice



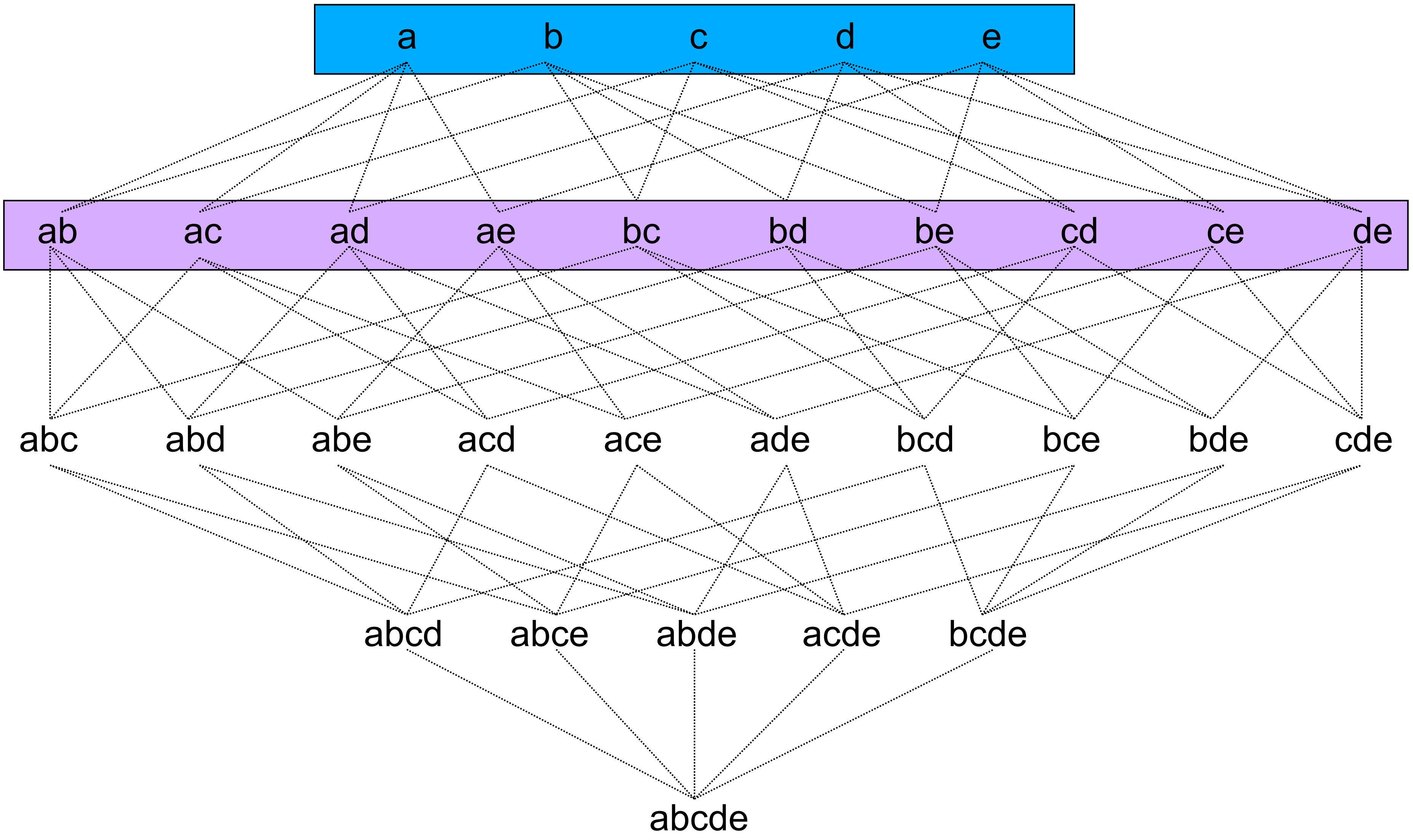
Generate the candidates of dimension 1



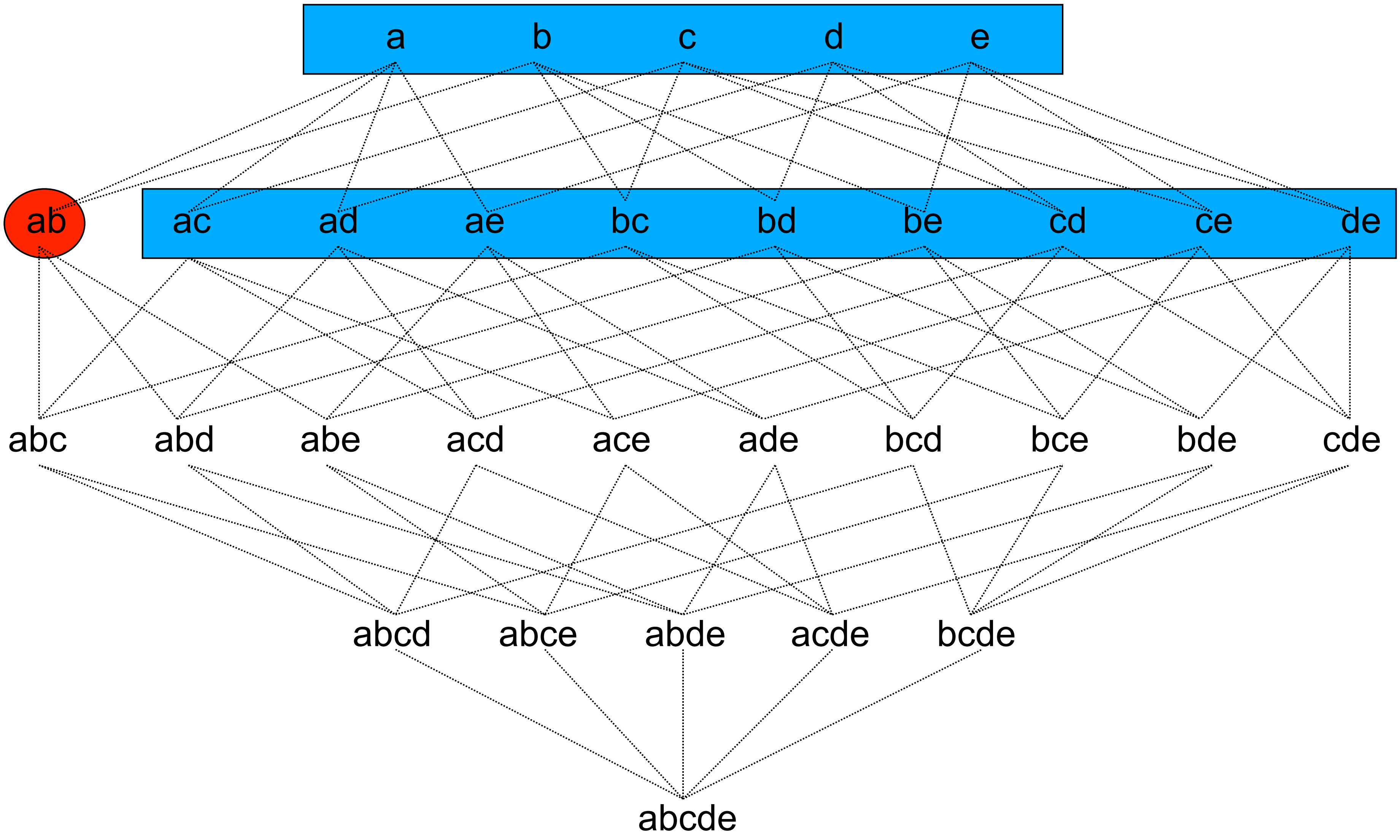
Compute the supports of the Candidates of dim. 1



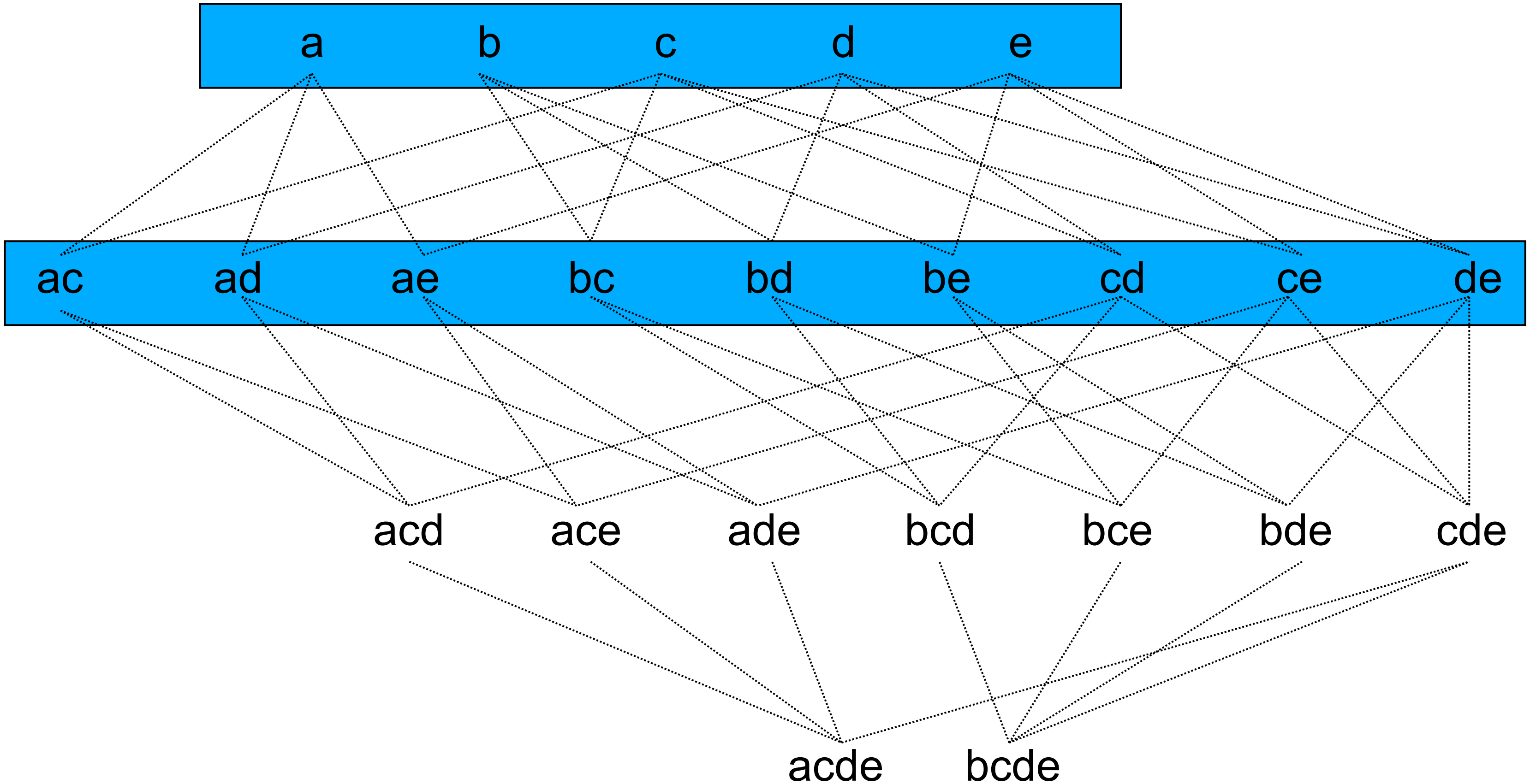
Generate the candidates of dimension 2



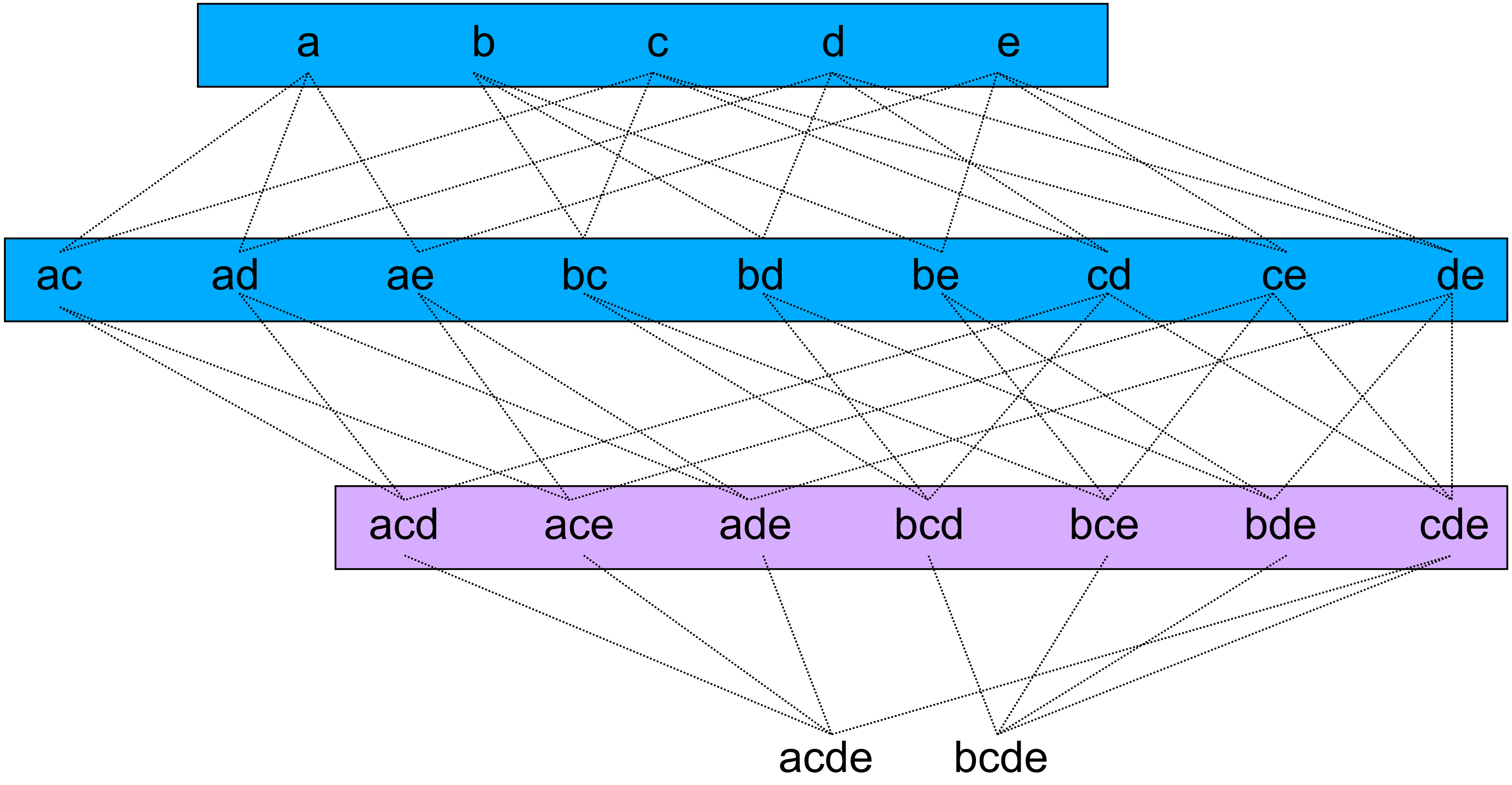
Compute the supports of the Candidates of dim. 2



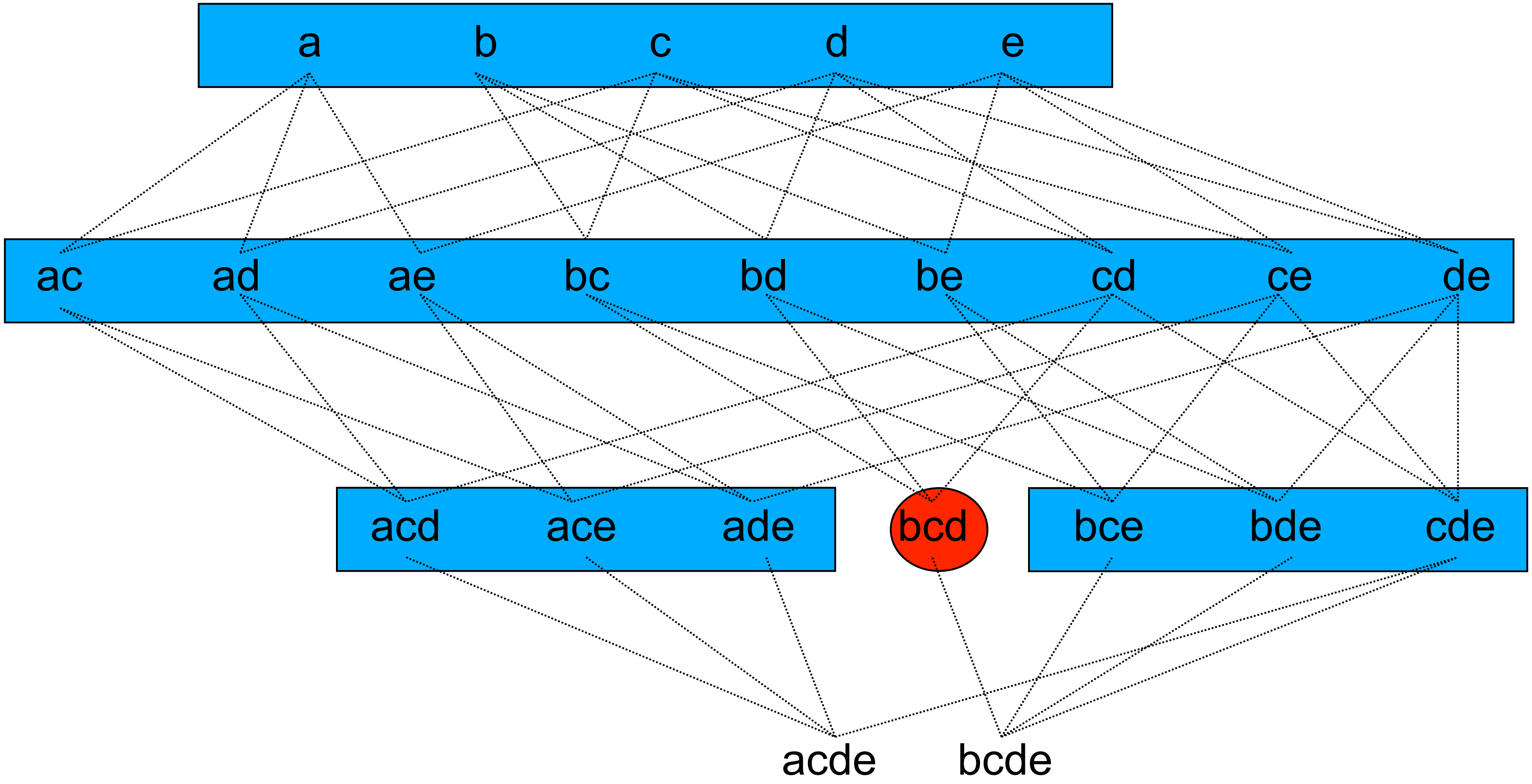
Prune the infrequent itemsets



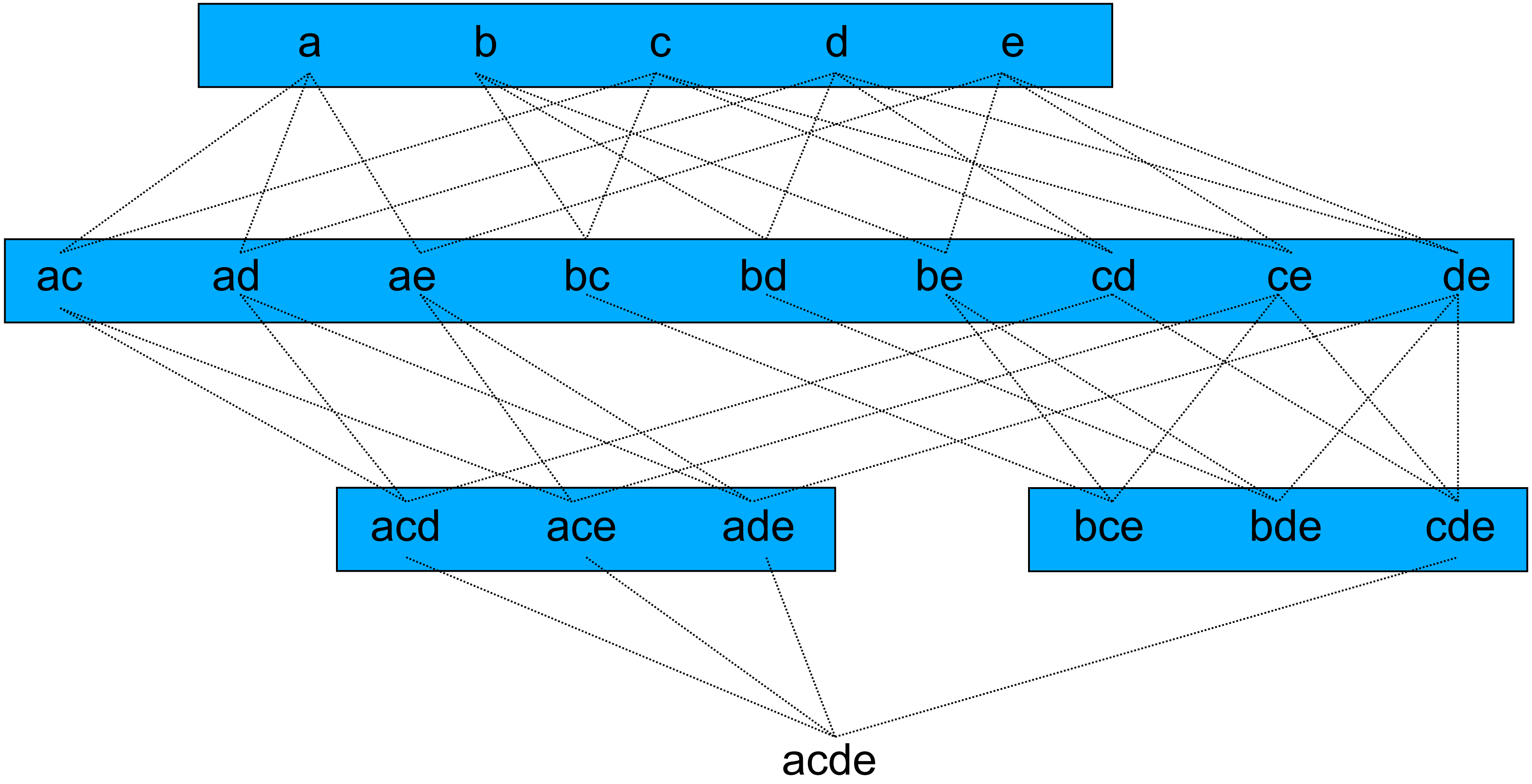
Generate the candidates of dimension 3



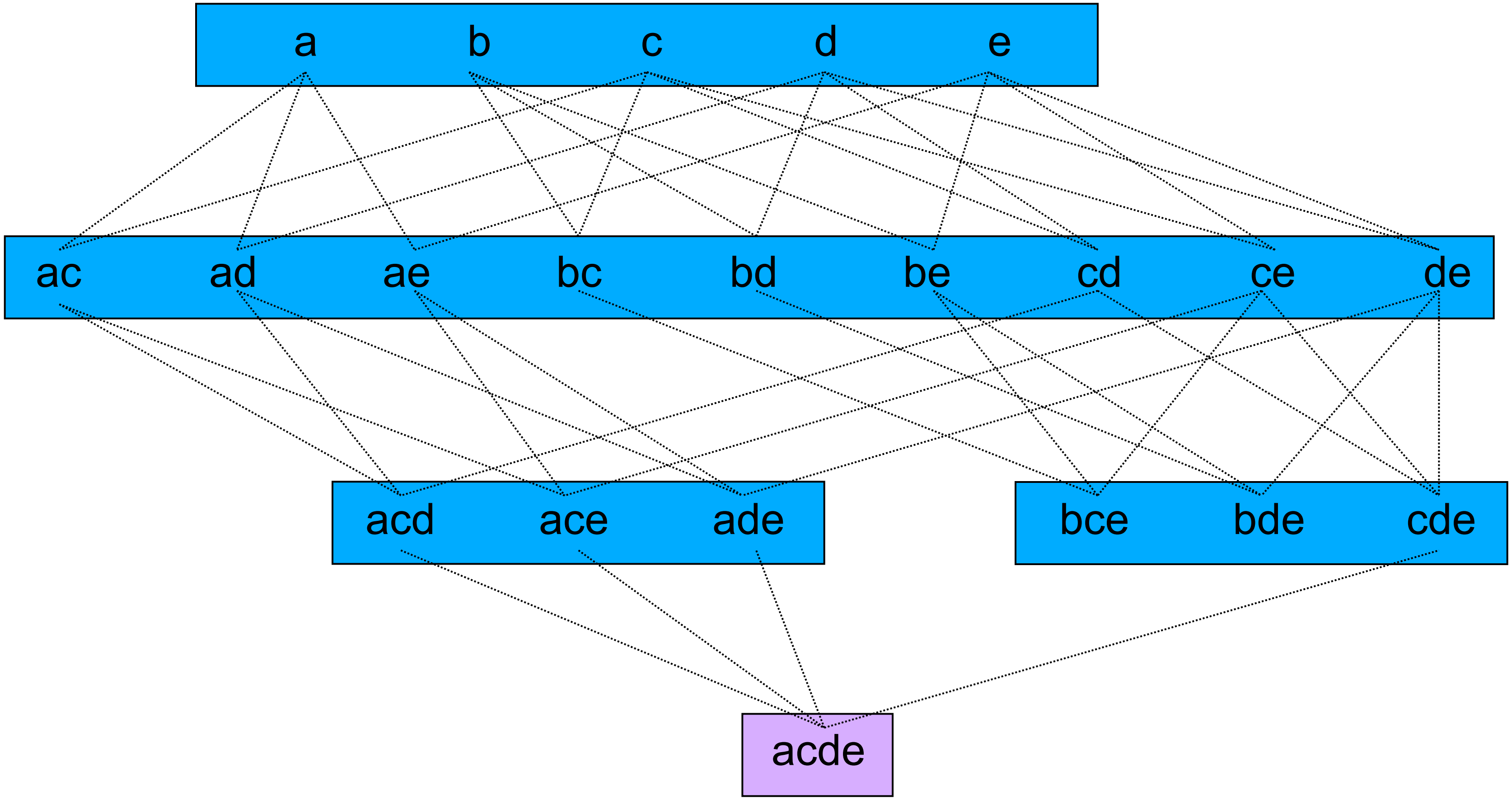
Compute the supports of the Candidates of dim. 3



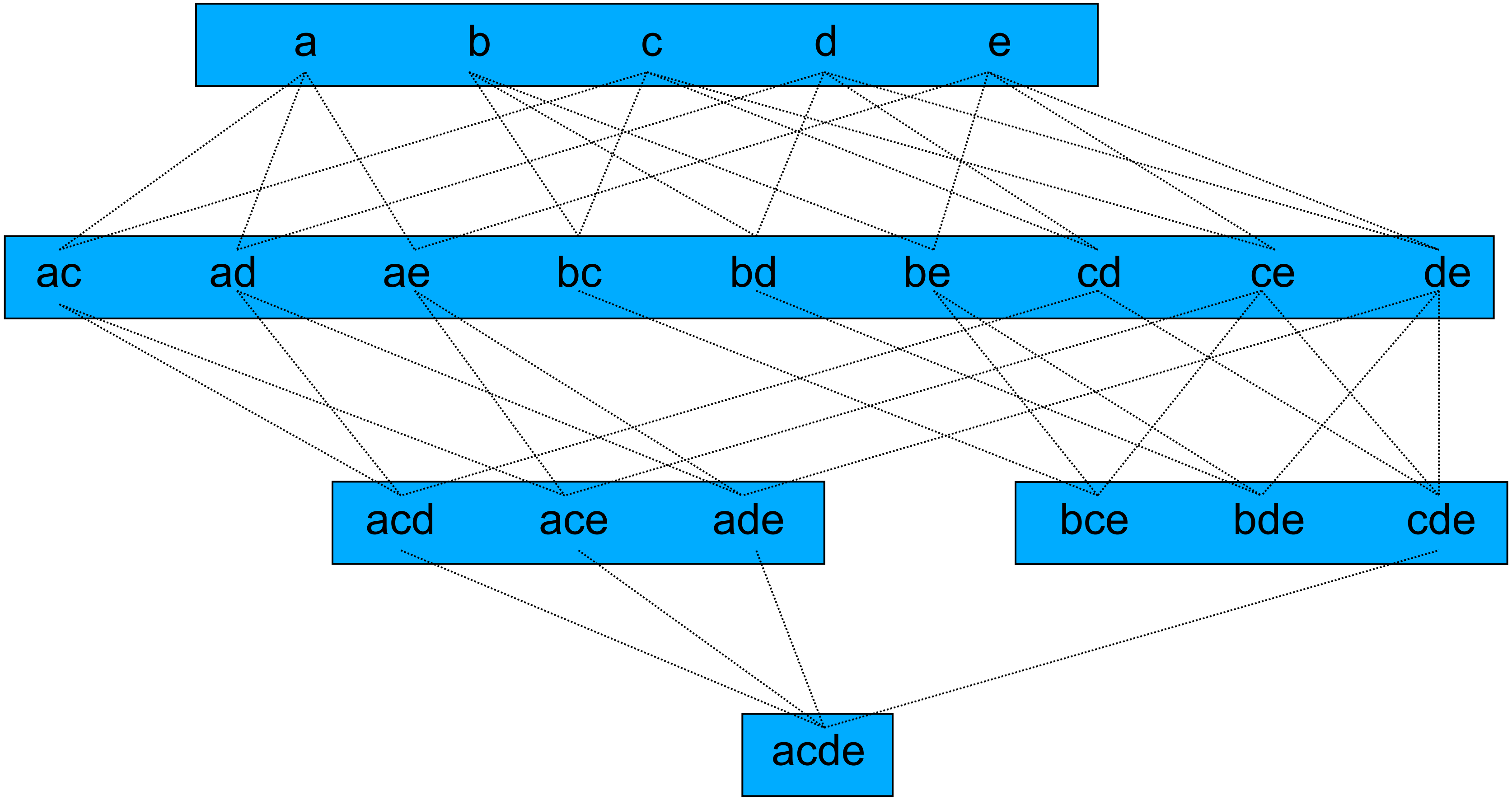
Prune the infrequent itemsets



Generate the candidates of dimension 3



Compute the supports of the Candidates of dim. 4



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

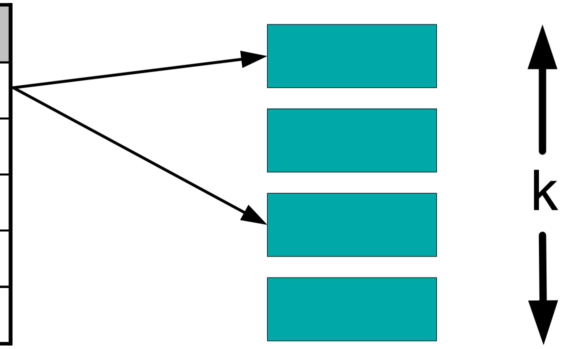
    Instead of matching each transaction against every candidate, match it against candidates contained in the hashed buckets

Transactions

N

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

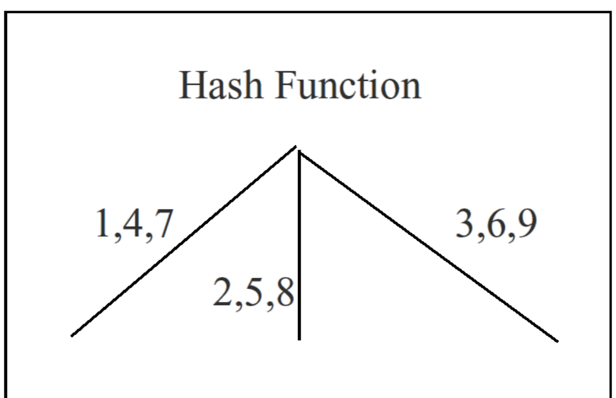
Hash Structure

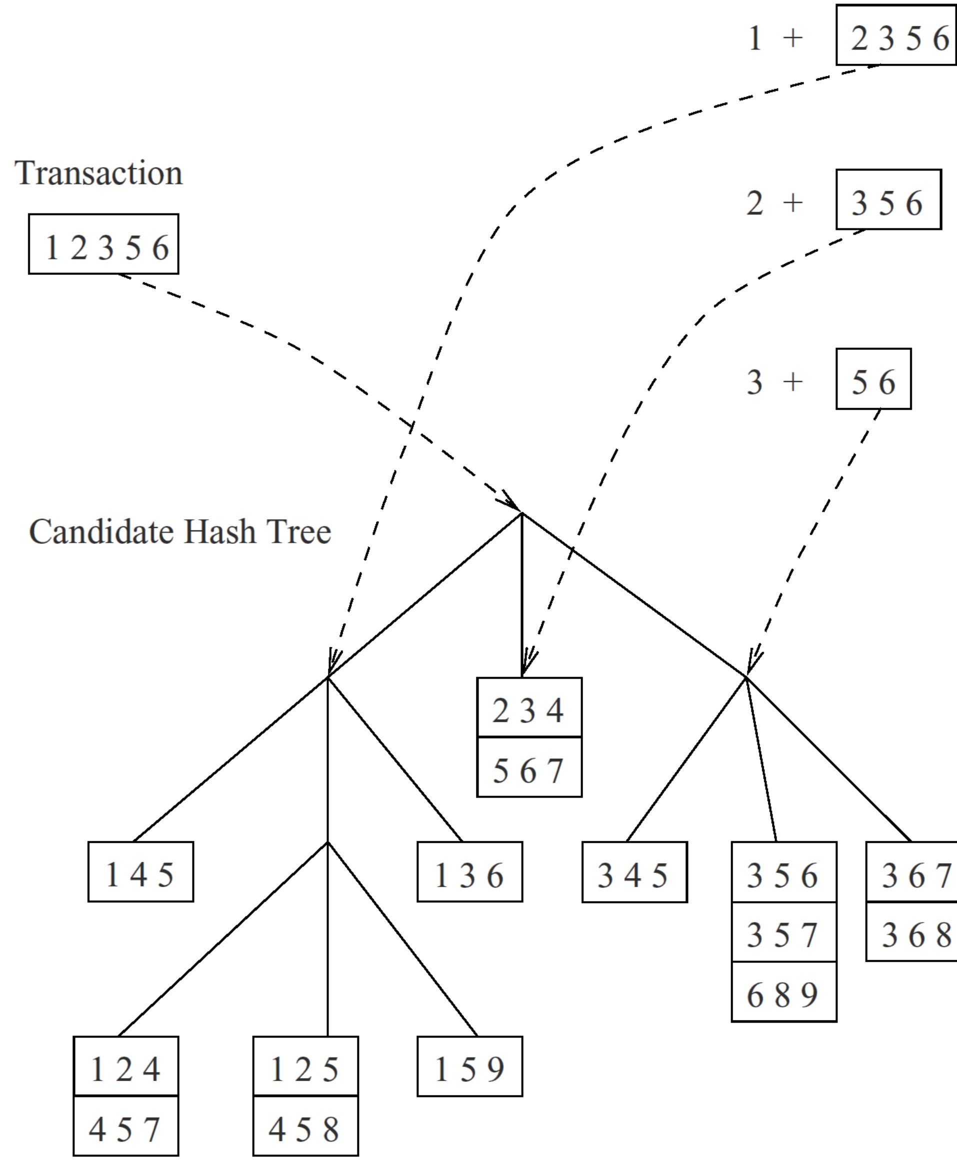


Buckets

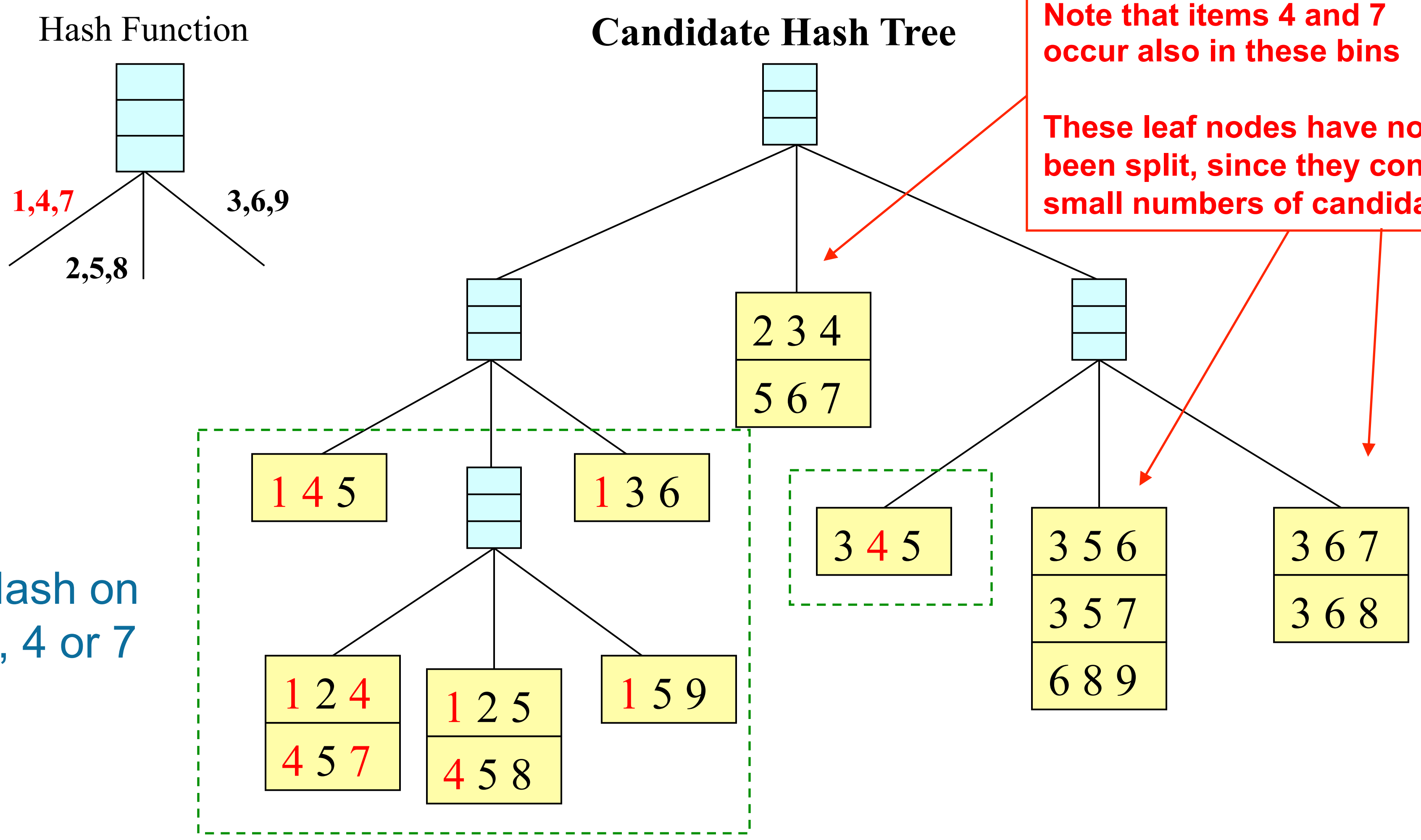
Generate Hash Tree

- **Hash-tree to store candidates in C_k of max depth k:**
  - **The selection of the path is done with a hash function over the items to select the path**
  - **Each leaf stores a list of candidates**
  - **<u>Max leaf size</u>: max number of itemsets stored in a leaf node (if number of candidate itemsets exceeds max leaf size, split the node)**





Generate Hash Tree

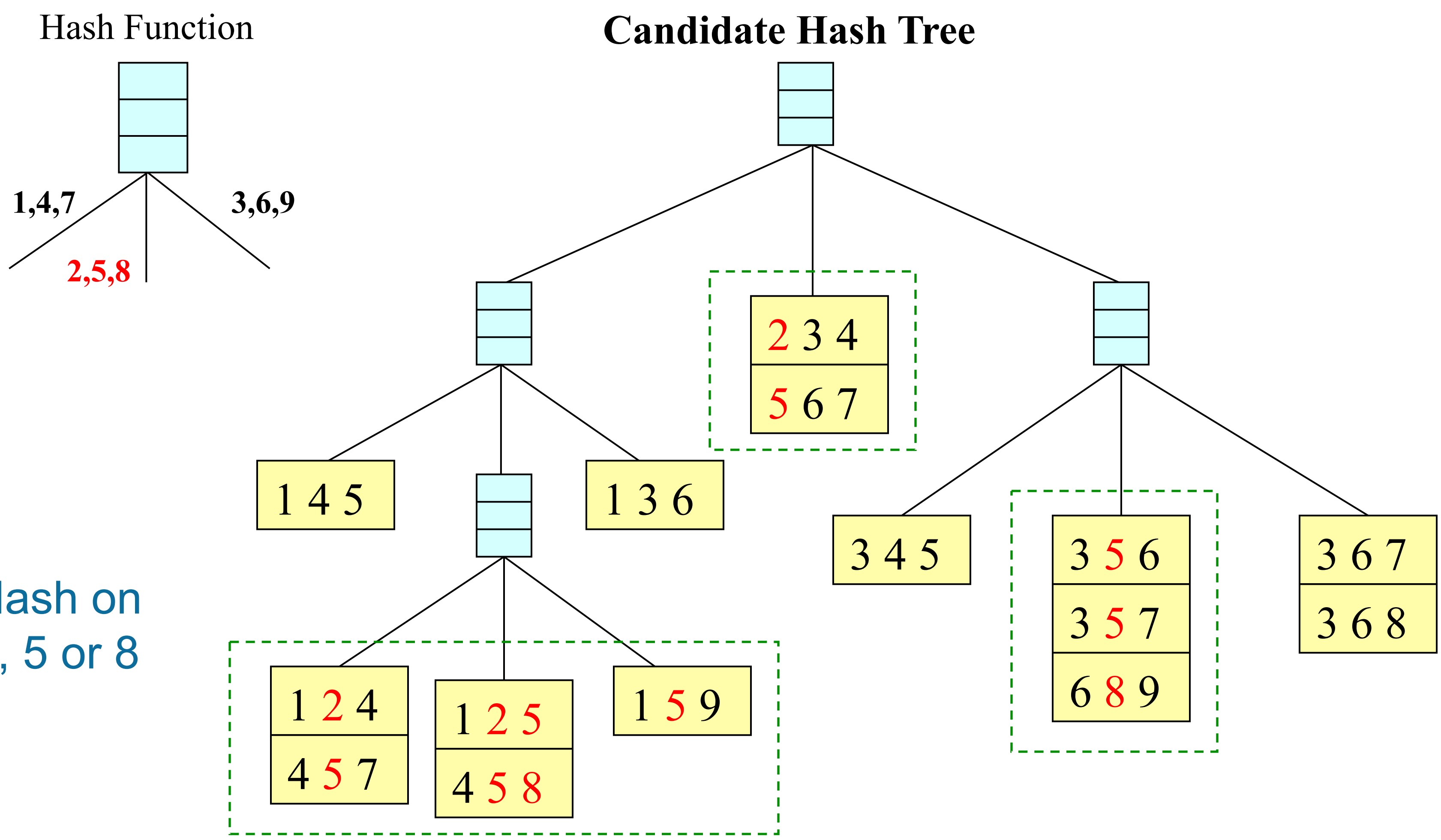


Note that items 4 and 7 occur also in these bins

These leaf nodes have not been split, since they contain small numbers of candidates

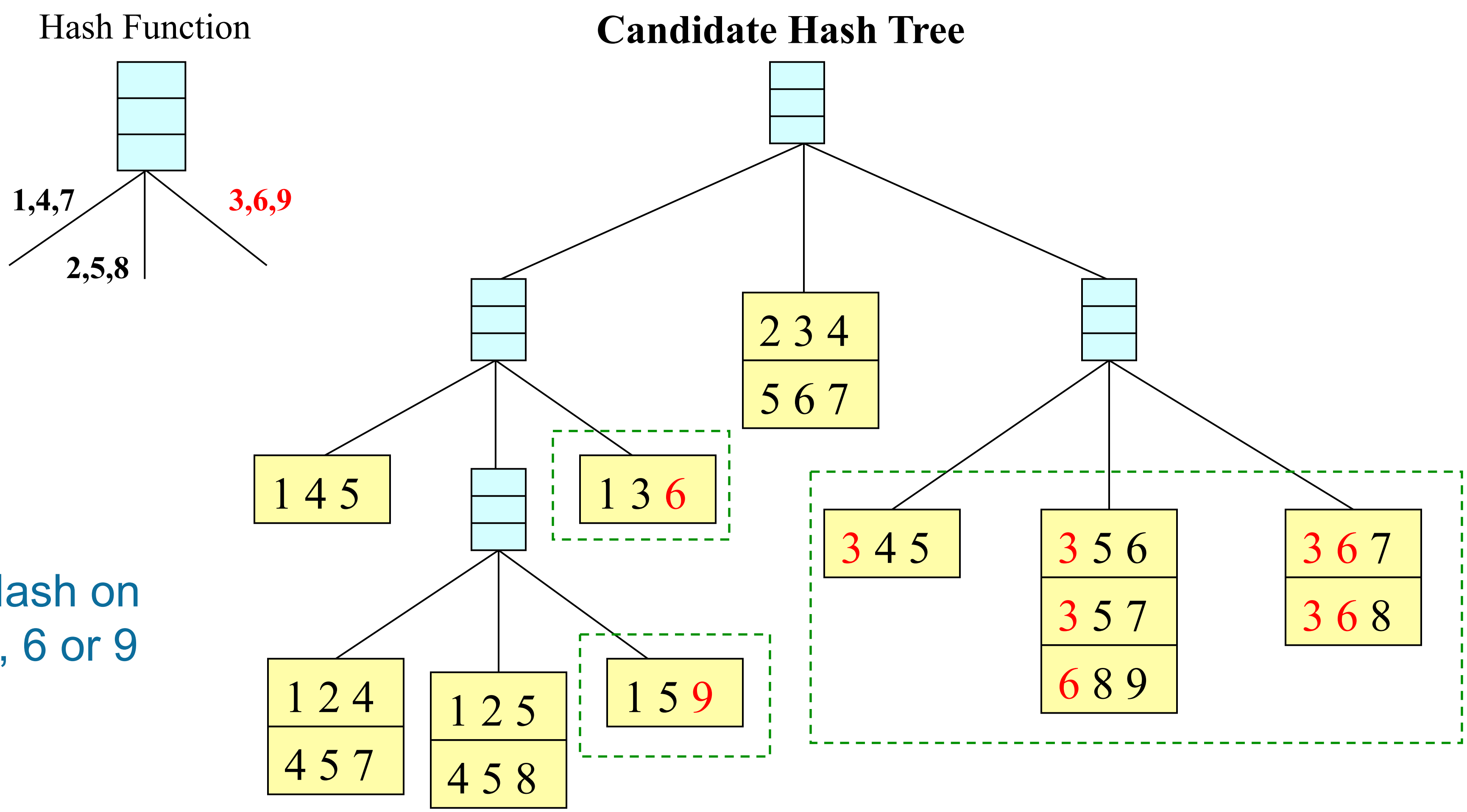
Hash on 1, 4 or 7

Generate Hash Tree



Hash on
2, 5 or 8

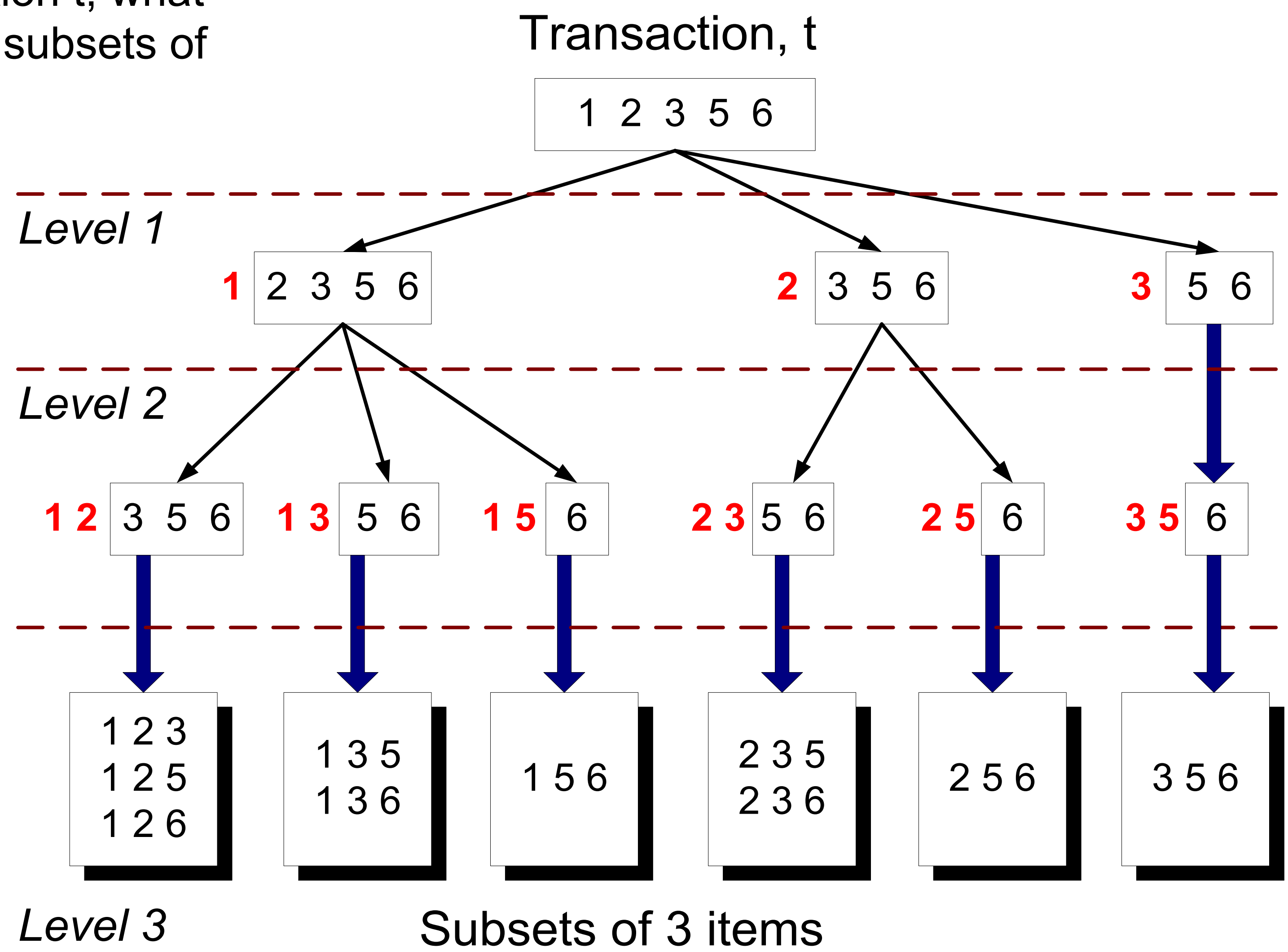
Generate Hash Tree



Hash on
3, 6 or 9

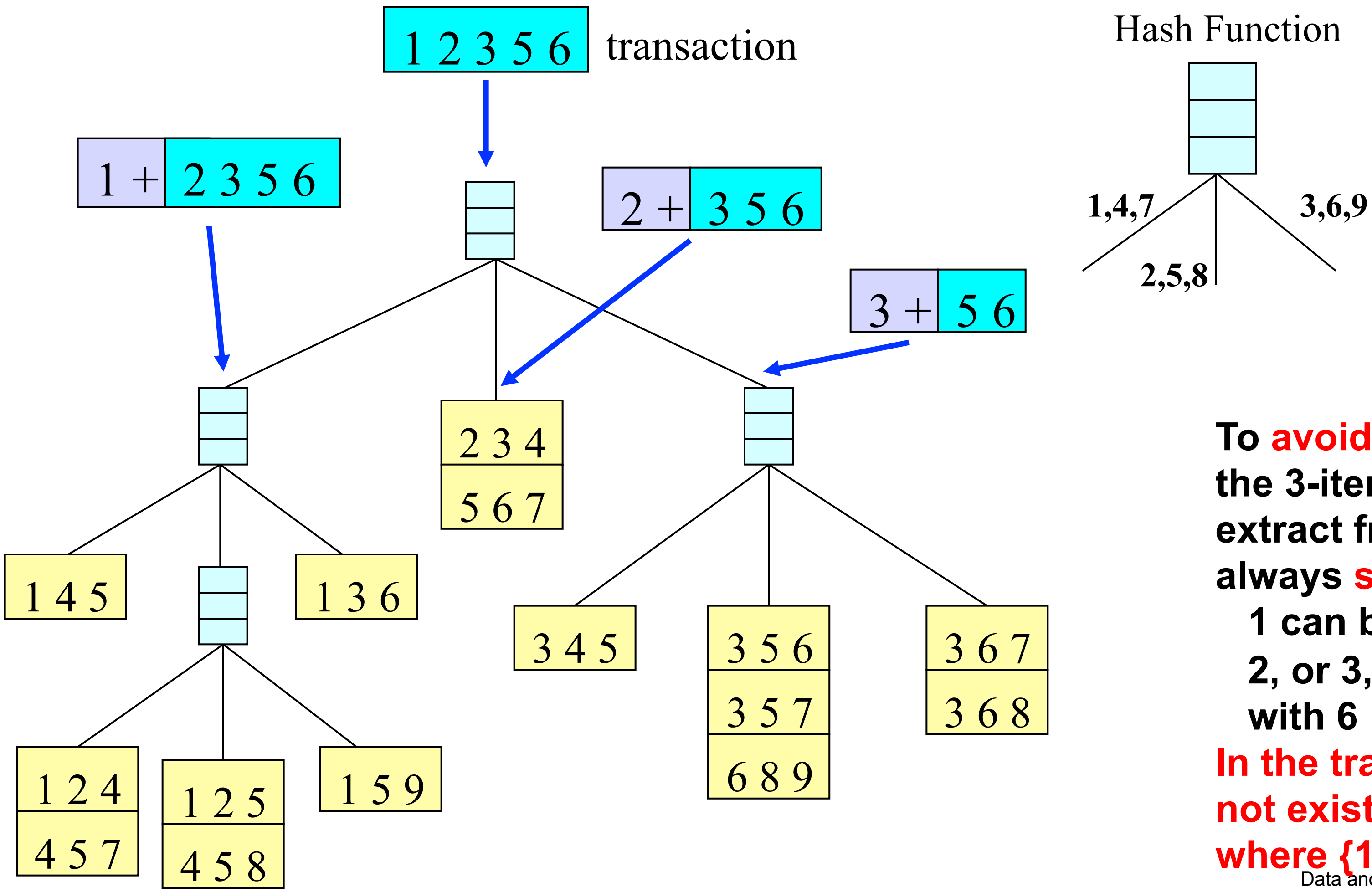
Subset Operation Using Hash Tree

Given a transaction t, what are the possible subsets of size 3?



Subset Operation Using Hash Tree

- **Recursive transaction subsetting**

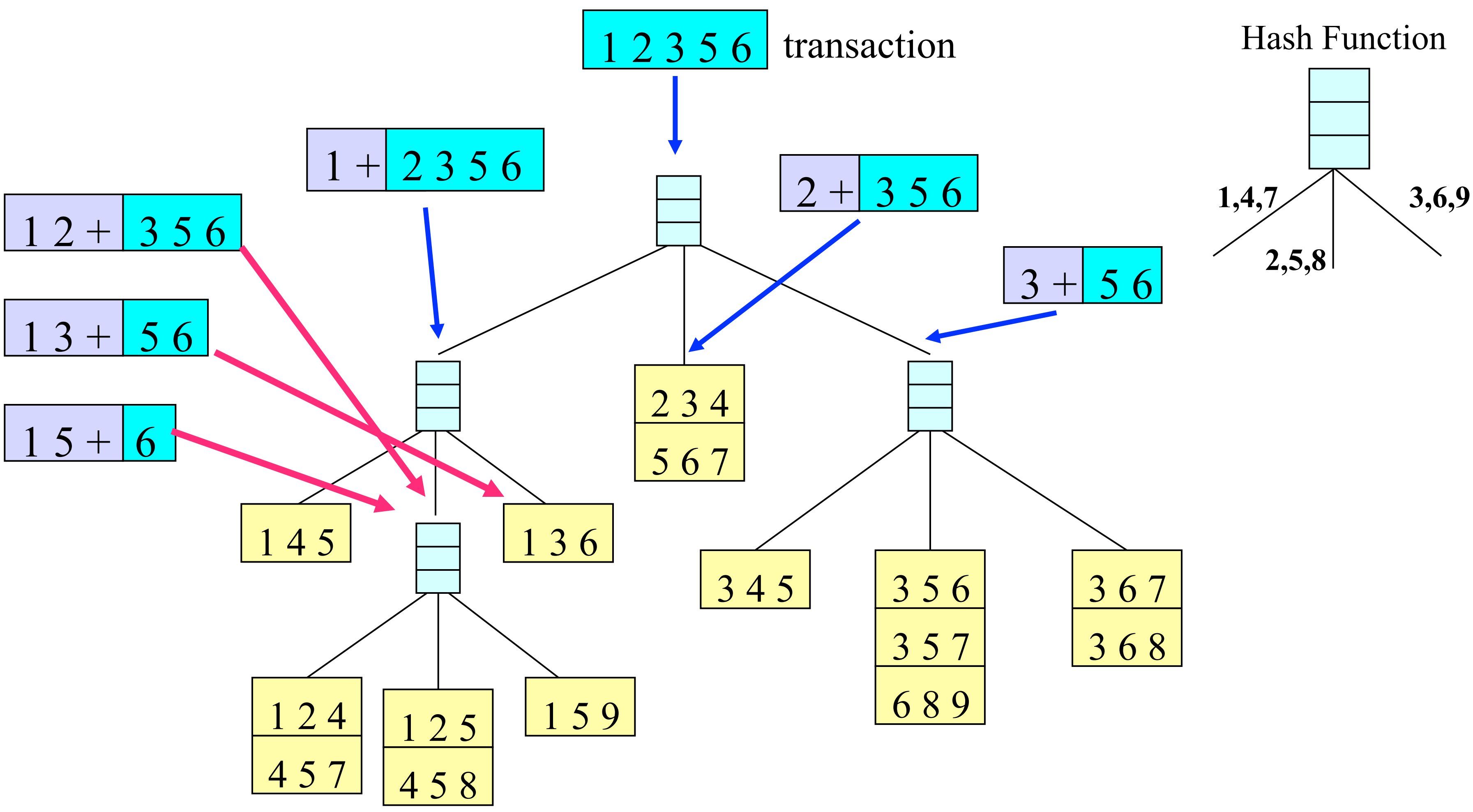


To avoid duplicates, the 3-itemsets we can extract from 123456 are always sorted

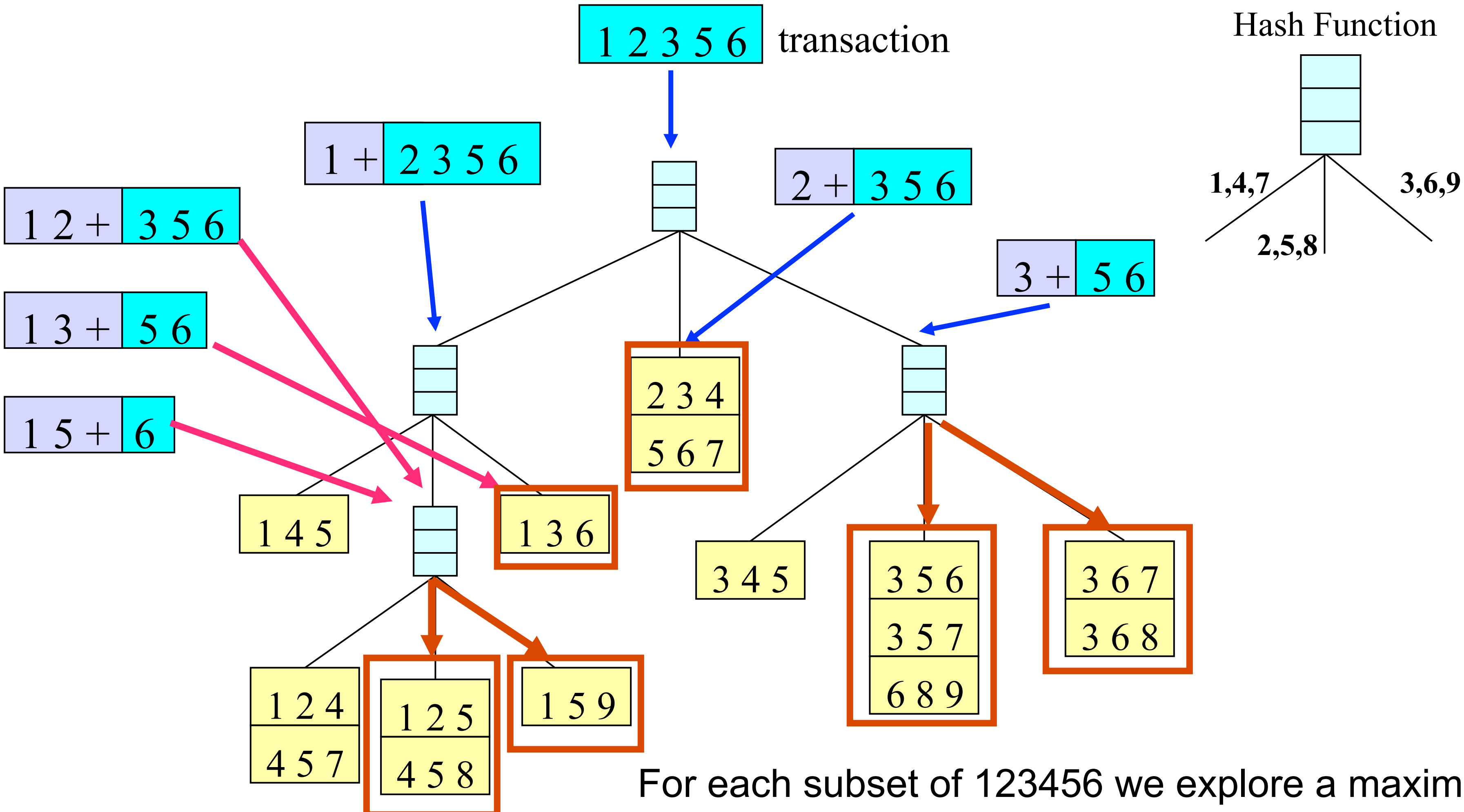
1 can be combined with 2, or 3, or 5, but <u>not</u> with 6

In the transaction there not exist any itemset, where {1,6,x}, x>6 !!!

Subset Operation Using Hash Tree



Subset Operation Using Hash Tree



For each subset of 123456 we explore a maximum of 3 candidates (the size of each leaf)
We explore 11 candidates to identify the 3 matching candidates.

Factors Affecting Complexity

- Choice of minimum support threshold
  - lowering support threshold results in more frequent itemsets
  - this may increase the number of candidates and the 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 also 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 the 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)

How to improve the efficiency of Apriori

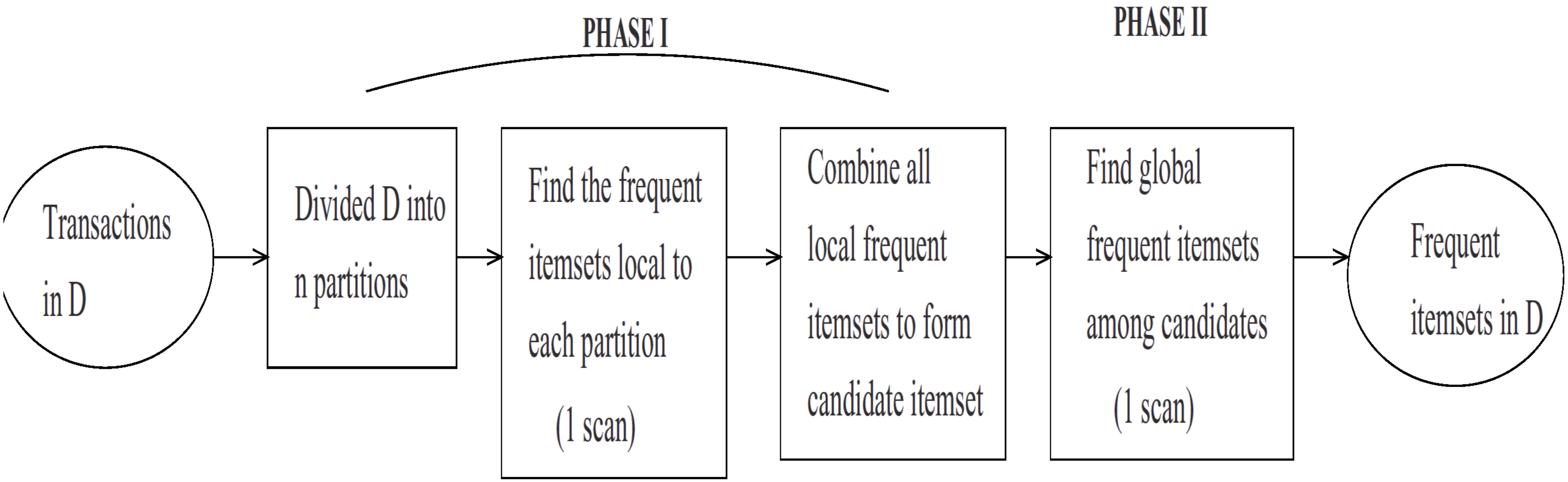
- **Hash-based itemset counting to reduce the size of C_k:**
  - **At iteration *k-1* try to forecast the itemsets that will NOT be part of C_k**
  - **The k-itemset (where items are mapped to integer IDs) occurring in a transaction are *mapped,* with a hash function, into a relatively small table**
    ⇒ less counters than in C_k
  - **All the *k*-itemsets mapped to the same hash bucket whose counter is less than a given minsup**
    - **cannot be frequent and thus can be pruned from C_k**
- **Example: at iteration *k*=1, create the hash table H2 , for items {I1,I2,I3,I4,I5,I6}**
  - **hash function:** $h(x, y) = (x * 10 + y) \bmod 7$
  - **min_supp = 3**
  - **Size hash table = 6   Number of subsets of 2 elements (max sz of C_k) = 15**

H_2

bucket address	0	1	2	3	4	5	6
bucket count	2	2	4	2	2	4	4
bucket contents	~~{I1,I4}~~ ~~{I3,I5}~~	~~{I1,I5}~~ ~~{I1,I5}~~	{I2,I3} {I2,I3} {I2,I3} {I2,I3}	~~{I2,I4}~~ ~~{I2,I4}~~	~~{I2,I5}~~ ~~{I2,I5}~~	{I1,I2} {I1,I2} {I1,I2} {I1,I2}	{I1,I3} {I1,I3} {I1,I3} {I1,I3}

How to improve the efficiency of Apriori

- **Transaction pruning**: A transaction that does not contain any frequent k-itemset, cannot contain any larger itemset, and can thus be pruned
- **Sampling**: mining a reduced number of transactions, but this reduces the *accuracy* of the result
- **Partitioning**: Small partitions of database D can be managed in main memory. An itemset that is frequent in D must be frequent in at least one of the partition of D.
  Unfortunately a frequent itemset in a partition of D could be infrequent in the whole database D.



Partitioning

- 3 partitions of D : $D1, D2, D3$
- If itemset X is globally frequent, then:

(1) $\sigma(X) = \sigma_{D1}(X) + \sigma_{D2}(X) + \sigma_{D3}(X)$ $>=$ $minsup\% \, (|D1| + |D2| + |D3|)$

$\forall i, \;\; \sigma_{Di}(X) < minsup\% \, |Di|$ $\Rightarrow$ X is globally infrequent, since property (1) does not hold

$\neg$ (X is globally infrequent) $\Rightarrow$ $\neg(\; \forall i, \;\; \sigma_{Di}(X) < minsup\% \, |Di| \;)$

X is globally frequent $\Rightarrow$ $\exists i, \; \sigma_{Di}(X) >= minsup\% \, |Di|$

X is globally frequent $\Rightarrow$ X is locally frequent in some dataset

Rule Generation

- **Non optimized algorithm**

c is frequent due Apriori property

for each frequent itemset ***l*** **do**

for each <u>proper</u> subset ***c*** of ***l*** **do**

if (support(***l***) / support(***l-c***) $\geq$ <u>*minconf*)</u> then

output rule ***(l-c)*** $\Rightarrow$ ***c***, with

confidence = support(*l*) / support(*l-c*)

support = support(*l*);

- **e.g.: If $X = \{A,B,C,D\}$ is frequent, candidate rules:**

ABC →D,	**ABD →C,**	**ACD →B,**	**BCD →A,**
A →BCD,	**B →ACD,**	**C →ABD,**	**D →ABC,**
AB →CD,	**AC → BD,**	**AD → BC,**	**BC →AD,**
BD →AC,	**CD → AB**		

- **If $|X| = m$, then there are $2^m - 2$ candidate association rules (ignoring $X \rightarrow \varnothing$ and $\varnothing \rightarrow X$)**

Efficient Rule Generation

- In general, confidence does not have an anti-monotone property

  c(ABC →D) can be larger or smaller than c(AB →D)

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

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

$$\frac{\sigma(ABCD)}{\sigma(ABC)} \geq \frac{\sigma(ABCD)}{\sigma(AB)} \geq \frac{\sigma(ABCD)}{\sigma(A)}$$

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

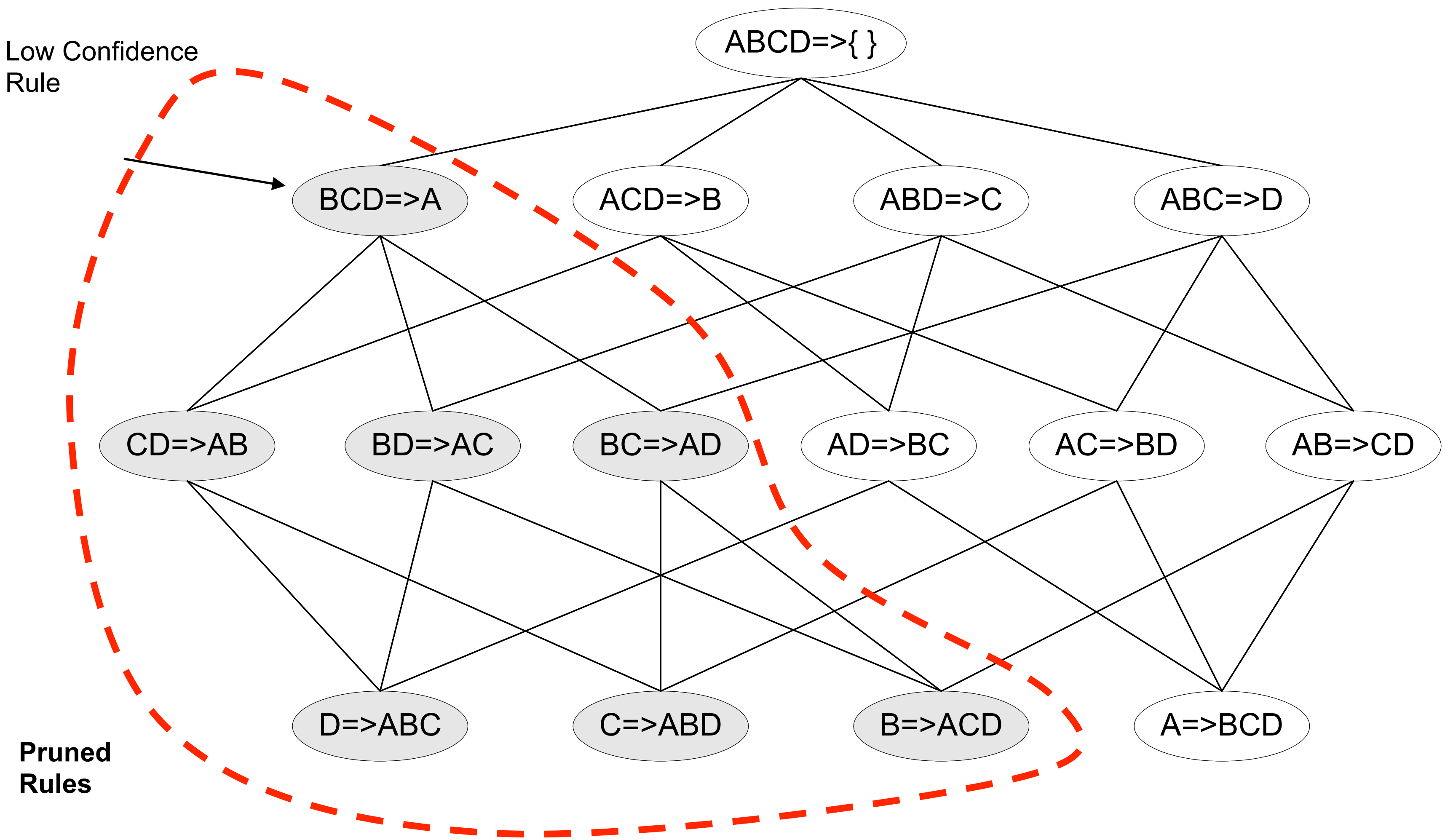
If *min_conf* > c(ABC → D) then

min_conf > c(ABC → D) ≥ c(AB → CD) ≥ c(A → BCD)

and thus we can prune c(AB → CD) and c(A → BCD)

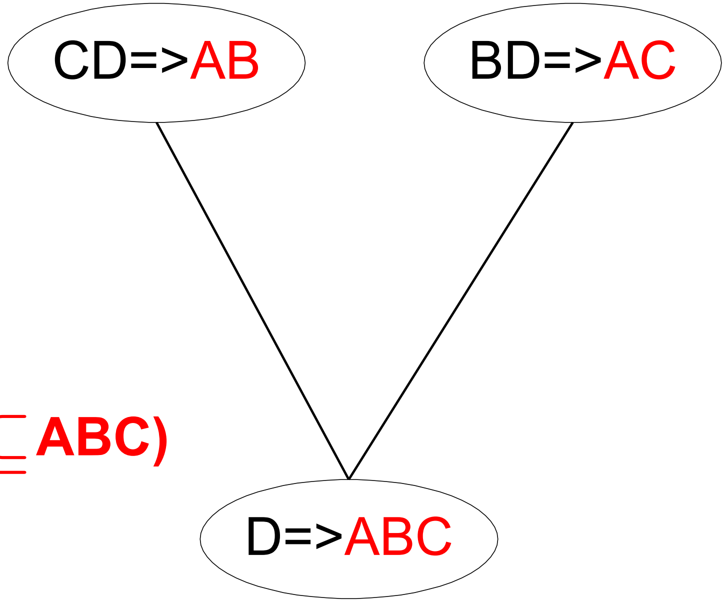
Efficient Rule Generation

Lattice of rules



Efficient Rule Generation

- **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 AD=>BC does not have high confidence**
- **(AD=>BC is not a a generator, but BC⊆ABC)**
- **Note that the other two "subsets" CD=>AB and BD=>AC are surely highly confident, since they are the generators of D => ABC**

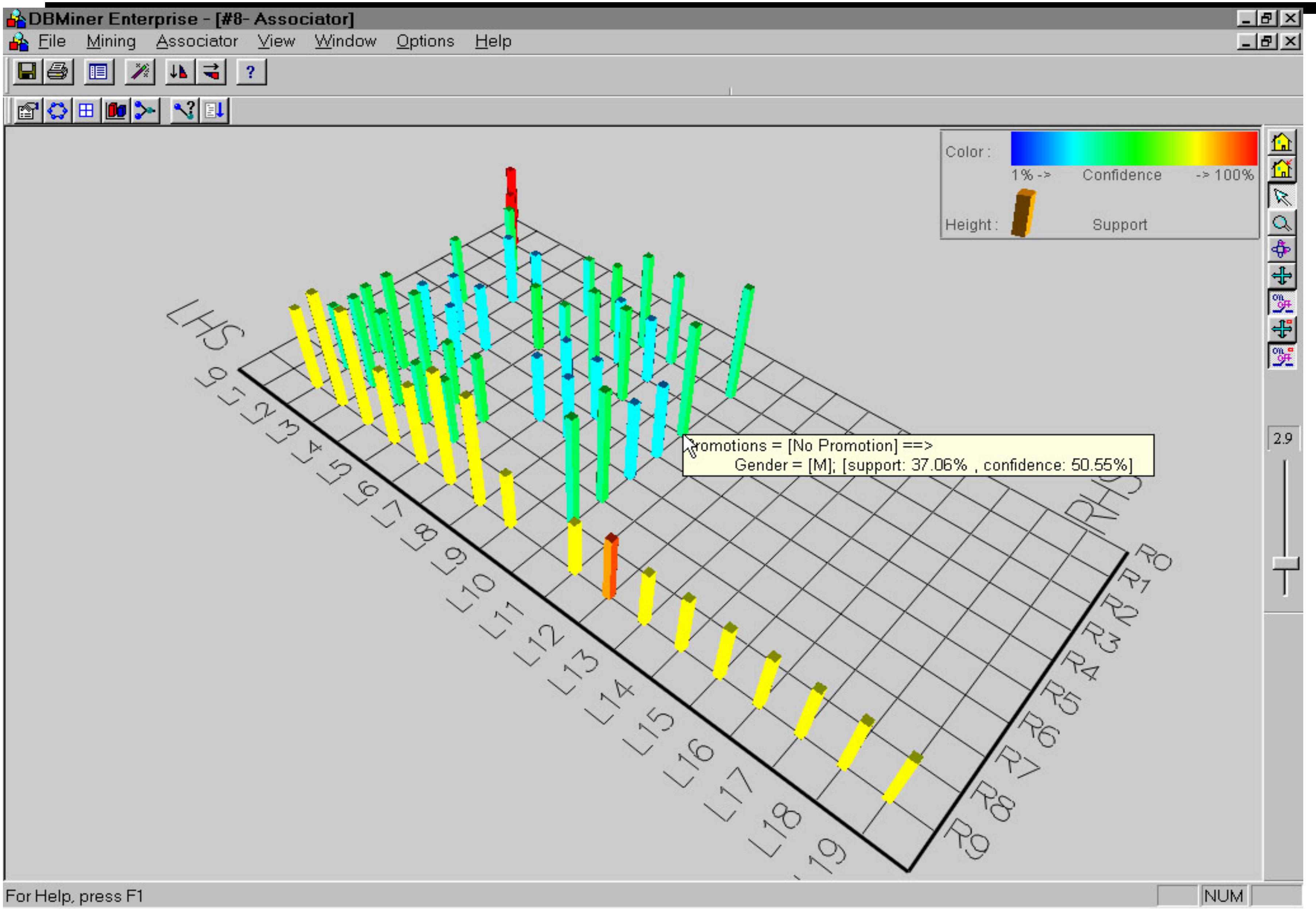


Presentation of Association Rules (Table Form)

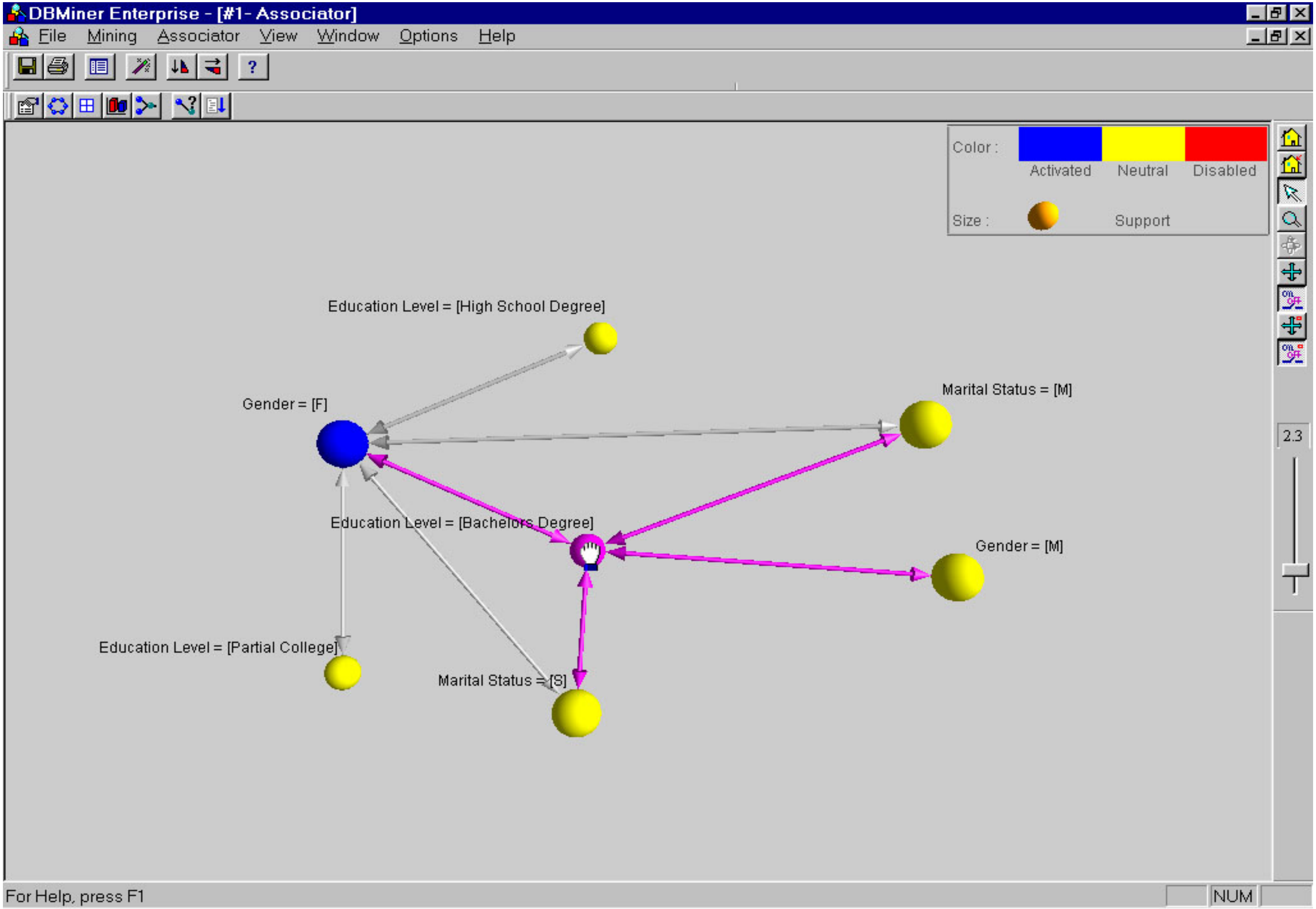
	Body	Implies	Head	Supp (%)	Conf (%)	F	G	H	I
1	cost(x) = '0.00~1000.00'	==>	revenue(x) = '0.00~500.00'	28.45	40.4				
2	cost(x) = '0.00~1000.00'	==>	revenue(x) = '500.00~1000.00'	20.46	29.05				
3	cost(x) = '0.00~1000.00'	==>	order_qty(x) = '0.00~100.00'	59.17	84.04				
4	cost(x) = '0.00~1000.00'	==>	revenue(x) = '1000.00~1500.00'	10.45	14.84				
5	cost(x) = '0.00~1000.00'	==>	region(x) = 'United States'	22.56	32.04				
6	cost(x) = '1000.00~2000.00'	==>	order_qty(x) = '0.00~100.00'	12.91	69.34				
7	order_qty(x) = '0.00~100.00'	==>	revenue(x) = '0.00~500.00'	28.45	34.54				
8	order_qty(x) = '0.00~100.00'	==>	cost(x) = '1000.00~2000.00'	12.91	15.67				
9	order_qty(x) = '0.00~100.00'	==>	region(x) = 'United States'	25.9	31.45				
10	order_qty(x) = '0.00~100.00'	==>	cost(x) = '0.00~1000.00'	59.17	71.86				
11	order_qty(x) = '0.00~100.00'	==>	product_line(x) = 'Tents'	13.52	16.42				
12	order_qty(x) = '0.00~100.00'	==>	revenue(x) = '500.00~1000.00'	19.67	23.88				
13	product_line(x) = 'Tents'	==>	order_qty(x) = '0.00~100.00'	13.52	98.72				
14	region(x) = 'United States'	==>	order_qty(x) = '0.00~100.00'	25.9	81.94				
15	region(x) = 'United States'	==>	cost(x) = '0.00~1000.00'	22.56	71.39				
16	revenue(x) = '0.00~500.00'	==>	cost(x) = '0.00~1000.00'	28.45	100				
17	revenue(x) = '0.00~500.00'	==>	order_qty(x) = '0.00~100.00'	28.45	100				
18	revenue(x) = '1000.00~1500.00'	==>	cost(x) = '0.00~1000.00'	10.45	96.75				
19	revenue(x) = '500.00~1000.00'	==>	cost(x) = '0.00~1000.00'	20.46	100				
20	revenue(x) = '500.00~1000.00'	==>	order_qty(x) = '0.00~100.00'	19.67	96.14				
21									
22									
23	cost(x) = '0.00~1000.00'	==>	revenue(x) = '0.00~500.00' AND order_qty(x) = '0.00~100.00'	28.45	40.4				
24	cost(x) = '0.00~1000.00'	==>	revenue(x) = '0.00~500.00' AND order_qty(x) = '0.00~100.00'	28.45	40.4				
25	cost(x) = '0.00~1000.00'	==>	revenue(x) = '500.00~1000.00' AND order_qty(x) = '0.00~100.00'	19.67	27.93				
26	cost(x) = '0.00~1000.00'	==>	revenue(x) = '500.00~1000.00' AND order_qty(x) = '0.00~100.00'	19.67	27.93				
27	cost(x) = '0.00~1000.00' AND order_qty(x) = '0.00~100.00'	==>	revenue(x) = '500.00~1000.00'	19.67	33.23				

Sheet1

Visualization of Association Rule Using Plane Graph



Visualization of Association Rule Using Rule Graph



Compact Representation of Frequent Itemsets

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

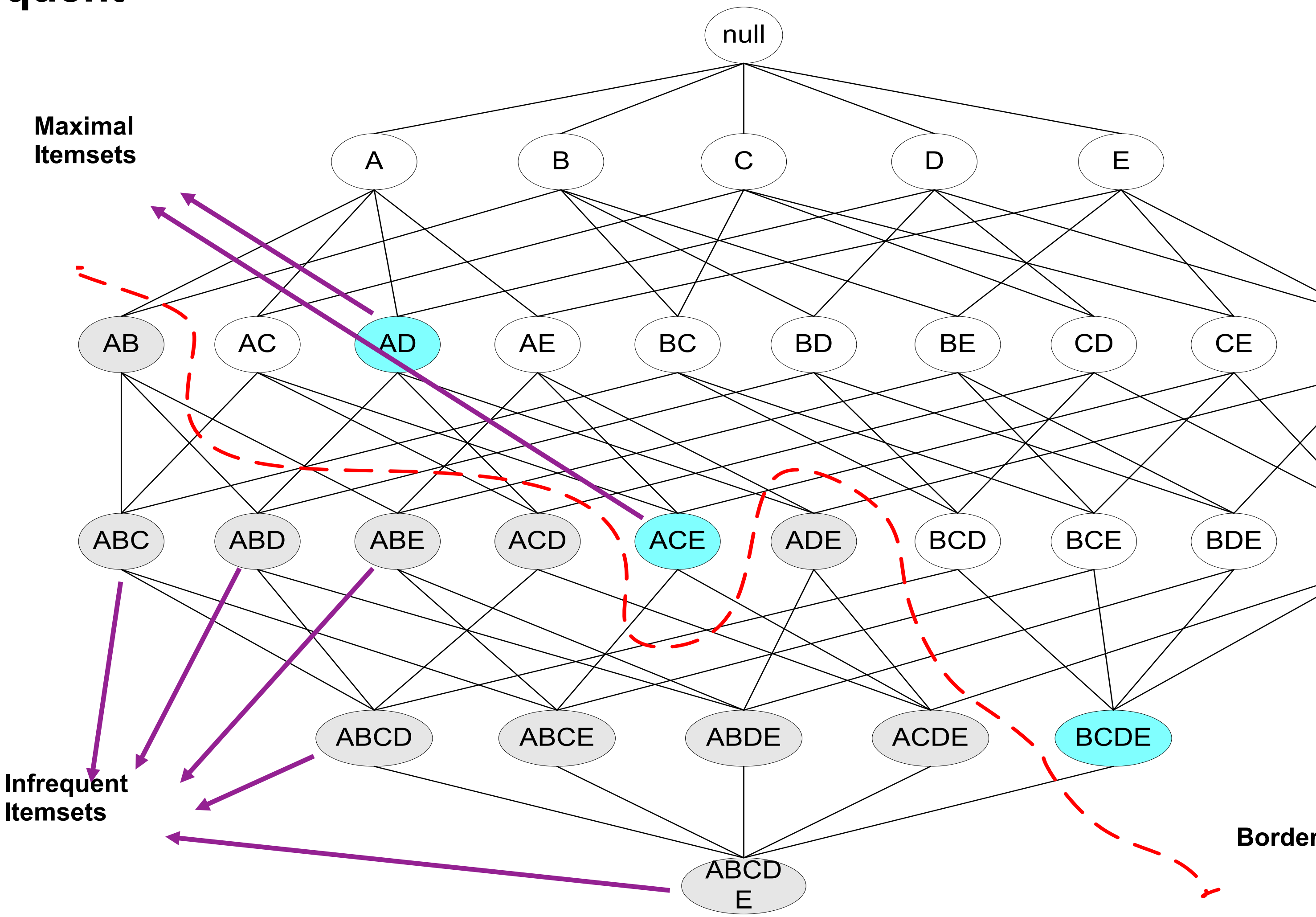
TID	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	1	1	1	1	1	1	1	1	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

- **Min support count:** σ **= 5**
- **Number of frequent itemsets** $= 3 \times \sum_{k=1}^{10} \binom{10}{k}$

- **Need a compact representation**

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 as the itemset**

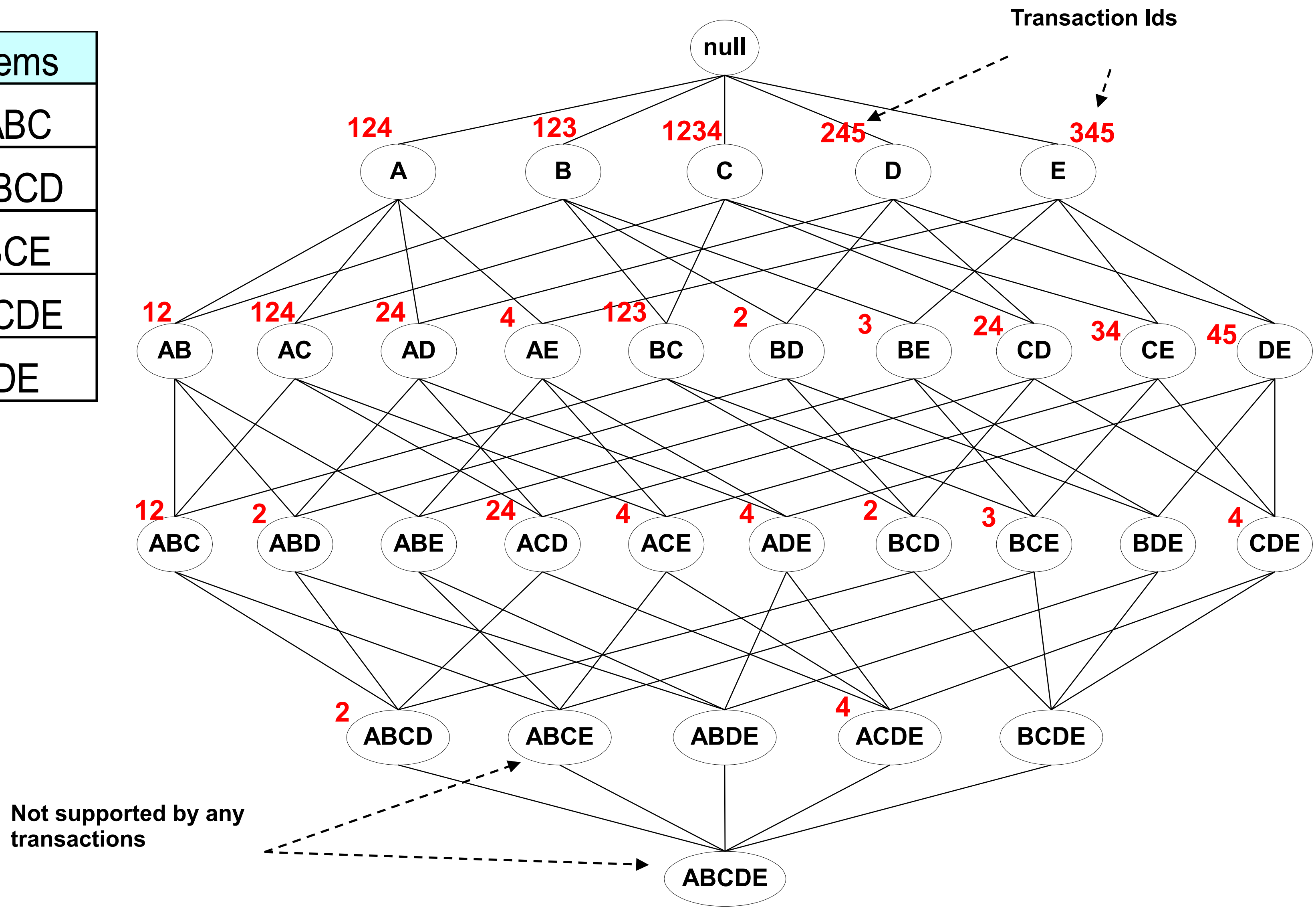
TID	Items
1	{A,B}
2	{B,C,D}
3	{A,B,C,D}
4	{A,B,D}
5	{A,B,C,D}

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

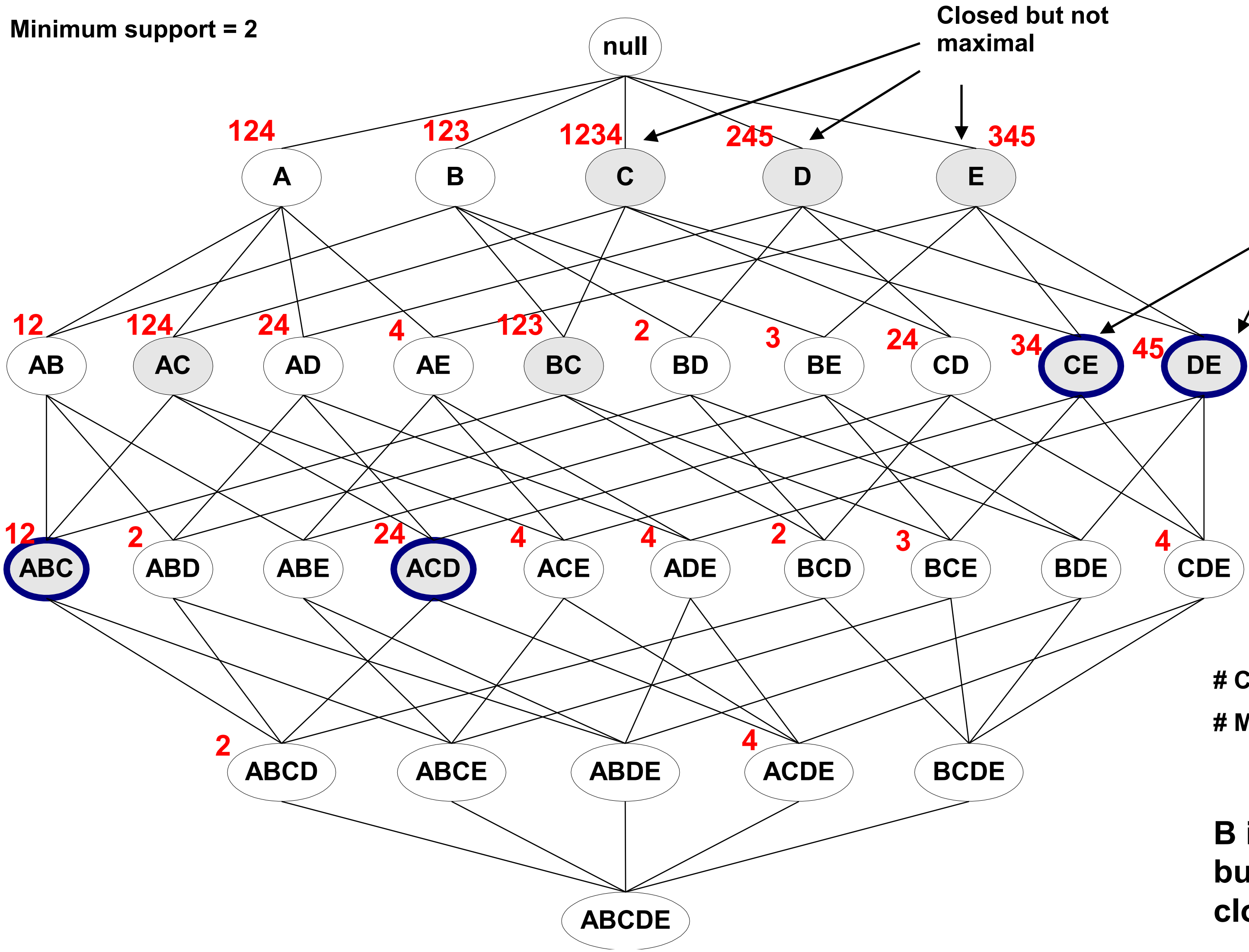
Itemset	Support
{A,B,C}	2
{A,B,D}	3
{A,C,D}	2
{B,C,D}	3
{A,B,C,D}	2

Maximal vs Closed Itemsets

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



Maximal vs Closed Frequent Itemsets



Closed = 9

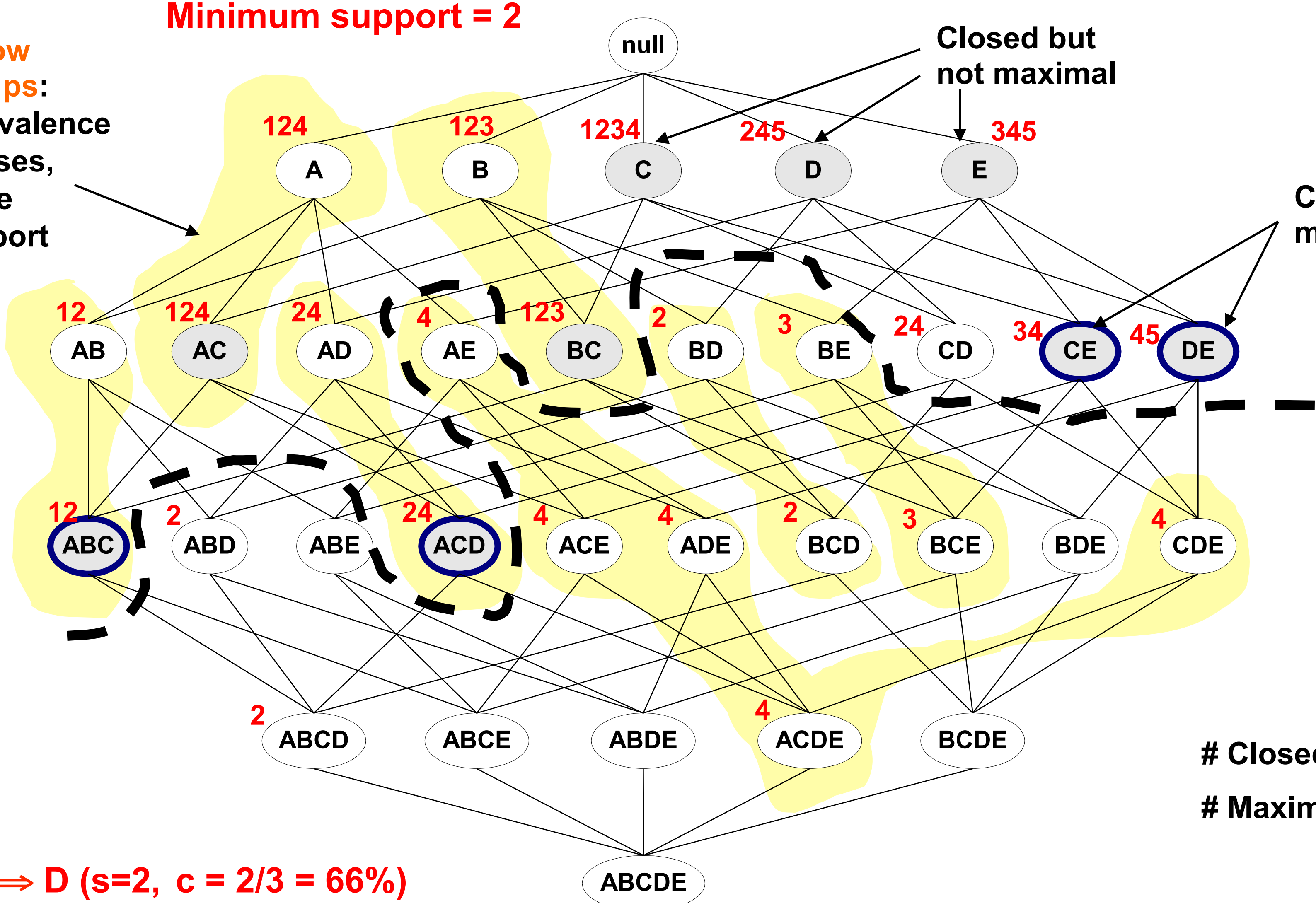
Maximal = 4

B is frequent, but is NOT closed. Why?

Maximal vs Closed Itemsets

Minimum support = 2

Yellow groups: equivalence classes, same support



Closed = 10

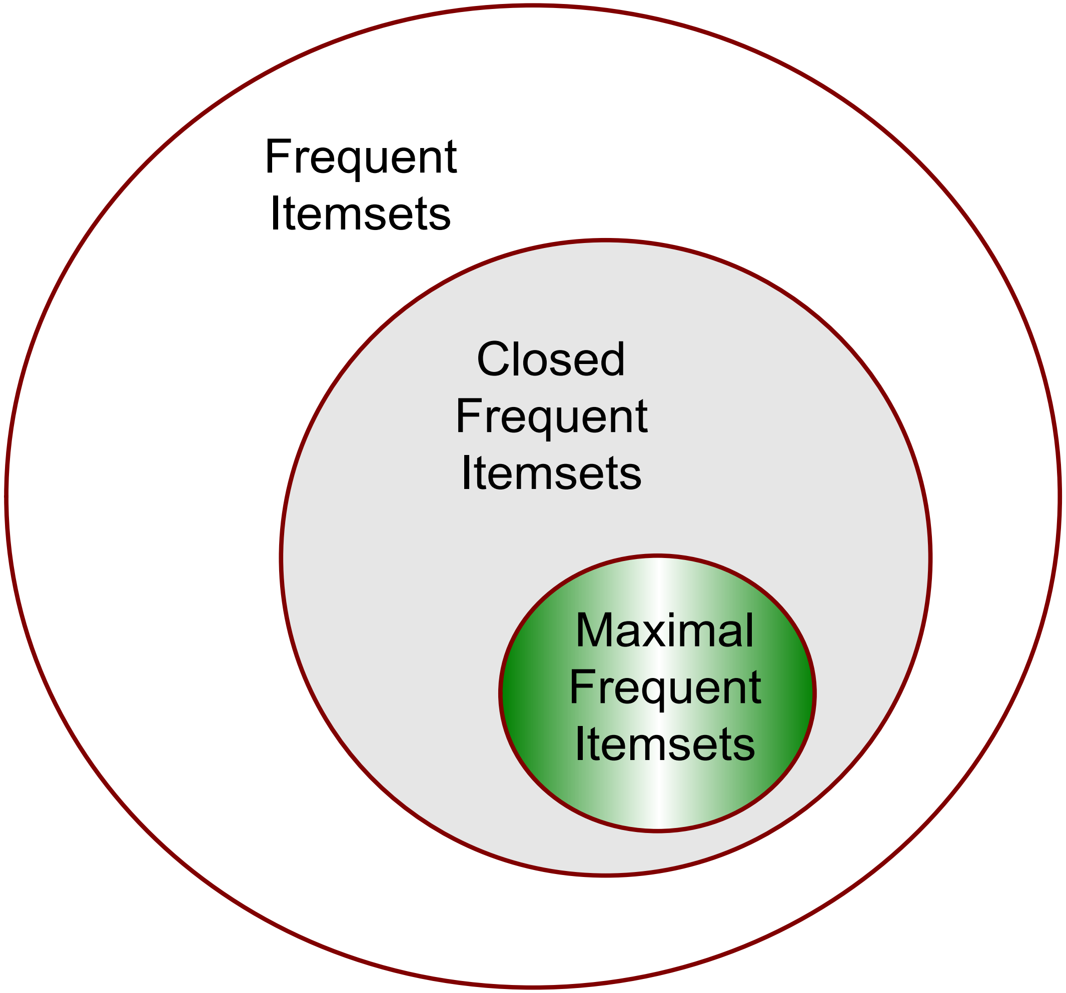
Maximal = 5

A $\Rightarrow$ D (s=2, c = 2/3 = 66%)

AC $\Rightarrow$ D (s=2, c = 2/3 = 66%)

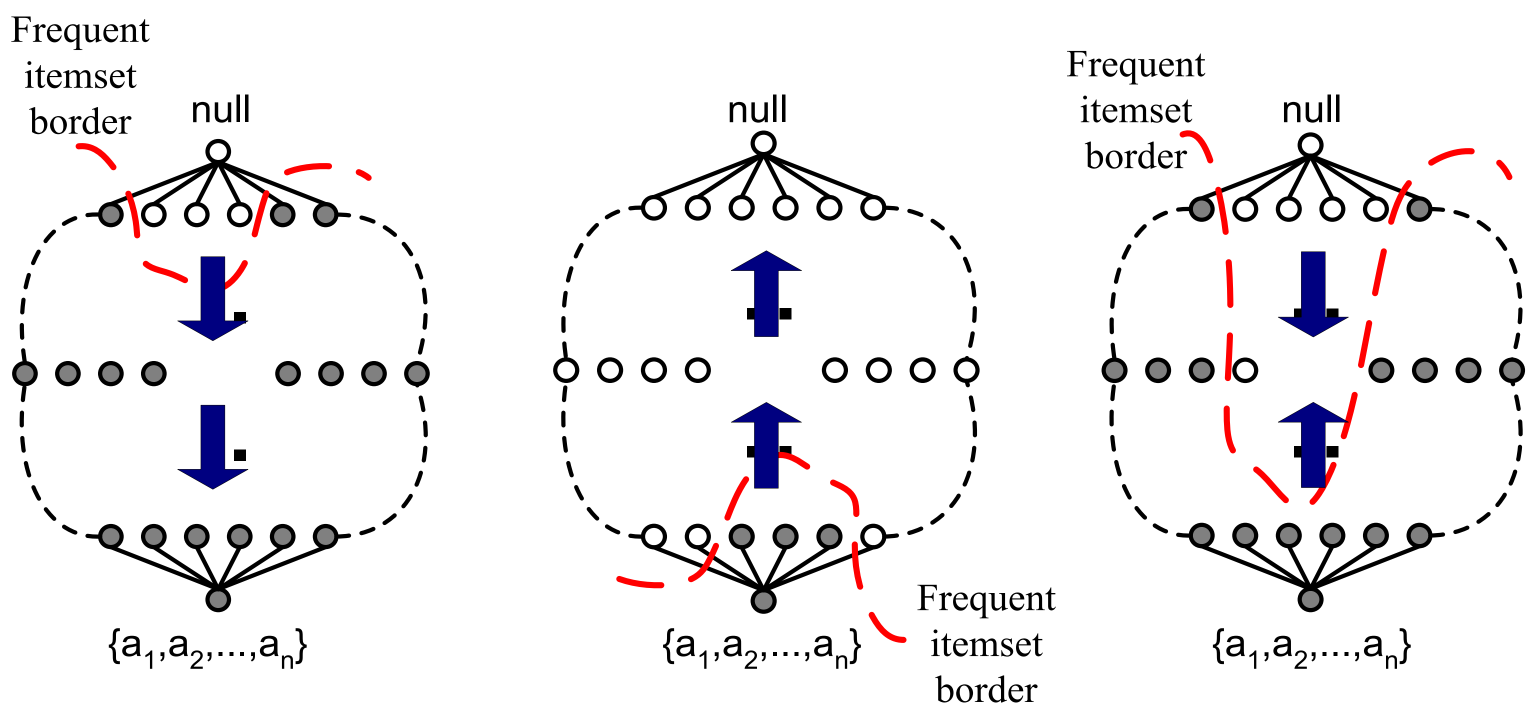
This rule is not generated if we start from the closed, since AD is not returned at all

Maximal vs Closed Itemsets



Alternative Methods for Frequent Itemset Generation

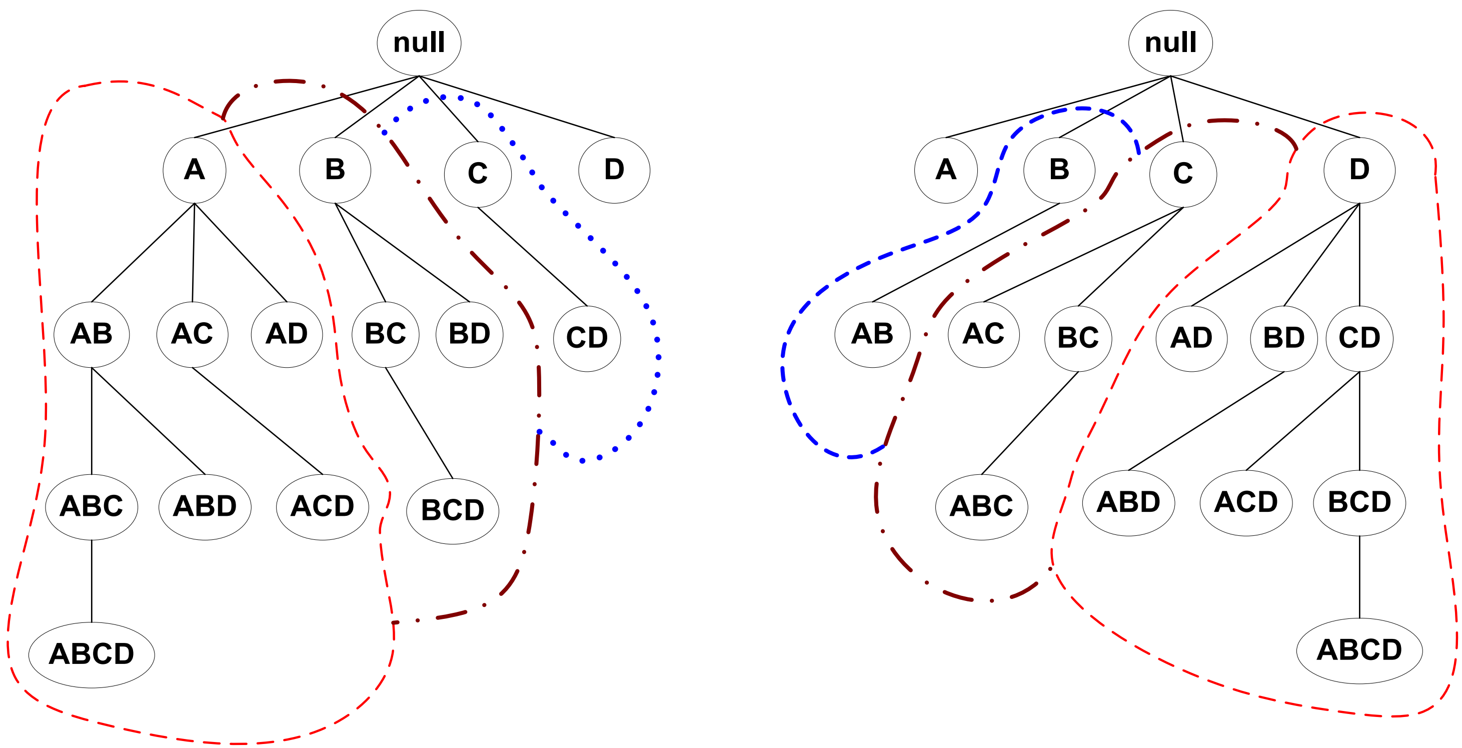
- **Traversal of Itemset Lattice**
  - **General-to-specific (Apriori method)   vs.   Specific-to-general**



(a) General-to-specific (b) Specific-to-general (c) Bidirectional

Alternative Methods for Frequent Itemset Generation

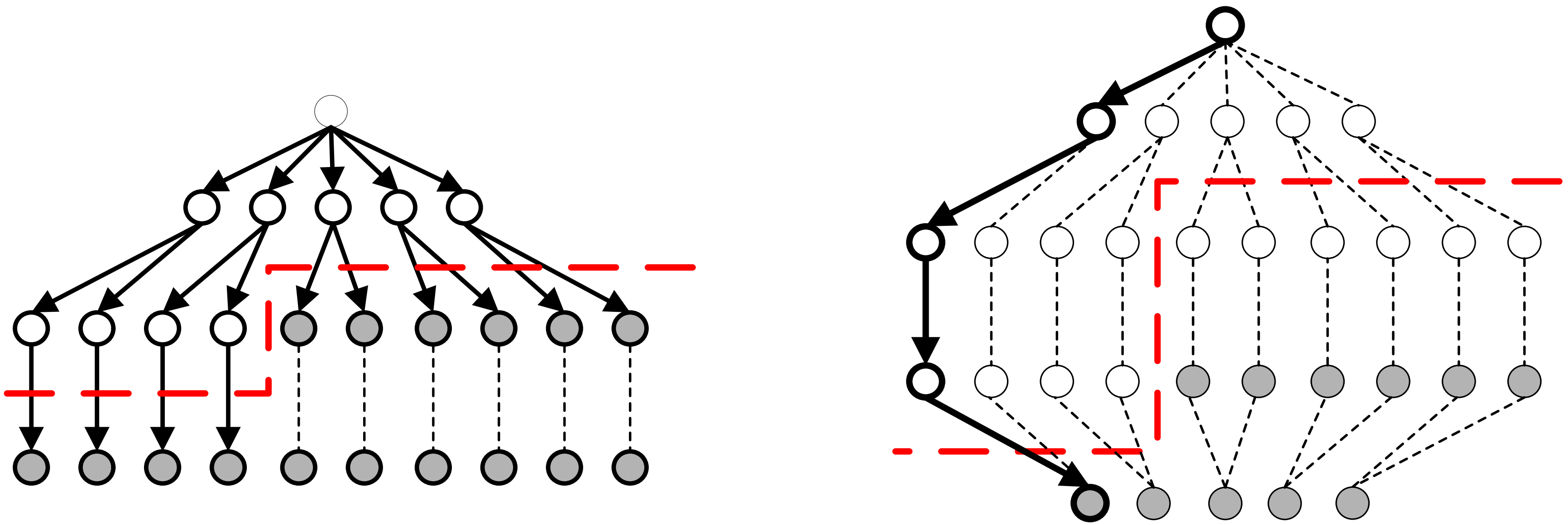
- **Traversal of Itemset Lattice**
  - **Equivalence Classes (either same prefix or suffix)**



(a) Prefix tree (b) Suffix tree

Alternative Methods for Frequent Itemset Generation

- **Traversal of Itemset Lattice**
  - **Breadth-first vs Depth-first**



(a) Breadth first

(b) Depth first

Apriori Performance Bottlenecks

- **The core of the Apriori algorithm:**
  - **Use frequent (*k* – 1)-itemsets to generate candidate frequent *k*-itemsets**
  - **Use database scan and pattern matching to collect counts for the candidate itemsets**
- **The bottleneck of *Apriori*: candidate generation**
  - **Huge candidate sets:**
    - **10^4 frequent 1-itemset will generate 10^7 candidate 2-itemsets**
    - **To discover a frequent pattern of size 100, e.g., $\{a_1, a_2, \ldots, a_{100}\}$, one needs to generate $2^{100} \approx 10^{30}$ candidates.**
  - **Multiple scans of database:**
    - **Needs (n +1) scans, n is the length of the longest pattern**

Mining Patterns Without Candidate Generation

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

Construct FP-tree from a Transaction DB

TID	*Items bought*	*(ordered) frequent items*
100	*{f, a, c, d, g, i, m, p}*	*{f, c, a, m, p}*
200	*{a, b, c, f, l, m, o}*	*{f, c, a, b, m}*
300	*{b, f, h, j, o}*	*{f, b}*
400	*{b, c, k, s, p}*	*{c, b, p}*
500	*{a, f, c, e, l, p, m, n}*	*{f, c, a, m, p}*

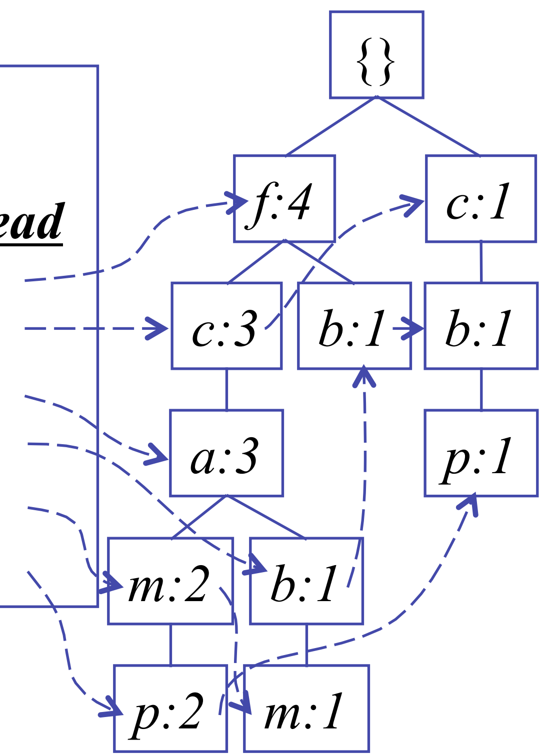
min_support = 3

Steps:

1. **Scan DB once, find frequent 1-itemset (single item pattern)**
2. **Order frequent items in frequency descending order**
3. **Scan DB again, construct FP-tree**

Header Table

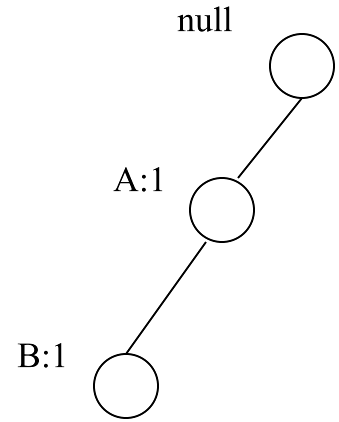
Item	*frequency*	*head*
f	4	
c	4	
a	3	
b	3	
m	3	
p	3	



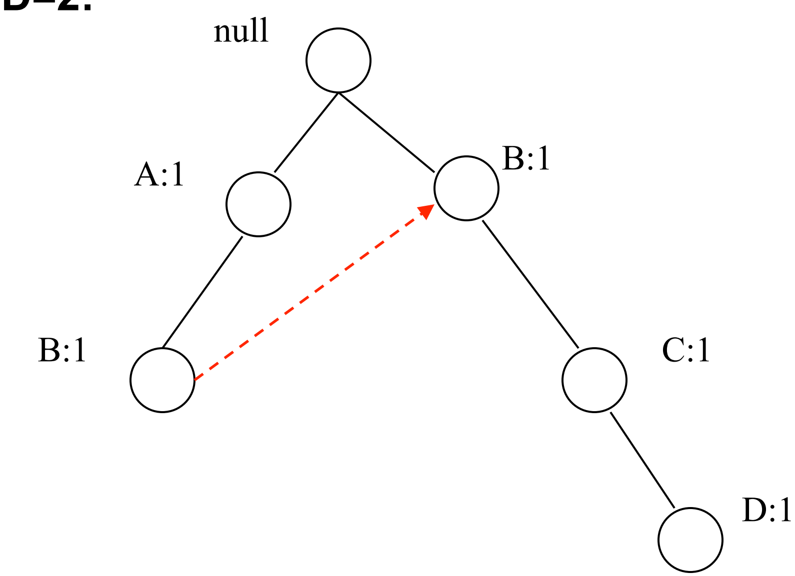
FP-tree construction

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:



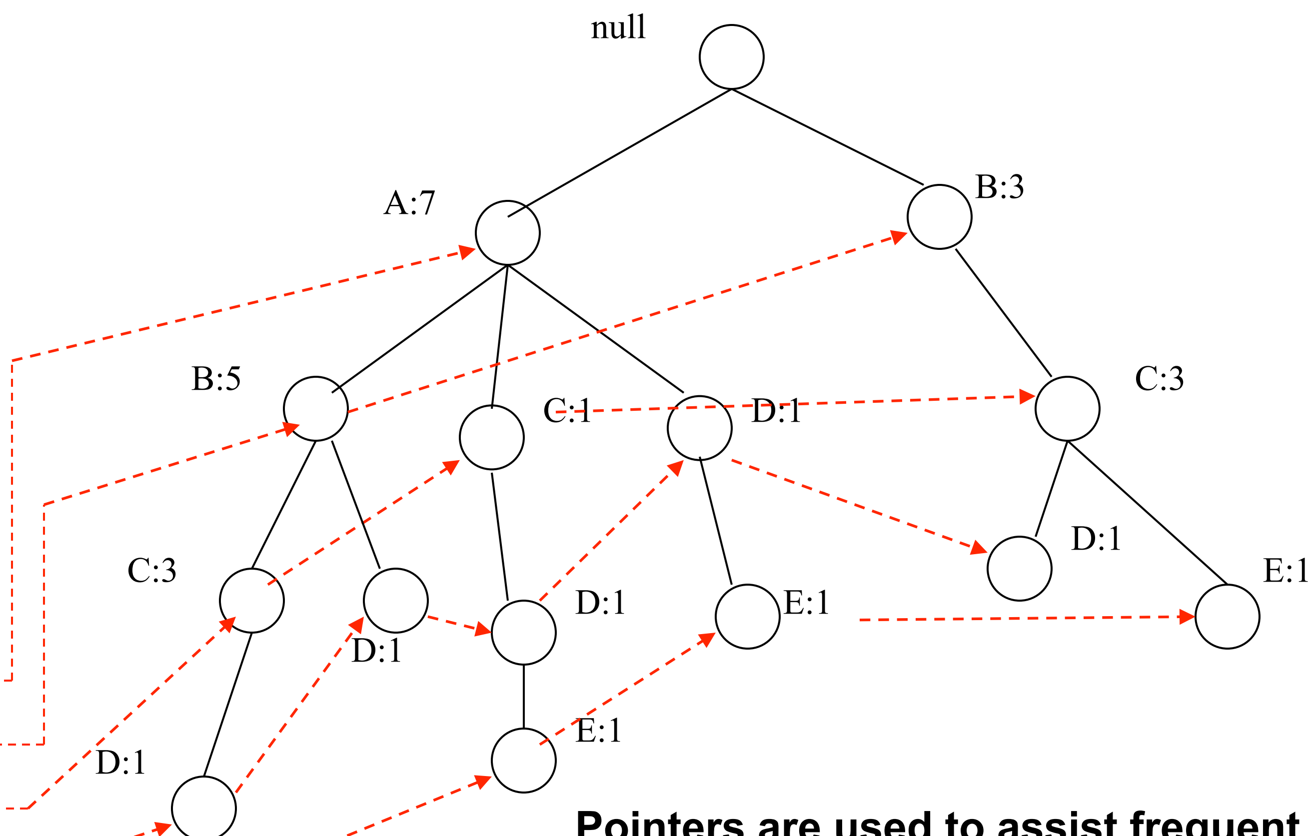
FP-Tree Construction

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

Benefits of the FP-tree Structure

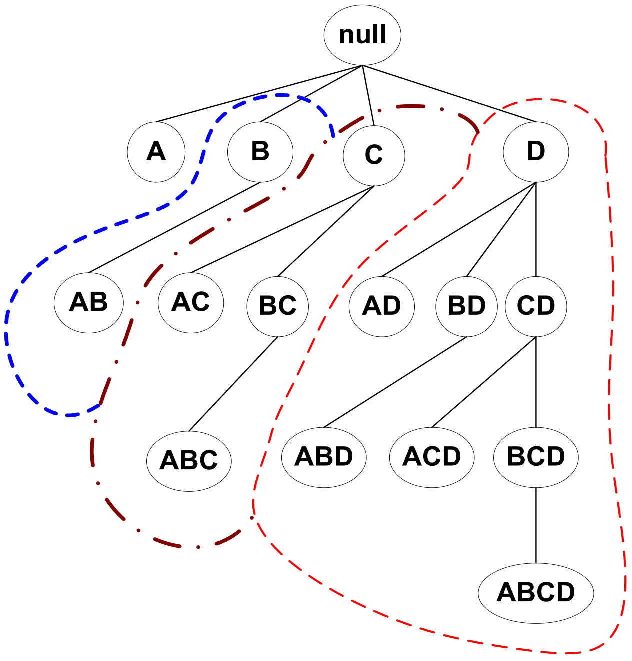
- Completeness:
  - never breaks a long pattern of any transaction
  - preserves complete information for frequent pattern mining
- Compactness
  - reduce irrelevant information—infrequent items are gone
  - frequency descending ordering: more frequent items are more likely to be shared
  - never be larger than the original database (if not count node-links and counts)
  - Example: For Connect-4 DB, compression ratio could be over 100

Mining Frequent Patterns Using FP-tree

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

Mining Frequent Patterns Using FP-tree

- **The lattice is explored depth-first**
  1. **First mine the patterns with suffix *p***
  2. **Then mine the patterns with suffix *m***
     – that not contain *p*
  3. **Then mine the pattern with suffix *b***
     – that not contain *p* and *m*
  4. **Then mine the pattern with suffix *a***
     – that not contain *p, m* and *b*
  5. **Then mine the pattern with suffix *c***
     – that not contain *p, m, b* and *a*
  6. **Then mine the pattern with suffix *f***
     – that not contain *p, m, b, a* and *c*



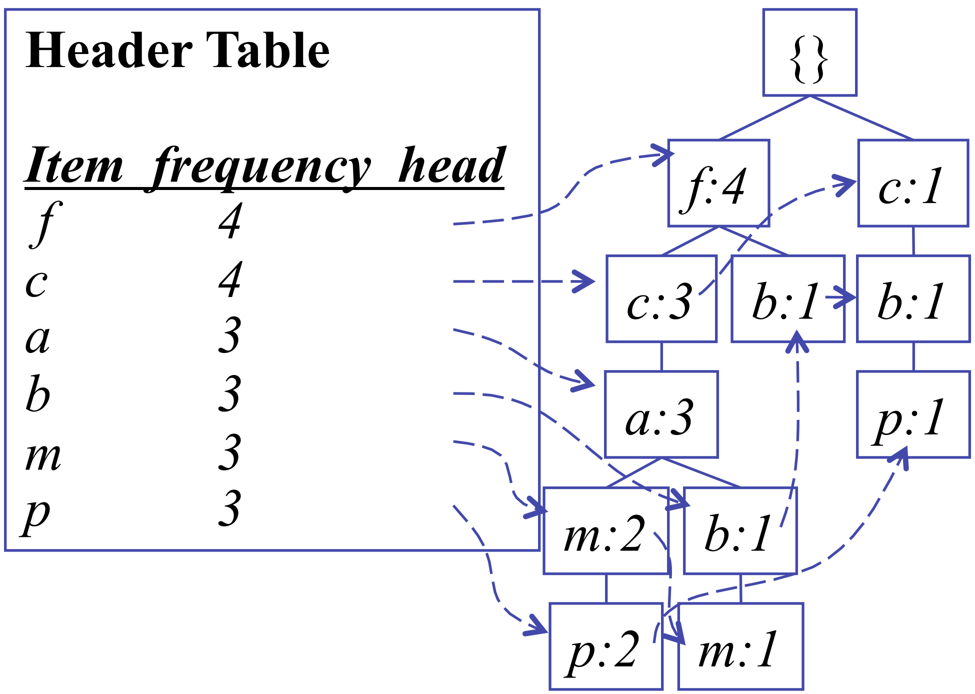
- **For each mining task, we can generate a projected DB by removing transactions and items**
  – **The FP-tree make it easy to generate the projected** DB

Major Steps to Mine FP-tree

1) Construct conditional pattern base for each node in the FP-tree
   - Projected DB
2) Construct conditional FP-tree from each conditional pattern-base
3) Recursively mine conditional FP-trees and grow frequent patterns obtained so far
   - If the conditional FP-tree contains a single path, simply enumerate all the patterns

Step 1: From FP-tree to Conditional Pattern Base

- Starting at the frequent header table in the FP-tree
- Traverse the FP-tree by following the link of each frequent item
- Accumulate all of transformed prefix paths of that item to form a conditional pattern base



Conditional pattern bases

item	***cond. pattern base***
c	***f:3***
a	***fc:3***
b	***fca:1, f:1, c:1***
m	***fca:2, fcab:1***
p	***fcam:2, cb:1***

Properties of FP-tree for Conditional Pattern Base Construction

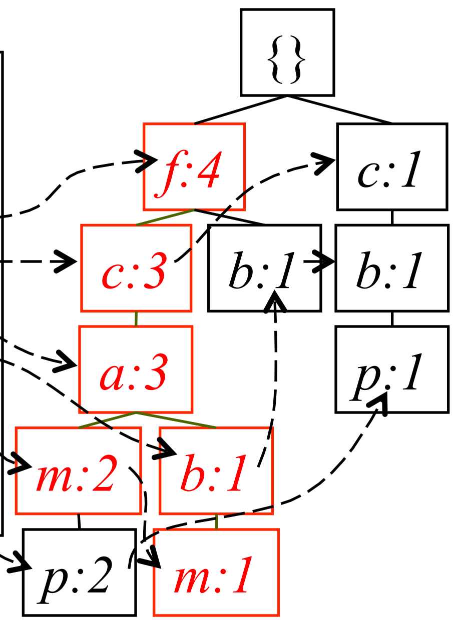
- **Node-link property**
  - For any frequent item a_i, all the possible frequent patterns that contain a_i can be obtained by following a_i's node-links, starting from a_i's head in the FP-tree header
- **Prefix path property**
  - To calculate the frequent patterns for a node a_i in a path P, only the prefix sub-path of a_i in P need to be accumulated, and its frequency count should carry the same count as node a_i.

Step 2: Construct Conditional FP-tree

- For each pattern-base
  - Accumulate the count for each item in the base
  - Construct the FP-tree for the <u>frequent items</u> of the pattern base

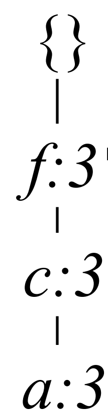
Header Table

Item	*frequency*	*head*
f	*4*	
c	*4*	
a	*3*	
b	*3*	
m	*3*	
p	*3*	



m-conditional **pattern base:**

fca:2, fcab:1



m-conditional **FP-tree**

Step 3

All the frequent patterns with suffix *m*:

m,

fm, cm, am,

fcm, fam, cam,

fcam

Mining Frequent Patterns by Creating Conditional Pattern-Bases

Item	Conditional pattern-base	Conditional FP-tree
p	{(fcam:2), (cb:1)}	{(c:3)}|p
m	{(fca:2), (fcab:1)}	{(f:3, c:3, a:3)}|m
b	{(fca:1), (f:1), (c:1)}	Empty
a	{(fc:3)}	{(f:3, c:3)}|a
c	{(f:3)}	{(f:3)}|c
f	Empty	Empty

Remove infrequent items (whose support count < 3) from the projected DB (Cond. Pattern-base) before generating the Cond. FP-tree

Step 3: Recursively mine the conditional FP-tree

- **Re-apply recursively FP-growth to the FP-tree generated**

{}
|
f:3
|
c:3
|
a:3

m-conditional **FP-tree**

Cond. pattern base di "am": (fc:3)

{}
|
f:3
|
c:3

am-conditional **FP-tree**

Cond. pattern base of "cm": (f:3)

{}
|
f:3

cm-conditional **FP-tree**

Cond. pattern base of "cam": (f:3)

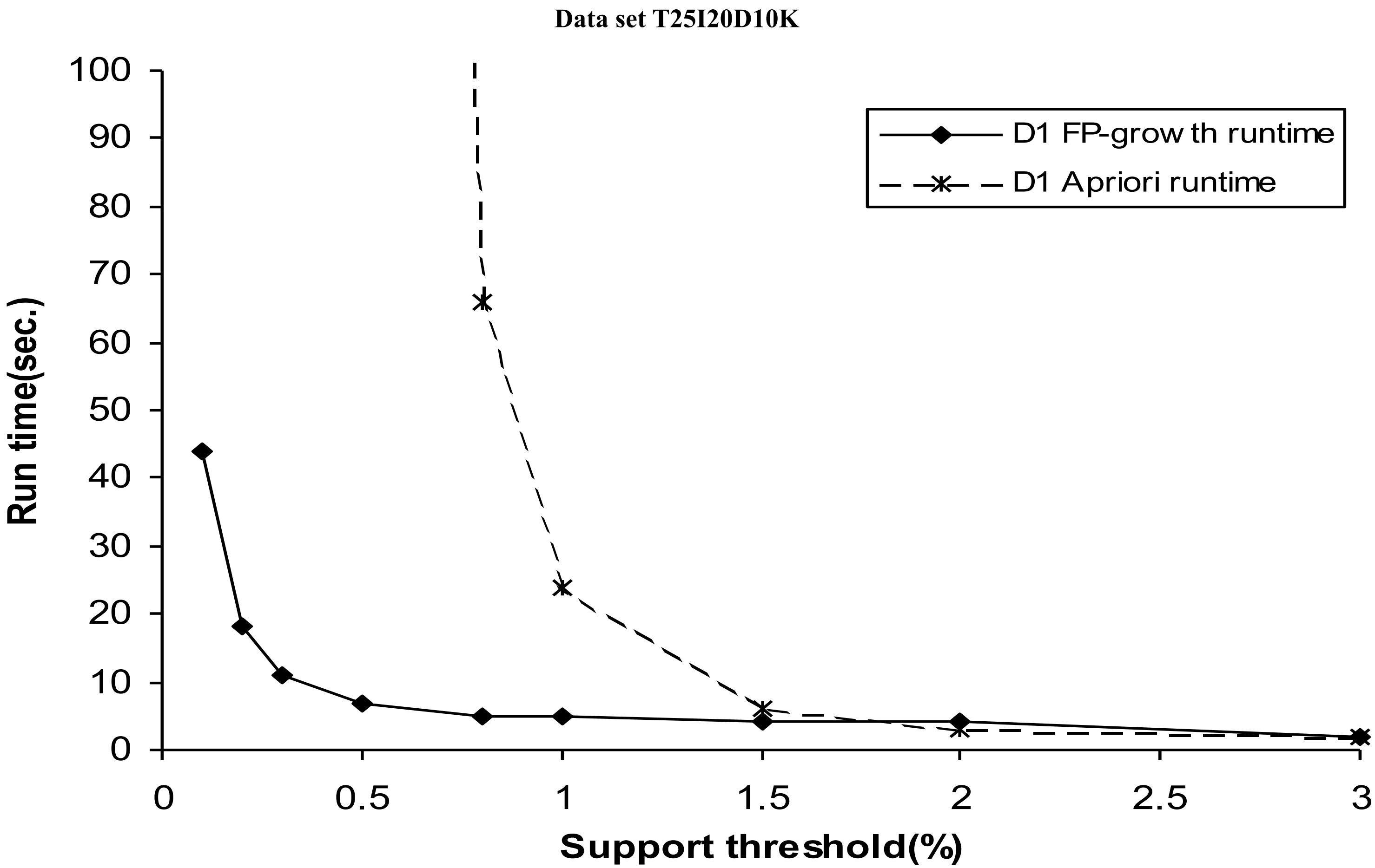
{}
|
f:3

cam-conditional **FP-tree**

Why Is Frequent Pattern Growth Fast?

- The performance study shows
  - FP-growth is an order of magnitude faster than Apriori for *dense dataset*
  - In this case we have the greatest sharing in the transaction, and the compression rate of FP-tree is very high
- Reasons
  - No candidate generation, no candidate test
  - Use compact data structure
  - Eliminate repeated database scan
  - Basic operation is counting and FP-tree building

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



Alternative Methods for Frequent Itemset Generation

- **Representation of Database**
  - **horizontal vs vertical data layout**

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				

Different databsse layout

- **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

FIM algorithms: methods to compute the supports

- **Count-based method**
  – is used by Apriori
  – exploits a <u>horizontal</u> database
  – subsetting of transactions and increment of counters, in turn associated with candidates

- **Intersection-based method**
  – is used by ECLAT
  – exploits a <u>vertical</u> database
  – for each candidate, <u>intersect</u> (set-intersection) the TID-lists associated with the itemsets/items occurring in the candidate
  – the cardinality of the resulting TID-list is the candidate support

ECLAT

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

A
1
4
5
6
7
8
9

$\wedge$

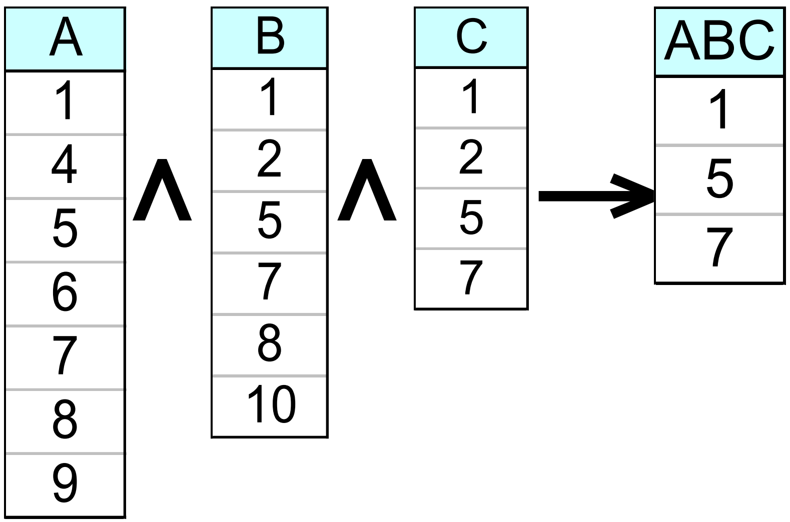
B
1
2
5
7
8
10

$\rightarrow$

AB
1
5
7
8

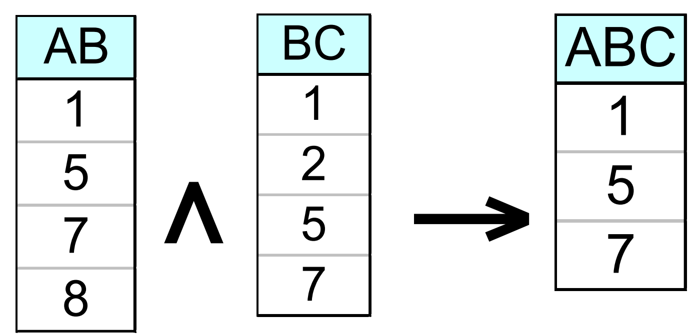
- **3 traversal approaches:**
  - **top-down, bottom-up and hybrid**
- **Advantage: very fast support counting**
- **Disadvantage: intermediate tid-lists may become too large for memory**

Intersection-based

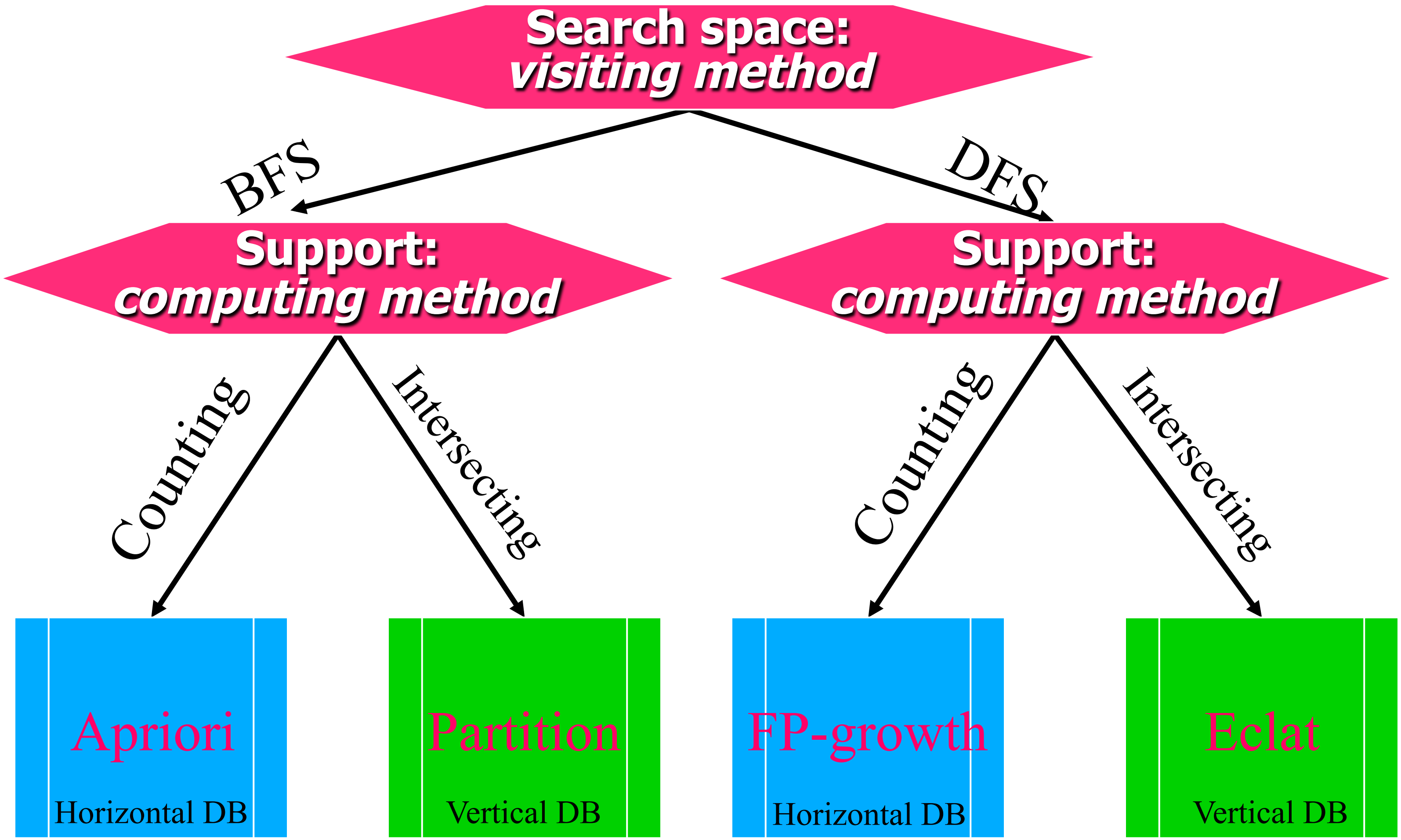


- **k-way intersection**
  - **set intersect the *atomic k TID-lists***
  - **PROS: small memory size to store TID-lists**
  - **CONS: expensive**

- **2-way intersection**
  - **set intersect only two TID-lists associated with two (k-1)-subsets**
  - **PROS: speed**
  - **CONS: : intermediate tid-lists may become too large to store in memory**



Various algoritms per FSC



Level-wise, strictly iterative

Divide & conquer, recursive

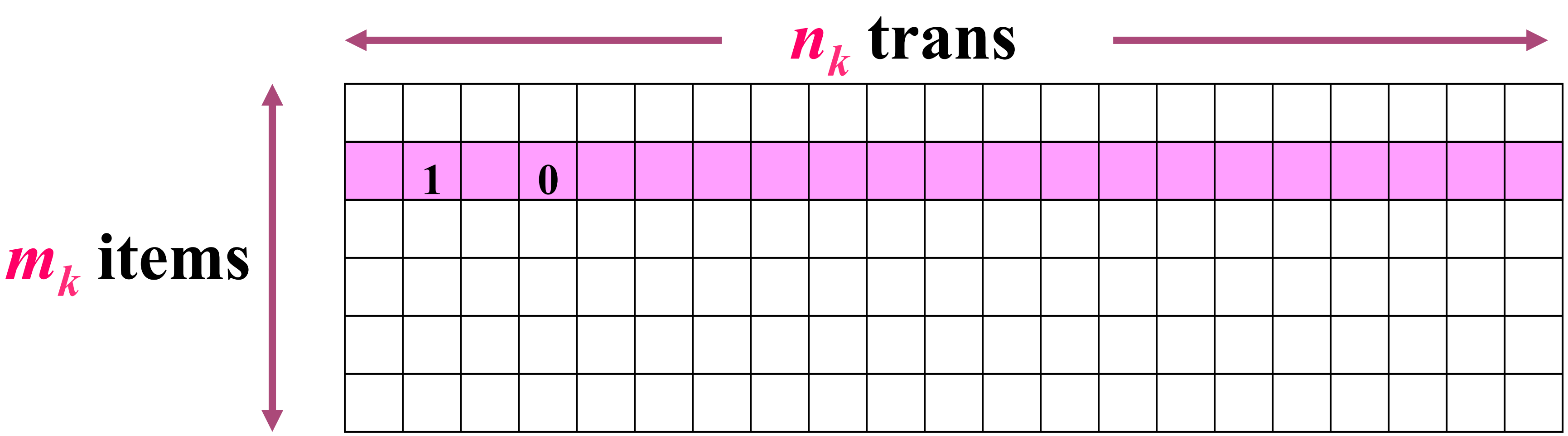
DCI: Direct Counting & Intersecting

- **Level-wise (BFS) algorithm**
- **Hybrid method for determining the supports of frequent itemsets**
  - **Counting-based during early iterations**
    - ***Innovative method for storing and accessing candidates to count their support***
    - ***Effective pruning of horizontal dataset***
  - **Intersection-based when *database* fits into the main memory ⇒ resource-aware**
    - ***Horizontal-to-Vertical transformation***
    - ***Fully optimized k-way intersections***

Orlando, P. Palmerini, R. Perego, and F. Silvestri, **“Adaptive and Resource-Aware Mining of Frequent Sets”** Proc. IEEE Int’l Conf. Data Mining, Dec. 2002.

DCI: intersection-based phase

- When the pruned database fits into the main memory, DCI builds on-the-fly an in-core bit-vector vertical dataset
- Due to the effectiveness of dataset pruning, this usually occurs at early iterations (2^{nd} or 3^{rd} iter)

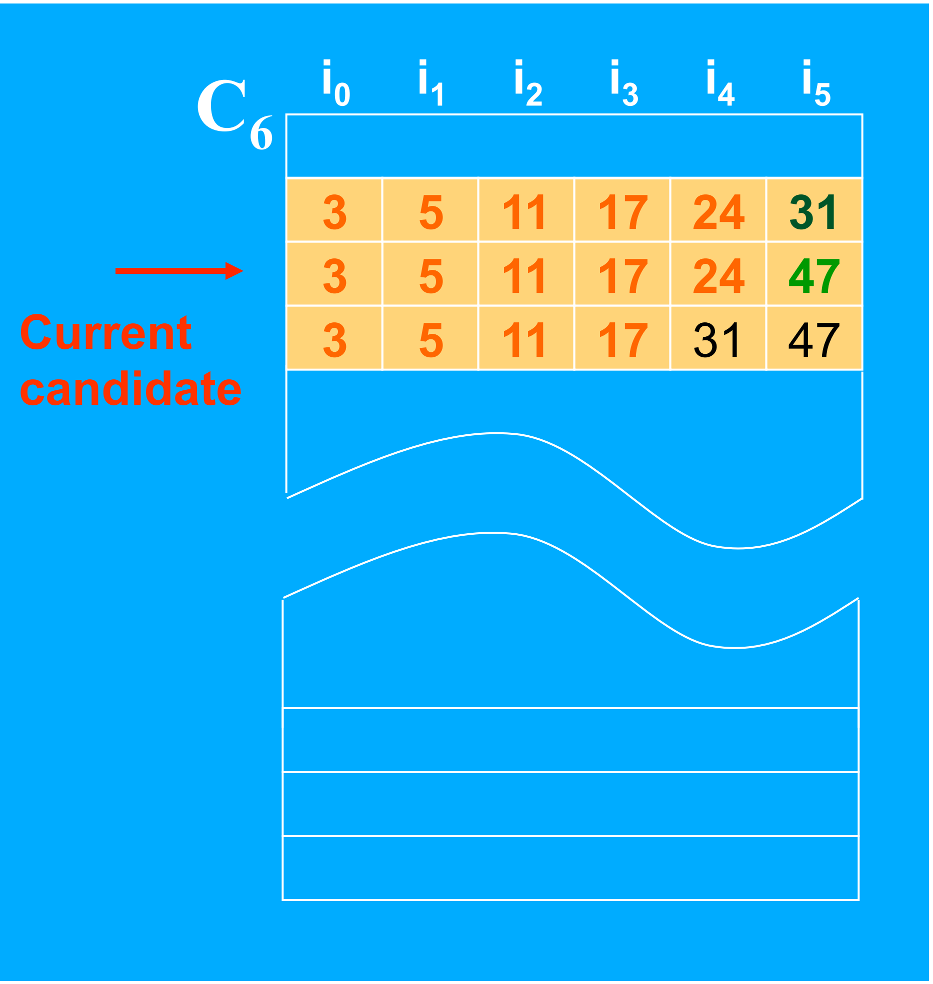


Cache: bitwise TID-list Intersection

- *k*-way intersections
  - intersect tidlists associated with single items
  - low memory requirements, but too many intersections!
- 2-way intersections
  - start from tidlists associated with frequent *(k-1)*-itemsets
  - huge memory requirements, but less intersections!
- DCI ⇒ tradeoff between 2-way and k-way
  - is based upon k-way intersections of bitvectors,
  - BUT caches all the partial intersections corresponding to the various prefixes of the current candidate itemset

Cache size: | **k-2** bitvector di n_k bit

Cache: bitwise TID-list Intersection

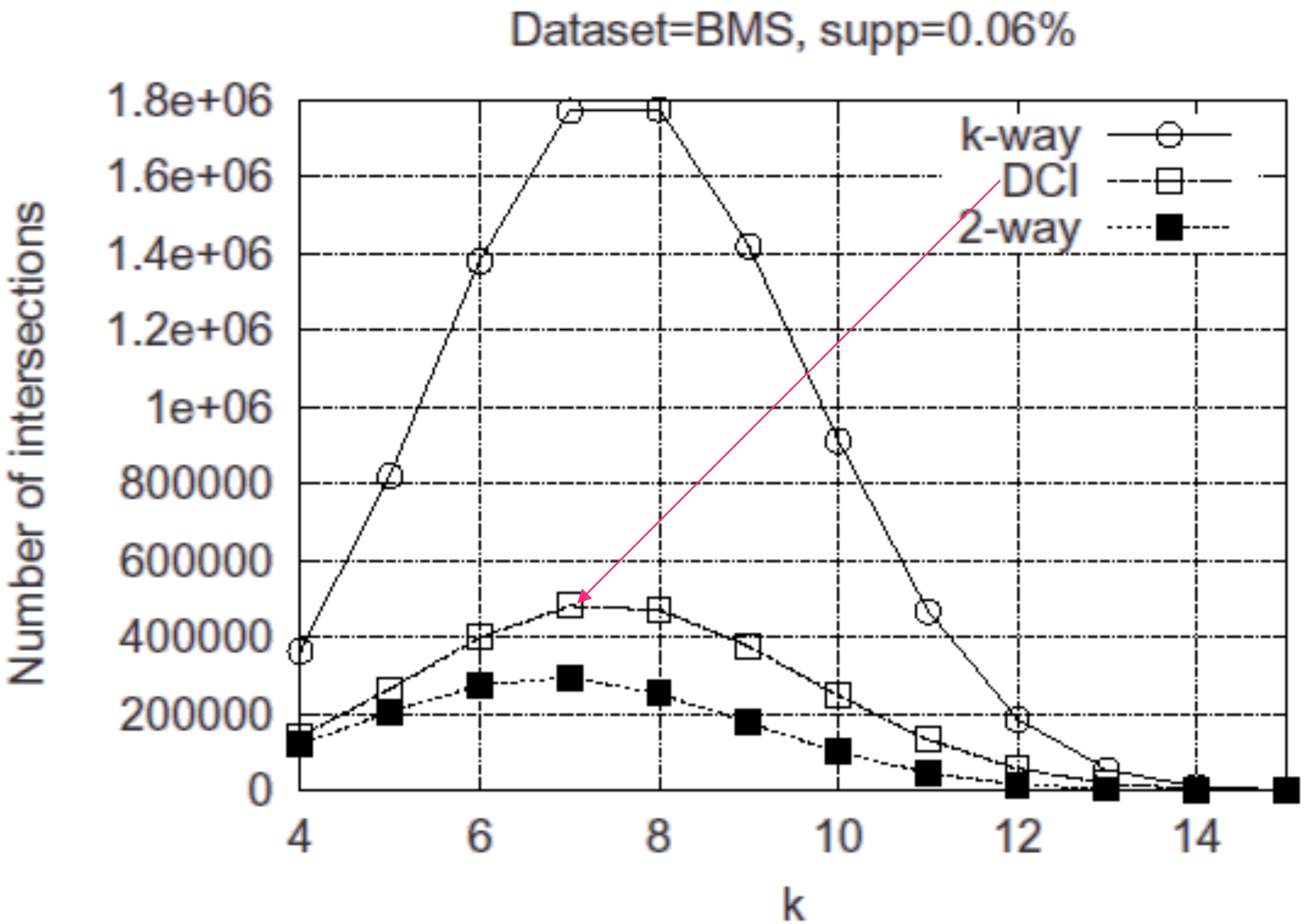


Buffer of (k-2) vectors of nk bits used for caching intermediate intersection results

3 & 5
3 & 5 & 11
3 & 5 & 11& 17
3 & 5 & 11& 17 & 24

Reuse of this cached intersection

DCI: number of intersections



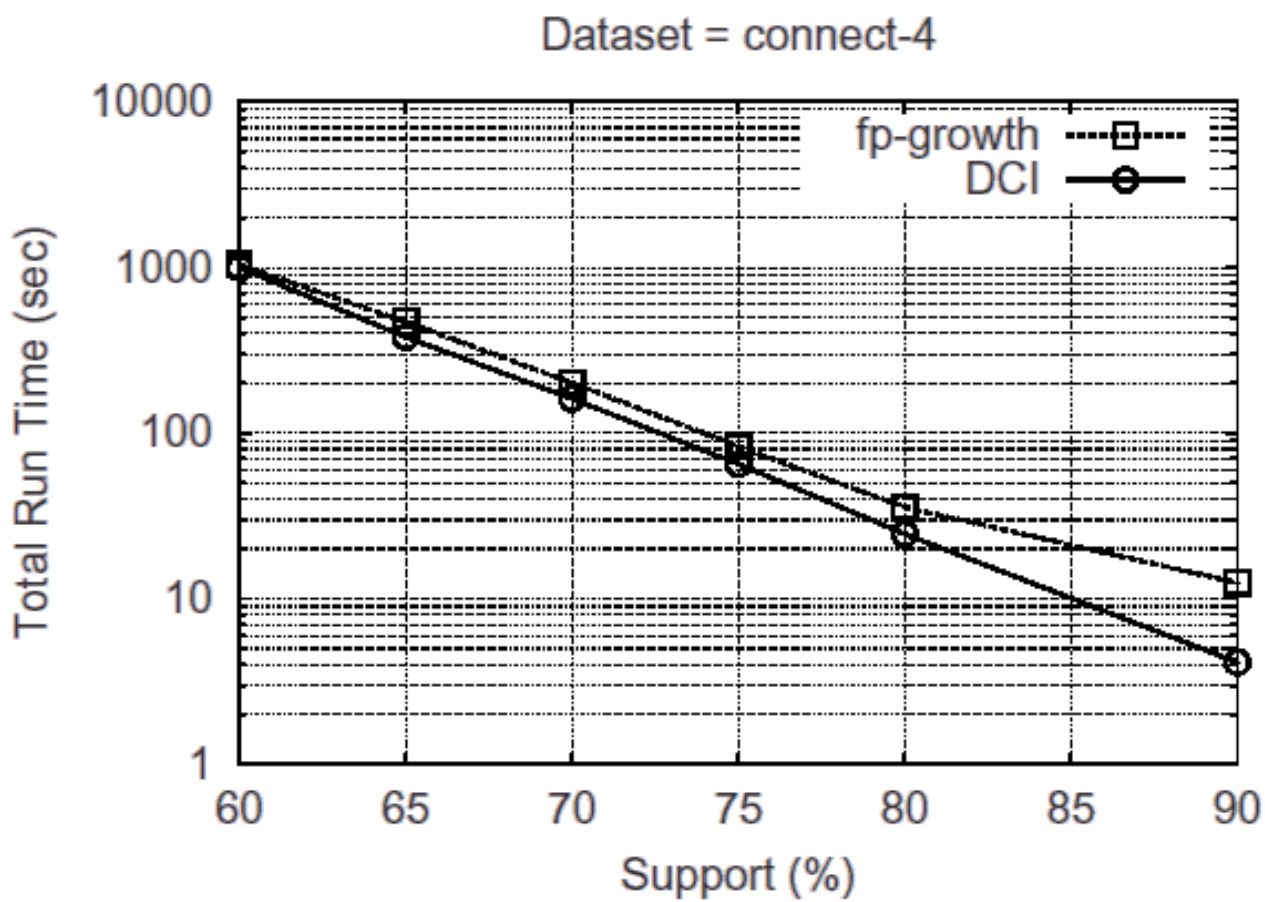
DCI: sparse vs. dense datasets

- **Sparse:**
  - The bit-vectors are sparse
    - A few 1 bits, and long sequences of 0
  - It is possible identify large sections of words equal to *zero*
  - We can skip these sections during the intersections

- **Dense**
  - Strong correlation among the most frequent items, whose associated bit-vectors are dense and very similar
    - Contain a few 0 bits, and long sequences of 1 bits
  - It is possible identify large sections of words equal to *one*
  - We can also skip these sections during the intersections

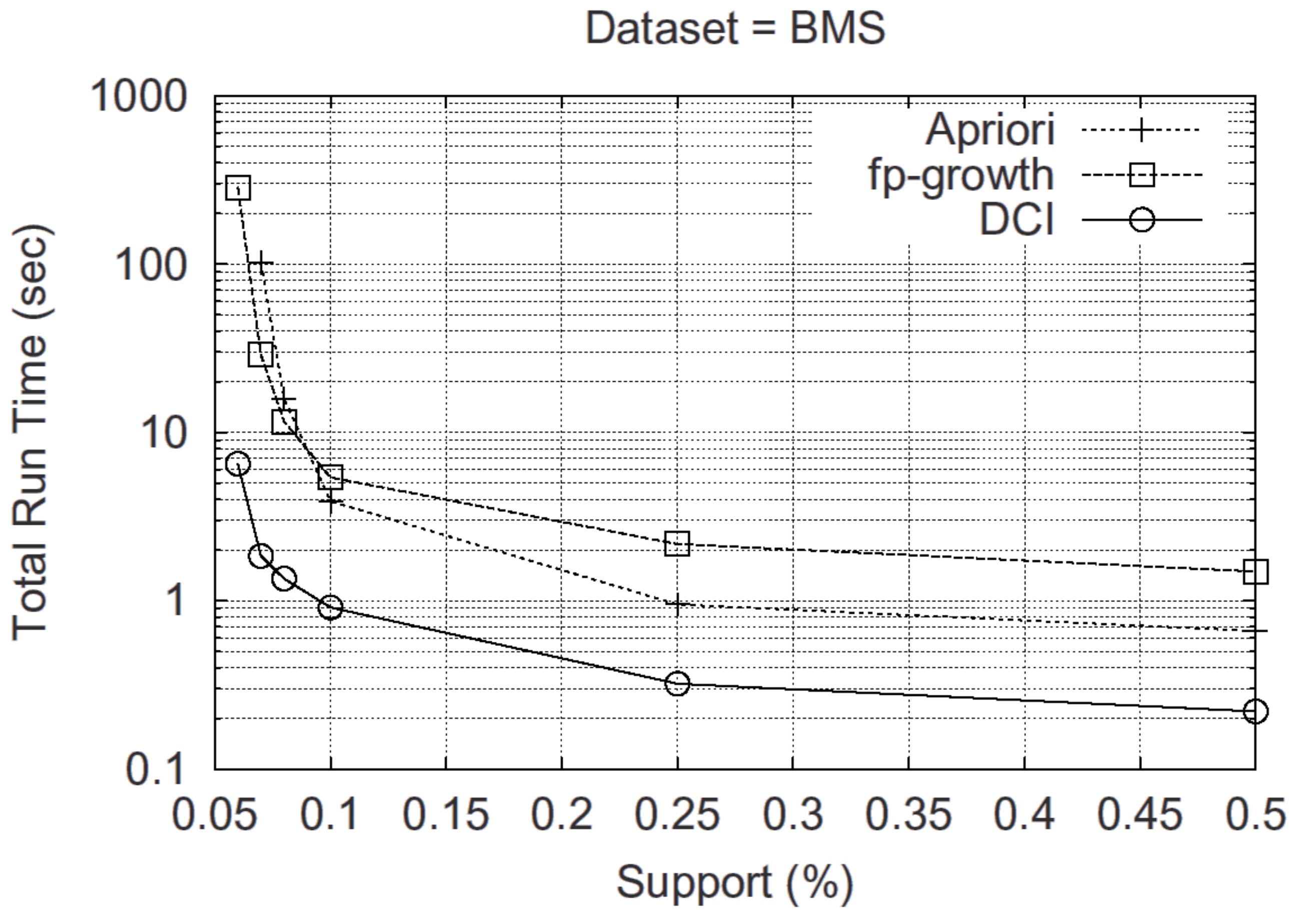
DCI: better than FP-growth for dense datasets

- Dense database
  - **Very long patterns**
- Apriori is inefficient on these datasets



DCI: better than FP-growth for dense datasets

- Sparse Database
  - **Click-stream collected from an e-commerce website**
- Patterns of average length for small supports



Constraints

- **Function C defined on a set of items I:**
  - $C : 2^I \Rightarrow \{true, false\}$
- **Main motivation:**
  - **Focus on further requirements/constraint of the analyst, besides the minimum support constraint, to avoiding flooding him with a lot of uninteresting patterns**
- **Example:**
  - $freq(X) \geq min\_supp$
  - $sum(X.price) \geq m$

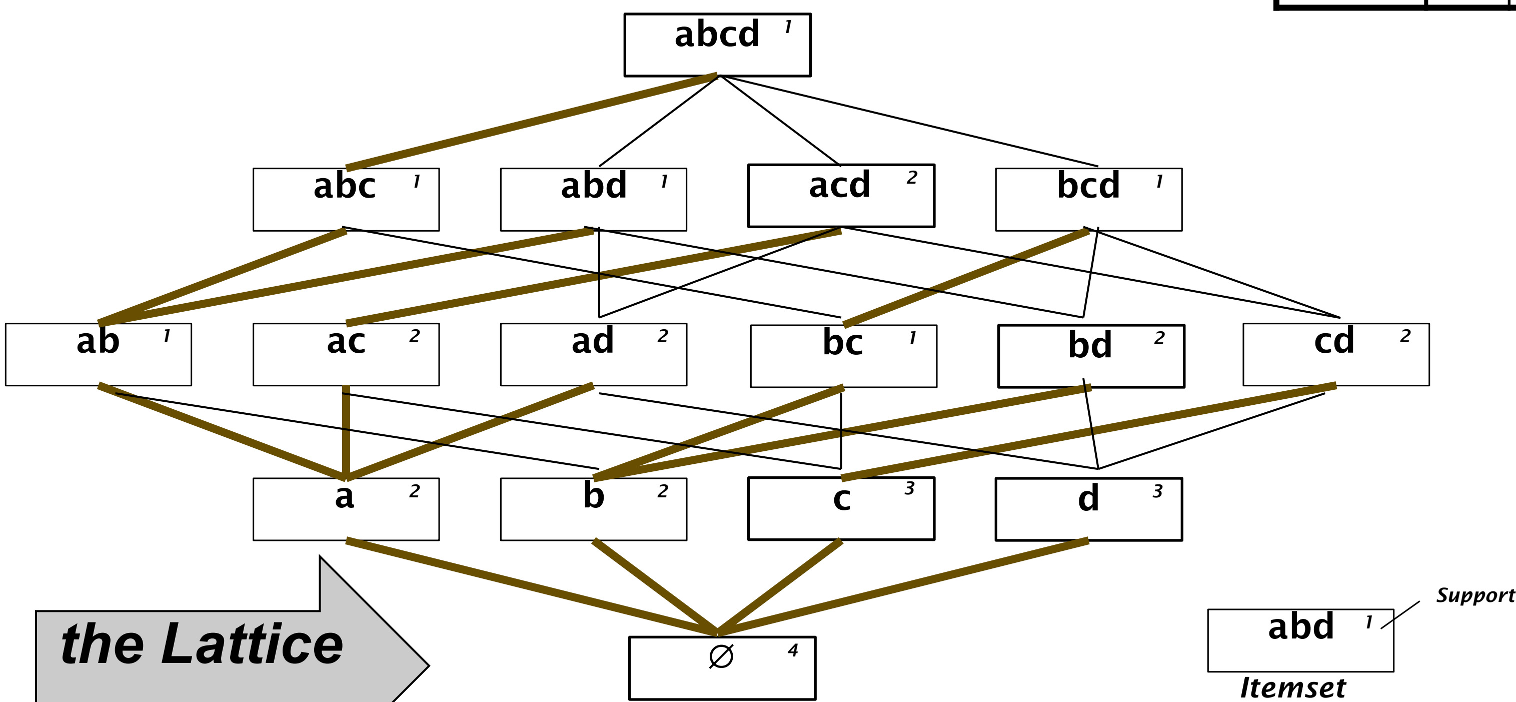
Exploiting the contraints

- **Understand the constraint properties:**
  - **to avoid an approach *generate&test***
  - **to reduce the search space**
  - **to prune the dataset**
- **Example with the minimum support constraint:**
  - **if *freq(a) ≤ min_supp***
    remove from the dataset every occurrence of item *a*, thus reducing the size of the dataset
- **Motivation:**
  - **Performance**
  - **Improved performance -> more constraints**
    -> more expressivity

The search space

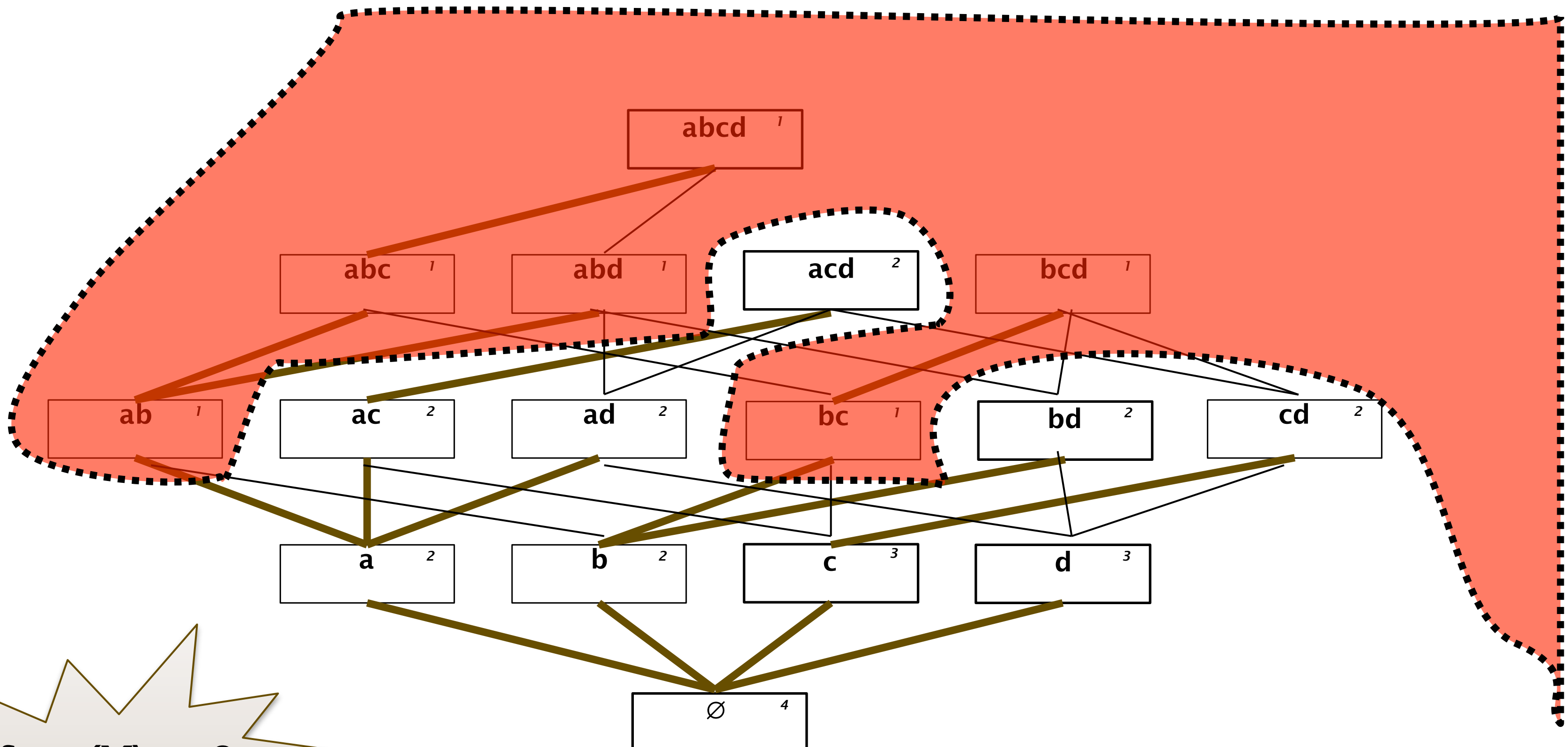
the Dataset

TID	items			
1	b	d		
2	a	b	c	d
3	a	c	d	
4	c			



the Lattice

Reducing the search space



freq(X) ≥ 2

ANTI-MONOTONE constraint

- **Definition:**
  - A constraint C is anti-monotone if $\forall\, X \subset Y:\ C(Y) \Rightarrow C(X)$
- **Example:**
  - *freq(X)* $\geq \sigma$, *sum(X.prices)* $< m$, ...
- **Corollary:**
  - $\forall\, X \subset Y:\ \neg C(X) \Rightarrow \neg C(Y)$

We use this corollary to prune the search space when we visit it level-wise and bottom-up

- **Strategy:**
  - If $\neg C(X)$ ignore X and its supersets
  - Bottom-up visit of the search space

(search space)

  - If an item *i* occurring in transaction *t* is not contained in almost *k* frequent itemsets of length *k*, then *i* will not occur in any frequent itemset of length *k+1* ➔ thus ignore *i* when generating the candidates of length *k+1* (*i* can be removed from the dataset)

(dataset)

MONOTONE constraints

- **Definition:**
  - A constraint *C* is monotone if $\forall\, X \subset Y : C(X) \Rightarrow C(Y)$
- **Corollary**
  - $\forall\, X \subset Y :\ \neg C(Y) \Rightarrow \neg C(X)$
- **Example:**
  - *sum(X.prices) ≥ m*, *max(X.prices) ≥ m*, ...
- **Strategy: Monotone + Anti-monotone.**
  - *bottom-up vs top-down* exploration of the search space

    (search space reduction)

Exante: how to exploit a monotone constraint

- **Idea: exploit the constraints before starting mining**
- **Property:**
  - **Apply the constraint to each transaction**
  - **If transaction Y do not satisfy the monotone constraint $\neg C(Y)$, then no of its subsets will satisfy the constraint ($\forall X \subset Y : \neg C(Y) \Rightarrow \neg C(X)$), and thus Y can be removed**
- **Side effects: when you remove transactions, some items can become infrequent, and thus not useful**
- **Result : virtuous cycle in pruning the dataset**
  - ***iterated step* to prune *transactions*, and subsequent pruning of *items***
- **The step can be repeated for each iteration *k* of Apriori**

Francesco Bonchi, Fosca Giannotti, Alessio Mazzanti, Dino Pedreschi: ***Exante: A Preprocessing Method for Frequent-Pattern Mining.*** *IEEE Intelligent Systems. 20(3): 25-31 (2005)*

Exante: example

item	price
a	5
b	8
c	14
d	30
e	20
f	15
g	6
h	12

tID	Itemset	Total price		
1	b,c,d,~~g~~	~~58~~	52	
~~2~~	~~a,b,d,e~~	~~63~~	~~38~~	
3	b,c,d,~~g~~,~~h~~	~~70~~	~~58~~	52
~~4~~	~~a,e,g~~	~~31~~		
~~5~~	~~c,d,f,g~~	~~65~~	~~50~~	44
6	~~a~~,b,c,d,~~e~~	~~77~~	52	
~~7~~	~~a,b,d,f,g,h~~	~~76~~	~~44~~	
8	b,c,d	52		
~~9~~	~~b,e,f,g~~	~~49~~	~~14~~	

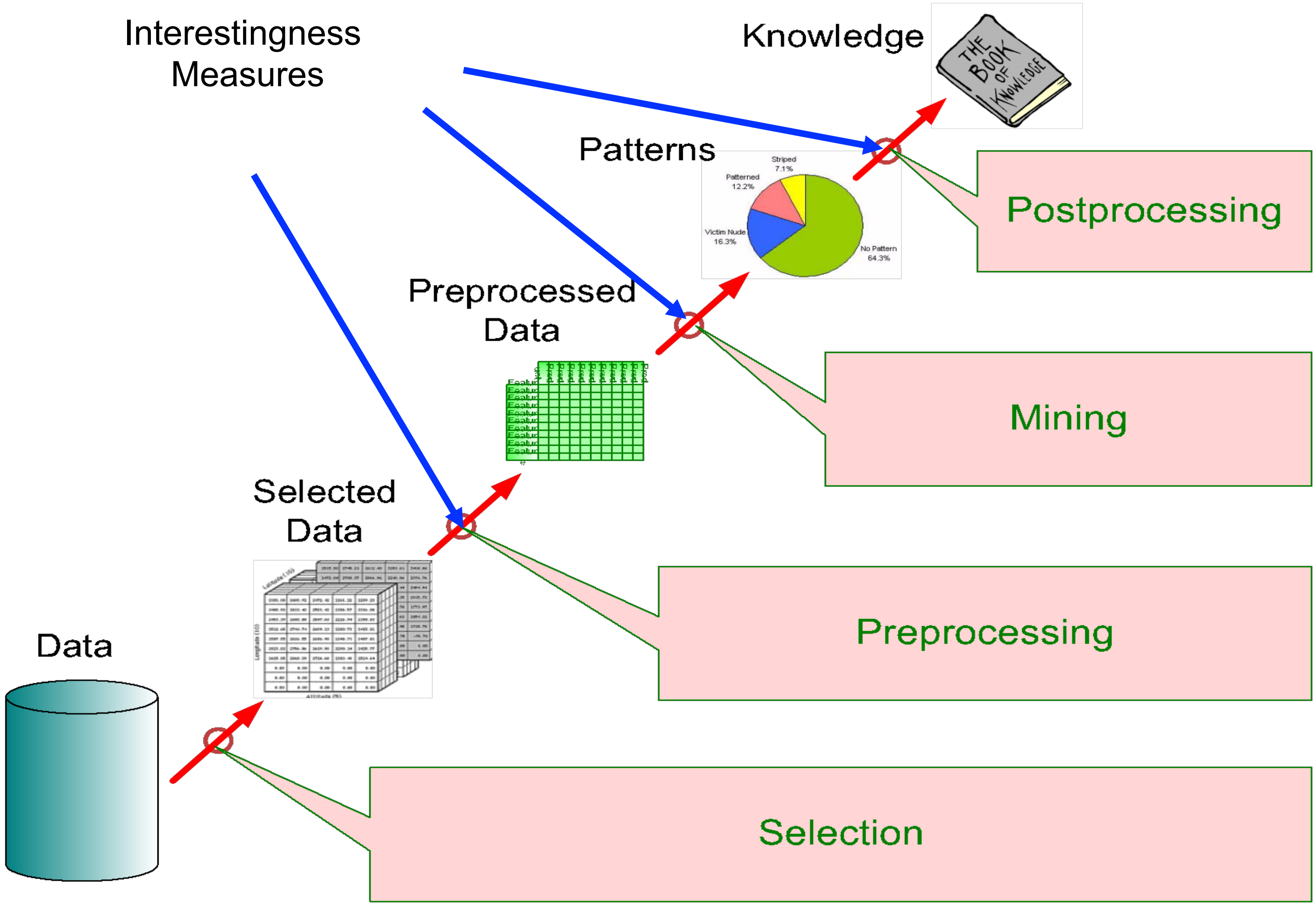
- Min_sup = 4
- $C_M \equiv sum(X.price) \geq 45$

Item	Support			
~~a~~	~~4~~	3	†	†
b	7	~~7~~	4	4
c	5	5	5	4
d	7	~~7~~	5	4
~~e~~	~~4~~	3	†	†
~~f~~	3	3	†	†
~~g~~	~~6~~	5	3	†
~~h~~	2	~~2~~	†	†

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

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

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

Can apply 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

Association Rule: Tea $\rightarrow$ Coffee

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

but P(Coffee) = 0.9

$\Longrightarrow$ Although confidence is high, rule is misleading

$\Longrightarrow$ $P(\text{Coffee}|\overline{\text{Tea}}) = 75/80 = 0.9375$

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)**

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

  - **$P(S \wedge B) = P(S) \times P(B)$ => Statistical independence**
  - **$P(S \wedge B) > P(S) \times P(B)$ => Positively correlated**
  - **$P(S \wedge B) < P(S) \times P(B)$ => Negatively correlated**

Statistical-based Measures

- **Measures that take into account statistical dependence**

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

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

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

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

Example: Lift/Interest

	Coffee	$\overline{\text{Coffee}}$	
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 it is negatively correlated)

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) => Lift = 1

Rare itemsets with low counts (low probability) which per chance occur a few times (or only once) together can produce enormous lift values.

Drawback of ϕ-Coefficient

- ϕ-coefficient is analogous to correlation coefficient for continuous variables

	Y	$\overline{Y}$	
X	60	10	70
$\overline{X}$	10	20	30
	70	30	100

	Y	$\overline{Y}$	
X	20	10	30
$\overline{X}$	10	60	70
	30	70	100

$$\phi = \frac{0.6 - 0.7 \times 0.7}{\sqrt{0.7 \times 0.3 \times 0.7 \times 0.3}} = 0.5238$$

$$\phi = \frac{0.2 - 0.3 \times 0.3}{\sqrt{0.7 \times 0.3 \times 0.7 \times 0.3}} = 0.5238$$

ϕ Coefficient is the same for both tables