

# Index Structure

for XML and Text

Debora Donato

Yahoo! Research – Barcelona, Spain

Seminars of Computer Networks

- 1 Introduction
- 2 Inverted Lists
- 3 Structure Indexes
- 4 Vector-Space Model

- 1 Introduction
- 2 Inverted Lists
- 3 Structure Indexes
- 4 Vector-Space Model

## In a nutshell

The extensible markup language (XML) is essentially a textual representation of the hierarchical (tree-like) data where a meaningful piece of data is bounded by matching starting and ending tags such as

```
<text>This is an XML document</text>
```

The nested structure of tags capture additional semantics of the information

## Advantages of the XML tagged structure

- XML structure exploited for better searching (compared to text or html search)
- Context information used for semantic search.

# Semantic Search over XML documents

## Semantic search

User can specify the context or tag in which a keyword occurs to get more accurate results

# Semantic Search over XML documents

## Semantic search

User can specify the context or tag in which a keyword occurs to get more accurate results

## Example

- “Verdicchio” Italian surname or a wine.

# Semantic Search over XML documents

## Semantic search

User can specify the context or tag in which a keyword occurs to get more accurate results

## Example

- “Verdicchio” Italian surname or a wine.
- Classical query: wine results

# Semantic Search over XML documents

## Semantic search

User can specify the context or tag in which a keyword occurs to get more accurate results

## Example

- “Verdicchio” Italian surname or a wine.
- Classical query: wine results
- If am I searching for a person?

# Semantic Search over XML documents

## Semantic search

User can specify the context or tag in which a keyword occurs to get more accurate results

## Example

- “Verdicchio” Italian surname or a wine.
- Classical query: wine results
- If am I searching for a person?
  - annotating text with the tag `<person>`,

# Semantic Search over XML documents

## Semantic search

User can specify the context or tag in which a keyword occurs to get more accurate results

## Example

- “Verdicchio” Italian surname or a wine.
- Classical query: wine results
- If am I searching for a person?
  - annotating text with the tag `<person>`,
  - query in a similar form: `<person>Verdicchio < \person>`.

- 1 Introduction
- 2 Inverted Lists**
- 3 Structure Indexes
- 4 Vector-Space Model

# Index Structures [Madhavrao, ]

- built and stored by search engines to navigate through data.
- goal: improving query processing speed and efficiency
- factors to be considered:
  - required processing time
  - index structure size
  - expected query complexity
  - amount of preprocessing done

# Inverted Lists

- widely used for text as well as XML.
- one entry for each term occurring in the data.
- additional information to locate the entry
- database/ html
  - for each occurrence of a lexicon
- XML
  - a entry also for each node structure.

# XML Tree Representation

XML document modeled as a tree

$$G = (V_E, V_T, E, root, oid, lab, ord)$$

- $V_E$ : set of element nodes;
- $V_T$ : set of term nodes (leaf nodes);
- $E$ : set of edges (spanning tree over  $V_E \cup V_T$ );
- $root \in V_E$ ;
- $lab$  is function that assigns a label to each node:
  - tag name for  $node \in V_E$ ;
  - keyword for  $node \in V_T$
- $oid$  is a function that assigns unique id to each node
- $ord$  is a function that assigns a unique ordinal number (sibling position)

## XML Database representation

A XML database is a collection of XML documents.

- modeled as a graph;
- *ROOT* with document *roots* as children;
- *oid* unique across the whole DB;
- the id of document root nodes is the document id.

## XML Inverted list representation

- For each structure node:  $\langle docid, start, end, level \rangle$
- For each text node:  $\langle docid, start, end \rangle$
- $level$  is a the depth in the document tree.
- Properties of  $start$  and  $end$ :
  - $n.start < n.end$  for each  $n$ ;
  - if  $n_1$  is an ancestor of an element node  $n_2$ :  
 $n_1.start < n_2.start < n_2.end < n_1.end$
  - if  $n_1$  is an ancestor of a text node  $n_2$ :  
 $n_1.start < n_2.start < n_1.end$
  - for sibling nodes  $n_1$  and  $n_2$  with  $ord(n_1) < ord(n_2)$ :  
 $n_1.end < n_2.end$

## Inverted List limits

- Conventional Inverted list can not be used directly in IR.
  - In XML search, node lists as query results (not document list)
  - Node that contains the keyword + all its ancestors
  - Redundant information, space overhead.

## Dewey Inverted Lists

- Need to capture the ancestor-descendant information: Dewey inverted lists
  - Tree representation: the path vector of sibling ordinal numbers from root to an element is unique and can be used as ID
  - Dewey ID of an ancestor is a prefix of the ID of a descendant

# Dewey Inverted Lists

- Need to capture the ancestor-descendant information: Dewey inverted lists
  - Tree representation: the path vector of sibling ordinal numbers from root to an element is unique and can be used as ID
  - Dewey ID of an ancestor is a prefix of the ID of a descendant
- Example: Dewey ID 5.0.1.3.4
  - the node is the 4th among its siblings;
  - its parents is the 3rd among its siblings;
  - ....

# Dewey Inverted Lists

Table: Dewey inverted List sorted by Dewey ID

keyword	Dewey ID	Rank	Position List
K1	5.0.3.0.0	88	32
	6.0.3.4.0	38	89
...	...	...	...
K2	5.0.3.0.1	84	38
	8.2.4.1.2	99	32

- 1 Introduction
- 2 Inverted Lists
- 3 Structure Indexes**
- 4 Vector-Space Model

# Path Expression Query

- Content Only (Google-style)
  - “aviation systems verification”
- Content and Structure
  - “In articles with abstract about aviation systems give sections about verification”
  - `//article[about(./abstract, aviation systems)]//section[about(., verification)]`
- Search forms
  - Restricted version of CaS queries

# Structure Indexes

## Structure Index

A structure index  $I(G)$  of a data graph  $G$  is a summary graph that preserves all the paths in the data graph but contains a fewer number of nodes and edges

# Structure Indexes

## Structure Index

A structure index  $I(G)$  of a data graph  $G$  is a summary graph that preserves all the paths in the data graph but contains a fewer number of nodes and edges

- Element node partitioned in equivalence classes.
  - Index node for each class:  $ext(n)$
  - Edge  $(A, B)$  if exists an edge from some data node in  $ext(A)$  to some data node  $ext(B)$ .
- Result of structure path expression  $R$  on  $I(G)$ :
  - union of the extends of the index nodes that match  $R$
- **Problem:**  $I(G)$  indexes only the structural part – not sufficient for path expression queries with text.

# Integrating Inverted Indexes and Structure Indexes

Kaushik approach:

- *indexid* added to the inverted list entries
  - Element node  $n$ :  $N$  structure node whose extent contains  $n$ , then  $n.indexid = id(N)$
  - Text node  $n$ :  $N$  structure node whose extent contains the parent of  $n$ , then  $n.indexid = id(N)$

- 1 Introduction
- 2 Inverted Lists
- 3 Structure Indexes
- 4 Vector-Space Model**

# Modifying Vector Space Model for XML (1)

- Vector Model standard in IR
- An index is defined by:

Index Units IU

Index Terms IT

Retrievable Units RU

Composition Function CF

## Modifying Vector Space Model for XML (2)

### Index Units

Index units represent the documents indexable as vectors:

- Text IR: each document.
- XML IR: each node.

## Modifying Vector Space Model for XML (3)

### Index Terms

Index terms are the axis of the vector space.

- Text IR: each keyword.
- XML IR: different possibilities.
  - all document subtrees : exponential number of index terms;  
no preprocessed index; query-time materialization.
  - only root-leaves path : additional effort to match subsequence queries

## Modifying Vector Space Model for XML (4)

### Retrievable Units

Nodes that can be returned as answers

- Text IR: entire documents.
- XML IR: a node containing all the keywords.

# Modifying Vector Space Model for XML (5)

## Composition Function

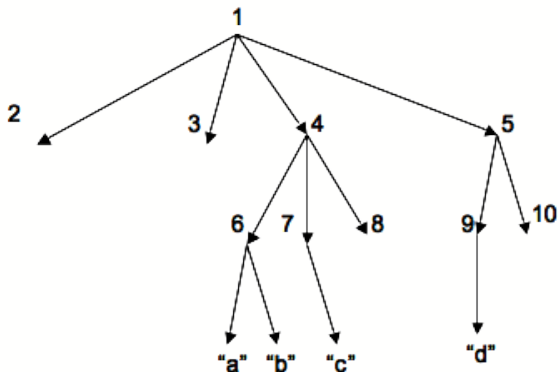
Let  $d$  be a document,  $v$  an index unit and  $t$  an index term, the composition function:

$$(d, v, t) \rightarrow r$$

$r \in \mathbb{R}^+$  is the weight of term  $t$  in the index unit  $v$ . The weight captures

- Inverse Document Frequency
- Positive Downgrade factor  $\gamma < 1$ .
  - $CF(v, t)$  weight of  $t$  in  $v$ ,
  - $a_h(v)$  ancestor of  $v$  at height  $h$ ,
  - $CF(a_h(v), t) = \gamma^h \times CF(v, t)$

# l-Path filters and $VeXML_{\gamma,l}$



Index entries  
for  $l=2$

$\langle 1,4,a \rangle$

$\langle 4,6,a \rangle$

$\langle 1,4,b \rangle$

$\langle 4,6,b \rangle$

$\langle 1,4,c \rangle$

$\langle 4,7,c \rangle$

$\langle 5,9,d \rangle$

$\langle 1,5,d \rangle$

# Index sizes

Two metrics used:

**Dictionary size** : depends by  $|IT|$  (dimensionality of the vector space)

**Posting size** : depends by  $|IU|$

## Randomized Index Costruction

- $VeXML_{\gamma,l}$  drawback: it does not address subpath matching between queries and document subtrees
- index augmented with index terms of length  $L$  randomly generated
  - $v_1, \dots, v_k, v$ : root to leaf path,  $v$  term
  - random choice of integer  $i, j$  s.t.  $1 \leq i < j \leq k$
  - index term :  $v_i v_j v$
  - randomization distribution weighted towards smaller value  $i, j$ .



Madhavrao, D. H.

Indexing, searching & ranking xml documents.

Master's thesis.