Frequent substructure mining - an introduction

R. Kessl

SUI, 7. May 2009

Outline

1. Introduction
2. Frequent itemset mining
3. The Apriori algorithm
4. Abstract problem formulation
5. The Eclat algorithm
6. The FPGrowth algorithm

Frequent substructure mining

- We have a database D of transactions T.
- T can be an arbitrary object.
- For example: itemsets (basket market), time sequences, graphs
- Mining of frequent substructures has exponential complexity (in the worst case)

Frequent itemset mining

Database D:

TID	Transaction
1	$\{1, 2, 3, 4\}$
2	$\{3, 5\}$
3	$\{1, 3, 4\}$
4	$\{1, 2\}$
5	$\{1, 3, 4, 5\}$
6	$\{1, 2, 3, 4, 5\}$

- The problem is to find a set of items (*itemsets*) that occurs in at least p% of *transactions*,
- exponential complexity with respect to the # of items (in the worst case)

Example:

Database D:

TID	Transaction
1	$\{1,2,3,4\}$
2	$\{3,5\}$
3	$\{1,3,4\}$
4	$\{1,2\}$
5	$\{1,3,4,5\}$
6	$\{1,2,3,4,5\}$

- are there any correlations among items in D ?

Example:

Database D:

TID	Transaction
1	$\{1, 2, 3, 4\}$
2	$\{3, 5\}$
3	$\{1, 3, 4\}$
4	$\{1, 2\}$
5	$\{1, 3, 4, 5\}$
6	$\{1, 2, 3, 4, 5\}$

- are there any correlations among items in D ?
- Itemset $\{3, 5\}$ occurs in 3 transactions $\Rightarrow$ $Supp(\{3, 5\}, D) = 3$

Example:

Database D:

TID	Transaction
1	$\{1,2,3,4\}$
2	$\{3,5\}$
3	$\{1,3,4\}$
4	$\{1,2\}$
5	$\{1,3,4,5\}$
6	$\{1,2,3,4,5\}$

- are there any correlations among items in D ?
- Itemset $\{3,5\}$ occurs in 3 transactions $\Rightarrow$ $Supp(\{3,5\},D) = 3$
- *Association rule* $\{3,5\} \rightarrow \{2\}$ in $33.3\overline{3}\%$ of cases $\Rightarrow$ *confidence* $Conf(\{3,5\},\{2\}) = 0.3\overline{3}$

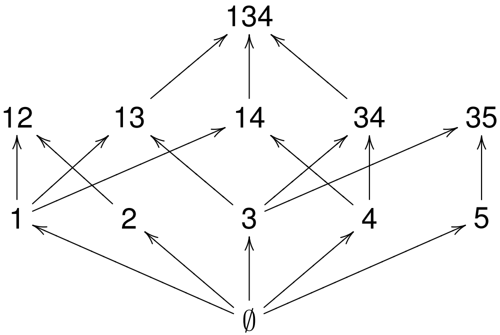
Example:

Database D:

TID	Transaction
1	$\{1, 2, 3, 4\}$
2	$\{3, 5\}$
3	$\{1, 3, 4\}$
4	$\{1, 2\}$
5	$\{1, 3, 4, 5\}$
6	$\{1, 2, 3, 4, 5\}$

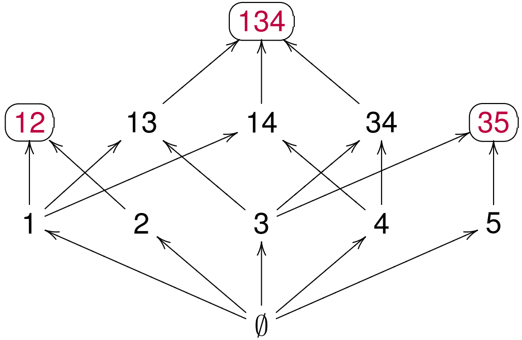
- are there any correlations among items in D ?
- Itemset $\{3, 5\}$ occurs in 3 transactions $\Rightarrow$ $Supp(\{3, 5\}, D) = 3$
- *Association rule* $\{3, 5\} \rightarrow \{2\}$ in $33.3\overline{3}\%$ of cases $\Rightarrow$ *confidence* $Conf(\{3, 5\}, \{2\}) = 0.3\overline{3}$
- $\Rightarrow$ associations(correlations) among items

Lattice of frequent itemsets for *min_support* $= 3$



$\Rightarrow$ maximal frequent itemsets (MFIs)
$\Rightarrow$ base set of items $B = \{1, 2, 3, 4, 5\}$

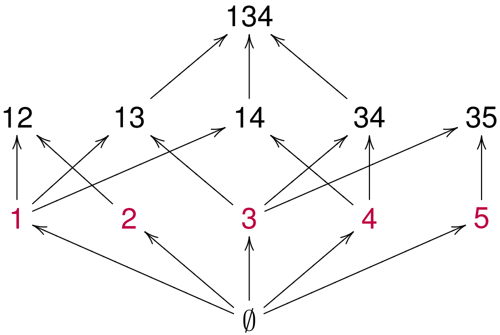
Lattice of frequent itemsets for *min_support* = 3



⇒ maximal frequent itemsets (MFIs)

⇒ base set of items $B = \{1, 2, 3, 4, 5\}$

Lattice of frequent itemsets for *min_support* $= 3$



$\Rightarrow$ maximal frequent itemsets (MFIs)

$\Rightarrow$ *base set of items* $B = \{1, 2, 3, 4, 5\}$

General schema of mining association rules

Input: database D, *min_support* and *min_conf*

1. generate frequent itemsets X with $Supp(X) \geq min\_support$ `/* most time consuming part */`
2. generate association rules with confidence $Conf(X) \geq min\_conf$

Algorithms

- The Apriori algorithm: the first and slowest algorithm for generation of frequent itemsets
- The FPGrowth algorithm: based on prefix tree (trie)
- The Eclat algorithm: uses vertical representation of the database

The Apriori algorithm

Lemma (Monotonicity of support)

Let $U \subseteq B$ be an itemset with support $Supp(U)$ in database D. For every superset V of U holds: $Supp(U) \geq Supp(V)$.

Proof.

(By induction)

(i) It is obvious that for an item b_i, $|D| = Supp(\emptyset) \geq Supp(\{b_i\})$.

(ii) If U has support $Supp(U)$, then for arbitrary $b_i \in B - U$ holds: $Supp(U) \geq Supp(U \cup \{b_i\})$, since not all transactions containing U must contain $U \cup \{b_i\}$.

$\square$

The Apriori algorithm contd.

- Based on the monotonicity property of support
- *Generate&test* algorithm
- **The Apriori algorithm can make $k = |B|$ scans of D**

A *candidate* itemset $U, |U| = k$:

- Support of U is unknown
- all $W \subset U, |W| = k - 1$ are frequent, i.e. $Supp(W) \geq min\_support$

From frequent itemsets F_k generate candidates C_{k+1}:

- $f_1, f_2 \in F_k$ such that f_1, f_2 identical in $k - 1$ items;
- $c = \{f_1[j] : j < k\} \cup \{f_1[k], f_2[k]\}$.

The Apriori algorithm contd.

APRIORI(**In:** Database D, **In:** Set B, **In:** Integer *min_support*, **In/Out:** Set F)

$k \leftarrow 1$
$C_k \leftarrow \{\{b_i\} : b_i \in B\}$
while C_k not empty **do**
    COMPUTE-SUPPORT(D, C_k)
    for all $c \in C_k$ **do**
        if $Supp(c) < min\_support$ **then**
            delete c from C_k
        end if
    end for
    $F_k \leftarrow C_k$
    $C_{k+1} \leftarrow$ GENERATE-CANDIDATES(F_k)
    $k \leftarrow k + 1$
end while
return $F_1 \cup F_2 \cup \ldots \cup F_{k-1}$

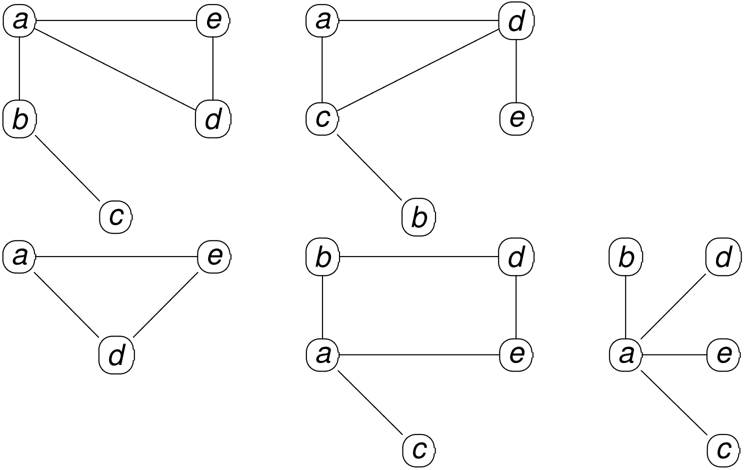
Abstract substructure mining

- A database D, a language $\mathcal{L}$;
- sentences $\varphi, \Phi \in \mathcal{L}$;
- a frequency criterion $q(\varphi) \in \{\text{true}, \text{false}\}$;
- a monotone specialization/generalization relation: $\varphi \preceq \Phi$
- $q(\Phi) = \text{true} \Rightarrow q(\varphi) = \text{true}$

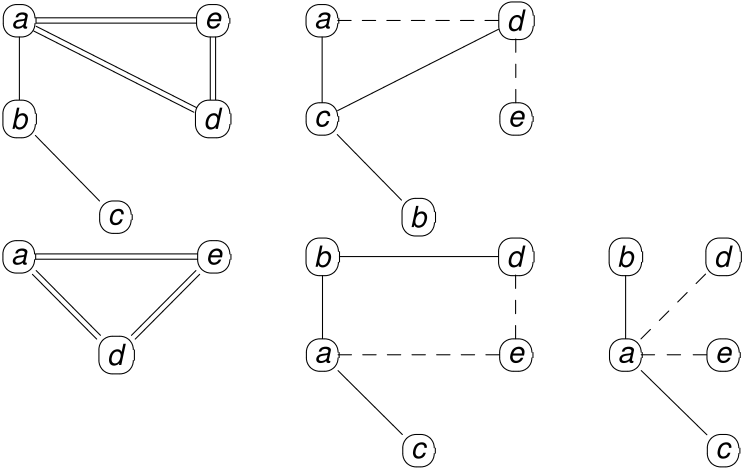
Generalization of the Apriori algorithm

$C_1 \leftarrow \{\varphi \in \mathcal{L} | \text{there is no } \varphi' \text{ such that } \varphi' \prec \varphi\}$
$i \leftarrow 1$
while C_i not empty **do**
    $F_i \leftarrow \{\varphi \in C_i | q(\varphi) = \mathit{true}\}$
    $C_{i+1} \leftarrow \{\varphi \in \mathcal{L} | \forall \varphi' \prec \varphi \text{ we have } \varphi' \in \cup_{j \leq i} F_j\} \setminus \cup_{j \leq i} C_j$
    $i \leftarrow i + 1$
end while
return $F_1 \cup F_2 \cup \ldots \cup F_{k-1}$

Subgraph mining



Subgraph mining



The basic characteristics of the Eclat alg.

- The Eclat algorithm is much quicker then the Apriori algorithm.
- It is a depth-first algorithm $\Rightarrow$ lower memory consumption then the Apriori algorithm.
- Based on the fact that the powerset $\mathcal{P}(B)$ forms a lattice.
- Uses so called prefix-based equivalence classes.
- Uses the vertical representation of the database.

The prefix-based equivalence classes

Definition (equivalence relation)

Let P be a set and $X, Y, Z \in P$ a relation $\equiv$ is called an equivalence relation if the relation is:

1. Reflexive: $X \equiv Y$.
2. Symmetric: if $X \equiv Y$ then $Y \equiv X$.
3. Transitive: if $X \equiv Y$ and $Y \equiv Z$ then $X \equiv Z$.

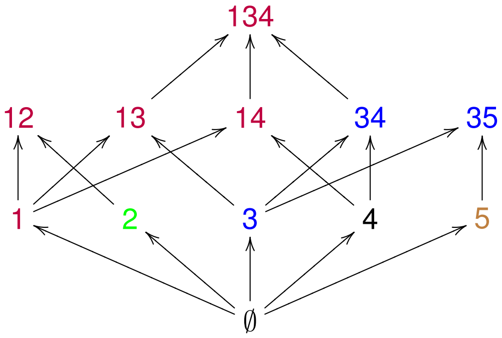
Definition (prefix-based equivalence class)

Let π be a function $\pi : \mathcal{P}(B) \times N \to \mathcal{P}(B)$ such that $\pi(X, l) = X[1 : l]$.
The *prefix-based equivalence* π_l on the lattice $\mathcal{L}$ is defined as follows:
$\forall U, V \subseteq B, U \equiv_{\pi_l} V \Leftrightarrow \pi(U, l) = \pi(V, l)$.

The prefix-based equivalence classes contd.

Lemma

Each equivalence class $[W]_\pi$ induced by the equivalence relation π is a sub-lattice of powerset $\mathcal{P}(B)$.



The vertical representation of a database

Database D:

TID	Transaction
1	$\{1, 2, 3, 4\}$
2	$\{3, 5\}$
3	$\{1, 3, 4\}$
4	$\{1, 2\}$
5	$\{1, 3, 4, 5\}$
6	$\{1, 2, 3, 4, 5\}$

Vertical representation of the database is a set of pairs $\{(b_i, \mathcal{T}(b_i))\}$, where $b_i \in B$ and $\mathcal{T}(b_i)$ is a *transaction list* of item b_i (tidlist in short)

Example:
$(1, \{1, 3, 4, 5, 6\})$
$(2, \{1, 4, 6\})$
$(3, \{1, 2, 3, 5, 6\})$
$(4, \{1, 3, 5, 6\})$
$(5, \{5, 6\})$

Support counting

Support of a set $U \subset B$ can be computed using the tidlists:
$Supp(U) = |\bigcap_{b_i \in U} \mathcal{T}(b_i)|$

- **The Eclat algorithm makes a single scan of D.**
- **The datastructures are processor-cache friendly** (it is just an array of numbers).
- Can be implemented using bitmaps.

itemset	$\{1\}$	$\{2\}$	$\{3\}$	$\{4\}$	$\{5\}$
TID	1	1	1	1	2
	3	4	2	3	5
	4	6	3	5	6
	5		5	6	
	6		6		

Frequent	$\times$	$\times$	$\times$	$\times$	$\times$	$\times$
itemset	$\{1,2\}$	$\{1,3\}$	$\{1,4\}$	$\{1,5\}$	$\{2,3\}$	$\{2,4\}$
TID	1	1	1	5	1	1
	4	3	3	6	6	6
	6	5	5			
		6	6			

		$\times$	$\times$	$\times$
itemset	$\{2,5\}$	$\{3,4\}$	$\{3,5\}$	$\{4,5\}$
TID	6	1	2	5
		3	5	6
		5	6	
		6		

The Eclat algorithm

ECLAT-BOTTOM-UP(**In:** Atoms $\mathcal{A}$, **In:** Itemset P, **Out:** Set F)

```
for all atom a_i ∈ A do
    A_i ← ∅
    for all atom a_j ∈ A, a_i < a_j do
        if |T(P ∪ {a_j})| ≥ min_support then
            A_i ← A_i ∪ {a_j}
            f ← P ∪ {a_j}
            F ← F ∪ f
        end if
    end for
    ECLAT-BOTTOM-UP(A_i, P ∪ {a_i}, F)
end for
```

Optimalizations to the Eclat algorithm

1. The “closed itemset” optimalization
2. Dynamic items ordering
3. The diffsets

Optimalizations – The “closed itemset” optimalization

TID	Transaction
1	{...}
2	{...}
3	{ 1,2 }
4	{ 1,2 }
5	{ 1,2,3,4,5}
6	{ 1,2,3,4,5}

Optimalizations – Dynamic items ordering

Let have a prefix $\Pi = \{\pi_1, \ldots, \pi_k\}, \pi_i \in B$, and extensions $\Sigma = \{\sigma_1, \ldots, \sigma_l\}, \sigma_i \in B$, and $Supp^*(\Pi \cup \{\sigma_1\}) \leq Supp^*(\Pi \cup \{\sigma_2\}) \leq \ldots \leq Supp^*(\Pi \cup \{\sigma_l\})$.
For the prefix Π:

Using the order it follows:

Optimalizations – Dynamic items ordering

Let have a prefix $\Pi = \{\pi_1, \ldots, \pi_k\}, \pi_i \in B$, and extensions $\Sigma = \{\sigma_1, \ldots, \sigma_l\}, \sigma_i \in B$, and $\mathit{Supp}^*(\Pi \cup \{\sigma_1\}) \leq \mathit{Supp}^*(\Pi \cup \{\sigma_2\}) \leq \ldots \leq \mathit{Supp}^*(\Pi \cup \{\sigma_l\})$.
For the prefix Π:

1. consider $\sigma_1, \sigma_2, \ldots, \sigma_l$ as extensions (in that order)

Using the order it follows:

1. the smallest portion of the database gets the largest portion of the search space.

Optimalizations – Dynamic items ordering

Let have a prefix $\Pi = \{\pi_1, \ldots, \pi_k\}, \pi_i \in B$, and extensions $\Sigma = \{\sigma_1, \ldots, \sigma_l\}, \sigma_i \in B$, and $Supp^*(\Pi \cup \{\sigma_1\}) \leq Supp^*(\Pi \cup \{\sigma_2\}) \leq \ldots \leq Supp^*(\Pi \cup \{\sigma_l\})$.
For the prefix Π:

1. consider $\sigma_1, \sigma_2, \ldots, \sigma_l$ as extensions (in that order)
2. consider $\sigma_l, \sigma_{l-1}, \ldots, \sigma_1$ as extensions (in that order)

Using the order it follows:

1. the smallest portion of the database gets the largest portion of the search space.
2. the largest portion of the database gets the largest portion of the search space.

Optimalizations – The diffsets

Definition

Let $U \subset B$ be an itemset and $i \in B - U$ an item. $\mathcal{T}(U)$ denotes a set of transaction id's. *Difference set* (or diffset in short) is $\mathcal{D}(U \cup \{i\}) = \mathcal{T}(U) - \mathcal{T}(\{i\})$

The support can be computed as:

$$Supp(U \cup \{i\}) = Supp(U) - |\mathcal{D}(U \cup \{i\})|$$

Computation of diffsets:

$$\begin{aligned}
\mathcal{D}(U \cup \{i\} \cup \{j\}) &= \mathcal{T}(U \cup \{i\}) - \mathcal{T}(U \cup \{j\}) \\
&= \mathcal{T}(U \cup \{i\}) - \mathcal{T}(U \cup \{j\}) + \mathcal{T}(U) - \mathcal{T}(U) \\
&= (\mathcal{T}(U) - \mathcal{T}(U \cup \{j\})) - (\mathcal{T}(U) - \mathcal{T}(U \cup \{i\})) \\
&= \mathcal{D}(U \cup \{j\}) - \mathcal{D}(U \cup \{i\})
\end{aligned}$$

The FPGrowth algorithm

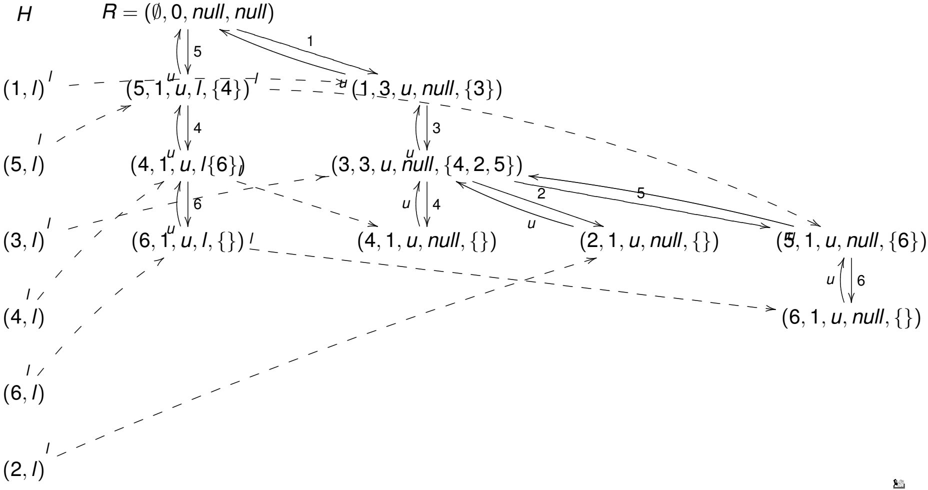
- Very similar to the Eclat algorithm
- the only difference is the used datastructure
- The datastructure (called FPTrie) is a prefix tree with linked nodes

The FPGrowth algorithm - the FPTrie

TID	Transaction
1	$\{1, 3, 4\}$
2	$\{5, 4, 6\}$
3	$\{1, 3, 5, 6\}$
4	$\{1, 3, 2\}$

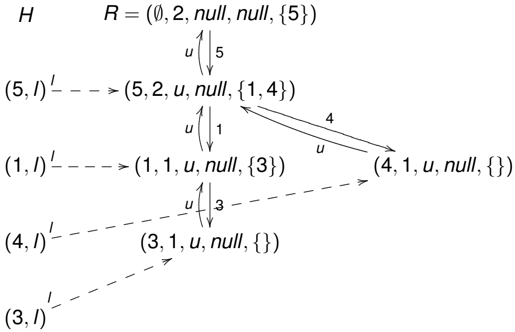
Item	Support
1	3
2	3
3	2
4	2
5	1
6	1

Each node is a tuple (`item, support, upper-link, header-link, children`).



6's conditional tree

Transaction	Support
$\{5, 4\}$	1
$\{5, 3, 1\}$	1



The FPGrowth algorithm

FPGROWTH(**In:** FP-Tree *tree*,**In:** Itemset I)

- **if** *tree* contains only single path P **then**
  - **for all** combination c in path P **do**
    - $s \leftarrow min\{s | i \in c, s = Supp(i)\}$
    - **if** $s \geq min\_support$ **then**
      - generate pattern $c \cup I$ with support s
    - **end if**
  - **end for**
- **else**
  - **for all** a_i in the header of *tree* **do**
    - **if** $Supp(a_i) \geq min\_support$ **then**
      - generate pattern $c = \{a_i\} \cup I$ with support $Supp(a_i)$
    - **end if**
    - construct c's conditional pattern base and
    - then c's conditional **FP-Tree** $tree_c$
    - **if** $Size(tree_c) \neq 0$ **then**
      - FPGROWTH($tree_c, c$)
    - **end if**
  - **end for**
- **end if**