The Eclat Algorithm

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


Source code and datasets available in the SPMF library
FREQUENT ITEMSET MINING
The problem of frequent itemset mining

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

<table>
<thead>
<tr>
<th>Transaction</th>
<th>Items appearing in the transaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>{pasta, lemon, bread, orange}</td>
</tr>
<tr>
<td>T2</td>
<td>{pasta, lemon}</td>
</tr>
<tr>
<td>T3</td>
<td>{pasta, orange, cake}</td>
</tr>
<tr>
<td>T4</td>
<td>{pasta, lemon, orange, cake}</td>
</tr>
</tbody>
</table>
### Example

<table>
<thead>
<tr>
<th>Transaction</th>
<th>Items appearing in the transaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>{pasta, lemon, bread, orange}</td>
</tr>
<tr>
<td>T2</td>
<td>{pasta, lemon}</td>
</tr>
<tr>
<td>T3</td>
<td>{pasta, orange, cake}</td>
</tr>
<tr>
<td>T4</td>
<td>{pasta, lemon, orange cake}</td>
</tr>
</tbody>
</table>

For $\text{minsup} = 2$, 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 $\text{minsup}$ 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

Eclat (Zaki, 2000)

- **ECLAT** (Equivalence CLAss Transformation)
- 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 \( I = \{I_1, I_2, \ldots, I_m\} \) be the set of items (products) sold in a retail store.

For example:
\( I = \{\text{pasta, lemon, bread, orange, cake}\} \)

- An itemset \( X \) is a set of items \( (X \subseteq I) \).
  e.g. \( \{\text{pasta, lemon}\} \) 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
The search space can be visualized as a set enumeration tree - Rymon, 1992

l = lemon
p = pasta
b = bread
O = orange
c = cake
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:
  
  \{	ext{pasta, lemon, bread}\},
  
  \{	ext{pasta, lemon, orange}\},
  
  \{	ext{pasta, lemon, cake}\}
An equivalence class:
Search space

Some other equivalence classes

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

Some other equivalence classes

\[ l = \text{l}e\text{mon} \]
\[ p = \text{p}asta \]
\[ b = \text{b}read \]
\[ 0 = \text{o}range \]
\[ c = \text{cake} \]
Search space

Some other equivalence classes

\[ l = \text{lemon} \]
\[ p = \text{pasta} \]
\[ b = \text{bread} \]
\[ o = \text{orange} \]
\[ c = \text{cake} \]
Search space

Another equivalence class

\( l = \text{lemon} \)
\( p = \text{pasta} \)
\( b = \text{bread} \)
\( o = \text{orange} \)
\( c = \text{cake} \)
The ECLAT algorithm

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

<table>
<thead>
<tr>
<th>Transaction</th>
<th>Items appearing in the transaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>{pasta, lemon, bread, orange}</td>
</tr>
<tr>
<td>T2</td>
<td>{pasta, lemon}</td>
</tr>
<tr>
<td>T3</td>
<td>{pasta, orange, cake}</td>
</tr>
<tr>
<td>T4</td>
<td>{pasta, lemon, orange, cake}</td>
</tr>
</tbody>
</table>
The ECLAT algorithm

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

<table>
<thead>
<tr>
<th>Transaction</th>
<th>Items appearing in the transaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>{pasta, lemon, bread, orange}</td>
</tr>
<tr>
<td>T2</td>
<td>{pasta, lemon}</td>
</tr>
<tr>
<td>T3</td>
<td>{pasta, orange, cake}</td>
</tr>
<tr>
<td>T4</td>
<td>{pasta, lemon, orange, cake}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>item</th>
<th>transactions containing the item</th>
</tr>
</thead>
<tbody>
<tr>
<td>pasta</td>
<td>T1, T2, T3, T4</td>
</tr>
<tr>
<td>lemon</td>
<td>T1, T2, T4</td>
</tr>
<tr>
<td>bread</td>
<td>T1</td>
</tr>
<tr>
<td>orange</td>
<td>T1, T3, T4</td>
</tr>
<tr>
<td>cake</td>
<td>T3, T4</td>
</tr>
</tbody>
</table>
**The ECLAT algorithm**

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

<table>
<thead>
<tr>
<th>Transaction</th>
<th>Items appearing in the transaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>{pasta, lemon, bread, orange}</td>
</tr>
<tr>
<td>T2</td>
<td>{pasta, lemon}</td>
</tr>
<tr>
<td>T3</td>
<td>{pasta, orange, cake}</td>
</tr>
<tr>
<td>T4</td>
<td>{pasta, lemon, orange, cake}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>item</th>
<th>transactions containing the item</th>
</tr>
</thead>
<tbody>
<tr>
<td>pasta</td>
<td>T1, T2, T3, T4</td>
</tr>
<tr>
<td>lemon</td>
<td>T1, T2, T4</td>
</tr>
<tr>
<td>bread</td>
<td>T1</td>
</tr>
<tr>
<td>orange</td>
<td>T1, T3, T4</td>
</tr>
<tr>
<td>cake</td>
<td>T3, T4</td>
</tr>
</tbody>
</table>

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

Step 2: This is the first equivalence class.

<table>
<thead>
<tr>
<th>Item</th>
<th>T1, T2, T3, T4</th>
</tr>
</thead>
<tbody>
<tr>
<td>pasta</td>
<td></td>
</tr>
<tr>
<td>lemon</td>
<td>T1, T2, T4</td>
</tr>
<tr>
<td>bread</td>
<td>T1</td>
</tr>
<tr>
<td>orange</td>
<td>T1, T3, T4</td>
</tr>
<tr>
<td>cake</td>
<td>T3, T4</td>
</tr>
</tbody>
</table>
The ECLAT algorithm

**Step 2:** This is the first equivalence class.

<table>
<thead>
<tr>
<th>Item</th>
<th>Transactions</th>
</tr>
</thead>
<tbody>
<tr>
<td>pasta</td>
<td>T1, T2, T3, T4</td>
</tr>
<tr>
<td>lemon</td>
<td>T1, T2, T4</td>
</tr>
<tr>
<td>bread</td>
<td>T1</td>
</tr>
<tr>
<td>orange</td>
<td>T1, T3, T4</td>
</tr>
<tr>
<td>cake</td>
<td>T3, T4</td>
</tr>
</tbody>
</table>

ECLAT eliminates infrequent itemsets

*(minsup = 2)*
The ECLAT algorithm

**Step 2:** This is the first equivalence class.

<table>
<thead>
<tr>
<th>Item</th>
<th>Transaction Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>pasta</td>
<td>T1, T2, T3, T4</td>
</tr>
<tr>
<td>lemon</td>
<td>T1, T2, T4</td>
</tr>
<tr>
<td>broad</td>
<td>T1</td>
</tr>
<tr>
<td>orange</td>
<td>T1, T3, T4</td>
</tr>
<tr>
<td>cake</td>
<td>T3, T4</td>
</tr>
</tbody>
</table>

ECLAT eliminates infrequent itemsets 

\( \text{\textit{mins}up} = 2 \)

ECLAT outputs the frequent itemsets with 1 items

\{pasta\}, \{lemon\}, \{orange\}, \{cake\}
The ECLAT algorithm

**Step 3:** ECLAT combines itemsets of the equivalence class to generate equivalence classes of size $K+1$
The ECLAT algorithm

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

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Pasta</td>
<td>T1, T2, T3, T4</td>
</tr>
<tr>
<td>lemon</td>
<td>T1, T2, T4</td>
</tr>
<tr>
<td>Orange</td>
<td>T1, T3, T4</td>
</tr>
<tr>
<td>Cake</td>
<td>T3, T4</td>
</tr>
</tbody>
</table>

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>pasta, lemon</td>
<td>T1, T2, T4</td>
</tr>
<tr>
<td>pasta, orange</td>
<td>T1, T3, T4</td>
</tr>
<tr>
<td>pasta, cake</td>
<td>T3, T4</td>
</tr>
<tr>
<td>lemon, orange</td>
<td>T1, T4</td>
</tr>
<tr>
<td>lemon, cake</td>
<td>T3, T4</td>
</tr>
</tbody>
</table>
Step 3: ECLAT combines itemsets of the equivalence class to generate equivalence classes of size $K+1$.
The ECLAT algorithm

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

<table>
<thead>
<tr>
<th>Pasta</th>
<th>T1, T2, T3, T4</th>
</tr>
</thead>
<tbody>
<tr>
<td>lemon</td>
<td>T1, T2, T4</td>
</tr>
<tr>
<td>Orange</td>
<td>T1, T3, T4</td>
</tr>
<tr>
<td>Cake</td>
<td>T3, T4</td>
</tr>
</tbody>
</table>

- 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:

<table>
<thead>
<tr>
<th>Items</th>
<th>Transactions</th>
</tr>
</thead>
<tbody>
<tr>
<td>pasta, lemon</td>
<td>T1, T2, T4</td>
</tr>
<tr>
<td>pasta, orange</td>
<td>T1, T3, T4</td>
</tr>
<tr>
<td>pasta, cake</td>
<td>T3, T4</td>
</tr>
</tbody>
</table>
**Step 4:** ECLAT recursively process each equivalence class in the same way. Consider the first one:

<table>
<thead>
<tr>
<th>Items</th>
<th>Transaction Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>pasta, lemon</td>
<td>T1, T2, T4</td>
</tr>
<tr>
<td>pasta, orange</td>
<td>T1, T3, T4</td>
</tr>
<tr>
<td>pasta, cake</td>
<td>T3, T4</td>
</tr>
</tbody>
</table>

- 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:

<table>
<thead>
<tr>
<th>pasta, lemon</th>
<th>T1, T2, T4</th>
</tr>
</thead>
<tbody>
<tr>
<td>pasta, orange</td>
<td>T1, T3, T4</td>
</tr>
<tr>
<td>pasta, cake</td>
<td>T3, T4</td>
</tr>
</tbody>
</table>

- pasta, lemon, orange | T1, T4  
- pasta, lemon, cake   | T4      
- pasta, orange, cake  | T3, T4  

### Table

<table>
<thead>
<tr>
<th>Item</th>
<th>Transaction Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>pasta, lemon</td>
<td>T1, T2, T4</td>
</tr>
<tr>
<td>pasta, orange</td>
<td>T1, T3, T4</td>
</tr>
<tr>
<td>pasta, cake</td>
<td>T3, T4</td>
</tr>
</tbody>
</table>
**The ECLAT algorithm**

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

<table>
<thead>
<tr>
<th>Itemset</th>
<th>Transactions</th>
</tr>
</thead>
<tbody>
<tr>
<td>pasta, lemon</td>
<td>T1, T2, T4</td>
</tr>
<tr>
<td>pasta, orange</td>
<td>T1, T3, T4</td>
</tr>
<tr>
<td>pasta, cake</td>
<td>T3, T4</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th></th>
<th>T1</th>
<th>lemon, orange, cake</th>
</tr>
</thead>
<tbody>
<tr>
<td>lemon, orange</td>
<td>T1</td>
<td></td>
</tr>
<tr>
<td>lemon, cake</td>
<td>T3, T4</td>
<td></td>
</tr>
</tbody>
</table>
Step 4: ECLAT recursively process each equivalence class in the same way. Consider the next equivalence class:

<table>
<thead>
<tr>
<th>itemset</th>
<th>T1</th>
<th>T3, T4</th>
</tr>
</thead>
<tbody>
<tr>
<td>lemon, orange</td>
<td></td>
<td></td>
</tr>
<tr>
<td>lemon, cake</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 $\prec$ influence the performance?
- Yes, but which one to choose?
  - The alphabetical order?
Optimization 1: total order

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

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

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

pasta < lemon < bread < orange < cake

The visited itemsets:

14 have been explored

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

bread≺ cake≺ lemon ≺ orange ≺ pasta

The visited itemsets:

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

<table>
<thead>
<tr>
<th>Transaction</th>
<th>Items appearing in the transaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>{pasta, lemon, bread, orange}</td>
</tr>
<tr>
<td>T2</td>
<td>{pasta, lemon}</td>
</tr>
<tr>
<td>T3</td>
<td>{pasta, orange, cake}</td>
</tr>
<tr>
<td>T4</td>
<td>{pasta, lemon, orange, cake}</td>
</tr>
</tbody>
</table>

### Item transactions containing the item

<table>
<thead>
<tr>
<th>Item</th>
<th>Transactions containing the item</th>
</tr>
</thead>
<tbody>
<tr>
<td>pasta</td>
<td>1111 (representing T1, T2, T3, T4)</td>
</tr>
<tr>
<td>lemon</td>
<td>1101</td>
</tr>
<tr>
<td>bread</td>
<td>1000</td>
</tr>
<tr>
<td>orange</td>
<td>1011</td>
</tr>
<tr>
<td>cake</td>
<td>0011</td>
</tr>
<tr>
<td>item</td>
<td>transactions containing the item</td>
</tr>
<tr>
<td>----------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>pasta</td>
<td>1111</td>
</tr>
<tr>
<td>lemon</td>
<td>1101</td>
</tr>
<tr>
<td>bread</td>
<td>1000</td>
</tr>
<tr>
<td>orange</td>
<td>1011</td>
</tr>
<tr>
<td>cake</td>
<td>0011</td>
</tr>
</tbody>
</table>

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

\[
\text{transactions(\{pasta\}) } \cap \text{transactions(\{lemon\})} = 1111 \text{ LOGICAL\_AND } 1101 = 1101
\]

Thus \{pasta, lemon\} has a support of 3
Optimization 3 - memory

Consider an equivalence class:
\{\text{ABCD, ABCE, ABCF, ABCG, ABCH}\}

It can be stored more efficiently as:
P = \{\text{ABC}\}
E = \{\text{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 \( X \prec Y \)
        \( R = X \cup Y \)
        \( t(R) = t(X) \cap t(Y) \)
        IF \( \text{sup}(R) \geq \text{minsup} \)
            THEN Output R
                \( T = T \cup \{R\} \)
    END FOR
END FOR
IF  \( T \neq \emptyset \) THEN  ECLAT(T)
END FOR
Conclusion

This video has presented:

- The Eclat algorithm
- Some optimizations
References