Introduction to Association Rules: Concept & the Apriori Algorithm

Dr. Vassilis S. Kontogiannis
Reader in Computational Intelligence
Email: V.Kodogiannis@westminster.ac.uk
https://scholar.google.co.uk/citations?user=meTTcLAAAAAJ&hl=en&oi=ao

Overview

- What is association rule mining?
- Frequent itemsets, support, and confidence
- Mining association rules
- The “Apriori” algorithm
- Rule generation

Question of the lecture

How can we mine **interesting patterns** and **useful rules** from data?

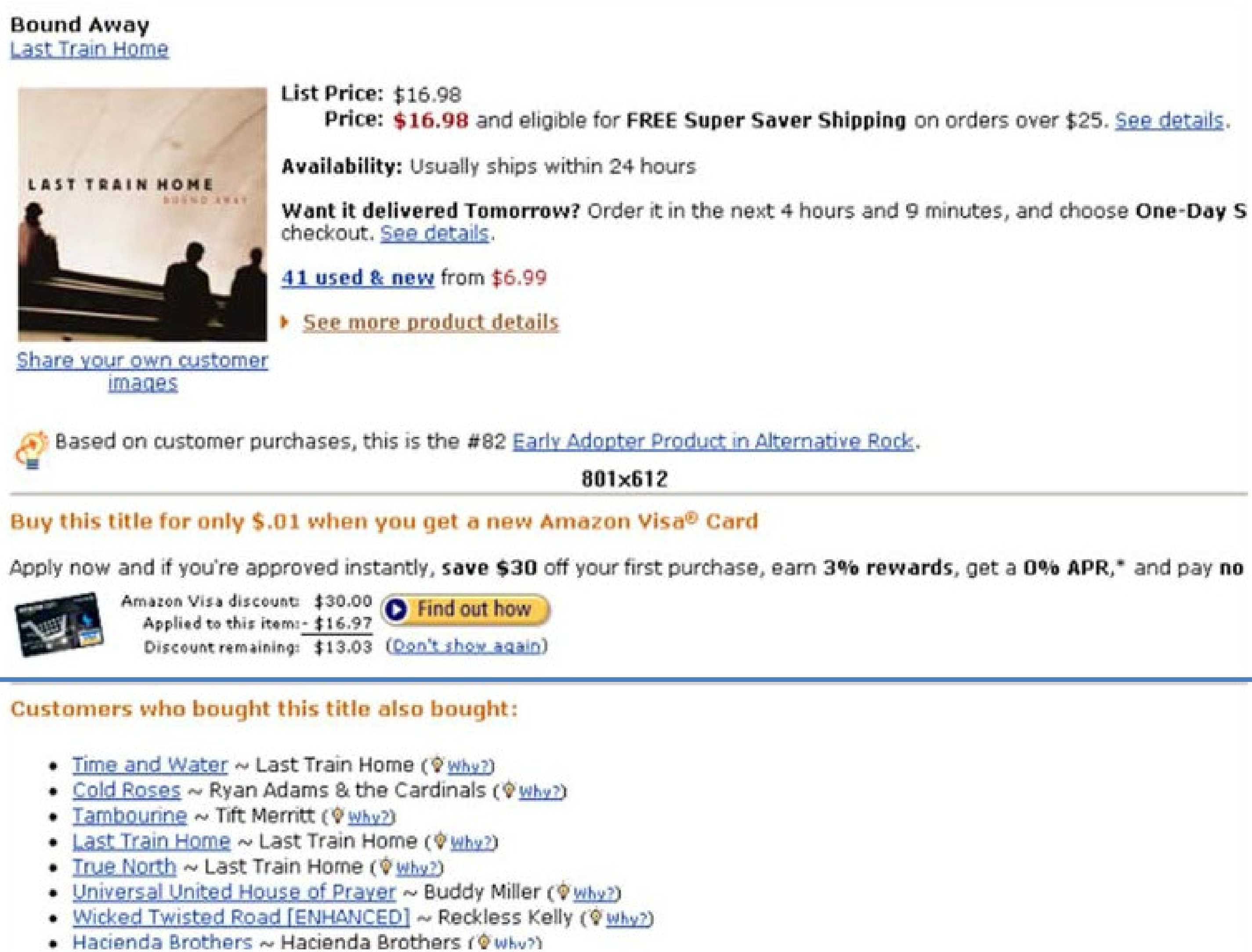
Motivational Example

You run an on-line store, and want to increase sales. You decide on **associative advertising**: show ads of relevant products **before** your users search for these

→

Easy, knowing the left-hand side. What if we don’t?

Used in many recommender systems



Applications

- **Market Basket Analysis:** given a database of customer transactions, where each transaction is a set of items the goal is to find groups of items which are frequently purchased together.
- **Telecommunication** (each customer is a transaction containing the set of phone calls)
- **Credit Cards/ Banking Services** (each card/account is a transaction containing the set of customer's payments)
- **Medical Treatments** (each patient is represented as a transaction containing the ordered set of diseases)
- **Basketball-Game Analysis** (each game is represented as a transaction containing the ordered set of ball passes)

Market Basket Analysis

- **Analysis of customer buying habits by finding associations and correlations between the different items that customers place in their "shopping basket"**

Milk, eggs, sugar, bread

Customer1

Milk, eggs, cereal, bread

Customer2

Eggs, sugar

Customer3

Market Basket Analysis

- **Retail – each customer purchases different set of products, different quantities, different times**
- **MBA uses this information to:**
  - Identify who customers are (not by name)
  - Understand why they make certain purchases
  - Gain insight about its merchandise (products):
    - Fast and slow movers
    - Products which are purchased together
    - Products which might benefit from promotion
  - Take action:
    - Store layouts
    - Which products to put on specials, promote, coupons…
- **Combining all of this with a customer loyalty card it becomes even more valuable**

Example of market-basket transactions

Bread
Peanuts
Milk
Fruit
Jam

Bread
Jam
Soda
Chips
Milk
Fruit

Steak
Jam
Soda
Chips
Bread

Jam
Soda
Peanuts
Milk
Fruit

Jam
Soda
Chips
Milk
Bread

Fruit
Soda
Chips
Milk

Fruit
Soda
Peanuts
Milk

Fruit
Peanuts
Cheese
Yogurt

What is association mining?

Finding frequent patterns, associations, correlations, or causal structures among sets of items or objects in transaction databases, relational databases, and other information repositories.

- **Extract information on purchasing behaviour**

  "**IF** buys beer and sausage, **THEN** also buy mustard with high probability“

- **Useful:**

  "On Thursdays, grocery store consumers often purchase diapers and beer together."

- **Trivial:**

  "Customers who purchase maintenance agreements are very likely to purchase large appliances."

What is association rule mining?

TID	Items
1	Bread, Peanuts, Milk, Fruit, Jam
2	Bread, Jam, Soda, Chips, Milk, Fruit
3	Steak, Jam, Soda, Chips, Bread
4	Jam, Soda, Peanuts, Milk, Fruit
5	Jam, Soda, Chips, Milk, Bread
6	Fruit, Soda, Chips, Milk
7	Fruit, Soda, Peanuts, Milk
8	Fruit, Peanuts, Cheese, Yogurt

Examples

$\{bread\} \Rightarrow \{milk\}$

$\{soda\} \Rightarrow \{chips\}$

$\{bread\} \Rightarrow \{jam\}$

- 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

Measures for Association Rules - I

Rule Evaluation Metrics

Support (s): Fraction of transactions that contain both X and Y

$$P(X \cup Y) = \frac{\#\,trans\ containing\ (X \cup Y)}{\#\,trans\ in\ D}$$

or

$$Support(A) = \frac{\text{number of transaction which contain A}}{\text{number of all transaction}}$$

Support calculates how often the product is purchased.

Confidence (c): Measures how often items in Y appear in transactions that contain X

$$P(X \mid Y) = \frac{\#\,trans\ containing\ (X \cup Y)}{\#\,trans\ containing\ X}$$

or

$$Confidence(A \rightarrow B) = \frac{Support(\text{A and B})}{Support(A)}$$

Measures for Association Rules - II

Confidence tells us what proportion of transactions which consist of product X, also contain product Y. It signifies the likelihood of item Y being purchased when item X is purchased. It can give some important insights, but it also has a major drawback. It only takes into account the popularity of the itemset *X* and not the popularity of *Y*.

As the last step, we calculate the "lift'. It is the value that tells us how likely item B is bought together with item A. Values greater than one indicate that the items are likely to be purchased together.

$$Lift(A \rightarrow B) = \frac{Support(\text{A and B})}{Support(A) \times Support(B)}$$

Measures for Association Rules - III

- How much better than chance is a rule?
- Lift (improvement) tells us how much better a rule is at predicting the result than just assuming the result in the first place
- **Lift** is the ratio of the records that support the entire rule to the number that would be expected, assuming there was no relationship between the products
- Calculating lift. When lift > 1 then the rule is better at predicting the result than guessing. When lift < 1, the rule is doing worse than informed guessing.

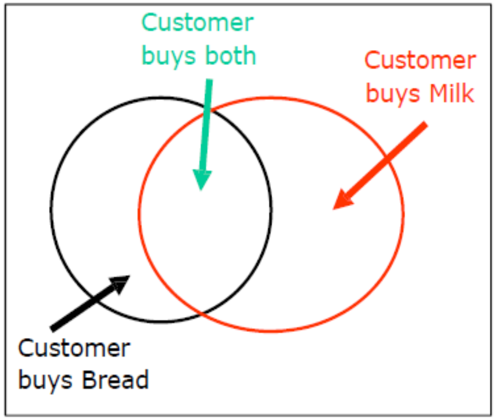
Definition: Frequent Itemset

- Itemset
  - A collection of one or more items, e.g., {milk, bread, jam}
  - k-itemset, an itemset that contains k items
- Support count (σ)
  - Frequency of occurrence of an itemset
  - σ({Milk, Bread}) = 3
    σ({Soda, Chips}) = 4
- Support
  - Fraction of transactions that contain an itemset
  - s({Milk, Bread}) = 3/8
    s({Soda, Chips}) = 4/8
- Frequent Itemset
  - An itemset whose support is greater than or equal to a minsup threshold

TID	Items
1	Bread, Peanuts, Milk, Fruit, Jam
2	Bread, Jam, Soda, Chips, Milk, Fruit
3	Steak, Jam, Soda, Chips, Bread
4	Jam, Soda, Peanuts, Milk, Fruit
5	Jam, Soda, Chips, Milk, Bread
6	Fruit, Soda, Chips, Milk
7	Fruit, Soda, Peanuts, Milk
8	Fruit, Peanuts, Cheese, Yogurt

What is an association rule?

- Implication of the form X $\Rightarrow$ Y, where X and Y are itemsets
- Example, {bread} $\Rightarrow$ {milk}
- Rule Evaluation Metrics, Suppor & Confidence



- Support (s)
  - Fraction of transactions that contain both X and Y

$$s = \frac{\sigma(\{\text{Bread}, \text{Milk}\})}{\#\text{ of transactions}} = 0.38$$

- Confidence (c)
  - Measures how often items in Y appear in transactions that contain X

$$c = \frac{\sigma(\{\text{Bread}, \text{Milk}\})}{\sigma(\{\text{Bread}\})} = 0.75$$

What is the goal?

- 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
- Brute-force approach is computationally prohibitive!

Mining Association Rules

$\{Bread, Jam\} \Rightarrow \{Milk\}$ s=0.4 c=0.75

$\{Milk, Jam\} \Rightarrow \{Bread\}$ s=0.4 c=0.75

$\{Bread\} \Rightarrow \{Milk, Jam\}$ s=0.4 c=0.75

$\{Jam\} \Rightarrow \{Bread, Milk\}$ s=0.4 c=0.6

$\{Milk\} \Rightarrow \{Bread, Jam\}$ s=0.4 c=0.5

- All the above rules are binary partitions of the same itemset:

{Milk, Bread, Jam}

- Rules originating from the same itemset have identical support but can have different confidence
- We can decouple the support and confidence requirements!

How Good is an Association Rule?

Customer	Items Purchased
1	Coca-Cola (CC), soda
2	Milk, CC, window cleaner
3	CC, detergent
4	CC, detergent, soda
5	Window cleaner, soda

POS Transactions

Co-occurrence of Products

	CC	Window cleaner	Milk	Soda	Detergent
CC	4	1	1	2	2
Window cleaner	1	2	1	1	0
Milk	1	1	1	0	0
Soda	2	1	0	3	1
Detergent	2	0	0	1	2

How Good is an Association Rule?

	CC	Window cleaner	Milk	Soda	Detergent
CC	4	1	1	2	2
Window cleaner	1	2	1	1	0
Milk	1	1	1	0	0
Soda	2	1	0	3	1
Detergent	2	0	0	1	2

Simple patterns:

1. CC and soda are more likely purchased together than any other two items
2. Detergent is never purchased with milk or window cleaner
3. Milk is never purchased with soda or detergent

How Good is an Association Rule?

Customer	Items Purchased
1	CC, soda
2	Milk, CC, window cleaner
3	CC, detergent
4	CC, detergent, soda
5	Window cleaner, soda

← POS Transactions

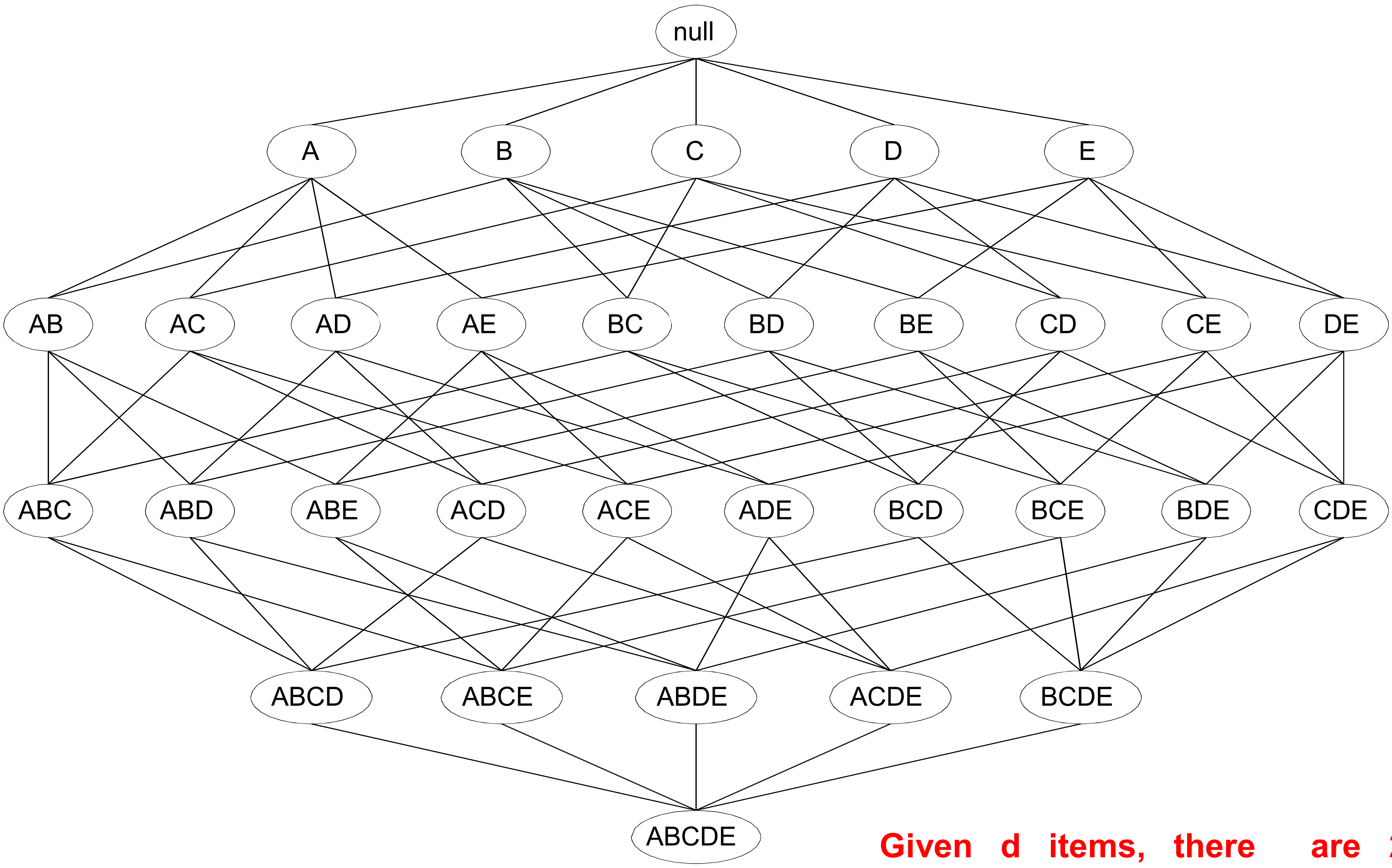
- What is the confidence for this rule:
  - If a customer purchases soda, then customer also purchases CC
  - 2 out of 3 soda purchases also include CC, so 67%
- What about the confidence of this rule reversed?
  - 2 out of 4 CC purchases also include soda, so 50%
- **Confidence** = Ratio of the number of transactions with all the items to the number of transactions with just the “if” items

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

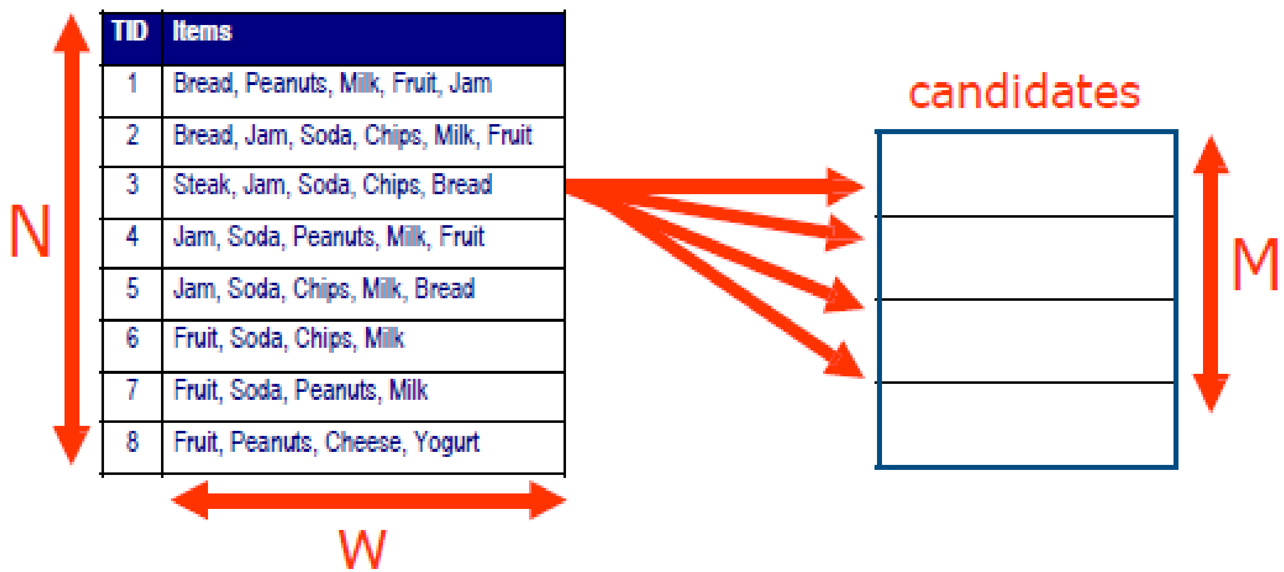
- Frequent itemset generation is still computationally expensive

Frequent Itemset Generation



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

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

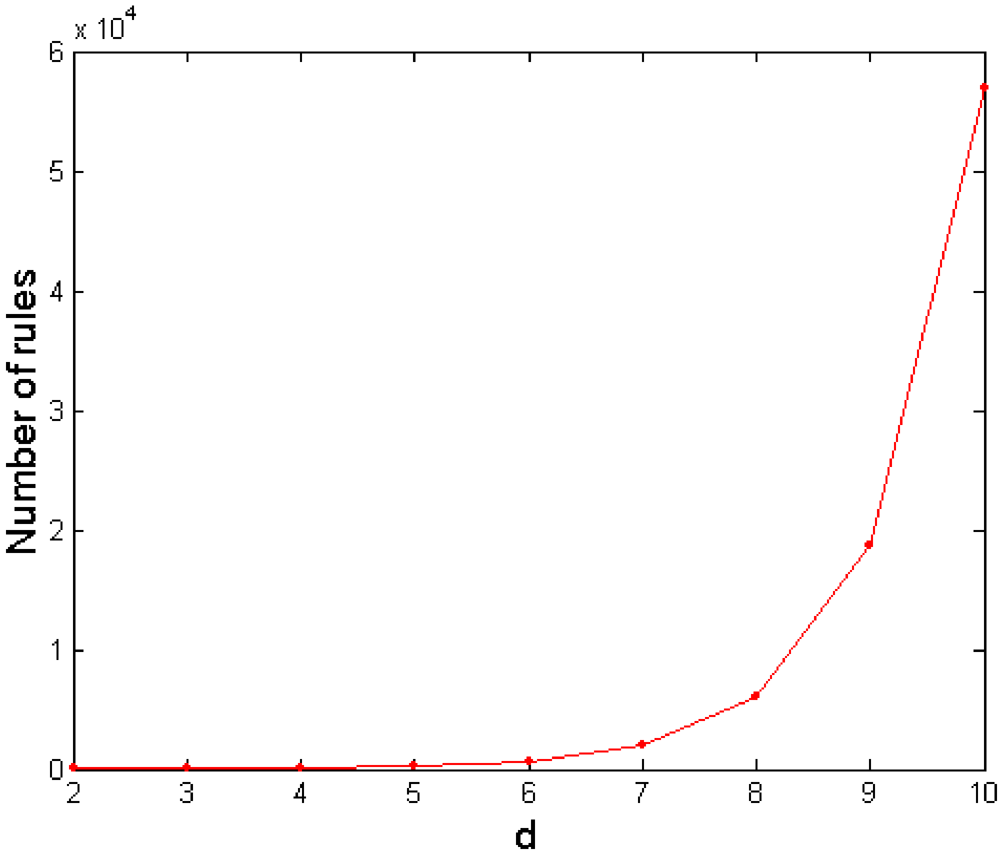


TID	Items
1	Bread, Peanuts, Milk, Fruit, Jam
2	Bread, Jam, Soda, Chips, Milk, Fruit
3	Steak, Jam, Soda, Chips, Bread
4	Jam, Soda, Peanuts, Milk, Fruit
5	Jam, Soda, Chips, Milk, Bread
6	Fruit, Soda, Chips, Milk
7	Fruit, Soda, Peanuts, Milk
8	Fruit, Peanuts, Cheese, Yogurt

  - 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:



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

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
- 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 the 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

Anti-Monotone Property

- Any subset of a *frequent* itemset must be also *frequent* — an anti-monotone property
  - Any transaction containing {beer, diaper, milk} also contains {beer, diaper}
  - {beer, diaper, milk} is frequent → {beer, diaper} must also be frequent
- In other words, any superset of an *infrequent* itemset must also be *infrequent*
  - No superset of any infrequent itemset should be generated or tested
  - Many item combinations can be pruned!

Terminology - I

k-itemset :   a set of *k* items. E.g.

{beer, cheese, eggs} is a 3-itemset

{cheese} is a 1-itemset

{honey, ice-cream} is a 2-itemset

support: an itemset has support *s*% if *s*% of the records in the DB contain that itemset.

minimum support: the Apriori algorithm starts with the specification of a minimum level of support, and will focus on itemsets with this level or above.

Terminology - II

large itemset: doesn't mean an itemset with many items. It means one whose support is at least minimum support.

L_k : the set of all large *k*-itemsets in the DB.

C_k : a set of *candidate* large *k*-itemsets. In the algorithm we will look at, it generates this set, which contains all the *k*-itemsets that might be large, and then eventually generates the set above.

The Apriori algorithm

Find all large 1-itemsets

For (k = 2 ; while L_{k-1} is non-empty; k++)

{C_k = apriori-gen(L_{k-1})

For each c in C_k, initialise $c.count$ to zero

For all records r in the DB

{C_r = subset(C_k, r); For each c in C_r , $c.count$++ }

Set L_k := all c in C_k whose *count* >= *minsup*

} /* end -- return all of the L_k sets.

The Apriori algorithm - steps

1: Find all large 1-itemsets

2: For (k = 2 ; while L_{k-1} is non-empty; k++)

3 {C_k = `apriori-gen`(L_{k-1})

Generate candidate 2-itemsets

4 For each c in C_k, initialise $c.count$ to zero

5 For all records r in the DB

6 {C_r = `subset`(C_k, r); For each c in C_r , $c.count$++ }

Prune them to leave the valid ones

7 Set L_k := all c in C_k whose $count$ >= $minsup$

(those with enough support)

8 } /* end -- return all of the L_k sets.

The Apriori algorithm - steps

```
Find all large 1-itemsets
For (k = 2 ; while L_{k-1} is non-empty; k++)
    {C_k = apriori-gen(L_{k-1})
     For each c in C_k, initialise c.count to zero
     For all records r in the DB
      {C_r = subset(C_k, r); For each c in C_r , c.count++ }
        Set L_k := all c in C_k whose count >= minsup
    } /* end -- return all of the L_k sets.
```

Generate candidate 3-itemsets

Prune them to leave the valid ones (those with enough support)

The Apriori algorithm - steps

```
Find all large 1-itemsets
For (k = 2 ; while L_{k-1} is non-empty; k++)
    {C_k = apriori-gen(L_{k-1})
     For each c in C_k, initialise c.count to zero
     For all records r in the DB
      {C_r = subset(C_k, r); For each c in C_r , c.count++ }
        Set L_k := all c in C_k whose count >= minsup
    } /* end -- return all of the L_k sets.
```

Generate candidate 4-itemsets

Prune them to leave the valid ones (those with enough support)

… etc …

The Apriori algorithm - explained

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. Use L_k to generate a collection of candidate itemsets C_{k+1} of size $k+1$
5. $k = k+1$

Apriori: Generating Frequent Item Sets

For *k* products…

1. User sets a minimum support criterion
2. Next, generate list of one-item sets that meet the support criterion
3. Use the list of one-item sets to generate list of two-item sets that meet the support criterion
4. Use list of two-item sets to generate list of three-item sets
5. Continue up through *k*-item sets

The Apriori algorithm - Example

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

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

Triplets (3-itemsets)

Itemset	Count
{Bread,Milk,Diaper}	2

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

If every subset is considered,

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

With support-based pruning,

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

The Apriori algorithm - Example

Database D

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

minsup=2=50%

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

Closed Itemset

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

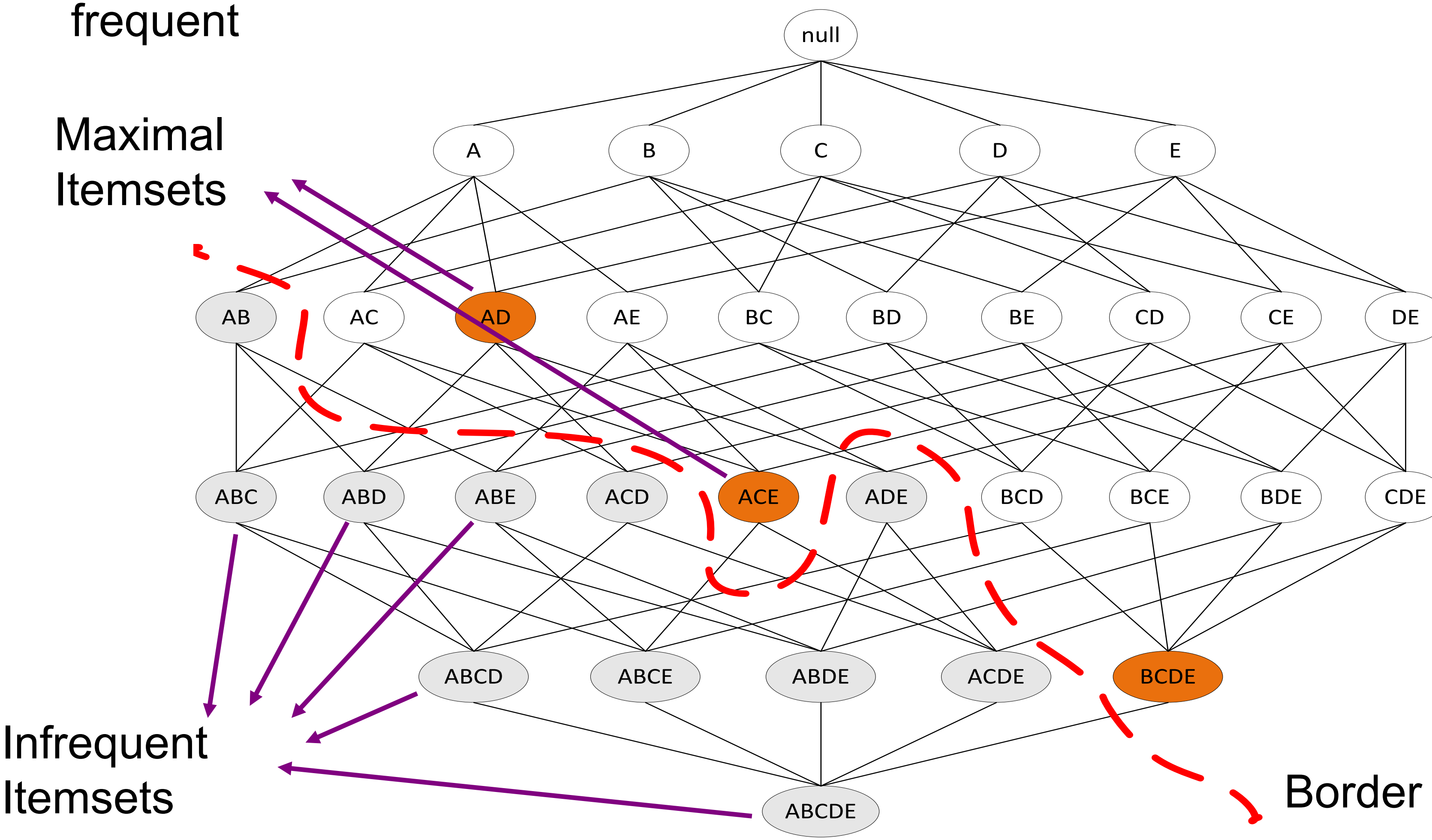
TID	Items
1	{A,B}
2	{B,C,D}
3	{A,B,C,D}
4	{A,B,D}
5	{A,B,C,D}

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}	2
{A,B,C,D}	2

Maximal Frequent Itemset

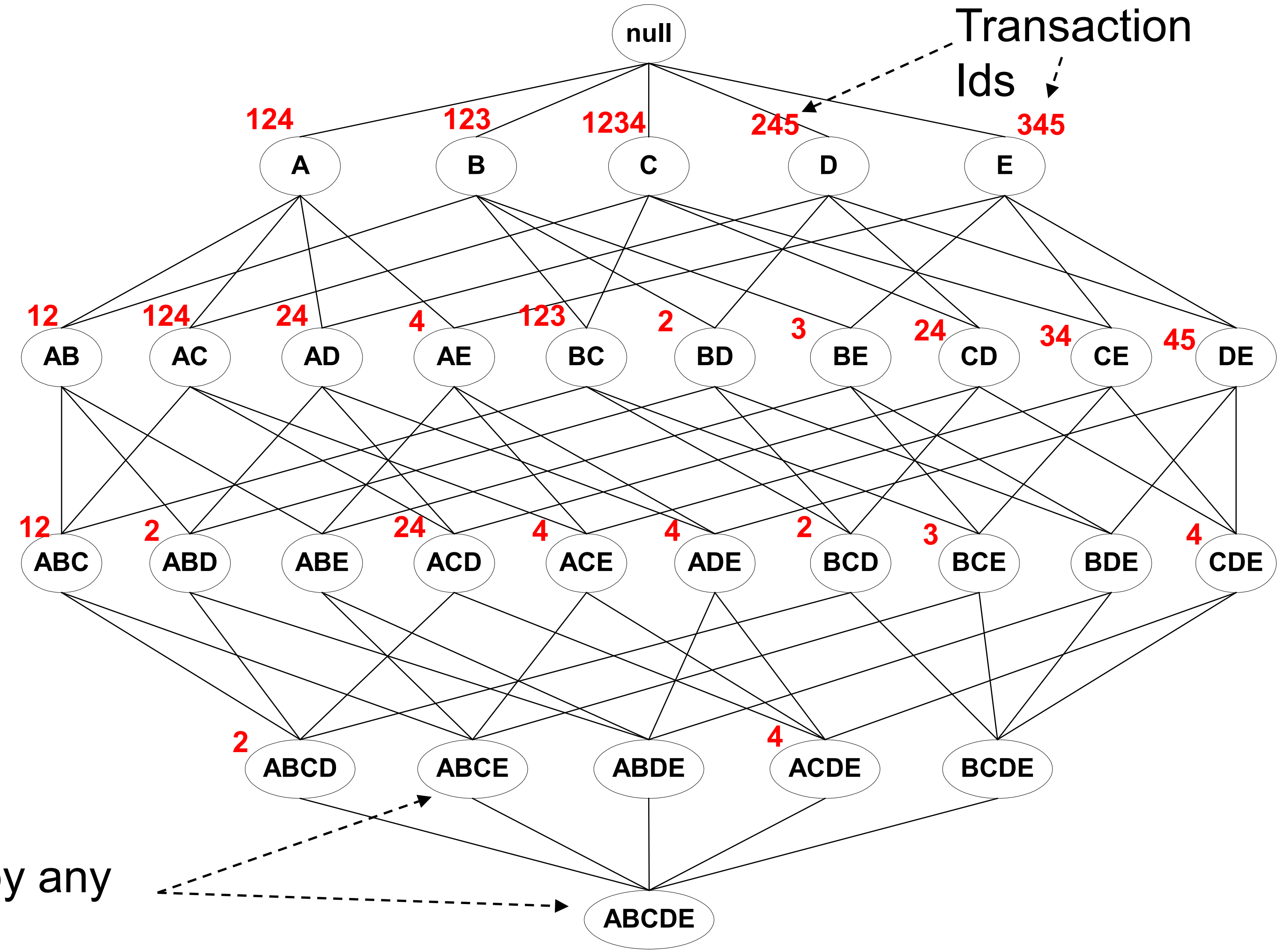
An itemset is maximal frequent if none of its immediate supersets is frequent



Maximal: no superset has this property

Maximal vs Closed Itemsets

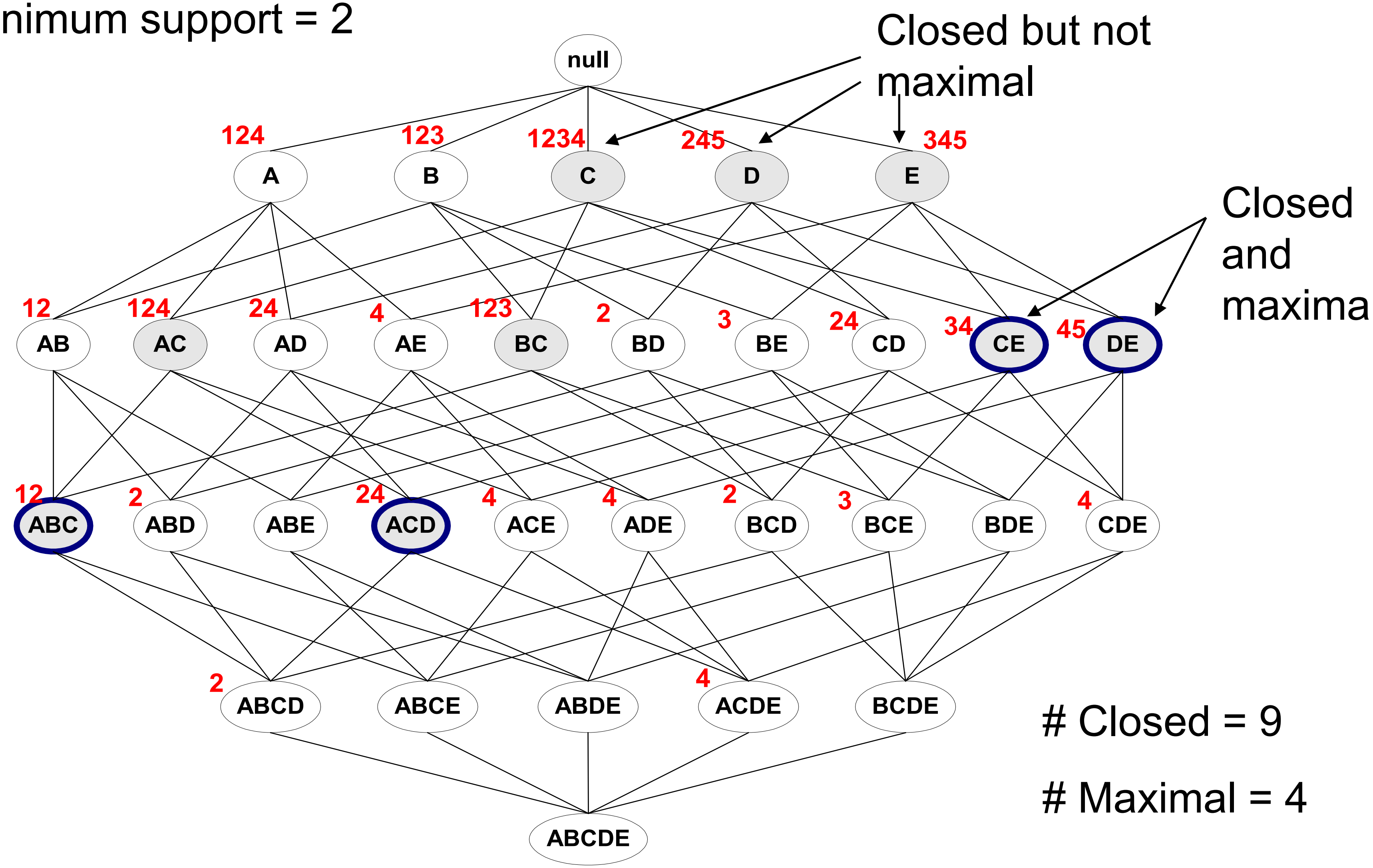
TID	Items
1	ABC
2	ABCD
3	BCE
4	ACDE
5	DE



Not supported by any transactions

Maximal vs Closed Frequent Itemsets

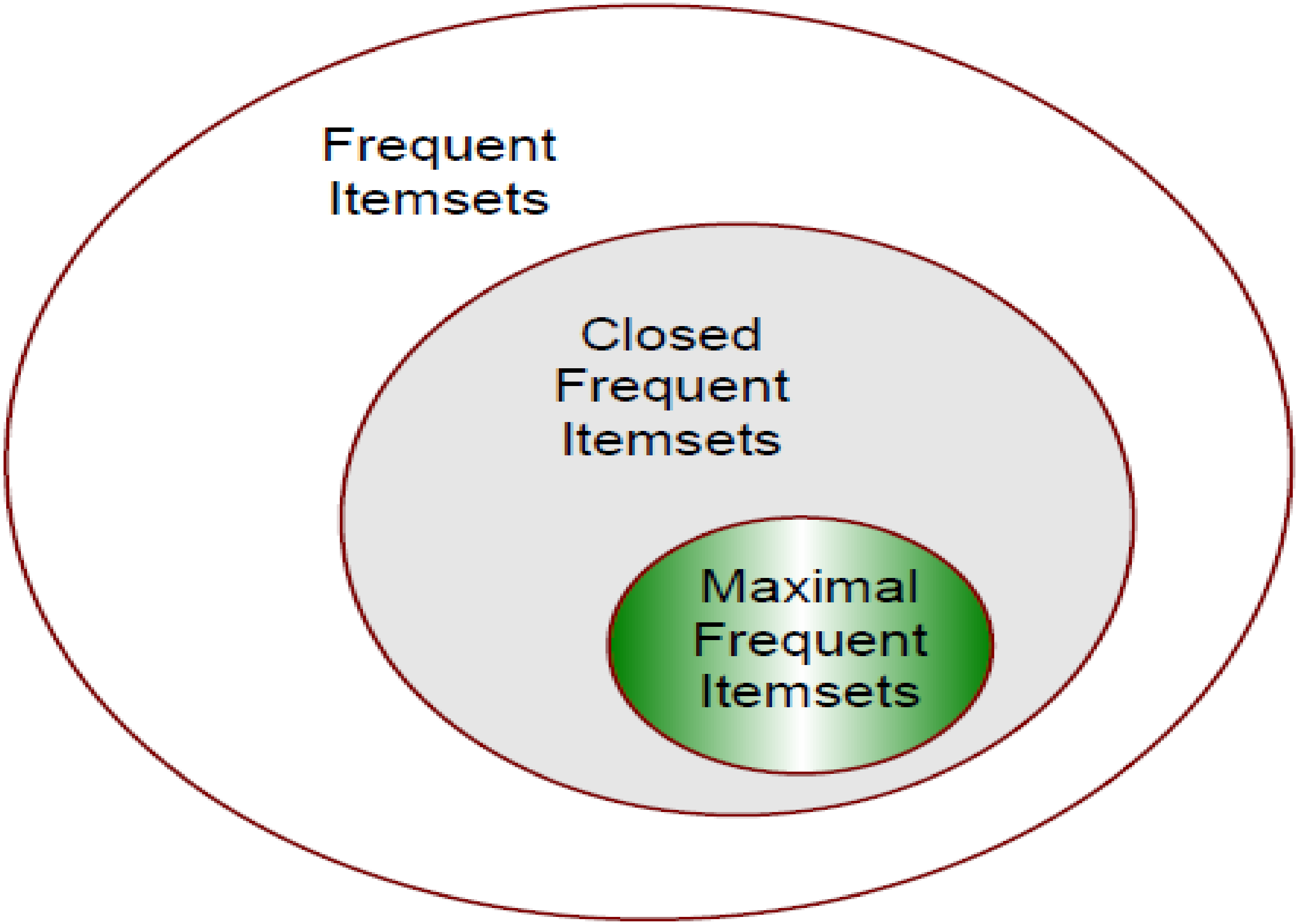
Minimum support = 2



Closed = 9

Maximal = 4

Maximal vs Closed Itemsets

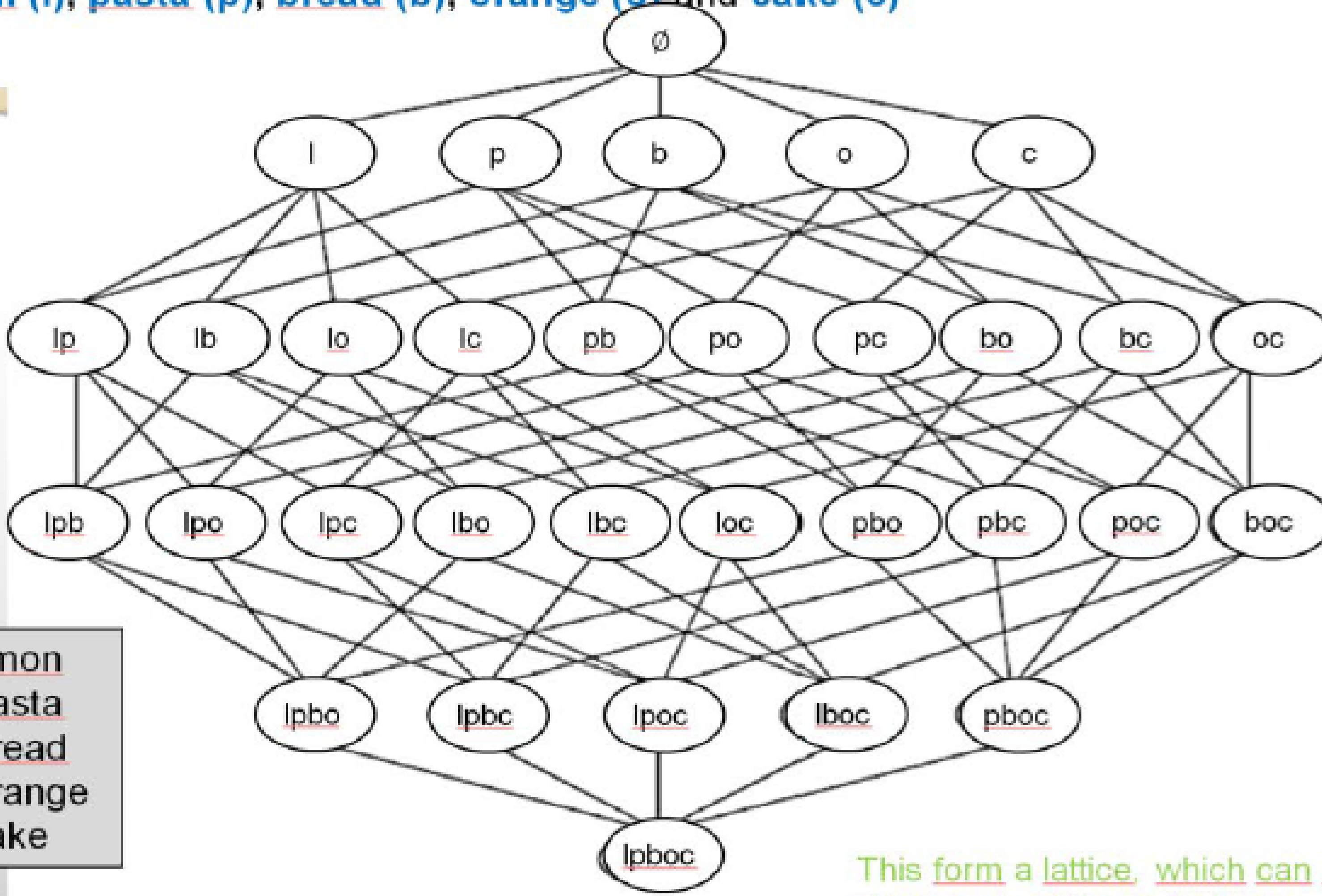


Example: Step-by-Step

Transaction	Items appearing in the transaction
T1	{pasta, lemon, bread, orange}
T2	{pasta, lemon}
T3	{pasta, orange, cake}
T4	{pasta, lemon, orange, cake}

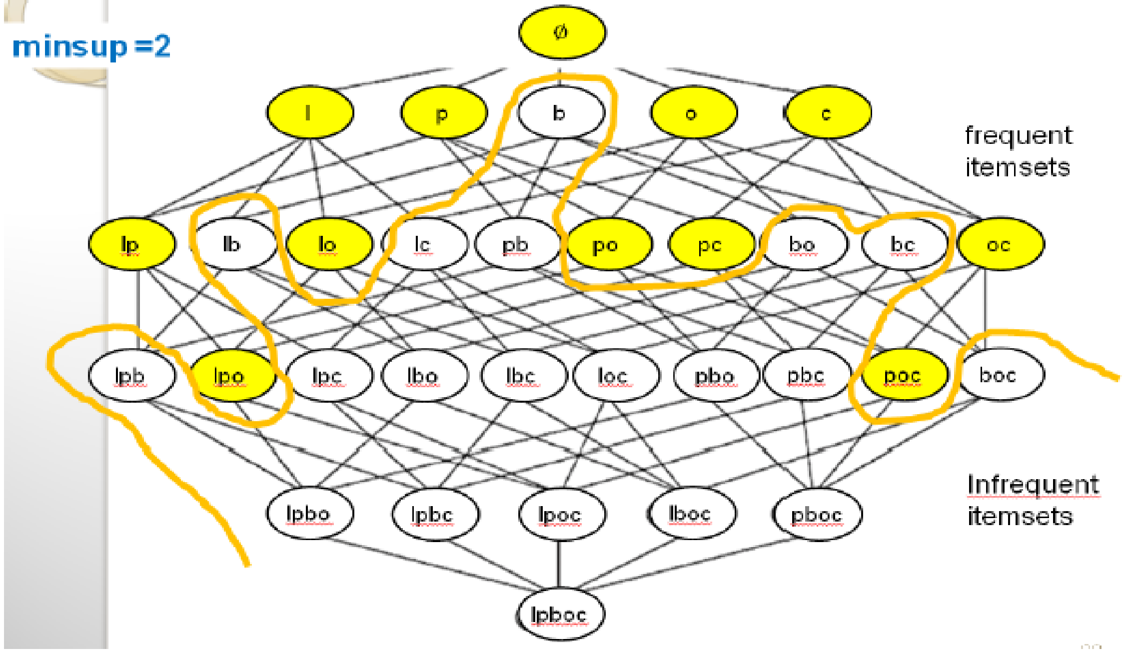
This database contains four **transactions**. Each transaction is a set of items purchased by a customer (an **itemset**). For example, the first transaction contains the items pasta, lemon, bread and orange, while the second transaction contains the items pasta and lemon.

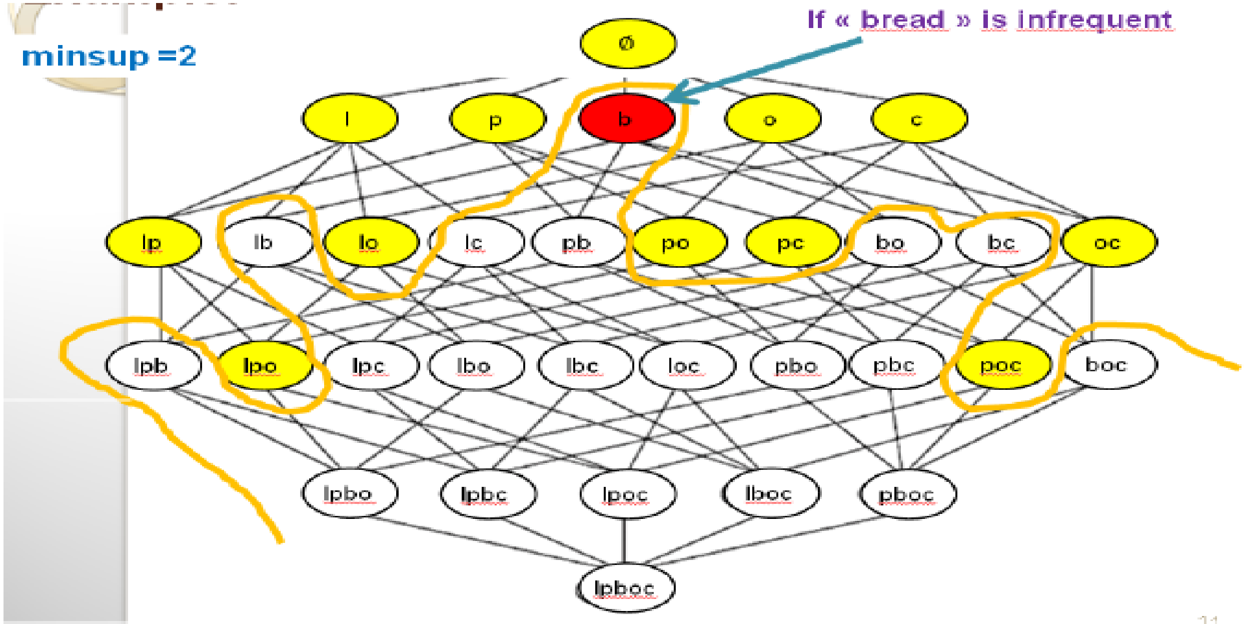
This is all the itemsets that can be formed with the items
lemon (l), pasta (p), bread (b), orange (o) and cake (c)



l = lemon
p = pasta
b = bread
0 = orange
c = cake

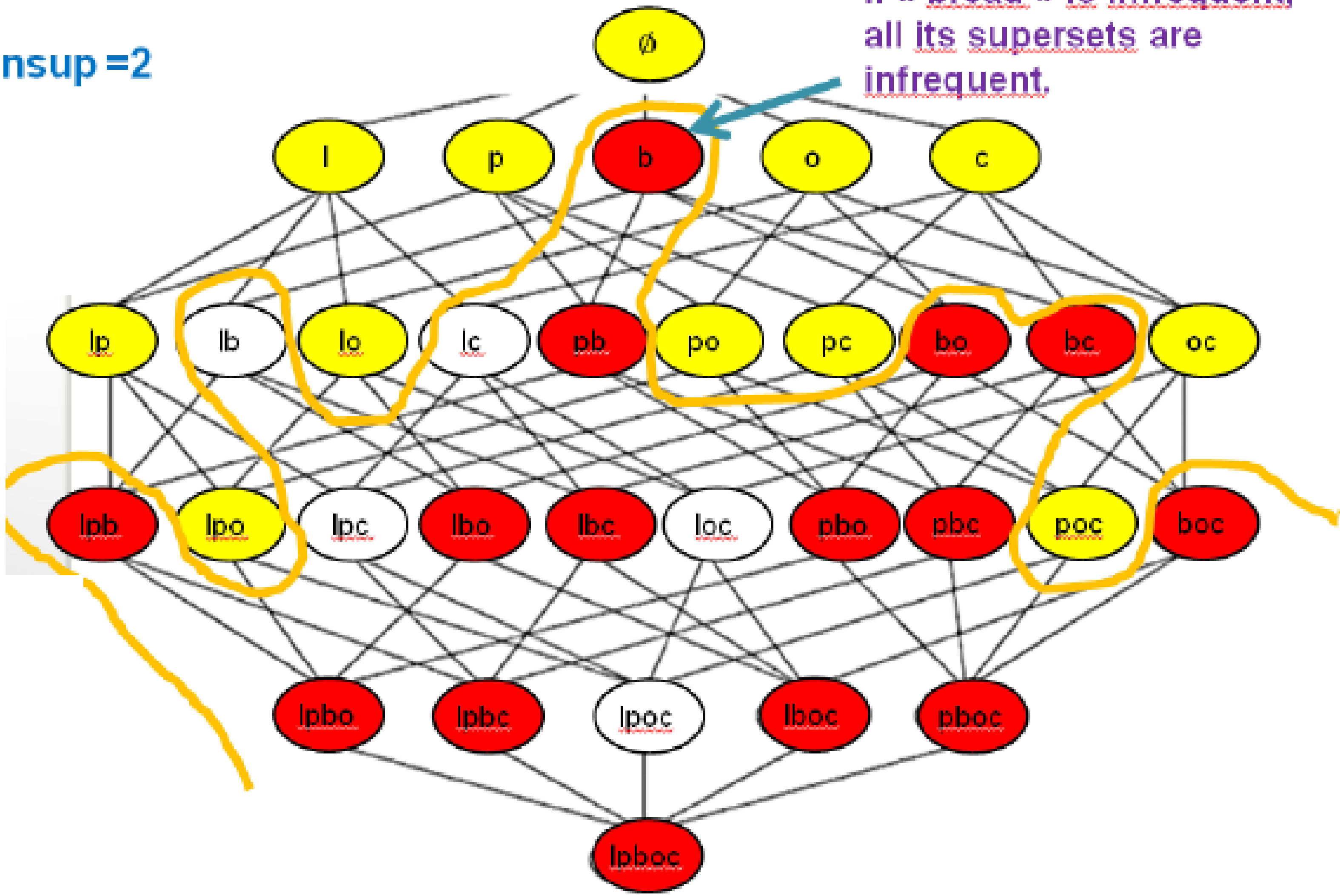
This form a lattice, which can be viewed as a Hasse diagram


minsup =2
Ø
l
p
b
o
c
frequent itemsets
lp
lb
lo
lc
pb
po
pc
bo
bc
oc
lpb
lpo
lpc
lbo
lbc
loc
pbo
pbc
poc
boc
Infrequent itemsets
lpbo
lpbc
lpoc
lboc
pboc
lpboc


minsup =2
If « bread » is infrequent
Ø
l
p
b
o
c
lp
lb
lo
lc
pb
po
pc
bo
bc
oc
lpb
lpo
lpc
lbo
lbc
loc
pbo
pbc
poc
boc
lpbo
lpbc
lpoc
lboc
pboc
lpboc

minsup =2

If « bread » is infrequent, all its supersets are infrequent.



Example:

- Consider {bread, lemon}.
- If we know that {bread} is infrequent, then we can infer that {bread, lemon} is also infrequent.

Transaction	Items appearing in the transaction
T1	{pasta, lemon, bread, orange}
T2	{pasta, lemon}
T3	{pasta, orange, cake}
T4	{pasta, lemon, orange, cake}

Step 1: scan the database to calculate the support of all itemsets of size 1.

e.g.

{pasta} support = 4
{lemon} support = 3
{bread} support = 1
{orange} support = 3
{cake} support = 2

After obtaining the support of single items, the second step is to **eliminate the infrequent itemsets**. Recall that the minsup parameter is set to 2 in this example. Thus we should eliminate all itemsets having a support that is less than 2. This is illustrated below:

Step 2: eliminate infrequent itemsets.

{pasta}	support = 4		{pasta}	support = 4
{lemon}	support = 3	⇨	{lemon}	support = 3
{bread}	support = 1		{orange}	support = 3
{orange}	support = 3		{cake}	support = 2
{cake}	support = 2			

Next the Apriori algorithm will find the **frequent itemsets containing 2 items**. To do that, the Apriori algorithm combines each frequent itemsets of size 1 (each single item) to obtain a set of candidate itemsets of size 2 (containing 2 items). This is illustrated below:

Step 3: generate candidates of size 2 by combining pairs of frequent itemsets of size 1.

Frequent items		Candidates of size 2
		{pasta, lemon}
		{pasta, orange}
{pasta}	⟶	{pasta, cake}
{lemon}		{lemon, orange}
{orange}		{lemon, cake}
{cake}		{orange, cake}

Step 4: Eliminate candidates of size 2 that have an infrequent subset (Property 2)

(none!)

Candidates of size 2

{pasta, lemon}

{pasta, orange}

{pasta, cake}

{lemon, orange}

{lemon, cake}

{orange, cake}

Step 5: scan the database to calculate the support of remaining candidate itemsets of size 2.

Candidates of size 2

{pasta, lemon} support: 3

{pasta, orange} support: 3

{pasta, cake} support: 2

{lemon, orange} support: 2

{lemon, cake} support: 1

{orange, cake} support: 2

Based on these support values, the **Apriori** algorithm next eliminates the infrequent candidate itemsets of size 2. The result is shown below:

Step 6: eliminate infrequent candidates of size 2

Candidates of size 2

{pasta, lemon} support: 3

{pasta, orange} support: 3

{pasta, cake} support: 2

{lemon, orange} support: 2

~~{lemon, cake} support: 1~~

{orange, cake} support: 2

Candidates of size 2

{pasta, lemon} support: 3

{pasta, orange} support: 3

{pasta, cake} support: 2

{lemon, orange} support: 2

{orange, cake} support: 2

Step 7: generate candidates of size 3 by combining frequent pairs of itemsets of size 2.

Frequent itemsets of size 2

{pasta, lemon}

{pasta, orange}

{pasta, cake}

{lemon, orange}

{orange, cake}

Candidates of size 3

{pasta, lemon, orange}

{pasta, lemon, cake}

{pasta, orange, cake}

{lemon, orange, cake}

Thereafter, **Apriori** will determine if these candidates are frequent itemsets. This is done by first checking the **second property**, which says that the subsets of a frequent itemset must also be frequent. Based on this property, we can eliminate some candidates. The **Apriori** algorithm checks if there exists a subset of size 2 that is not frequent for each candidate itemset. Two candidates are eliminated as shown below.

Step 8: eliminate candidates of size 3 having a subset of size 2 that is infrequent.

Frequent itemsets of size 2

{pasta, lemon}

{pasta, orange}

{pasta, cake}

{lemon, orange}

{orange, cake}

Candidates of size 3

{pasta, lemon, orange}

~~{pasta, lemon, cake}~~

{pasta, orange, cake}

~~{lemon, orange, cake}~~

Because {lemon, cake} is infrequent!

For example, in the above illustration, the itemset {lemon, orange, cake} has been eliminated because one of its subset of size 2 is infrequent (the itemset {lemon cake}). Thus, after performing this step, only two candidate itemsets of size 3 are left.

Step 9: scan the database to calculate the support of the remaining candidates of size 3.

Candidates of size 2

{pasta, lemon, orange} support: 2

{pasta, orange, cake} support: 2

Based on these support values, the **Apriori** algorithm next eliminates the infrequent candidate itemsets of size 3 o obtain the frequent itemset of size 3. The result is shown below:

Step 10: eliminate infrequent candidates (none!)

frequent itemsets of size 3

{pasta, lemon, orange} support: 2

{pasta, orange, cake} support: 2

There was no infrequent itemsets among the candidate itemsets of size 3, so no itemset was eliminated. The two candidate itemsets of size 3 are thus frequent and are output to the user.

Next, the **Apriori** algorithm will try to generate **candidate itemsets of size** 4. This is done by combining pairs of **frequent itemsets** of size 3. This is done as follows:

Step 11: generate candidates of size 4 by combining pairs of frequent itemsets of size 3.

Frequent itemsets of size 3	Candidates of size 4
{pasta, lemon, orange}	{pasta, lemon, orange, cake}
{pasta, orange, cake}	

Only one candidate itemset was generated. hereafter, **Apriori** will determine if this candidate is frequent. This is done by first checking the **second property**, which says that the subsets of a frequent itemset must also be frequent. The Apriori algorithm checks if there exist a subset of size 3 that is not frequent for the candidate itemset.

Step 12: eliminate candidates of size 4 having a subset of size 3 that is infrequent.

Frequent itemsets of size 3	Candidates of size 4
{pasta, lemon, orange}	~~{pasta, lemon, orange, cake}~~
{pasta, orange, cake}	

During the above step, the candidate itemset {pasta, lemon, orange, cake} is eliminated because it contains at least one subset of size 3 that is infrequent. For example, {pasta, lemon cake} is infrequent.

Now, since there is no more candidate left. The **Apriori algorithm** has to stop and do not need to consider larger itemsets (for example, itemsets containing five items).

The final result found by the algorithm is this **set of frequent itemsets**.

Final result

{pasta}	support = 4
{lemon}	support = 3
{orange}	support = 3
{cake}	support = 2
{pasta, lemon}	support: 3
{pasta, orange}	support: 3
{pasta, cake}	support: 2
{lemon, orange}	support: 2
{orange, cake}	support: 2
{pasta, lemon, orange}	support: 2
{pasta, orange, cake}	support: 2

Thus, the **Apriori algorithm** has found **11 frequent itemsets**. The **Apriori algorithm** is said to be a recursive algorithm as it recursively explores larger itemsets starting from itemsets of size 1.