Data Mining 2013
Frequent Pattern Mining (2)

Ad Feelders

Universiteit Utrecht

October 11, 2013

Frequent Pattern Mining

1. Frequent Item Set Mining
2. Sequence Mining
3. Tree Mining
4. Graph Mining

Node Labeled Graph

Definition (Node Labeled Graph)

A node labeled graph is a quadruple $G = (V, E, \Sigma, L)$ where:

1. V is the set of nodes,
2. E is the set of edges,
3. Σ is a set of labels, and
4. $L : V \rightarrow \Sigma$ is a labeling function that assigns labels from Σ to nodes in V.

Labeled Rooted Unordered Tree

Definition (Labeled Rooted Unordered Tree)

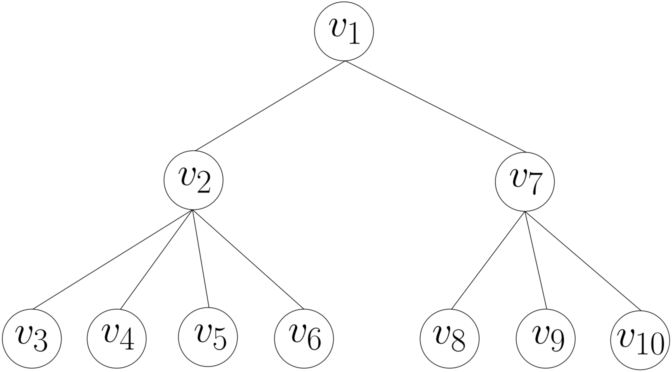
A labeled rooted unordered tree $U = (V, E, \Sigma, L, v^r)$ is an acyclic undirected connected graph $G = (V, E, \Sigma, L)$ with a special node v^r called the root of the tree such that there exists exactly one path between the root node and any other node in V.

Labeled Rooted Ordered Tree

Definition (Labeled Rooted Ordered Tree)

A labeled rooted ordered tree $T = (V, E, \Sigma, L, v^r, \leq)$ is an unordered tree $U = (V, E, \Sigma, L, v^r)$ where between all the siblings an order $\leq$ is defined. To every node in an ordered tree a preorder (pre(v)) number is assigned according to the depth-first (or preorder) traversal of the tree.

Node Numbering according to Preorder Traversal



Tree Inclusion Relations

1. Bottom-up subtree.
2. Induced subtree.
3. Embedded subtree.

Induced Subtree

Let $\pi(v)$ denote the parent of node v.

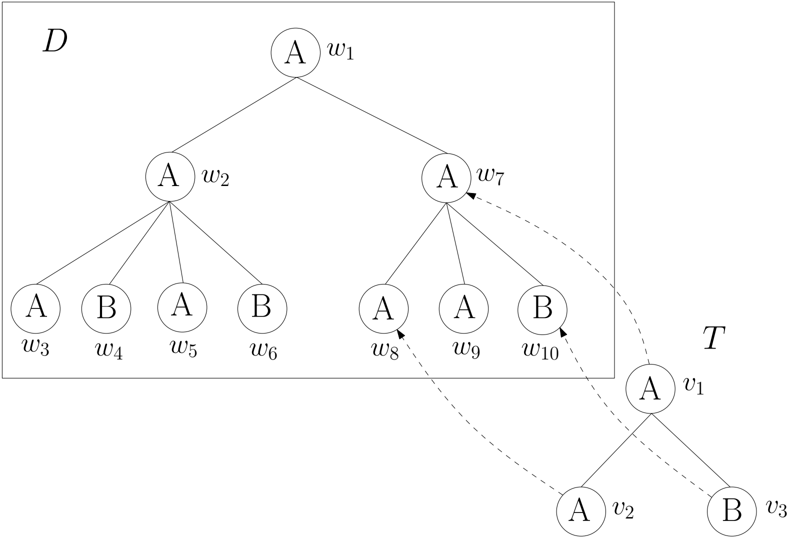
Definition (Induced Subtree)

Given two ordered trees D and T, we call T an induced subtree of D if there exists an injective matching function ϕ of V_T into V_D satisfying the following conditions:

1. ϕ preserves the labels: $L_T(v) = L_D(\phi(v))$.
2. ϕ preserves the parent-child relation: $v_i = \pi_T(v_j) \Leftrightarrow \phi(v_i) = \pi_D(\phi(v_j))$.
3. ϕ preserves the left to right order between the nodes: $\text{pre}(v_i) < \text{pre}(v_j) \Leftrightarrow \text{pre}(\phi(v_i))) < \text{pre}(\phi(v_j))$.

An induced subtree T can be obtained from a tree D by repeatedly removing leaf nodes, or possibly the root node if it has only one child.

Induced Subtree



Induced Subtree

Matching function

1. $\phi(v_1) = w_7$
2. $\phi(v_2) = w_8$
3. $\phi(v_3) = w_{10}$

Verify that

1. $L_T(v_1) = L_D(w_7) = A$
2. $L_T(v_2) = L_D(w_8) = A$
3. $L_T(v_3) = L_D(w_{10}) = B$

Likewise, we can verify that the other conditions are met, so T is an induced subtree of D.

Embedded Subtree

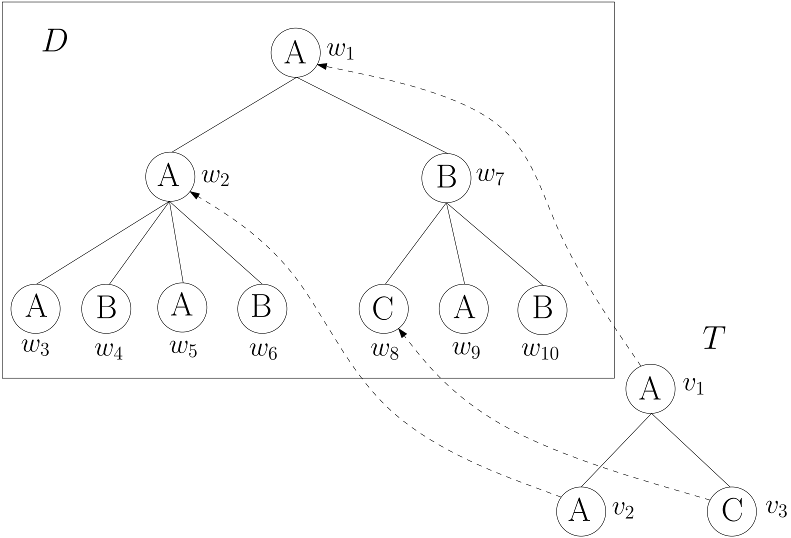
Let $\pi^*(v)$ denote the set of ancestors of v.

Definition (Embedded Subtree)

Given two ordered trees D and T, we call T an embedded subtree of D if there exists an injective matching function ϕ of V_T into V_D satisfying the following conditions:

1. ϕ preserves the labels: $L_T(v) = L_D(\phi(v))$.
2. ϕ preserves the ancestor-descendant relation: $v_i \in \{\pi^*_T(v_j)\} \Leftrightarrow \phi(v_i) \in \{\pi^*_D(\phi(v_j))\}$.
3. ϕ preserves the left to right order between the nodes: $\text{pre}(v_i) < \text{pre}(v_j) \Leftrightarrow \text{pre}(\phi(v_i))) < \text{pre}(\phi(v_j))$.

Embedded Subtree



Frequent Tree Mining

Given a database of trees $D = \{d_1, d_2, \ldots, d_n\}$ and a tree inclusion relation $\preceq$, we define the support of a tree T as

$$\mathrm{supp}(T, D) = \frac{|\{d \in D \mid T \preceq d\}|}{|D|}$$

Given a minimum support threshold σ, compute

$$\mathcal{F}(\sigma, D, \preceq) = \{T \mid \mathrm{supp}(T, D) \geq \sigma\}$$

Anti-Monotonicity Property

Given a database of trees D, and two trees T_1 and T_2, then

$$T_1 \preceq T_2 \Rightarrow \text{supp}(T_1, D) \geq \text{supp}(T_2, D),$$

because $\forall d \in D : T_2 \preceq d \Rightarrow T_1 \preceq d$.

Hence, in a level-wise search for frequent trees, there is no point in expanding infrequent trees.

Mining Frequent Induced Trees with FREQT

We must address two basic issues:

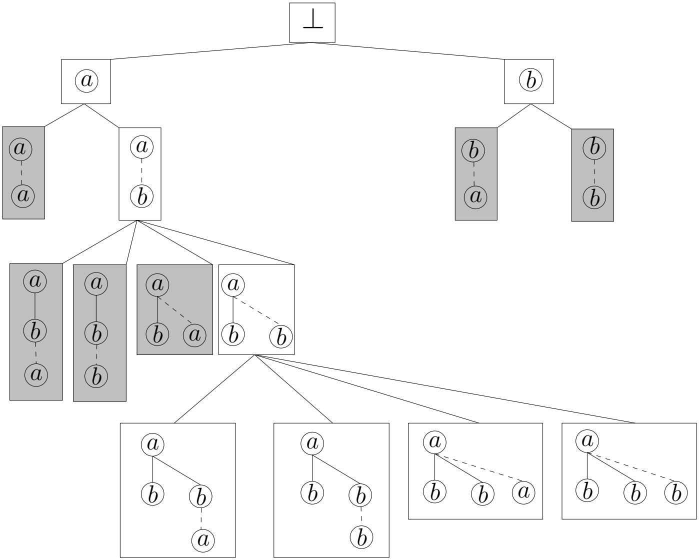
1. Generate candidate frequent trees: add a single node with a frequent label to a frequent tree. This is done by so-called *right-most extension*.
2. Record the occurrences of the candidate trees in the data trees, and determine whether they are frequent.

Right-most Extension

Let T_k denote a tree of size k (a tree with k nodes).

- Consider the node numbering of T_k according to pre-order (depth-first) traversal of the tree.
- The right-most branch of the tree is the path from the root node to the right-most leaf (i.e. the node with number k).
- To expand the tree T_k, it is only allowed to add a node as the right-most child of a node on the right-most branch of T_k. This node gets number $k + 1$, as it is the last node in the pre-order traversal of T_{k+1}.

Right-most Extension with label set $\Sigma = \{a, b\}$



Right-most Extension

The right-most extension technique generates each tree at most once.

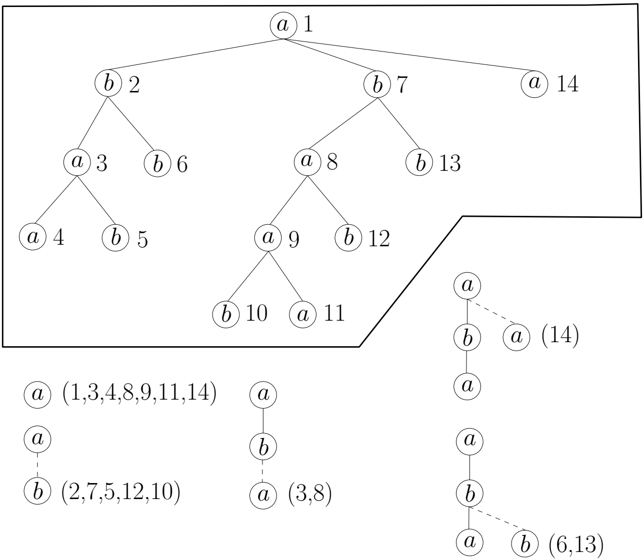
Consider any tree T_{k+1}. This tree only has one predecessor (in the generation sequence), namely the tree T_k that is obtained by removing the right-most leaf of T_{k+1} (i.e. the node with number $k+1$ in the pre-order traversal).

Also, the right-most expansion technique generates every possible tree, so each tree is generated exactly once.

Occurrence List

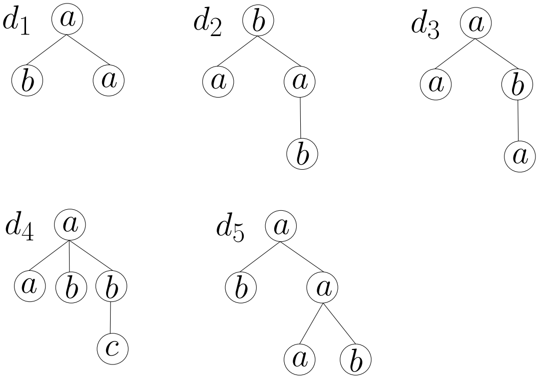
- For counting the frequency of a pattern tree an occurrence list is maintained that contains the list of nodes in the data tree to which the nodes in the pattern tree can be mapped.
- FREQT only stores the nodes of the data tree to which the right-most node in the pattern tree can be mapped.
- This is sufficient since only the nodes on the right-most branch are needed for future extension.

Right-most Occurrence List



Example

Consider the following database of labeled ordered trees:



Find all frequent induced subtrees with support at least 3.

Example: Level 1

At level 1 we have the following three candidates:

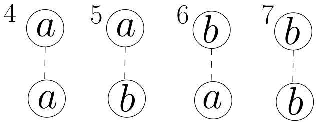
$$^{1}\textcircled{a} \quad ^{2}\textcircled{b} \quad ^{3}\textcircled{c}$$

The right-most occurrence lists are:

	(1)	(2)	(3)
d_1	(1,3)	(2)	–
d_2	(2,3)	(1,4)	–
d_3	(1,2,4)	(3)	–
d_4	(1,2)	(3,4)	(5)
d_5	(1,3,4)	(2,5)	–
Support	5	5	1
Frequent?	Y	Y	N

Example: Level 2

At level 2 we have the following candidates:

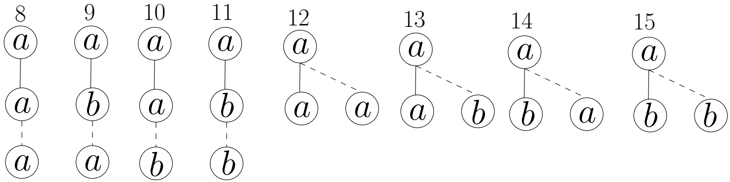


The RMO-lists are:

	(4)	(5)	(6)	(7)
d_1	(3)	(2)	–	–
d_2	–	(4)	(2,3)	–
d_3	(2)	(3)	(4)	–
d_4	(2)	(3,4)	–	–
d_5	(3,4)	(2,5)	–	–
Support	4	5	2	0
Frequent?	Y	Y	N	N

Example: Level 3

The level 3 candidates are:

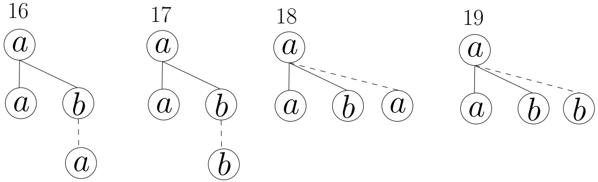


The RMO-lists are:

	(8)	(9)	(10)	(11)	(12)	(13)	(14)	(15)
d_1	–	–	–	–	–	–	(3)	–
d_2	–	–	–	–	–	–	–	–
d_3	–	(4)	–	–	–	(3)	–	(4)
d_4	–	–	–	–	–	(3,4)	–	–
d_5	(4)	–	(5)	–	–	(5)	(3)	–
Support	1	1	1	0	0	3	2	1
Frequent?	N	N	N	N	N	Y	N	N

Example: Level 4

The level 4 candidates are:



The RMO-lists are:

	(16)	(17)	(18)	(19)
d_1	–	–	–	–
d_2	–	–	–	–
d_3	(4)	–	–	–
d_4	–	–	–	(4)
d_5	–	–	–	–
Support	1	0	0	1
Frequent?	N	N	N	N

Applications of frequent tree mining

- Mining usage patterns in Web logs.
- Mining frequent query patterns from XML queries.
- Classification of XML documents according to subtree structures.
- ...

Mining usage patterns in web logs

Mining data from web server log files to:

- Study customer behavior.
- Better organize web pages.

LOGML

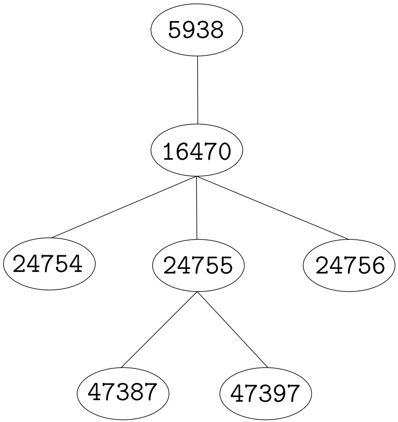
- LOGML is a publicly available XML application to describe log reports of web servers.
- LOGML provides an XML vocabulary to structurally express the contents of the log file in a compact manner.
- LOGML documents have three parts
  1. A web graph induced by the source-target pairs in the raw logs.
  2. A summary of statistics.
  3. A list of user sessions (subgraphs of the web graph) extracted from the logs.

Example user session

Each user session has a session id (IP or host name), a list of edges (`uedges`) giving source and target node pairs, and the time (`utime`) when a link is traversed. Example user session:

```
<userSession name="ppp0-69.ank2.isbank.net.tr" ...>
<uedge source="5938" target="16470" utime="7:53:46"/>
<uedge source="16470" target="24754" utime="7:56:13"/>
<uedge source="16470" target="24755" utime="7:56:36"/>
<uedge source="24755" target="47387" utime="7:57:14"/>
<uedge source="24755" target="47397" utime="7:57:28"/>
<uedge source="16470" target="24756" utime="7:58:30"/>
```

Tree representation of example user session



Example of frequent subtree found

One day's logs from CS web site (Rensselaer Polytechnic Institute). The pattern refers to a popular Turkish poetry site maintained by one of the department members.

```
Let Path=http://www.cs.rpi.edu/~name/poetry
Let Akova = Path/poems/akgun_akova
FREQUENCY=59, NODES = 5938 16395 38699 -1 38698 -1 38700
                   Path/sair_listesi.html
                           |
           Path/poems/akgun_akova/index.html
          /                |                 \
Akova/picture.html  Akova/contents.html  Akova/biyografi.html
```

Mining frequent query patterns

- As XML prevails over the internet, the efficient retrieval of XML data becomes more important.
- Research to improve query response times has largely concentrated on indexing XML documents and processing regular path expressions.
- Another approach is to discover frequent query patterns since the answers to those queries can be stored and indexed.

Mining frequent query patterns

Given an XML data source and the history of XML queries $\{q_1, \dots, q_N\}$ issued against it, transform them into a corresponding history of query pattern trees $D = \{QPT_1, \dots, QPT_N\}$.

Mining frequent query patterns is equivalent to finding the rooted subtrees that occur frequently over the set of pattern trees D.

Document Type Definition

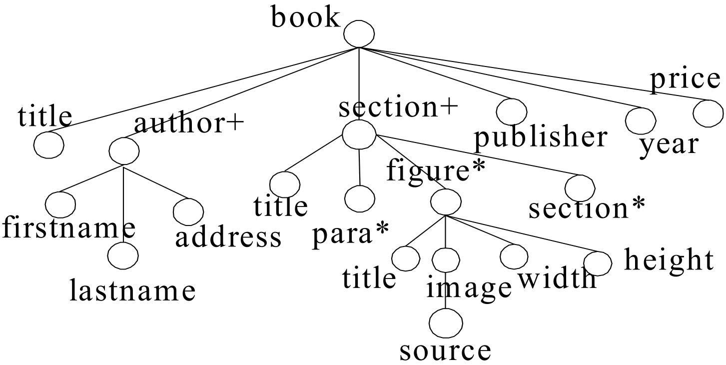


Figure 1. Book DTD Tree.

The purpose of a DTD (Document Type Definition) is to define the legal building blocks of an XML document.

A query and its corresponding query pattern tree

```
Q1: for $b in document(book.xml) /book
     where some $a in $b/author
         satisfies $a/lastname/data()="Buneman"
    return <result>
                  <book>{$b/title, $b/author, $b/price}<book>
           </result>
```

We extract the following information from Q_1:

- resultpattern={/book/author, /book/title, book/price}
- predicates={/book/author/lastname/data()="Buneman"}
- documents={book.html}

A query and its corresponding query pattern tree

To construct a query pattern tree we:

- Extract the paths from the set *predicates* by ignoring the selection conditions.
- Combine these extracted path expressions with the paths in the set *resultpattern* to generate the query pattern tree.
- Exactly *how* they are combined is unfortunately not clear from the source article!

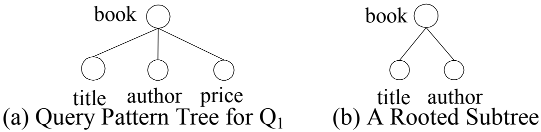


Figure 2. Query Pattern Tree for Q_1.

Example frequent rooted subtree

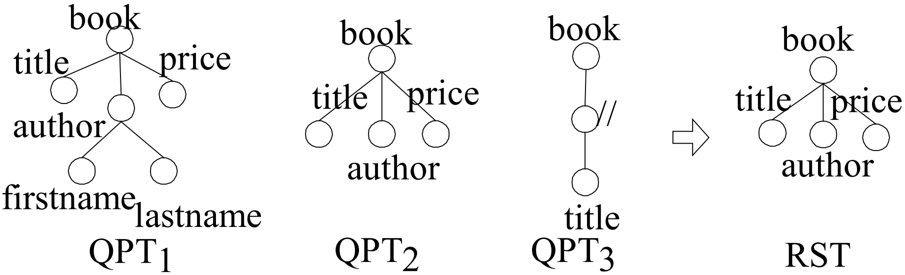


Figure 3. Database of Query Pattern Trees and a Frequent Rooted Subtree.

Special Labels

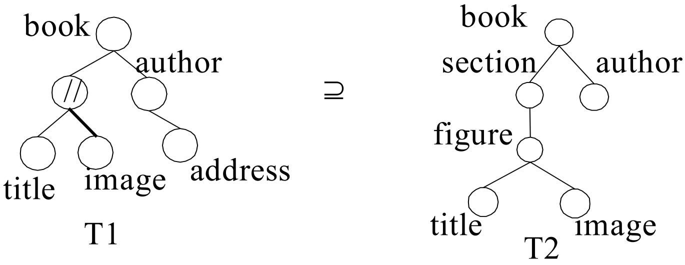


Figure 4. Example of Pattern Tree Containment.

Frequent Pattern Mining and Classification

Frequent pattern mining can also be used to extract features for classification tasks:

1. Find frequent patterns per class.
2. Define *discriminating* patterns, for example, as patterns that are frequent in one class but not in the other.
3. Use the presence/absence of such a discriminating pattern as a (binary) feature for constructing a classifier (e.g. classification tree!).

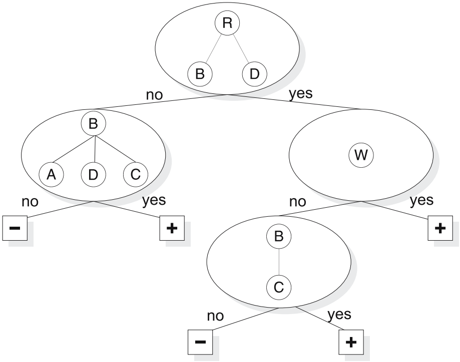


Fig. 4. A decision tree as produced by the TREE^2 algorithm