Data Mining and Applications

Association Rules Mining

Outline

- Introduction
- Frequent itemset mining
- Association Rule Mining Task
- Generating Association Rules from Frequent Itemsets
- Interestingness Measure for Association Rules

References

[1] Tan, Steinbach, Karpatne, Kumar, Introduction to Data Mining, 2nd Edition, 2018.

[2] Jiawei Han, Micheline Kamber, "Data Mining: Concepts and Techniques", Second Edition, Morgan Kaufmann Publishers, 2006.

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

Introduction

- Many retail stores collect data about customers.
- **e.g.** customer transactions
- Need to analyze this data to understand customer behavior
- Why?
  - for marketing purposes,
  - inventory management,
  - customer relationship management

Introduction

Discovering patterns and associations

- Discovering interesting relationships hidden or frequent pattern (a pattern is a set of items, subsequences, substructures, etc.) that occurs frequently in large databases.
  - **beer and diapers are often sold together**
- **pattern mining** is a fundamental data mining problem with many applications in various fields.
- First introduced by Agrawal (1993) in the context of frequent itemsets and association rule mining
- Many extensions of this problem to discover patterns in graphs, sequences, and other kinds of data.

Introduction

- Motivation: Finding inherent regularities in data
  - What products were often purchased together?— Beer and diapers?!
  - What are the subsequent purchases after buying a PC?
  - What kinds of DNA are sensitive to this new drug?
  - Can we automatically classify web documents?
- Applications
  - Basket data analysis, cross-marketing, catalog design, sale campaign analysis, Web log (click stream) analysis, and DNA sequence analysis.

Frequent itemset mining

Definitions

- Let $I = \{I_1, I_2, I_3, \ldots, I_n\}$ be the set of items (products) that appear (sold) in the database (in a retail store).

  e.g., I= {pasta, lemon, bread, orange, cake}

- A **transaction database** D is a set of transactions.

  $$D = \{T1, T2, \ldots Tr\} \text{ such that } Ta \subseteq I \ (1 \leq a \leq r).$$

  Number of transaction in D denoted |D|

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

Definitions

- Each transaction has a unique identifier called its **Transaction ID** (TID).

  e.g. the transaction ID of T4 is 4.

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

Definitions

A transaction is a set of items (*an itemset*).

e.g. T2= {pasta, lemon}

An **item** (a symbol) may not appear or appear once in each transaction.

Each transaction is unordered.

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

Definitions

A transaction database can be viewed as a binary matrix:

Transaction	pasta	lemon	bread	orange	cake
T1	1	1	1	1	0
T2	1	1	0	0	0
T3	1	0	0	1	1
T4	1	1	0	1	1

- Asymetrical binary attributes (because 1 is more important than 0)
- There is no information about purchase quantities and prices.

Definitions

Let I bet the set of all items:

I= {pasta, lemon, bread, orange, cake}

There are $2^{|I|} - 1 = 2^5 - 1 = 31$ subsets :

{pasta}, {lemon}, {bread}, {orange}, {cake}

{pasta, lemon}, {pasta, bread} {pasta, orange}, {pasta, cake}, {lemon, bread}, {lemon orange},

{lemon, cake}, {bread, orange}, {bread cake}

…

{pasta, lemon, bread, orange, cake}

Definitions

- ❑ **Itemset** is a set of one or more items
  - E.g.: {pasta, lemon, bread}
- ❑ **An itemset is said to be of size *k*, if it contains *k* items.**

  - Itemsets of size 1:

    {pasta}, {lemon}, {bread}, {orange}, {cake}

  - Itemsets of size 2:

{pasta, lemon}, {pasta, bread} {pasta, orange}, {pasta, cake}, {lemon, bread}, {lemon orange}, …

Definitions

The **support (frequency) of an itemset X** is the number of transactions that contains **X**.

$$\sup(X) = |\{T \mid X \subseteq T \wedge T \in D\}|$$

For example: The support of {pasta, orange} is 3.

which is written as: $sup(\{\text{pasta, orange}\}) = 3$

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

Definitions

The **support of an itemset X** can also be written as a ratio (*absolute support*).

Example: The support of {pasta, orange} is 75% because it appears in 3 out of 4 transactions.

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

The problem of frequent itemset mining

- Let there be a numerical value ***minsup***, set by the user.
- Frequent itemset mining (FIM) consists of enumerating all **frequent itemsets,** that is itemsets having a support greater or equal to *minsup*.

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

Example

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

For ***minsup*** = 2, the **frequent itemsets** are:
{lemon}, {pasta}, {orange}, {cake}, {lemon, pasta}, {lemon, orange}, {pasta, orange}, {pasta, cake}, {orange, cake}, {lemon, pasta, orange}

For the user, choosing a high ***minsup*** value,

- will reduce the number of frequent itemsets,
- will increase the speed and decrease the memory required for finding the frequent itemsets

The problem of frequent itemset mining

- Example: Let I = {A, B, C, D, E, F} and a transaction database as:

T_1	{A, B, C, D}
T_2	{A, C, E}
T_3	{A, E}
T_4	{A, E, F}
T_5	{A, B, C, E, F}

With X = *{A, E}* , *minsup=60%*

→ $sup(X) = 4$

or $sup(X) = 4/5 = 80\%$

→ *X* is frequent itemset

The problem of frequent itemset mining

Given I = { Beer, Bread, Jelly, Milk, PeanutButter} and a transaction database as:

TID	Items
10	Bread, Jelly, PeanutButter
20	Bread, PeanutButter
30	Bread, Milk, PeanutButter
40	Beer, Bread
50	Beer, Milk

minsup = 60%

Find the support for each below itemset and which itemset is frequent itemset

X1= {Bread,PeanutButter}

X_2 = {Bread}

X_3 = {PeanutButter}

X_4 = {Milk}

X_5={Milk, Bread}

Several algorithms

- Algorithms:
  - Apriori, AprioriTID (1993)
  - Eclat (1997)
  - FPGrowth (2000)
  - Hmine (2001)
  - LCM, …
  - …
- Moreover, numerous extensions of the FIM problem: uncertain data, fuzzy data, purchase quantities, profit, weight, time, rare itemsets, closed itemsets, etc.

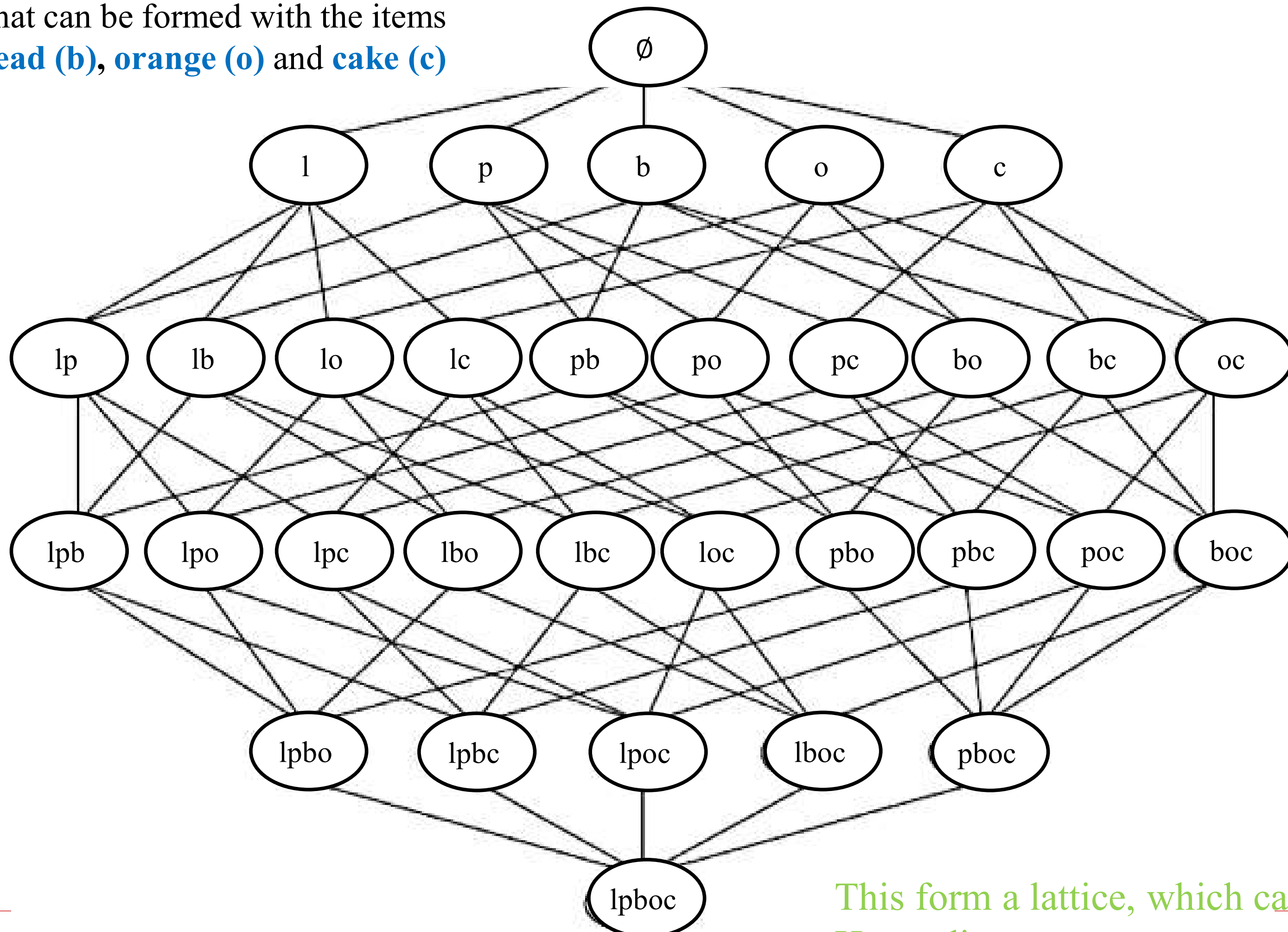
Algorithms

Naïve approach

- If there are n items in a database, there are $2^n - 1$ itemsets may be frequent.
- **Naïve approach**: count the support of all these itemsets.
- To do that, we would need to read each transaction in the database to count the support of each itemset.
- This would be inefficient:
  - need to perform too many comparisons
  - requires too much memory

Search space

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

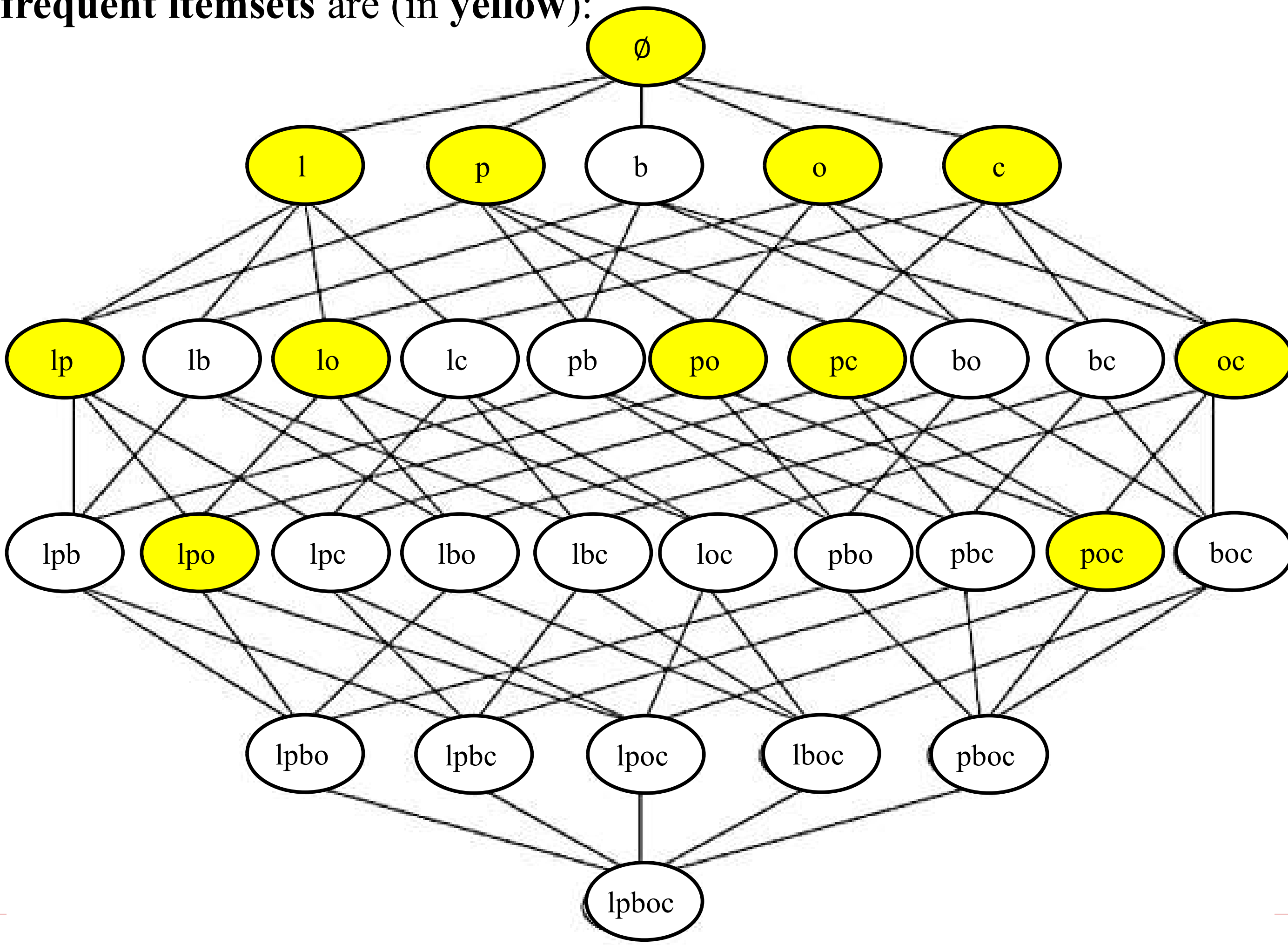


l = lemon
p = pasta
b = bread
0 = orange
c = cake

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

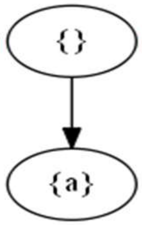
Search space

If **minsup = 2**, the **frequent itemsets** are (in **yellow**):

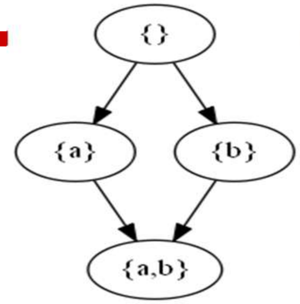


l = lemon
p = pasta
b = bread
0 = orange
c = cake

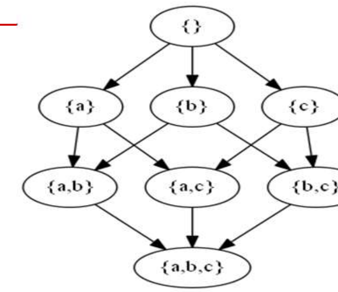
I={A}



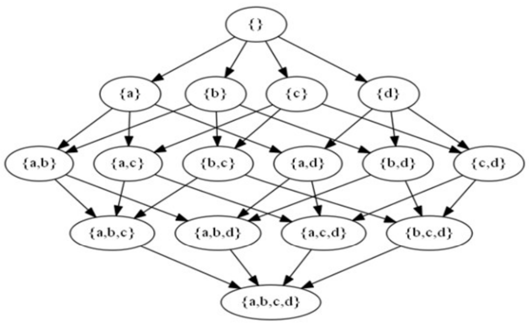
I={A, B}



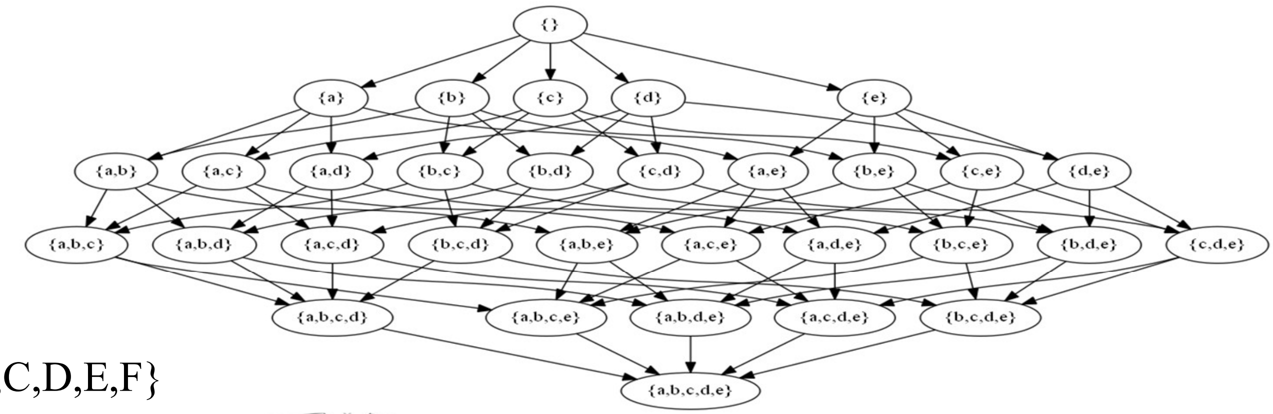
I={A, B,C}



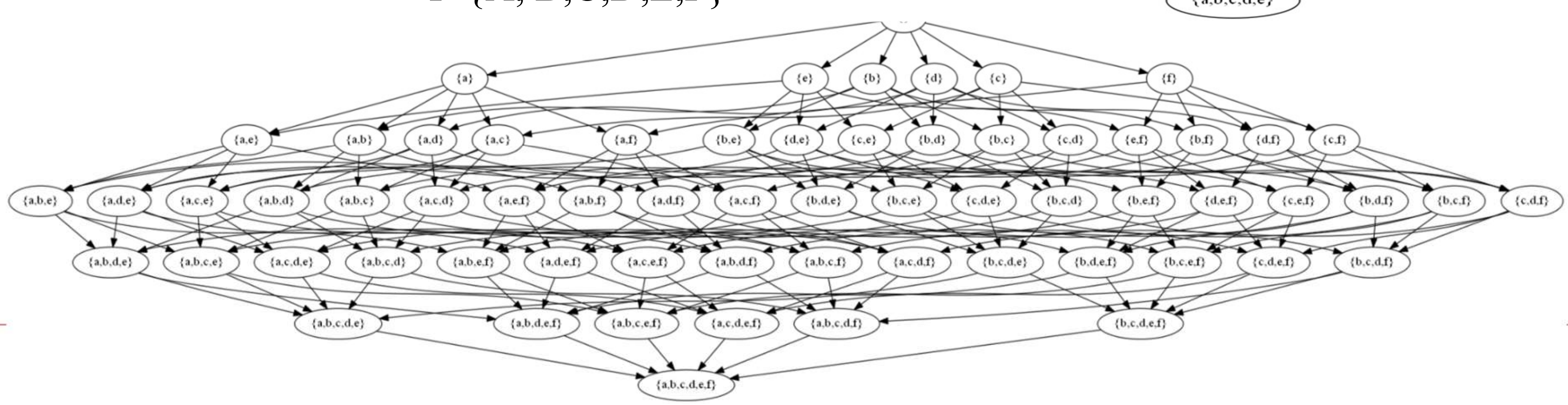
I={A, B,C,D}



I={A, B,C,D,E}



I={A, B,C,D,E,F}



How to find the frequent itemsets?

Two challenges:

- How to count the support of itemsets in an efficient way (not spend too much time or memory)?
- How to reduce the search space (we do not want to consider all the possibilities)?

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

The APRIORI algorithm (Agrawal & Srikant, 1993/1994)

R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. Research Report RJ 9839, IBM Almaden Research Center, San Jose, California, June 1994.

Introduction

Apriori is a famous algorithm

- which is not the most efficient algorithm,
- but has inspired many other algorithms!
- has been applied in many fields,
- has been adapted for many other similar problems.

Apriori is based on two important properties

Apriori property: Let there be two itemsets X and Y.
If $X \subset Y$, the support of Y is less than or equal to the support of X.

Example:

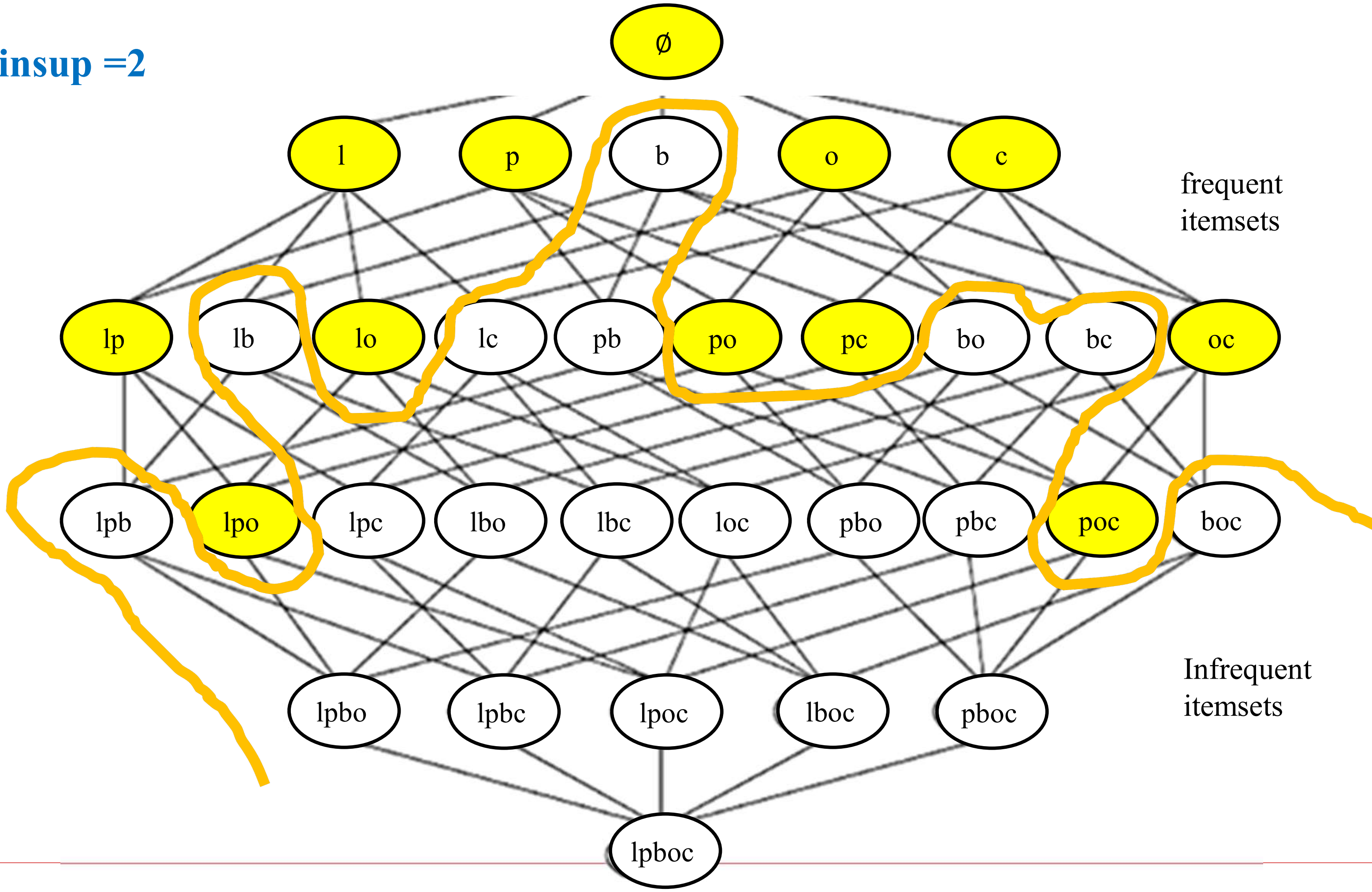
- The support of {pasta} is 4
- The support of {pasta, lemon} is 3
- The support of {pasta, lemon, orange} is 2

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

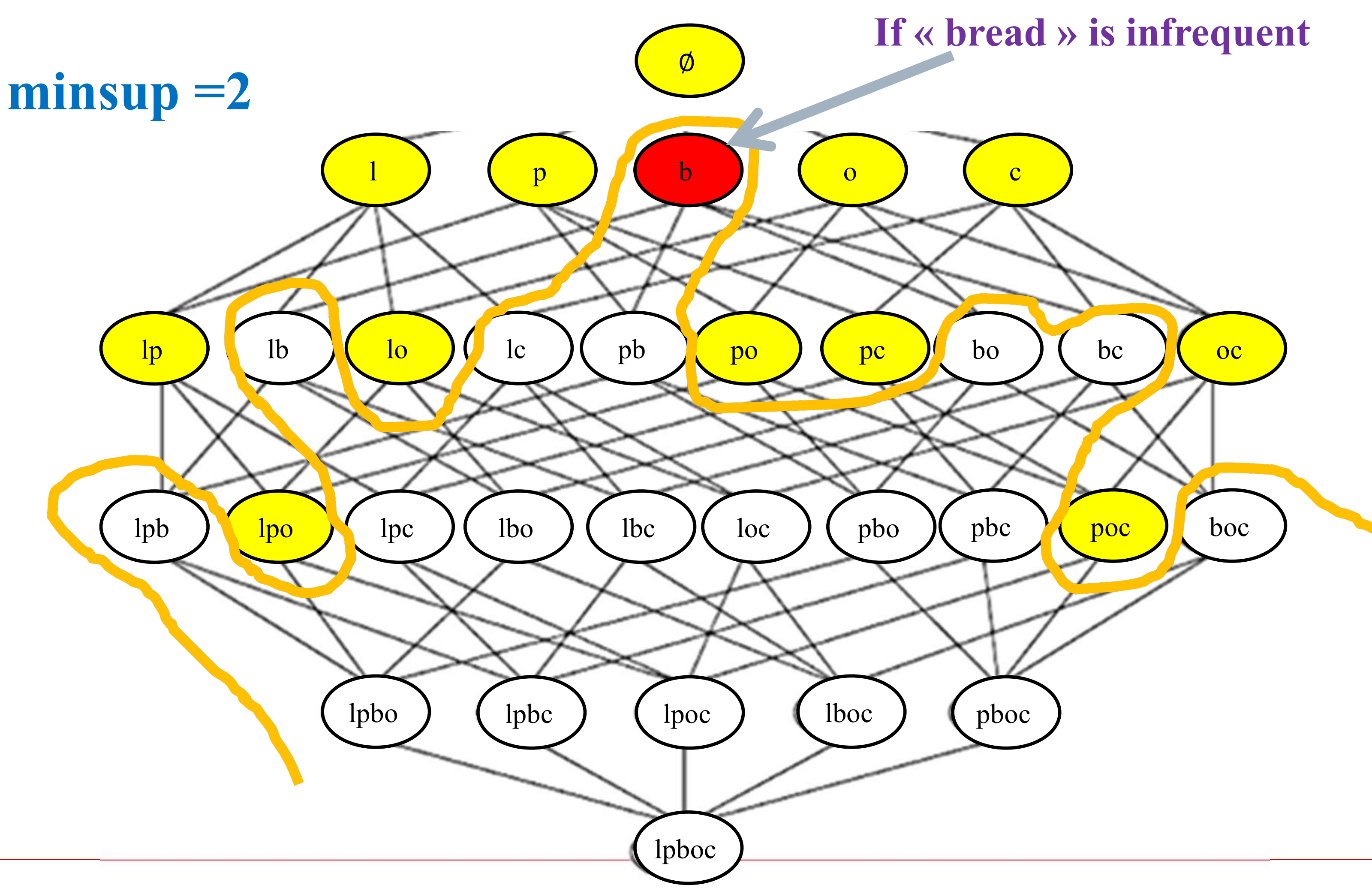
(support is anti-monotonic)

Illustration

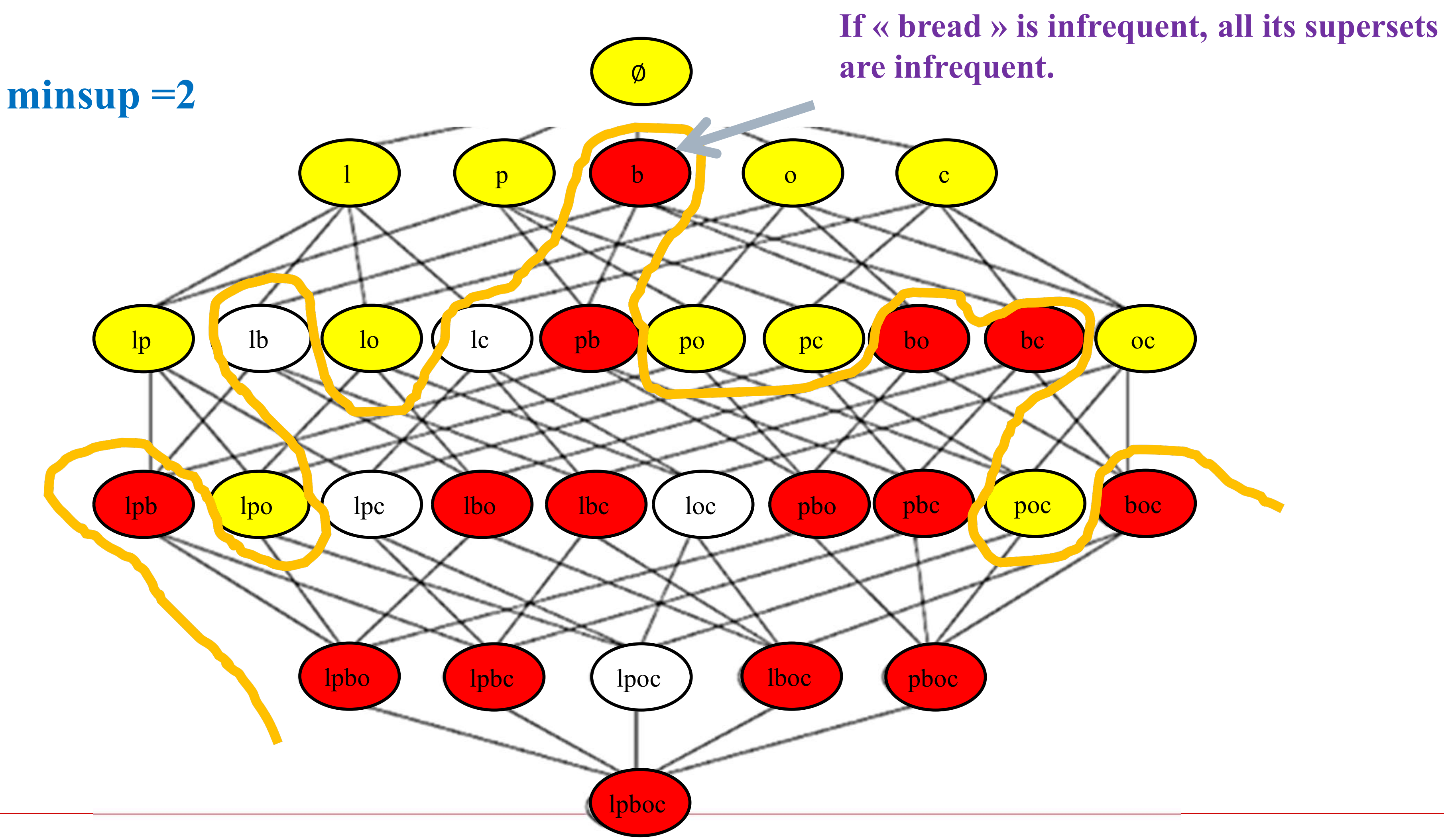
minsup =2



This property is useful to reduce the search space. **Example:**



This property is useful to reduce the search space. **Example:**



Property 2: Let there be an itemset Y.
If there exists an itemset $X \subset Y$ such that X is infrequent, then Y is 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}

The Apriori algorithm

- I will now explain how the Apriori algorithm works
- **Input:**
  - *minsup*
  - a transactional database
- **Output:**
  - all the frequent itemsets

Consider ***minsup =2*.**

Apriori Algorithm

- F_k: frequent k-itemsets
- L_k: candidate k-itemsets

- Algorithm
  - Let k=1
  - Generate F_1 = {frequent 1-itemsets}
  - Repeat until F_k is empty
    - **Candidate Generation**: Generate L_{k+1} from F_k
    - **Candidate Pruning**: Prune candidate itemsets in L_{k+1} containing subsets of length k that are infrequent
    - **Support Counting**: Count the support of each candidate in L_{k+1} by scanning the DB
    - **Candidate Elimination**: Eliminate candidates in L_{k+1} that are infrequent, leaving only those that are frequent => F_{k+1}

The Apriori algorithm

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

The Apriori algorithm

Step 2: eliminate infrequent itemsets.

e.g.

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

The Apriori algorithm

Step 2: eliminate infrequent itemsets.

e.g.

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

The Apriori algorithm

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

Frequent items

{pasta}
{lemon}
{orange}
{cake}

→

Candidates of size 2

{pasta, lemon}
{pasta, orange}
{pasta, cake}
{lemon, orange}
{lemon, cake}
{orange, cake}

The Apriori algorithm

Step 4: Eliminate candidates of size 2 that have an infrequent subset (Property 2) (none!)

Frequent items

{pasta}

{lemon}

{orange}

{cake}

Candidates of size 2

{pasta, lemon}

{pasta, orange}

{pasta, cake}

{lemon, orange}

{lemon, cake}

{orange, cake}

The Apriori algorithm

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

The Apriori algorithm

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

The Apriori algorithm

Step 6: eliminate infrequent candidates of size 2

Frequent itemsets of size 2

{pasta, lemon} support: 3

{pasta, orange} support: 3

{pasta, cake} support: 2

{lemon, orange} support: 2

{orange, cake} support: 2

The Apriori algorithm

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}

The Apriori algorithm

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!

The Apriori algorithm

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, orange, cake}

Because {lemon, cake} is infrequent!

The Apriori algorithm

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

The Apriori algorithm

Step 10: eliminate infrequent candidates (none!)

frequent itemsets of size 3

{pasta, lemon, orange} support: 2

{pasta, orange, cake} support: 2

The Apriori algorithm

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}		

The Apriori algorithm

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

Frequent itemsets of size 3

{pasta, lemon, orange}

{pasta, orange, cake}

→

Candidates of size 4

~~{pasta, lemon, orange, cake}~~

The Apriori algorithm

Step 12: Since there is no more candidates, we cannot generate candidates of size 5 and the algorithm stops.

Candidates of size 4

~~{pasta, lemon, orange, cake}~~

Result →

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

Technical details

Combining different itemsets can generate the same candidate.

Example:

{A, B} and { A,E} → {A, B, E}

{B, E} and { A,E} → {A, B, E}

problem: some candidates are generated several times!

Technical details

Combining different itemsets can generate the same candidate.

Example:

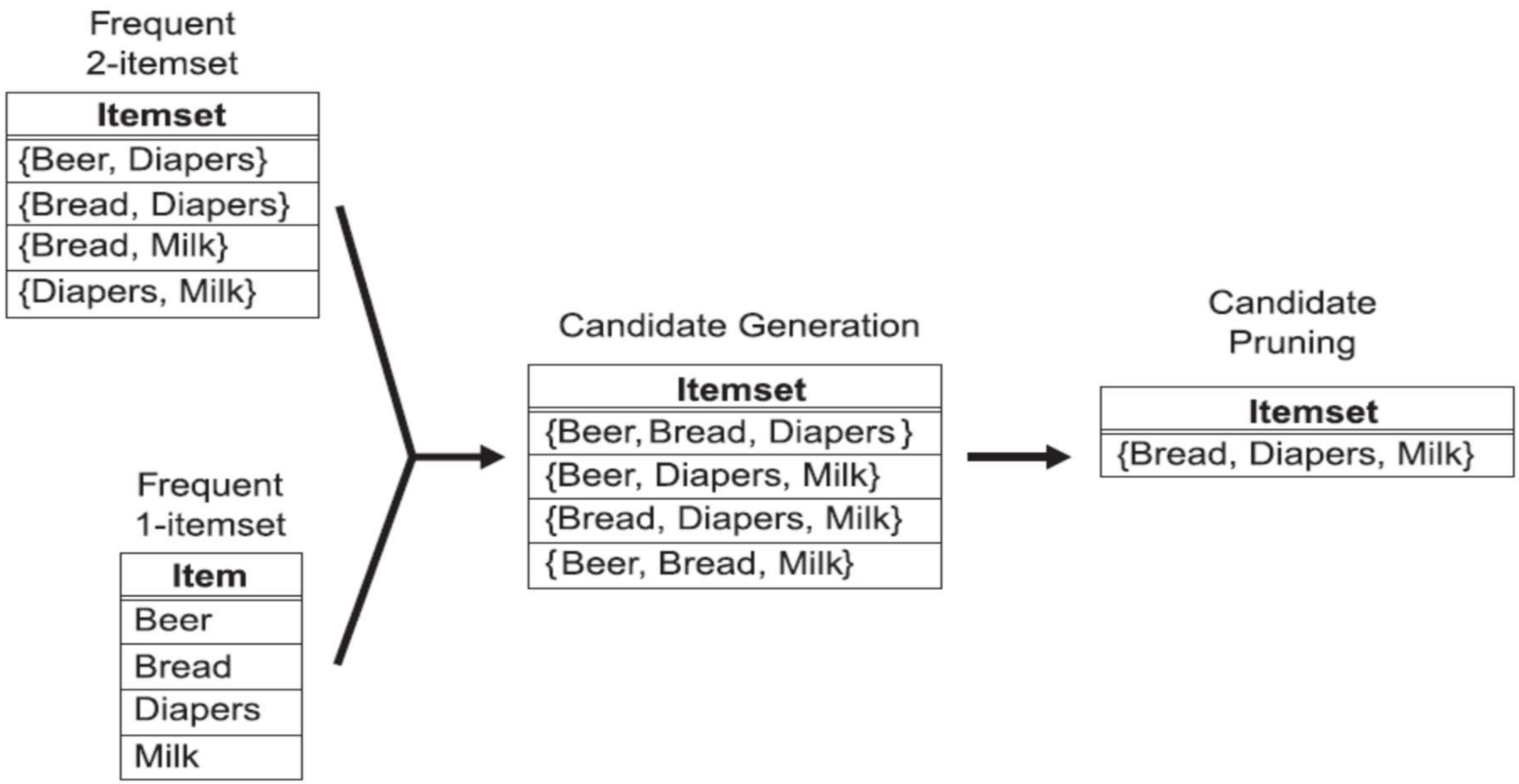
{A, B} and { A,E} → {A, B, E}

~~{B, E} and { A,E} → {A, B, E}~~

Solution:

- Sort items in each itemsets (e.g. by alphabetical order)
- Combine two itemsets only if all items are the same except the last one.

Candidate Generation: Merge Fk-1 and F1 itemsets



Generating and pruning candidate k-itemsets by merging a frequent $(k-1)$-itemset with a frequent item. Note that some of the candidates are unnecessary because their subsets are infrequent.

Candidate Generation: F_{k-1} x F_{k-1} Method

- Create a candidate itemset of size k, by joining two itemsets of size k-1 if their first (k-2) items are identical

- Example: F_3 = {ABC,ABD,ABE,ACD,BCD,BDE,CDE}
  - Join(<u>**AB**</u>C, <u>**AB**</u>D) = <u>**AB**</u>CD
  - Join(<u>**AB**</u>C, <u>**AB**</u>E) = <u>**AB**</u>CE
  - Join(<u>**AB**</u>D, <u>**AB**</u>E) = <u>**AB**</u>DE

  - Do not join(<u>**A**</u>BD,<u>**A**</u>CD) because they share only prefix of length 1 instead of length 2

Alternate Fk-1 x Fk-1 Method

- Merge two frequent (k-1)-itemsets if the last (k-2) items of the first one is identical to the first (k-2) items of the second.

- F_3 = {ABC,ABD,ABE,ACD,BCD,BDE,CDE}
  - Merge(A<u>**BC**</u>, <u>**BC**</u>D) = A<u>**BC**</u>D
  - Merge(A<u>**BD**</u>, <u>**BD**</u>E) = A<u>**BD**</u>E
  - Merge(A<u>**CD**</u>, <u>**CD**</u>E) = A<u>**CD**</u>E
  - Merge(B<u>**CD**</u>, <u>**CD**</u>E) = B<u>**CD**</u>E

Candidate Pruning

- For each candidate k-itemset create all subset (k-1)-itemsets
- Remove a candidate if it contains a subset (k-1)-itemset that is not frequent
- Example:
  - Let F_3 = {ABC,ABD,ABE,ACD,BCD,BDE,CDE} be the set of frequent 3-itemsets
  - L_4 = {ABCD,ABCE,ABDE} is the set of candidate 4-itemsets generated (from previous slide)
  - Candidate pruning
    - Prune ABCE because ACE and BCE are infrequent
    - Prune ABDE because ADE is infrequent
  - After candidate pruning: L_4 = {ABCD}

The Apriori Algorithm—An Example

$Sup_{min} = 2$

Database TDB

Tid	Items
10	A, C, D
20	B, C, E
30	A, B, C, E
40	B, E

1^{st} scan

L1

Itemset	sup
{A}	2
{B}	3
{C}	3
{D}	1
{E}	3

F_1

Itemset	sup
{A}	2
{B}	3
{C}	3
{E}	3

L_2

Itemset
{A, B}
{A, C}
{A, E}
{B, C}
{B, E}
{C, E}

2^{nd} scan

L_2

Itemset	sup
{A, B}	1
{A, C}	2
{A, E}	1
{B, C}	2
{B, E}	3
{C, E}	2

F_2

Itemset	sup
{A, C}	2
{B, C}	2
{B, E}	3
{C, E}	2

L_3

Itemset
{B, C, E}

3^{rd} scan

F_3

Itemset	sup
{B, C, E}	2

Apriori vs the naïve algorithm

- The Apriori property can considerably reduce the number of itemsets to be considered.
- In the previous example:
  - Naïve approach:
    2^5-1 = 31 itemsets are considered
  - By using the Apriori property:
    18 itemsets are considered

Factors Affecting Complexity of Apriori

- Choice of minimum support threshold
  - lowering support threshold results in more frequent itemsets
  - this may increase number of candidates and max length of frequent itemsets
- Dimensionality (number of items) of the data set
  - More space is needed to store support count of itemsets
  - if number of frequent itemsets also increases, both computation and I/O costs may also increase
- Size of database
  - run time of algorithm increases with number of transactions
- Average transaction width
  - transaction width increases the max length of frequent itemsets
  - number of subsets in a transaction increases with its width, increasing computation time for support counting
- Solution: Mine *closed patterns* and *max-patterns*

Maximal Frequent Itemset

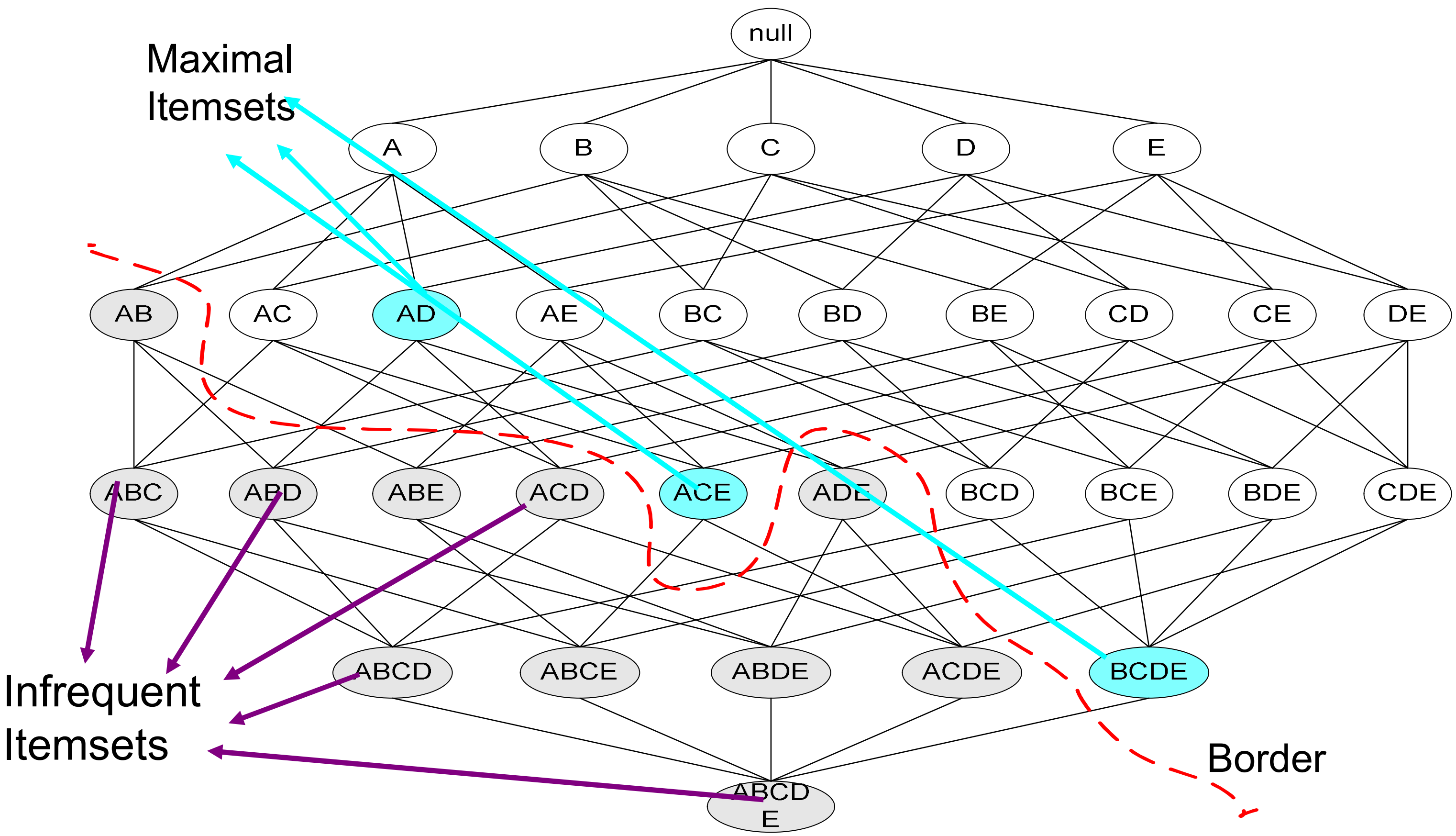
- An itemset X is maximal frequent if it is frequent and none of its immediate supersets is frequent (Bayardo – SIGMOD'98)
- Example:

Tid	Items
10	A,B,C,D,E
20	B,C,D,E,
30	A,C,D,F

Minsupp=2

  - {B, C, D, E}, {A, C, D} – maximal frequent itemsets
  - {B, C, D} – non-maximal frequent itemset

Maximal Frequent Itemset



Closed Itemset

- An itemset X is closed if none of its immediate supersets has the same support as the itemset X.
- X is not closed if at least one of its immediate supersets has support count as X.

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

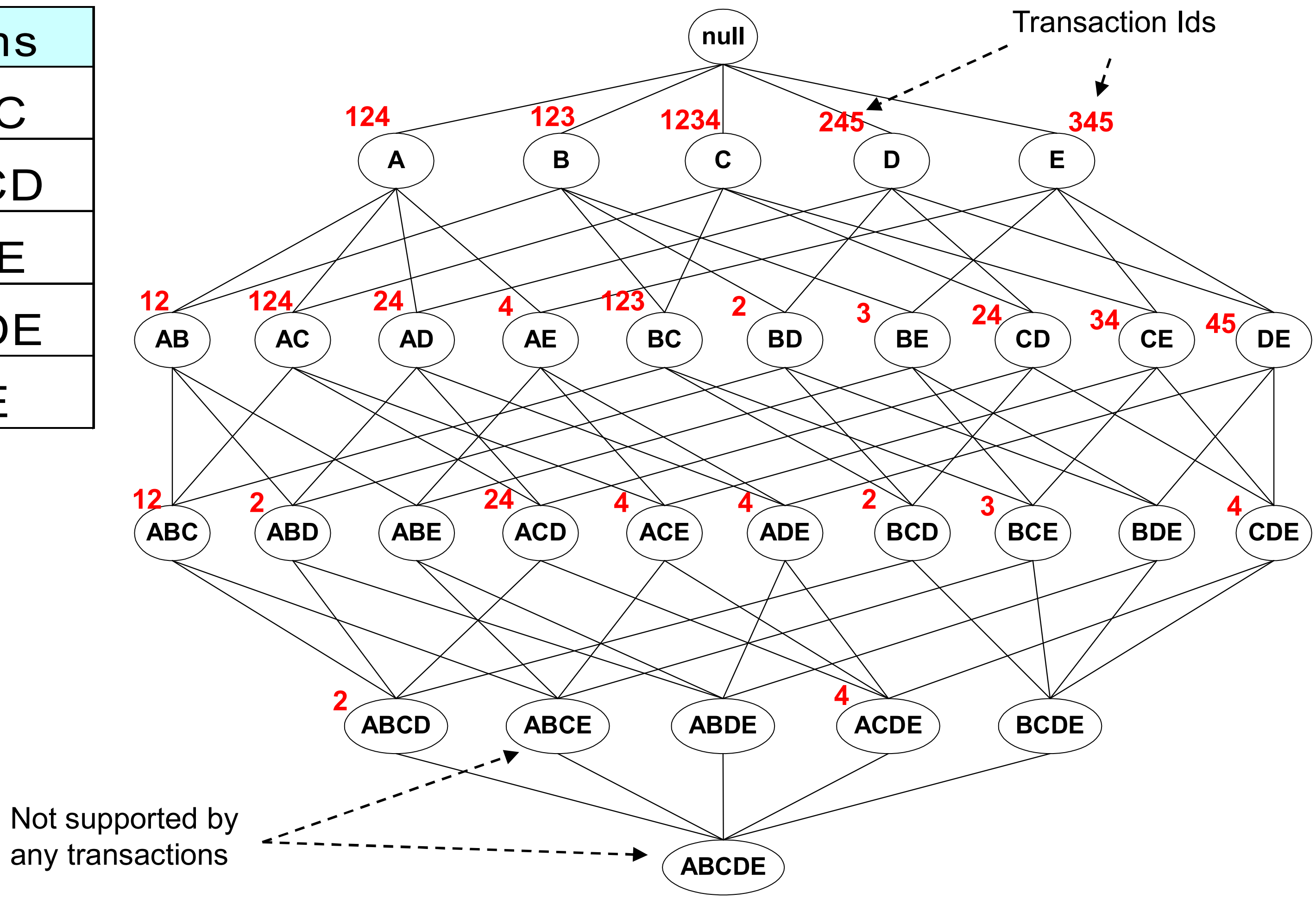
Itemset	Support
{A}	4
{B}	5
{C}	3
{D}	4
{A,B}	4
{A,C}	2
{A,D}	3
{B,C}	3
{B,D}	4
{C,D}	3

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

Minsupp=2

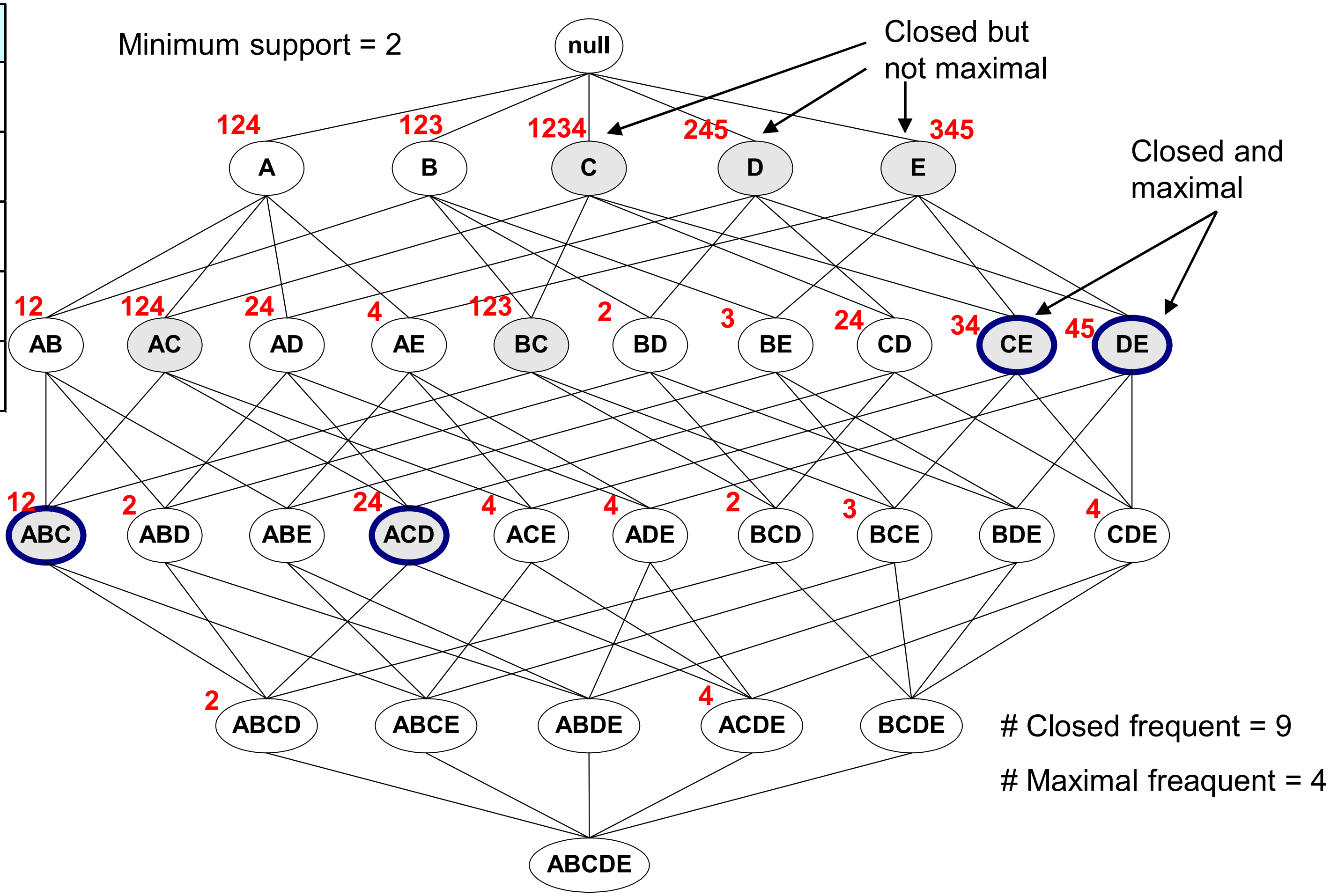
Maximal vs Closed Itemsets

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

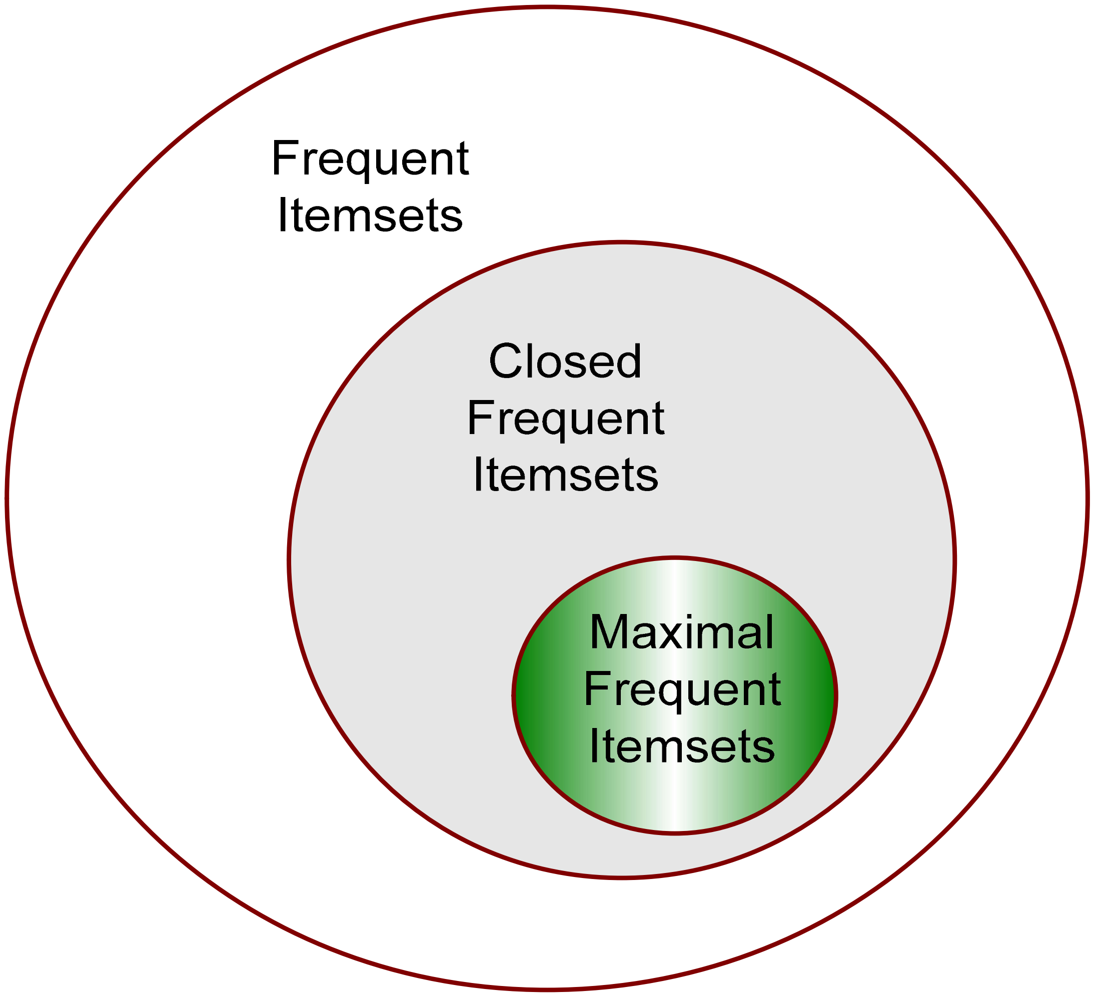


Maximal Frequent vs Closed Frequent Itemsets

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



Maximal vs Closed Itemsets



Exercise

Exercise 1: Let I = {A, B, C, D, E, F} and a transaction database as:

T1	{A, B, C, F}
T2	{A, B, E, F}
T3	{A, C}
T4	{D, E}
T5	{B, F}

1. Apply Apriori algorithm to find all frequent itemsets with minsupp = 25%
2. List all maximal frequent itemsets and closed frequent itemsets.

Exercise

Exercise 2: Given I = {A, B, C, D, E, F} and a transaction database as:

T1	{D, E}
T2	{A, B, D, E}
T3	{A, B, D}
T4	{C, D, E}
T5	{F}
T6	{B, C, D}

1. Apply Apriori algorithm to find all of frequent itemsets with minsupp = 20%

2. List all maximal frequent itemsets and closed frequent itemsets.