The Eclat Algorithm

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

Mohammed Javeed Zaki: Scalable Algorithms for Association Mining. IEEE Trans. Knowl. Data Eng. 12(3): 372-390 (2000)

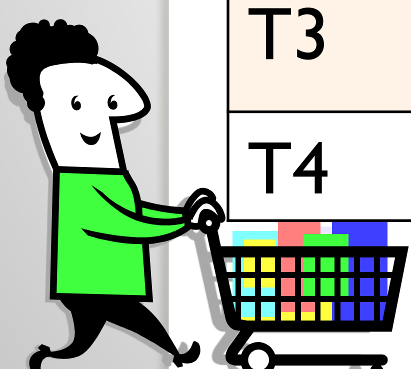
Source code and datasets available in the SPMF library

FREQUENT ITEMSET MINING

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 ECLAT ALGORITHM

Mohammed Javeed Zaki: Scalable Algorithms for Association Mining. IEEE Trans. Knowl. Data Eng. 12(3): 372-390 (2000)

Eclat (Zaki, 2000)

- ECLAT (**E**quivalence **CLA**ss **T**ransformation)
- An algorithm that is generally faster than **Apriori**.
- Utilize a **depth-first search** (contrarily to Apriori/AprioriTID).
- Utilize a ***vertical database*** (as AprioriTID)
- Utilize the concept of **equivalence classes of itemsets** sharing the same prefix.

Definitions

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

  For example:
  I= {pasta, lemon, bread, orange, cake}

- **An itemset** ***X*** **is a set of items** $(X \subseteq I)$.

  e.g. {pasta, lemon} size = 2

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}, …

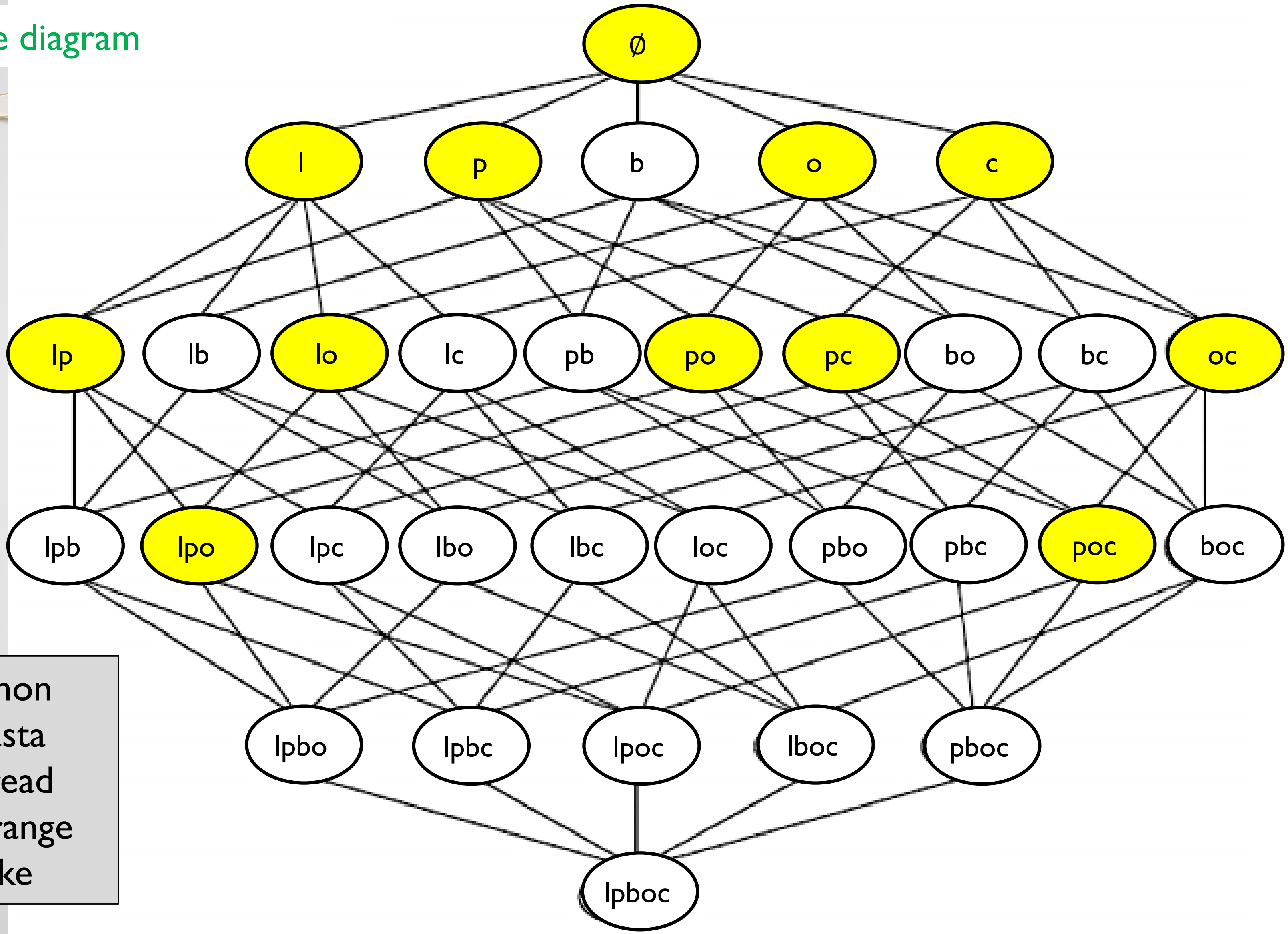
Definitions

Total order

- Without loss of generality, we suppose that all transactions and itemsets are sorted according to a **total order** $\prec$.
- This total order $\prec$ can for example be the alphabetical order.
- **e.g.**
  pasta $\prec$ lemon $\prec$ bread $\prec$ orange $\prec$ cake

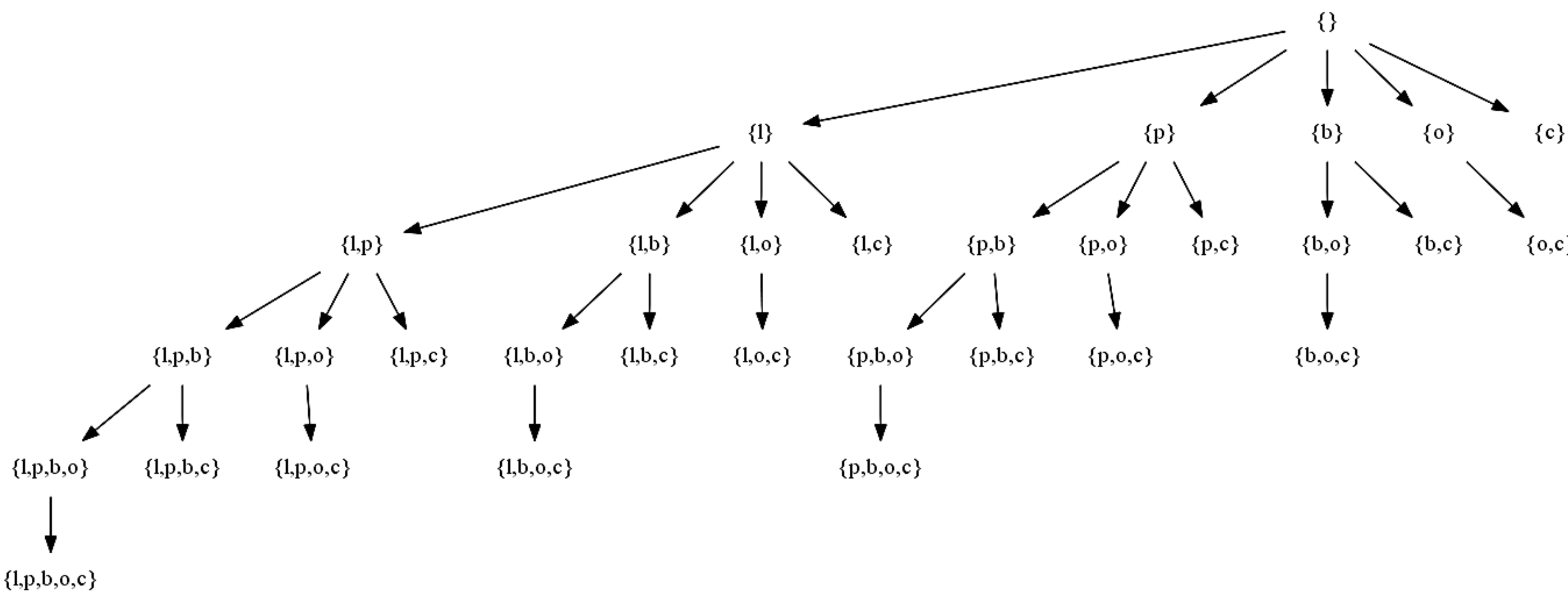
Search space

Hasse diagram



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

Search space



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

The search space can be visualized as a **set enumeration tree**
- Rymon, 1992

Definitions

Equivalence class

- Let there be two itemsets **X** and **Y** of size *k*.
- **X** and **Y** belong to the same equivalence class if the $k - 1$ first items of **X** and **Y** are the same according to the total order.
- **e.g.** An equivalence class:

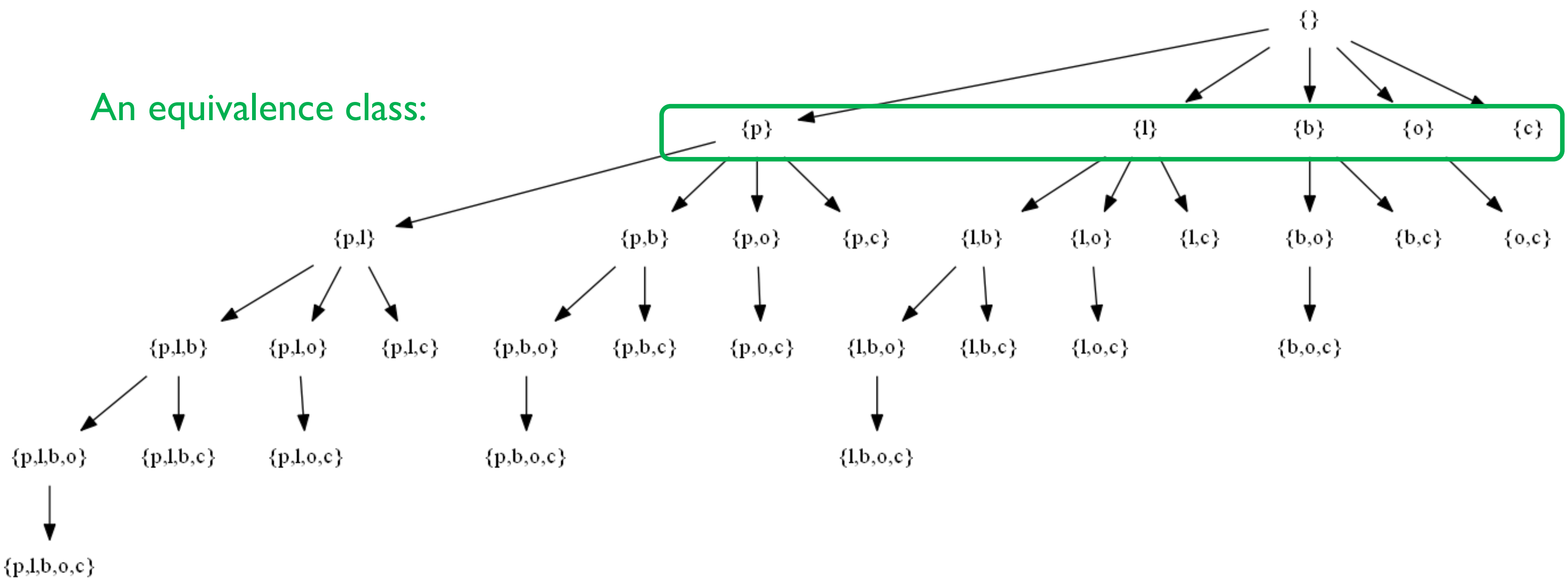
  {pasta, lemon, bread},

  {pasta, lemon, orange},

  {pasta, lemon, cake}

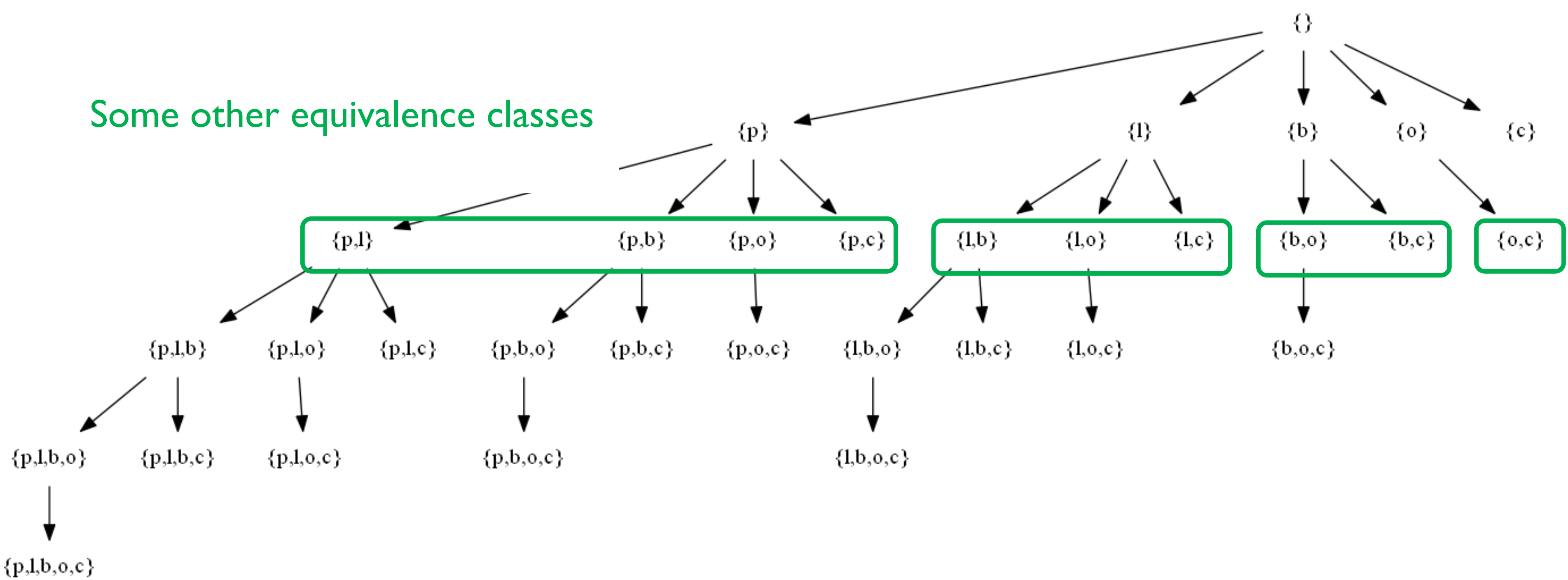
Search space

An equivalence class:



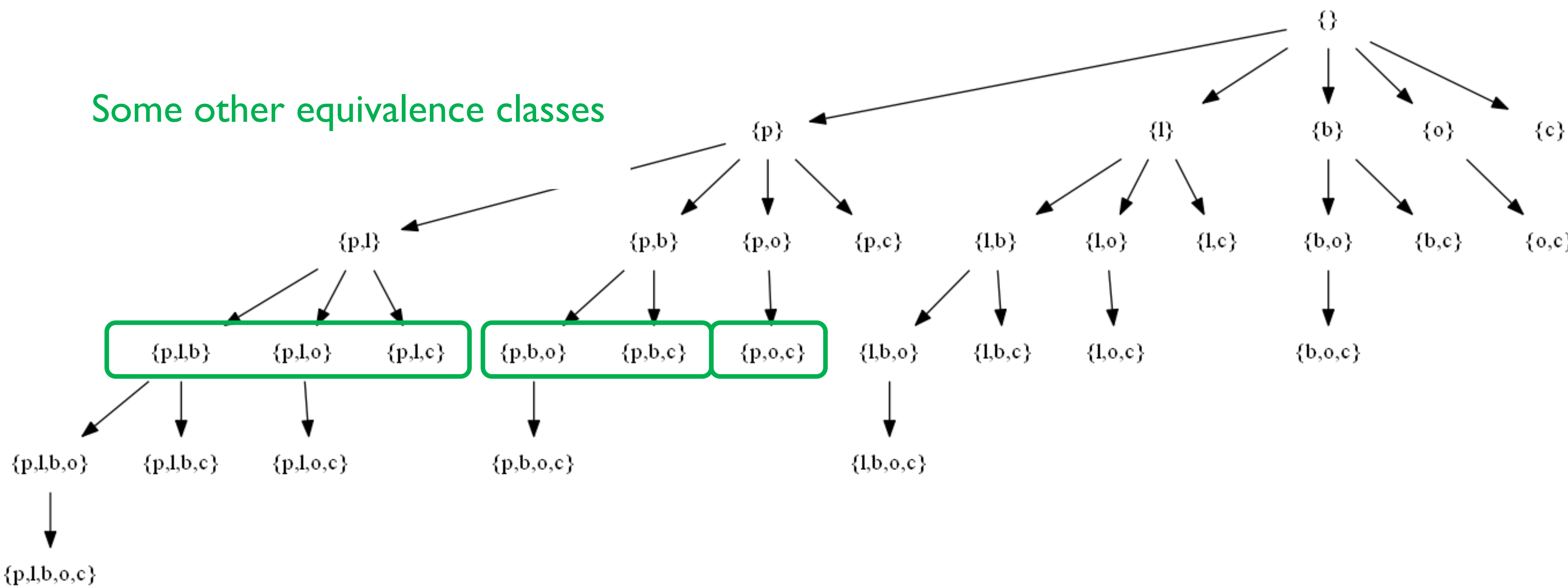
l = lemon
p = pasta
b = bread
0 = orange
c = cake

Search space



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

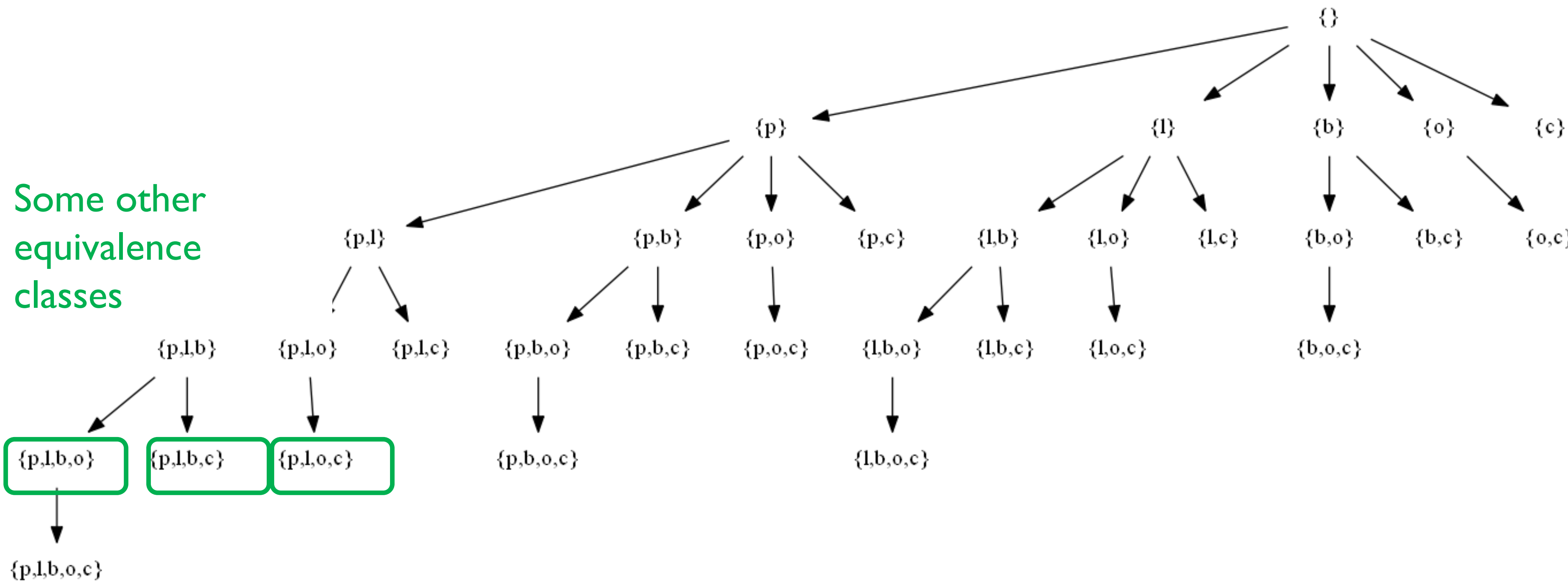
Search space



Some other equivalence classes

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

Search space

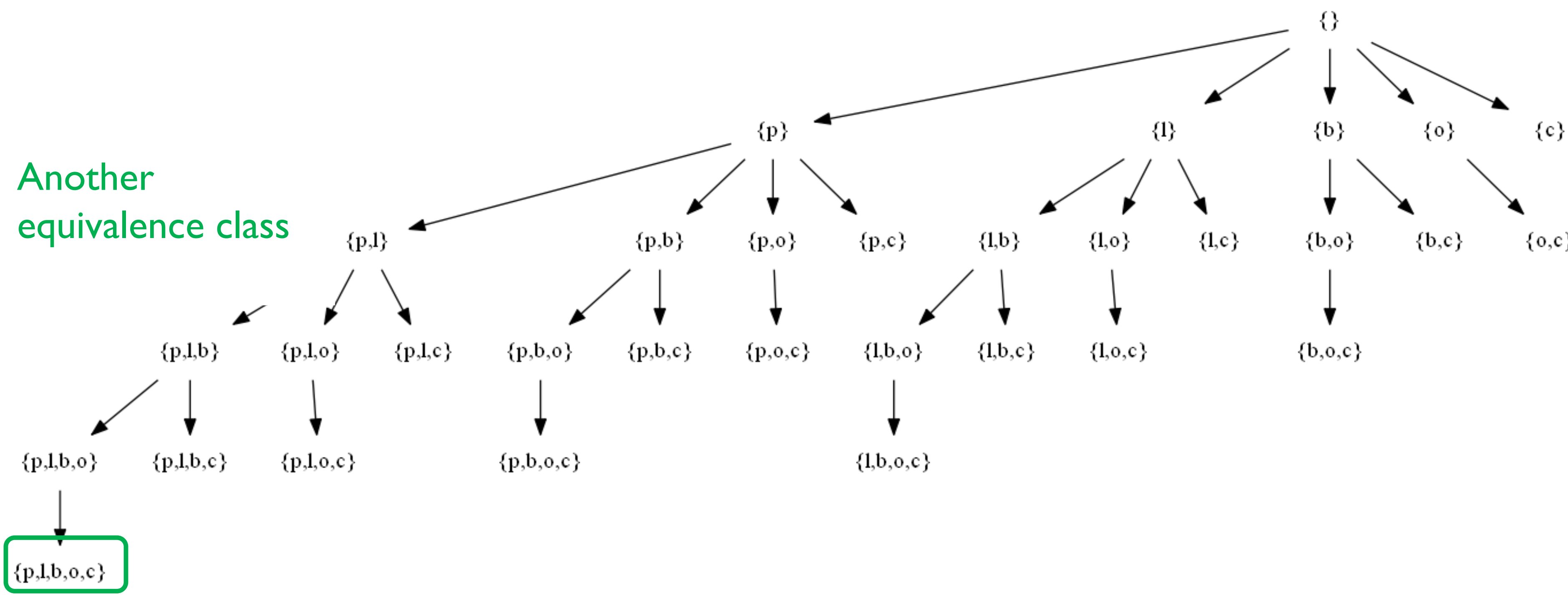


Some other equivalence classes

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

Search space

Another equivalence class



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

The ECLAT algorithm

Step 1: Scan the database to create a vertical representation of the database.

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

The ECLAT algorithm

Step 1: Scan the database to create a vertical representation of the database.

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 ,T4
bread	T1
orange	T1, T3, T4
cake	T3, T4

The ECLAT algorithm

Step 1: Scan the database to create a vertical representation of the database.

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 ,T4
bread	T1
orange	T1, T3, T4
cake	T3, T4

Each line is called a *TID-list* (Transaction ID List)

The ECLAT algorithm

Step 2: This is the first equivalence class.

pasta	T1, T2, T3, T4
lemon	T1, T2 ,T4
bread	T1
orange	T1, T3, T4
cake	T3, T4

The ECLAT algorithm

Step 2: This is the first equivalence class.

pasta	T1, T2, T3, T4
lemon	T1, T2 ,T4
~~bread~~	~~T1~~
orange	T1, T3, T4
cake	T3, T4

ECLAT eliminates infrequent itemsets

(*minsup* = 2)

The ECLAT algorithm

Step 2: This is the first equivalence class.

pasta	T1, T2, T3, T4
lemon	T1, T2 ,T4
~~bread~~	~~T1~~
orange	T1, T3, T4
cake	T3, T4

ECLAT eliminates infrequent itemsets

(*minsup* = 2)

ECLAT outputs the frequent itemsets with 1 items

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

The ECLAT algorithm

Step 3: ECLAT combines combine itemsets of the equivalence class to generate equivalence classes of size K+1

Pasta	T1, T2, T3, T4
lemon	T1, T2 T4
Orange	T1, T3, T4
Cake	T3, T4

pasta, lemon	T1,T2,T4
pasta, orange	T1,T3,T4
pasta, cake	T3,T4

The ECLAT algorithm

Step 3: ECLAT combines combine itemsets of the equivalence class to generate equivalence classes of size K+1

Pasta	T1, T2, T3, T4
lemon	T1, T2 T4
Orange	T1, T3, T4
Cake	T3, T4

pasta, lemon	T1,T2,T4
pasta, orange	T1,T3,T4
pasta, cake	T3,T4

lemon, orange	T1,T4
lemon, cake	T3,T4

The ECLAT algorithm

Step 3: ECLAT combines combine itemsets of the equivalence class to generate equivalence classes of size K+1

Pasta	T1, T2, T3, T4
lemon	T1, T2 T4
Orange	T1, T3, T4
Cake	T3, T4

pasta, lemon	T1,T2,T4
pasta, orange	T1,T3,T4
pasta, cake	T3,T4

lemon, orange	T1,T4
lemon, cake	T3,T4

orange, cake	T3,T4

The ECLAT algorithm

Step 3: ECLAT combines combine itemsets of the equivalence class to generate equivalence classes of size K+1

Pasta	T1, T2, T3, T4
lemon	T1, T2 T4
Orange	T1, T3, T4
Cake	T3, T4

pasta, lemon	T1,T2,T4
pasta, orange	T1,T3,T4
pasta, cake	T3,T4

lemon, orange	T1,T4
lemon, cake	T3,T4

orange, cake	T3,T4

Then, ECLAT eliminates infrequent itemsets and output the frequent itemsets: {pasta,lemon}, {pasta,orange}, {pasta,cake}, {lemon,cake}, {orange, cake},{lemon,orange}

The ECLAT algorithm

Step 4: ECLAT recursively process each equivalence class in the same way. Consider the first one:

pasta, lemon	T1, T2, T4
pasta, orange	T1, T3, T4
pasta, cake	T3, T4

The ECLAT algorithm

Step 4: ECLAT recursively process each equivalence class in the same way. Consider the first one:

pasta, lemon	T1,T2,T4
pasta, orange	T1,T3,T4
pasta, cake	T3,T4

pasta, lemon, orange	T1,T4
pasta, lemon, cake	T4

The ECLAT algorithm

Step 4: ECLAT recursively process each equivalence class in the same way. Consider the first one:

pasta, lemon	T1,T2,T4
pasta, orange	T1,T3,T4
pasta, cake	T3,T4

pasta, lemon, orange	T1,T4
pasta, lemon, cake	T4

pasta, orange, cake	T3,T4

The ECLAT algorithm

Step 4: ECLAT recursively process each equivalence class in the same way. Consider the first one:

pasta, lemon	T1,T2,T4
pasta, orange	T1,T3,T4
pasta, cake	T3,T4

pasta, lemon, orange	T1,T4
~~pasta, lemon, cake~~	~~T4~~

pasta, orange, cake	T3,T4

ECLAT eliminates infrequent itemsets and output the frequent itemsets

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

The ECLAT algorithm

Step 4: ECLAT recursively process each equivalence class in the same way. Consider the next equivalence class:

lemon, orange	T1
lemon, cake	T3, T4

The ECLAT algorithm

Step 4: ECLAT recursively process each equivalence class in the same way. Consider the next equivalence class:

lemon, orange	T1
lemon, cake	T3, T4

lemon,orange,cake	

The ECLAT algorithm

Step 4: ECLAT recursively process each equivalence class in the same way. Consider the next equivalence class:

lemon, orange	T1
lemon, cake	T3, T4

~~lemon,orange,cake~~	

This itemset is infrequent.
It is eliminated.

The ECLAT algorithm

All other equivalence classes contain a single itemset. Thus, no candidates can be generated.

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

Performance

- In this example:
  - **ECLAT** has explored 14 itemsets.
  - **Apriori** would have explored 18 itemsets.
- How is the performance of ECLAT?
  - ECLAT scans the database a single time to create a vertical database.
  - Then, the most costly operation is the intersection of TID-lists.
  - In the worst case, these lists have the size of the database.

  - **Several possible optimizations→**

Optimization 1: total order

- Does the choice of a total order $\prec$ a influences the performance?

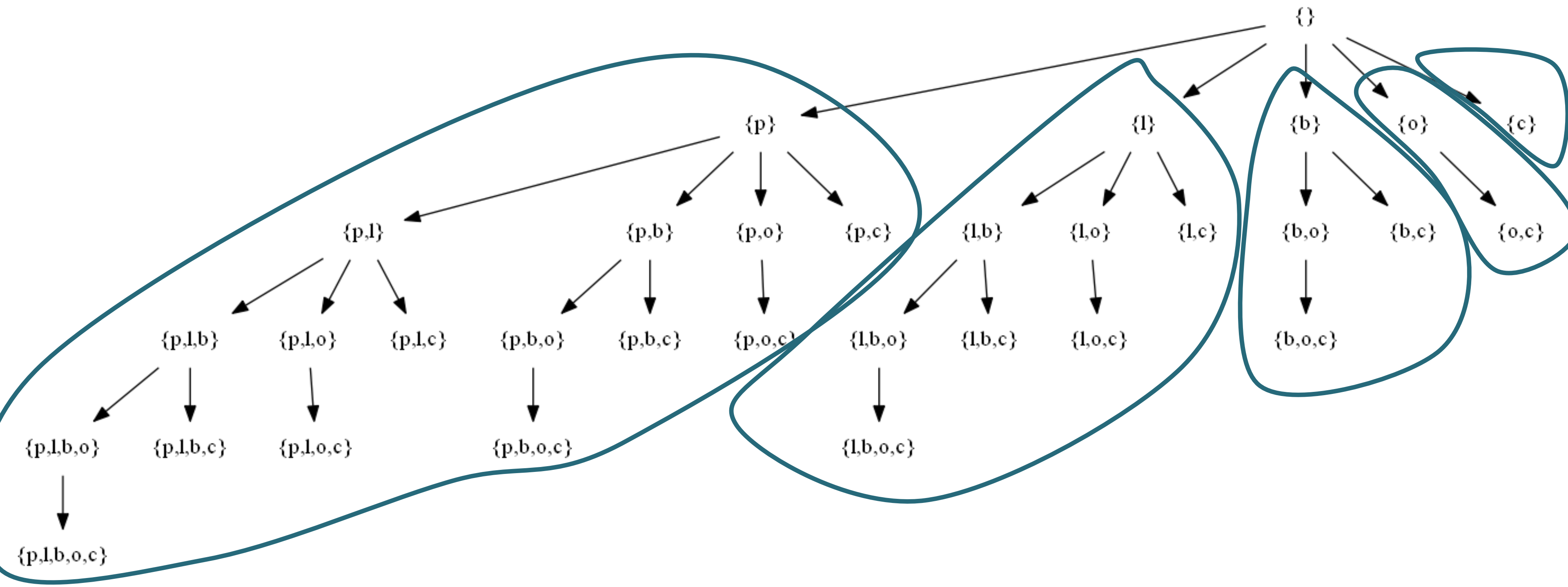
Optimization 1: total order

- Does the choice of a total order $\prec$ a influences the performance?
- Yes, but which one to choose?
  - The alphabetical order?

Optimization 1: total order

- Does the choice of a total order $\prec$ a influences the performance?
- Yes, but which one to choose?
  - The alphabetical order?
- A better choice: the order of increasing support.

Observation



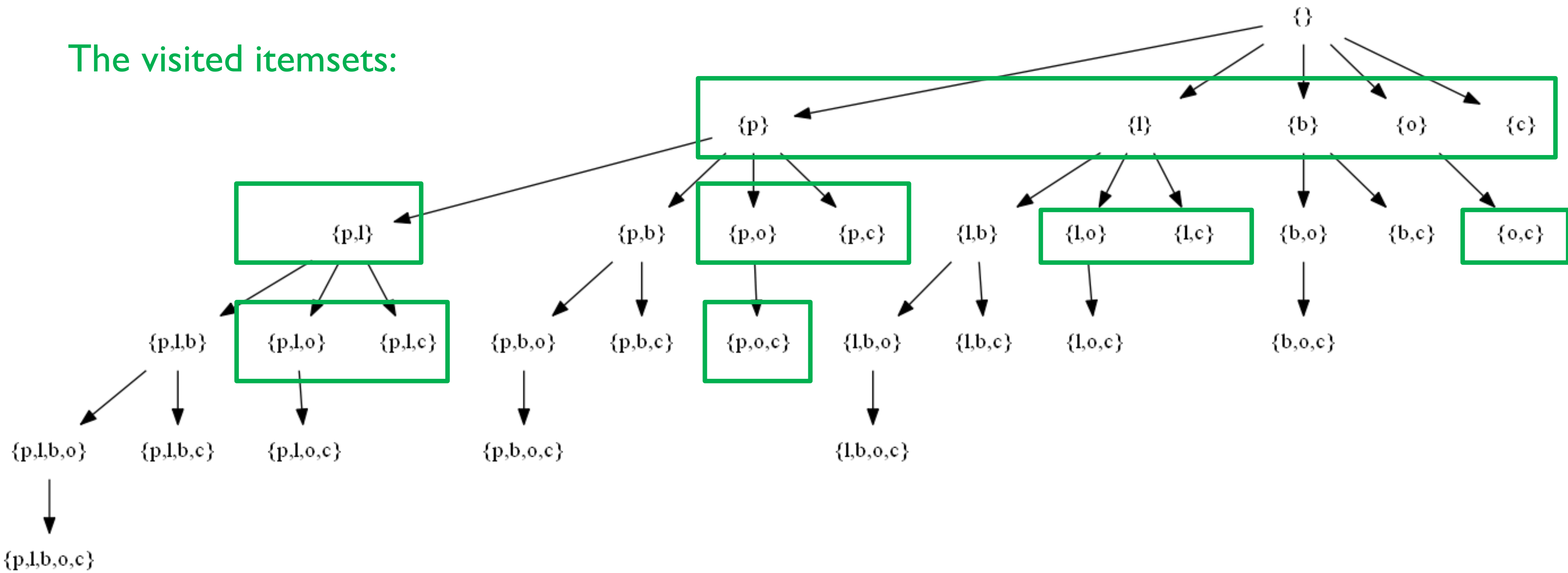
l = lemon
p = pasta
b = bread
0 = orange
c = cake

If an item is smaller according to the total order, its sub-tree will be larger.

Search space

pasta ≺ lemon≺ bread ≺ orange ≺ cake

The visited itemsets:



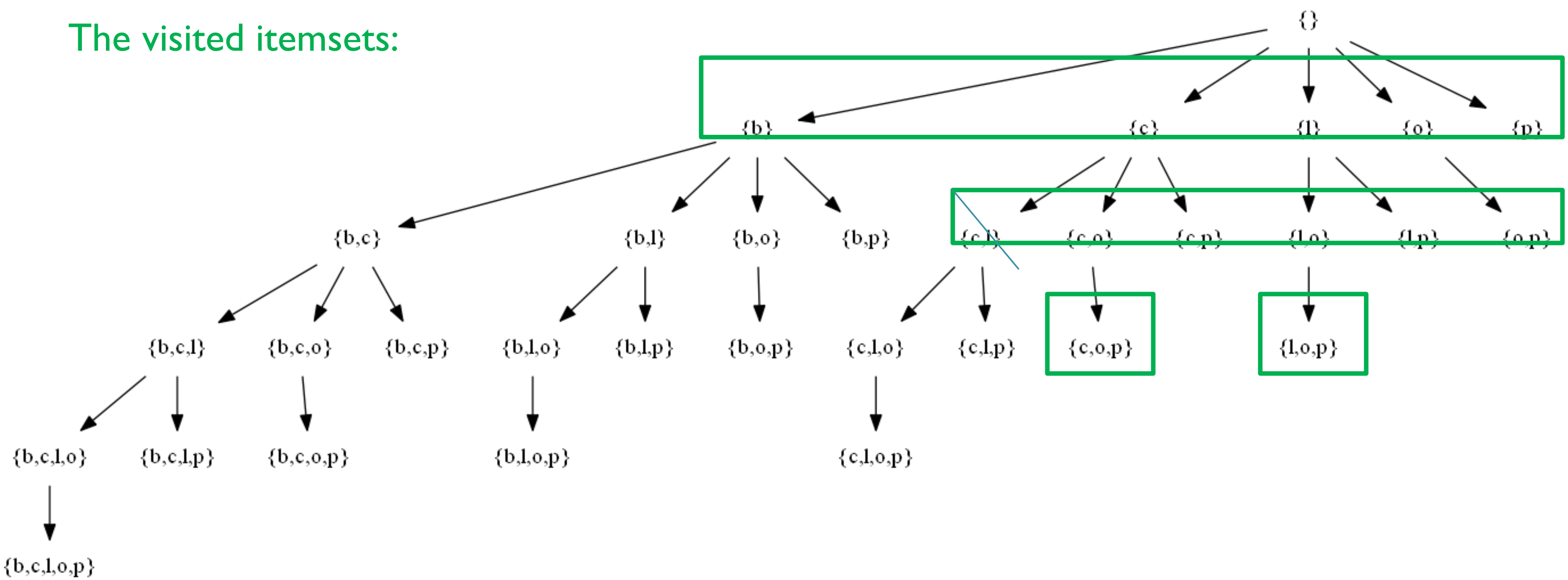
l = lemon
p = pasta
b = bread
0 = orange
c = cake

14 have been explored

Search space

bread≺ cake≺ lemon ≺ orange ≺ pasta

The visited itemsets:



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

13 have been explored

Optimization 2 - intersection

How to reduce the cost of intersections?

- Utilize bit vectors the represent the lists of transaction ids
- Will be fast if the number of 1 is large compared to the number of zeros

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

Optimization 3 - memory

Consider an equivalence class:

{ABCD, ABCE, ABCF, ABCG, ABCH}

It can be stored more efficiently as:

P = {ABC}

E = {E, F, G, H}

Pseudocode of ECLAT

ECLAT(an equivalence class C)
FOR EACH $X \in C$
    $T = \emptyset$
    FOR EACH $Y \in C$ such that X $\prec$ Y
        $R = X \cup Y$
        $\boldsymbol{t(R) = t(X) \cap t(Y)}$
        IF $sup(R) \geq minsup$
            THEN Output R
                $T = T \cup \{R\}$
    END FOR
    IF T $\neq \emptyset$ **THEN ECLAT(T)**
END FOR

Conclusion

This video has presented:

- The **Eclat** algorithm
- Some optimizations

References

- Chapter 8 and 9. Han and Kamber (2011), Data Mining: Concepts and Techniques, 3rd edition, Morgan Kaufmann Publishers,
- Chapter 4. Tan, Steinbach & Kumar (2006), Introduction to Data Mining, Pearson education, ISBN-10: 0321321367
- *Mohammed Javeed Zaki: Scalable Algorithms for Association Mining. IEEE Trans. Knowl. Data Eng. 12(3): 372-390 (2000)*
- *Zaki, M.J., Gouda, K.: Fast vertical mining using diffsets. Technical Report 01-1, Computer Science Dept., Rensselaer Polytechnic Institute (March 2001) 10*