Frequent Itemset Mining and The Apriori algorithm

Philippe Fournier-Viger
http://www.philippe-Fournier-viger.com

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.

Source code and datasets available in the SPMF library

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 in large databases.
- **e.g. beer and diapers are often sold together**
- **pattern mining** is a fundamental data mining problem with many applications in various fields.
- Introduced by Agrawal (1993).
- Many extensions of this problem to discover patterns in graphs, sequences, and other kinds of data.

FREQUENT ITEMSET MINING

Definitions

Let $I = \{I_1, I_2, \ldots\ Im\}$ be the set of items (products) sold in a retail store.

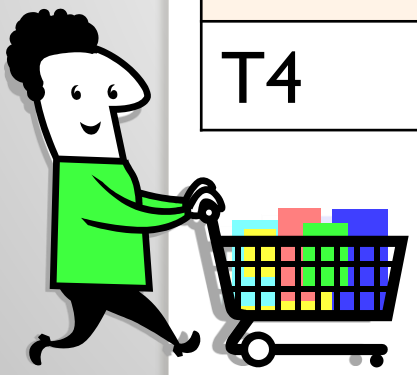
For example:

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

Definitions

A **transaction database** D is a set of **transactions.**
D = $\{T_1, T_2, \ldots T_r\}$ such that $T_a \subseteq I$ $(1 \leq a \leq r)$.

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

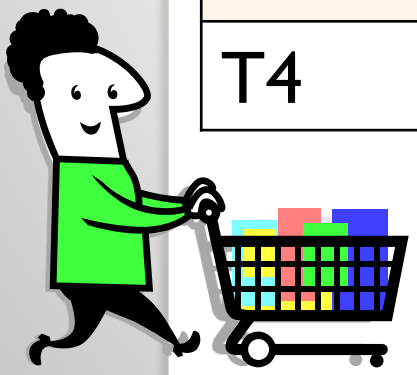


Definitions

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

e.g. the transaction ID of T_4 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}



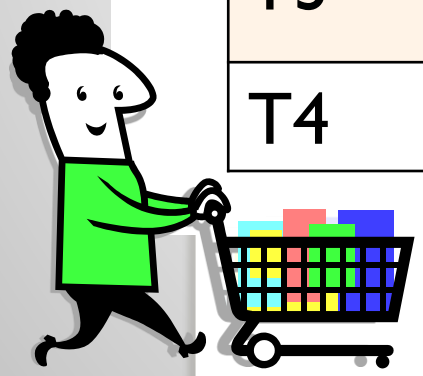
Definitions

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

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

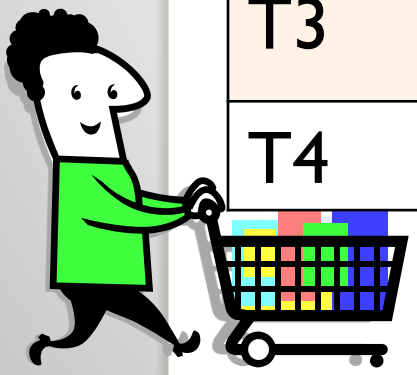
The **support (frequency) of an itemset X** is the number of transactions that contains **X**.

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

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}



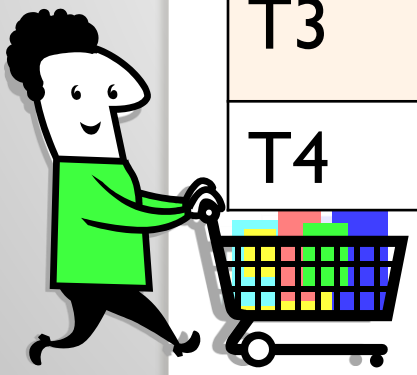
support = 支持

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.

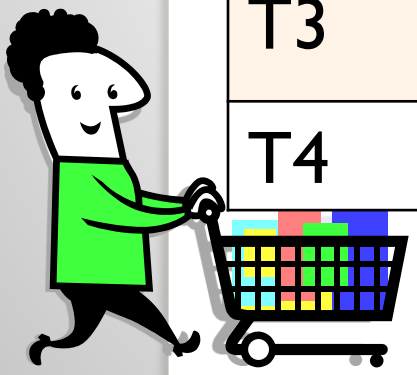
Transaction	Items appearing in the transaction
T1	{pasta, lemon, bread, orange}
T2	{pasta, lemon}
T3	{pasta, orange, cake}
T4	{pasta, lemon, orange, cake}



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

Numerous applications

Frequent itemset mining has numerous applications.

- medical applications,
- chemistry,
- biology,
- e-learning,
- etc.

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.

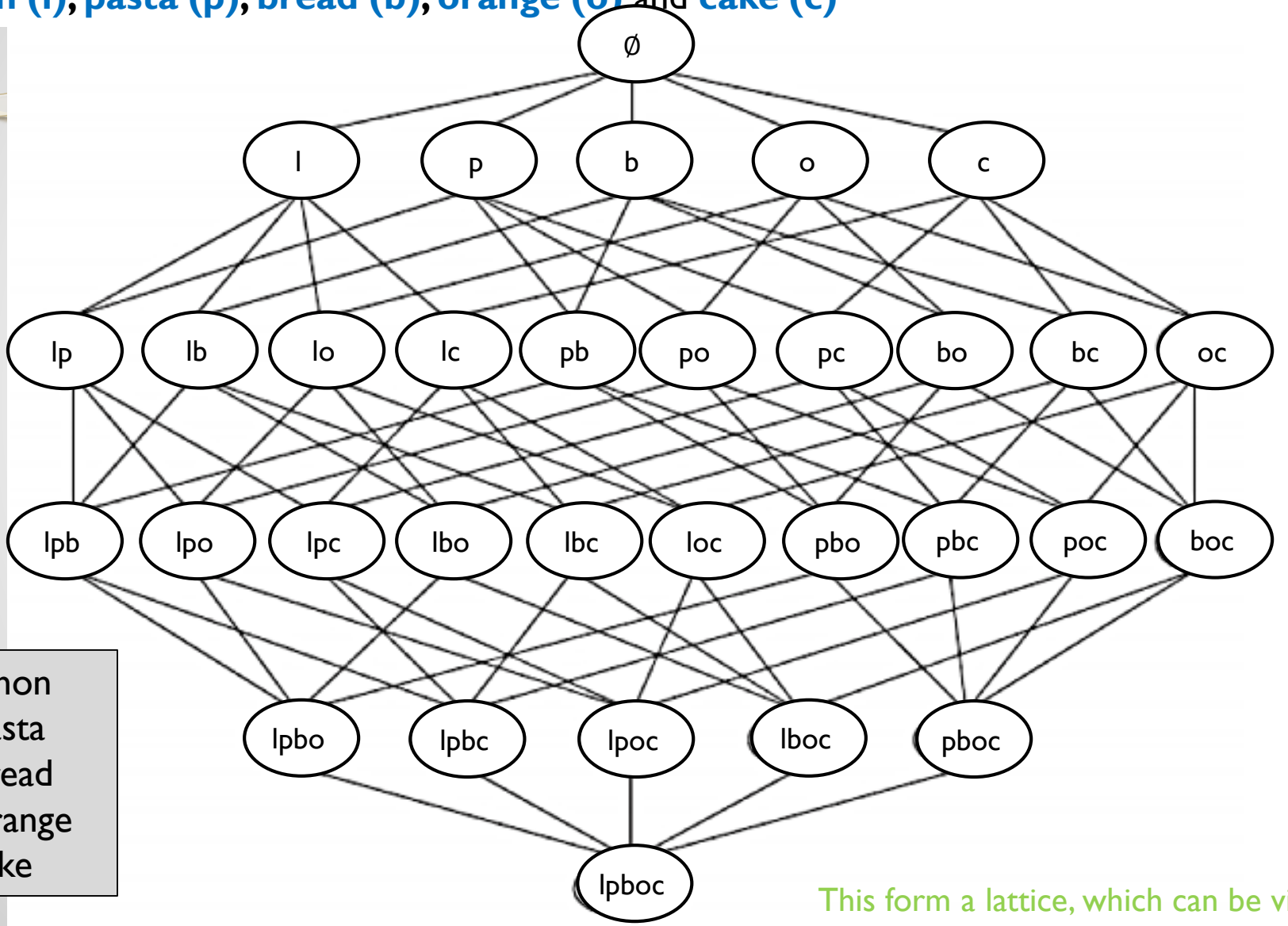
ALGORITHMS

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)**

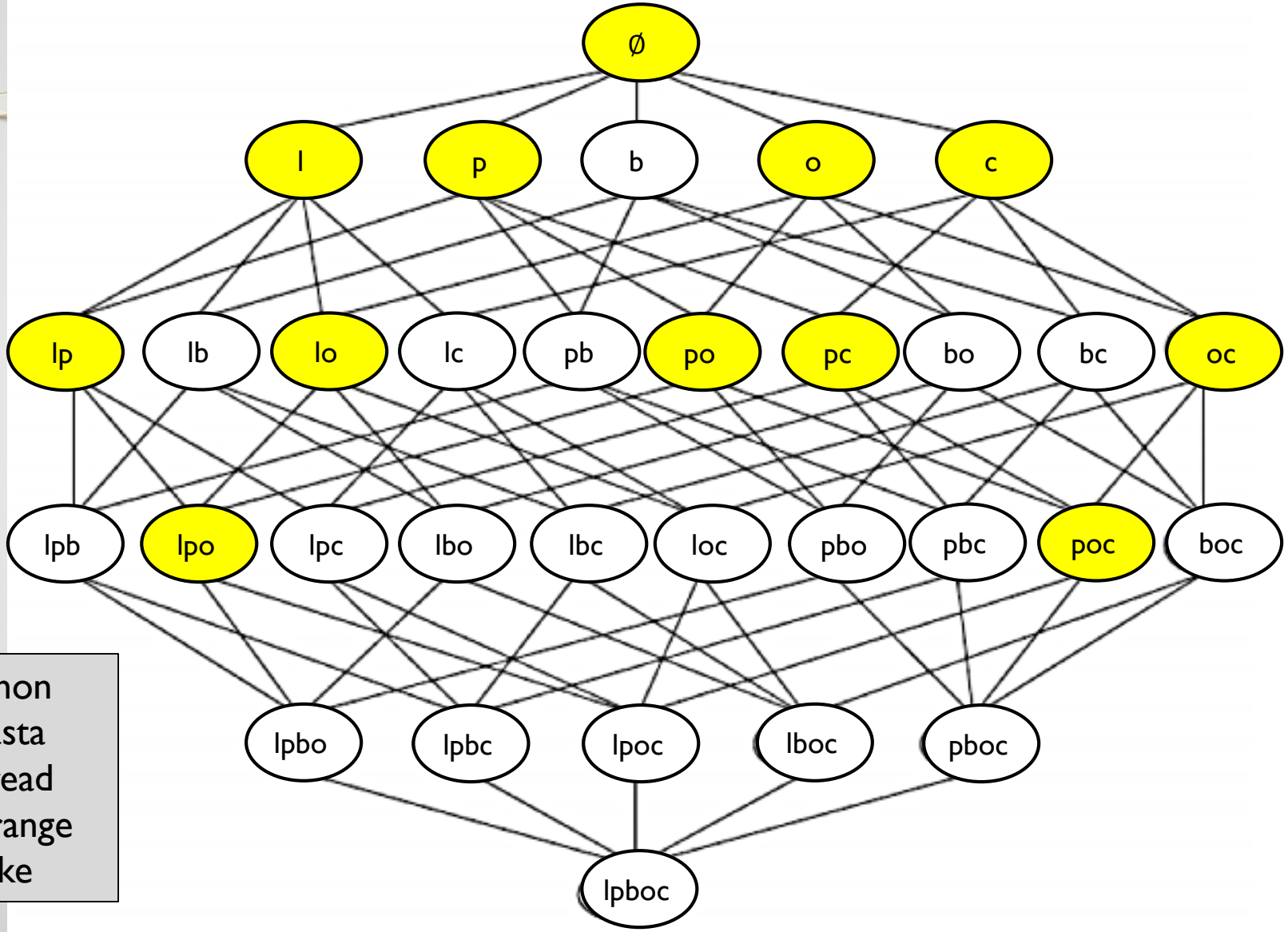


l = lemon
p = pasta
b = bread
0 = orange
c = cake

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

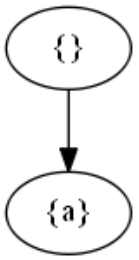
Search space

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

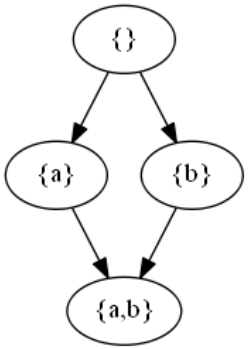


l = lemon
p = pasta
b = bread
0 = orange
c = cake

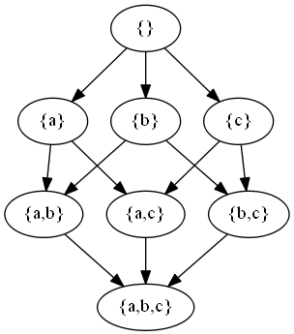
I={A}



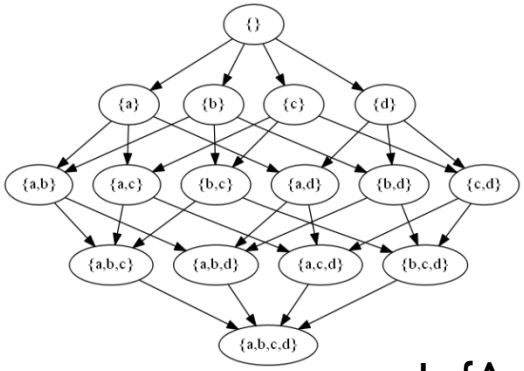
I={A, B}



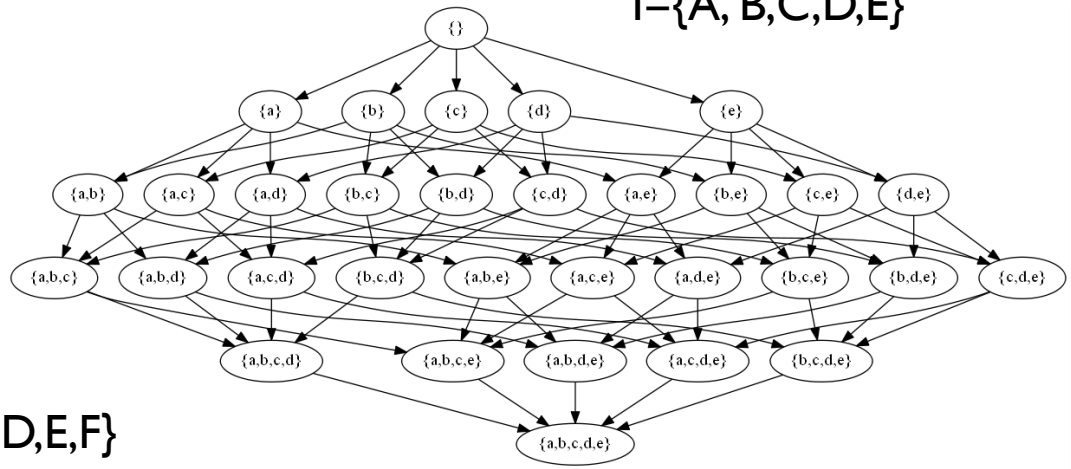
I={A, B,C}



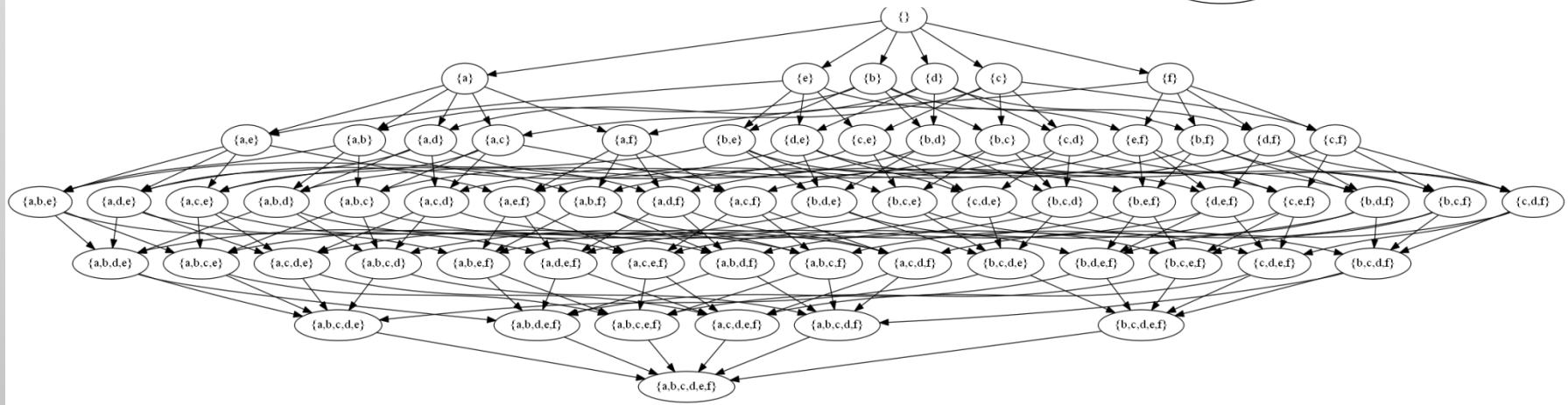
I={A, B,C,D}



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)?

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:

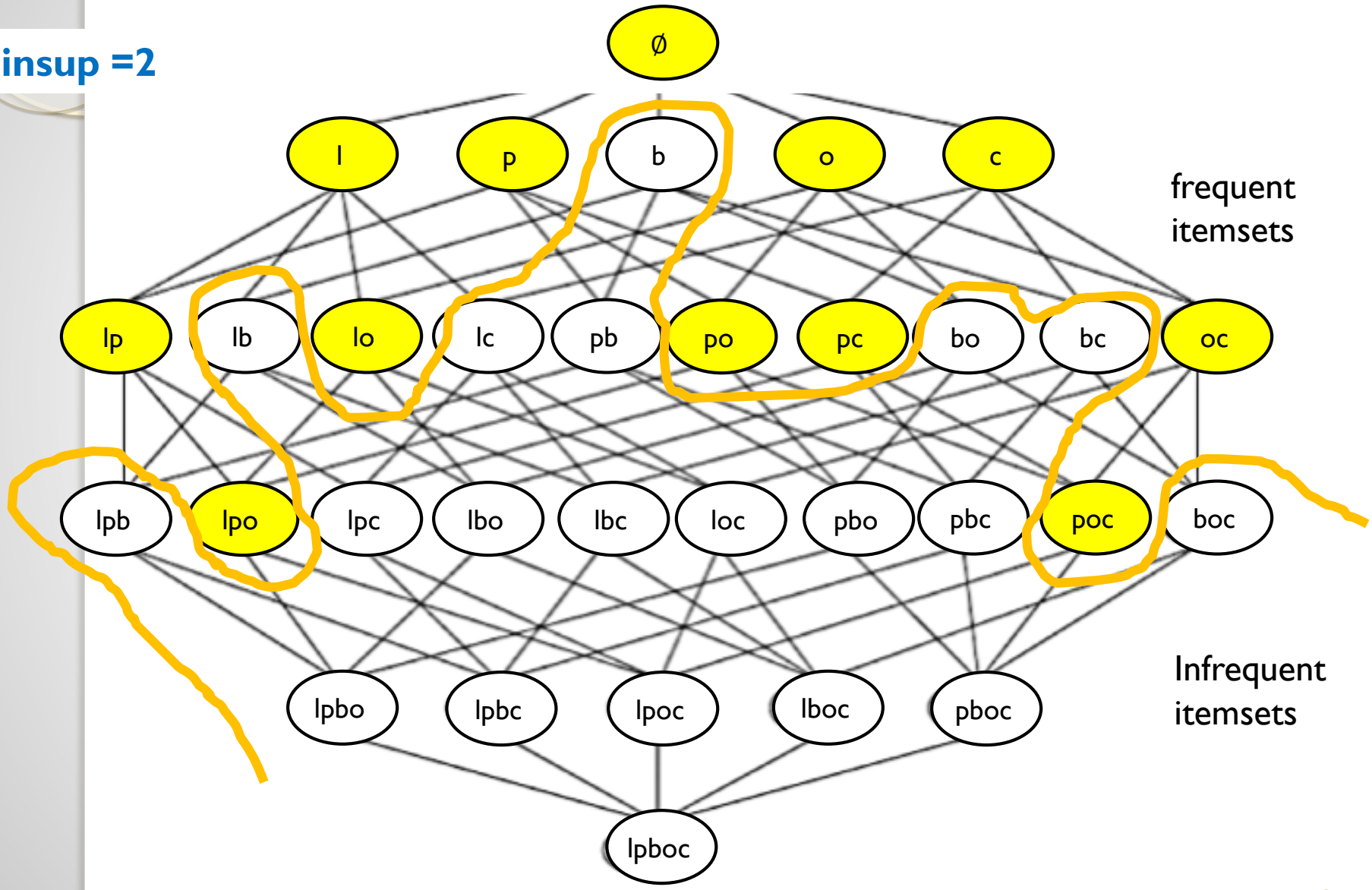
- The support of {pasta} is 4
- The support of {pasta, lemon} is 3
- The support of {pasta, lemon, orange} is 2

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

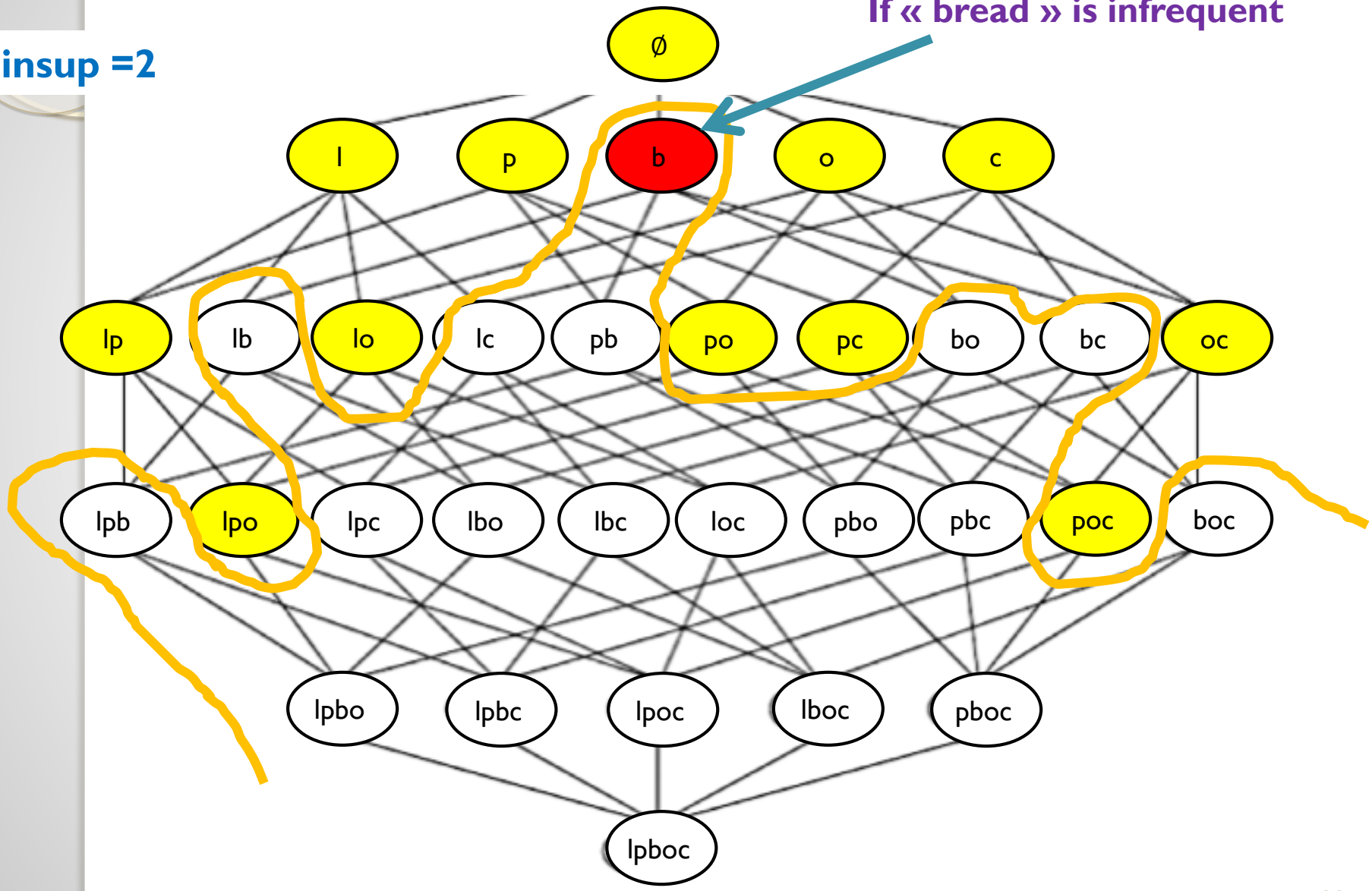
minsup =2



This property is useful to reduce the search space.

Example:

minsup =2

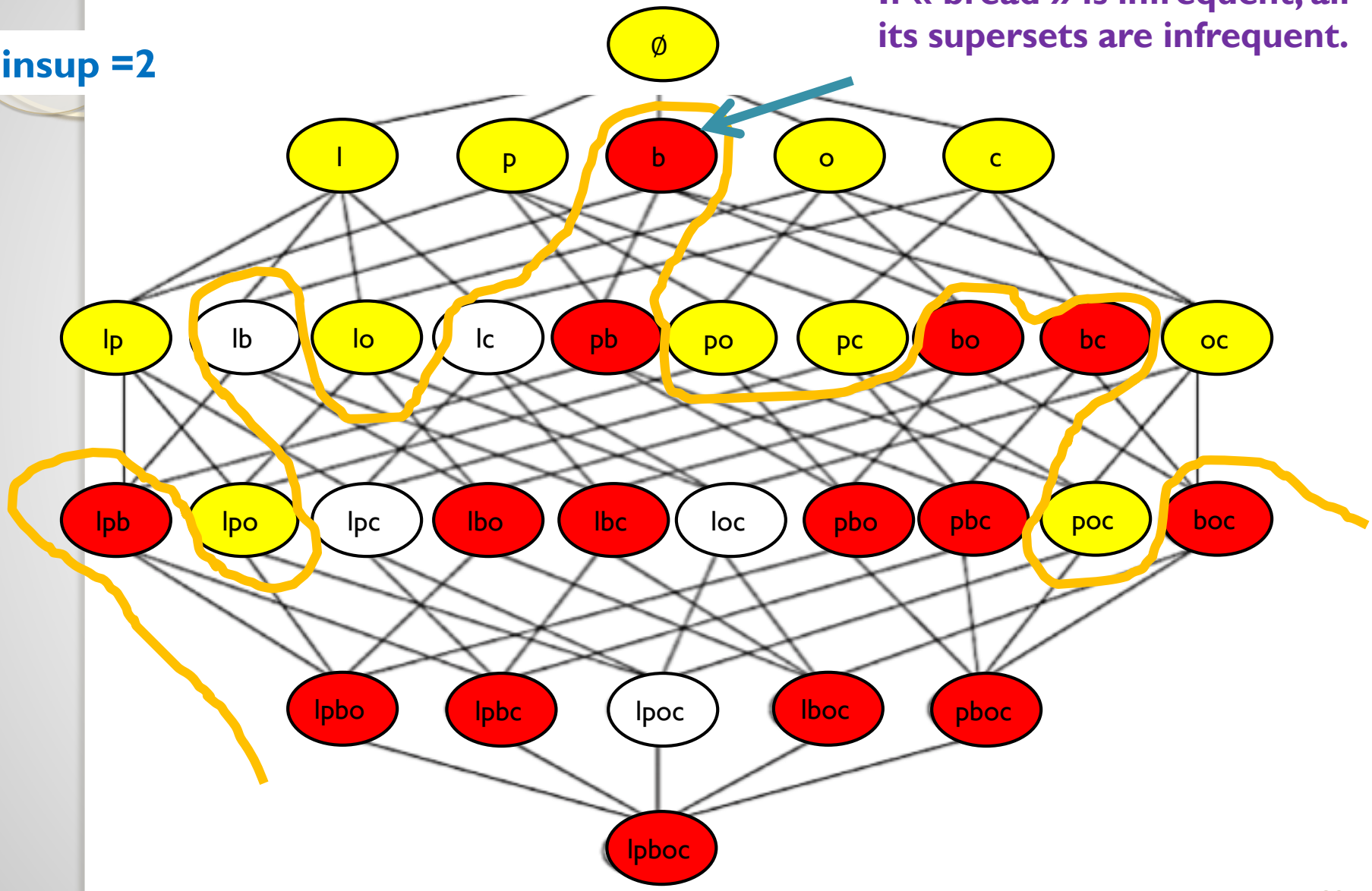


This property is useful to reduce the search space.

Example:

minsup =2

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



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.**

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

{pasta, lemon, orange}

{pasta, orange, cake}

Candidates of size 4

{pasta, lemon, 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

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.

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:

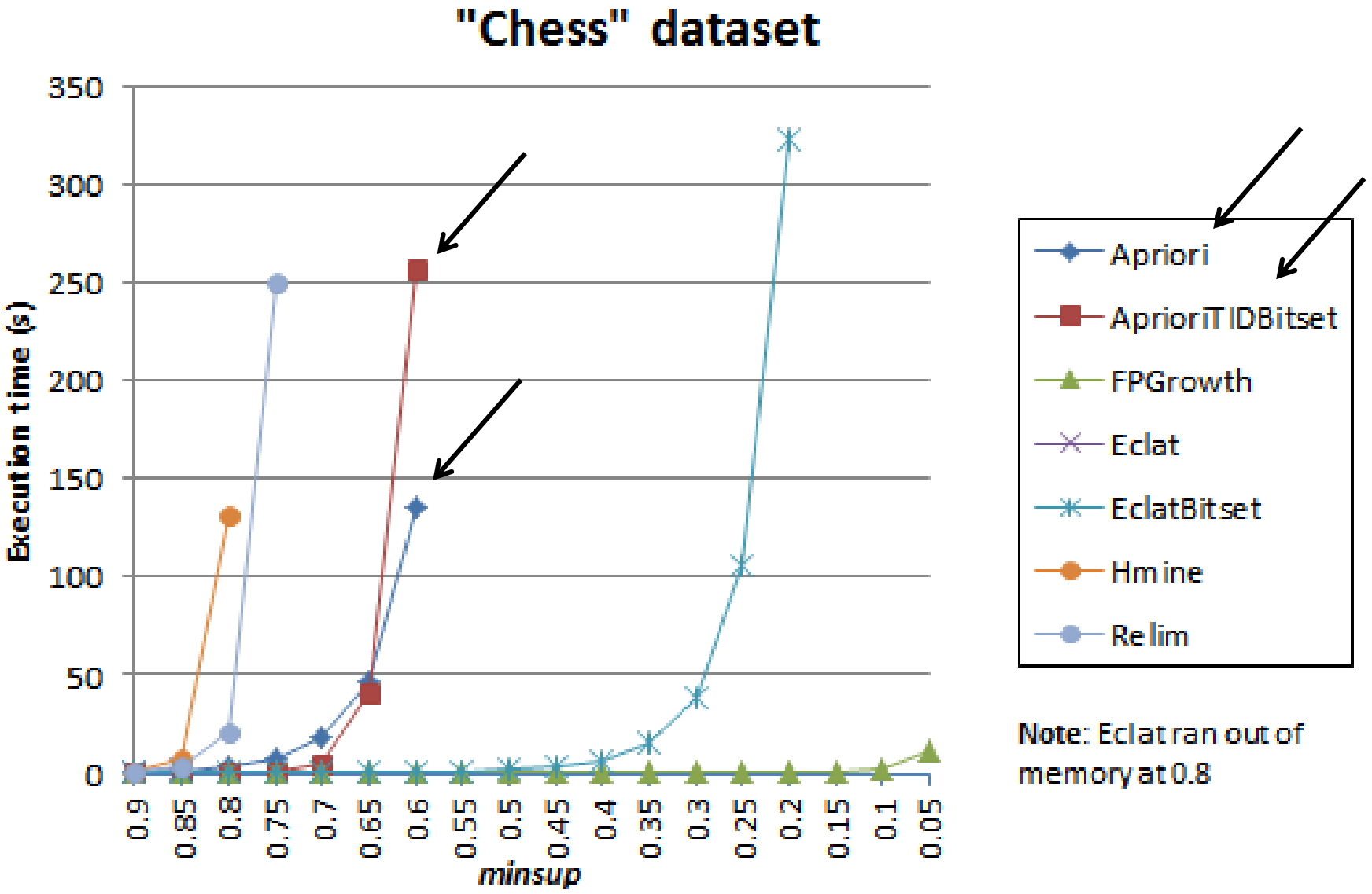
    18 itemsets are considered

PERFORMANCE COMPARISON

How to evaluate this type of algorithms?

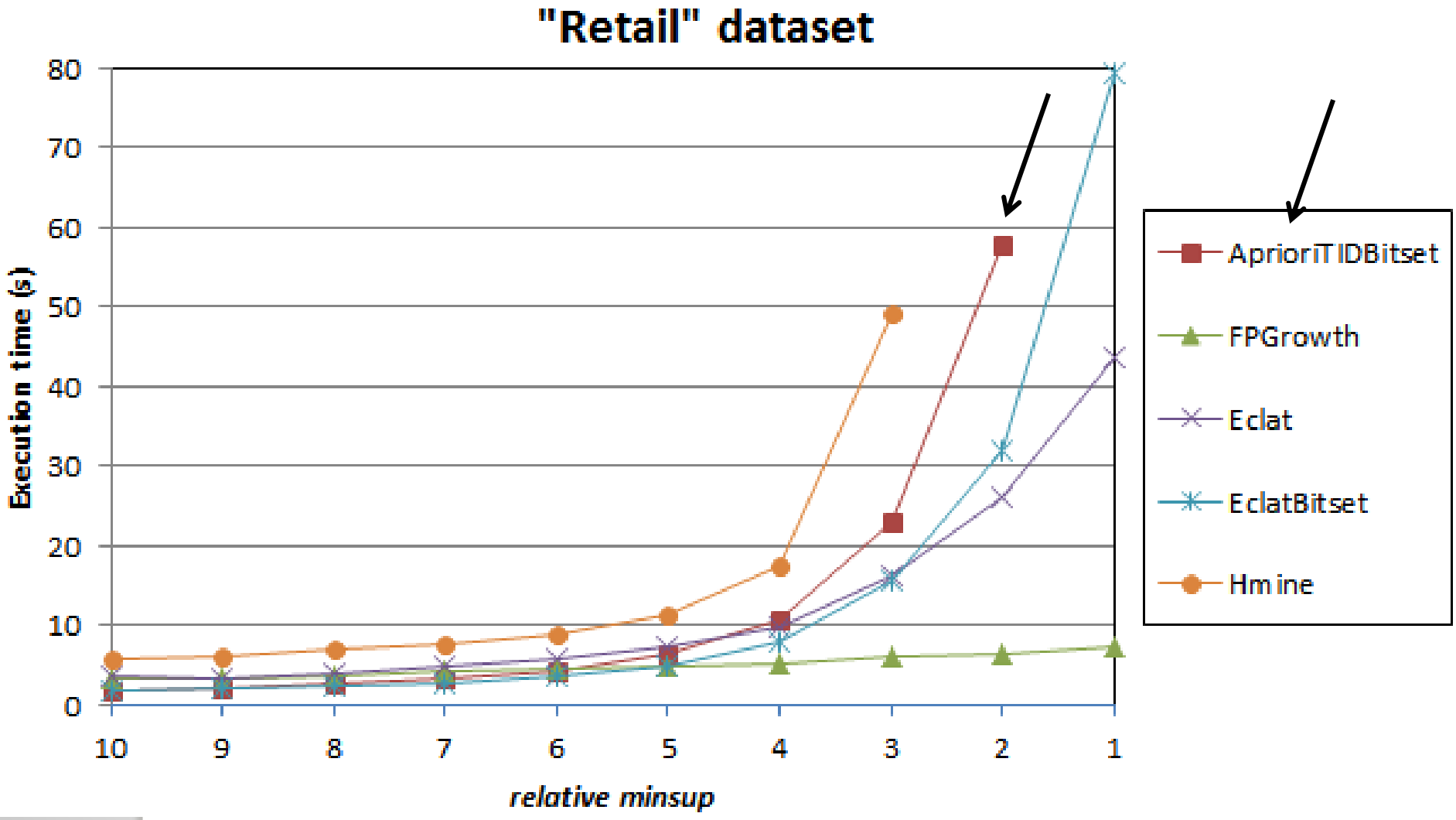
- Execution time,
- Memory used,
- *Scalability:* how the performance is influenced by the number of transactions
- Performance on different types of data:
  - real data,
  - synthetic (fake) data,
  - dense vs sparse data,…
- …

Performance (execution time)



Note: Eclat ran out of memory at 0.8

Performance (execution time)



Performance of Apriori

The performance of Apriori depends on several factors:

- **the *minsup* parameter**: the more it is set low, the larger the search space and the number of itemsets will be.
- **the number of items,**
- **the number of transactions,**
- **The average transaction length.**

Problems of Apriori

- can generate numerous candidates
- requires to scan the database numerous times.
- candidates may not exist in the database.
- …

A FEW OPTIMIZATIONS FOR THE APRIORI ALGORITHM

This is an **advanced topic**

Optimization 1

In terms of data structure:

- Store all items as integers:
  e.g. 1= pasta, 2= orange, 3= bread…

- **Why?**
  - it is faster to compare two integers than to compare two character strings,
  - requires less memory.

Optimization 2

To reduce the time required to calculate the support of itemsets

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

sort transactions by ascending length

- To calculate the support of an itemset of size *k*, only the transactions of size >= k are used.

Optimization 3

To reduce the time required to calculate the support of itemsets

- Replace all identical transactions by a single transactions with a weight.

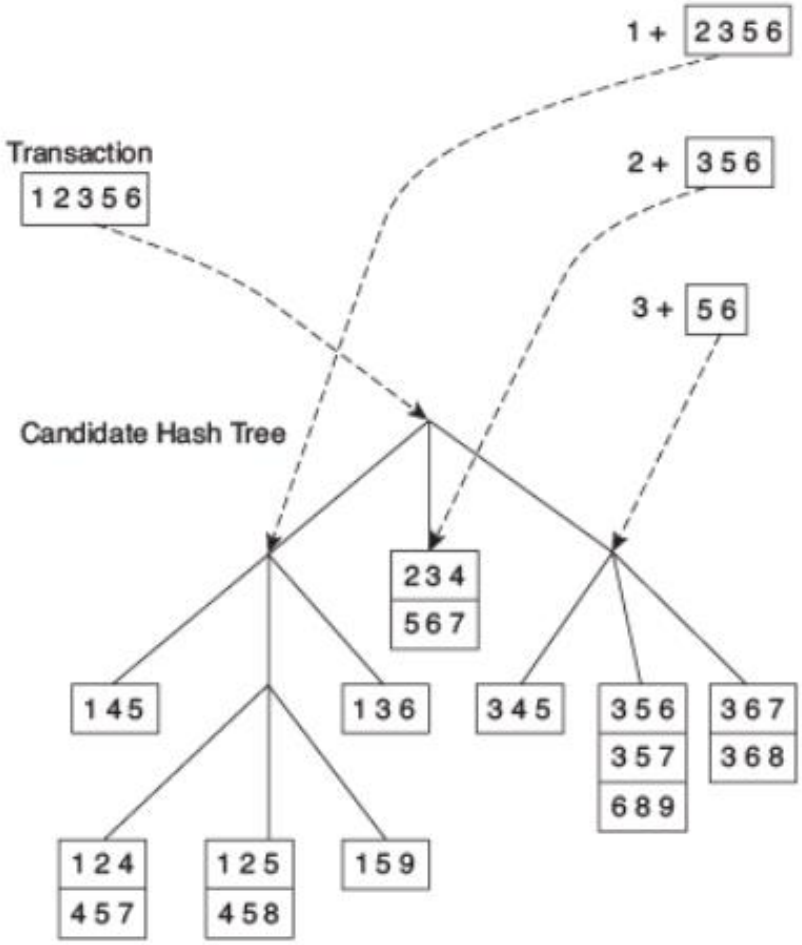
Optimization 4

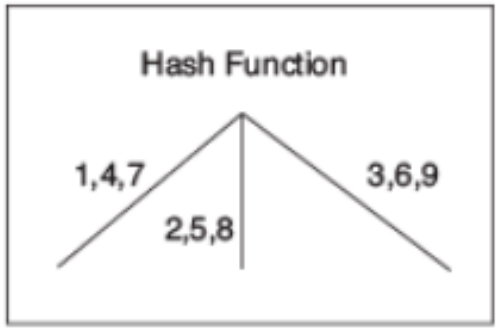
To reduce the time require to calculate the support of itemsets:

- Sort items in transactions according to a total order (**e.g.** alphabetical order).
- Utilize binary search to quickly check if an item appears in a transaction.

Optimization 5

- Store candidates in a hash tree
- To calculate the support of candidates
  - Calculate a hash value based on a transaction to determine if candidates are contained in thetransaction.





Other optimizations

- Sampling and partitioning

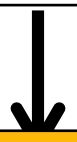
AprioriTID: a varition

AprioriTID:

- Annotate each itemset with the ids of transactions that contain it,
- use the intersection ($\cap$) to calculate the support of itemsets instead of reading the database.

Example →

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



item	transactions containing the item
pasta	T1, T2, T3, T4
lemon	T1, T2, T3
bread	T1
orange	T1, T3, T4
cake	T3, T4

item	transactions containing the item
pasta	T1, T2, T3, T4
lemon	T1, T2, T4
bread	T1
orange	T1, T3, T4
cake	T3, T4

Example: calculating the support of {pasta, lemon} :

transactions({pasta}) ∩ transactions({lemon})

= {T1, T2, T3, T4} ∩ {T1, T2, T4}

= {T1, T2, T4 }

Thus {pasta, lemon} has a support of 3

AprioriTID_Bitset

AprioriTID_bitset:

- Same idea, except that bit vectors are used instead of lists of ids.
- This allows to calculate the intersection using the **Logical_AND**, which is often very fast.

Example →

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

item	transactions containing the item
pasta	1111 (representing T1, T2, T3, T4)
lemon	1101
bread	1000
orange	1011
cake	0011

item	transactions containing the item
pasta	1111
lemon	1101
bread	1000
orange	1011
cake	0011

Example: Calculate the support of {pasta, lemon} :

transactions({pasta}) ∩ transactions({lemon})

= 1111 LOGICAL_AND 1101

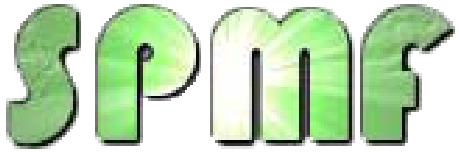
= 1101

Thus {pasta, lemon} has a support of 3

Conclusion

This video has presented:

- The problem of frequent itemset mining
- The **Apriori** algorithm
- Some optimizations



References

- Han and Kamber (2011), Data Mining: Concepts and Techniques, 3rd edition, Morgan Kaufmann Publishers,
- Tan, Steinbach & Kumar (2006), Introduction to Data Mining, Pearson education, ISBN-10: 0321321367
- …