XML-background

Different kinds of data
• Structured
  - Highly regular, homogeneous structure
  - Row sets, Comma delimited files
• Semi-Structured
  - Heterogeneous structure
  - Sparse occurrences of data
  - Denormalized
  - HTML and XML documents
• Unstructured
  - Documents/Content

Types of XML documents
• Data-centric
  - Regular structure; Order, Customer, Part
  - Fine grained
  - Basis for Web-services
• Document-centric
  - Irregular structure; Book, Email, XHTML doc.
  - Coarse grained
  - With fewer markups
• No hard separation between the two document types
  - One document may be a combination of both, e.g. semi-structured
Semi-structured data
• Schema-less and self-describing: the schema is attached to the data.
• Application
  - information integration; integrate different databases together

Two different modes of XML
• Well-formed XML
  - Semi-structured data, schema-less and using user defined tags
• Valid XML; Using DTD
  - to specify tags and the grammar of the document. It has semi-strict schema. DTD uses regular expression such as *, +, ? to express the number of occurrence of the element. It is not as schema-less as semi-structured data.

Well-formed XML
<? XML VERSION="1.0" STANDALONE="yes" ?>
<my-Tag>
  <inner>…</inner>
  ...
</inner>
</my-Tag>

Valid XML
<? XML VERSION="1.0" ?>
<!DOCTYPE BOOKS ["!ELEMENT BOOKS (BOOK*)">
  <!ELEMENT BOOK (ISBN, TITLE, AUTHOR+, YEAR)>
  <!ELEMENT ISBN (#PCDATA)>
  <!ELEMENT TITLE (#PCDATA)>
  <!ELEMENT AUTHOR (#PCDATA)>
  <!ELEMENT YEAR (#PCDATA)>
]>
XML-background

- **XML document** - an ordered, labeled tree
  - **character data** - leaf nodes containing the actual data (text strings)
    - data nodes must be non-empty and non-adjacent to other character data nodes
  - **element nodes** - labeled with a name (often called the element type),
    - a set of attributes, each consisting of a name and a value,
    - can have child nodes

**XML-background**

```
<chapter id="chap1">
  <chap_title>
    Introduction
  </chap_title>
  <para>
    This chapter is an introduction about the history of IBM
  </para>
  <trade_mark>
    IBM
  </trade_mark>
  <company>
    company.
  </company>
</chapter>
```

**Table of Contents**

- XML background
- XML query languages
- ApproXQL
- The approach
- Conclusion
XML Query Languages

• Why?
  - To integrate different databases and construct new one.
  - Example
    April 1, 2002, three major Japanese banks (Dai-ichi Kangyo, Fuji Bank, and Industrial Bank of Japan) were merged into Mizuho Bank to make world’s biggest bank. It was a good news, but bad news was, database integration conflicts caused six days of chaos involving more than 30,000 transaction errors and 2.5 million delayed debits in ABM transactions.

• Most popular XML query languages
  - YaTL
  - XML-QL
  - Lorel
  - WebSQL
  - YAXQL
  - Xquery
  - TREQ-Q

Characteristics for a good query language

• Query formulation must be simple even for occasional users, and must require only partial knowledge about the collection to be queried.
• The language must support expressive queries that specify selection conditions on both the content and the hierarchical structure of XML documents.
• The interpretation of a query must be vague to find not only exact matches, but also results with a structure and content similar to the selection conditions of the query.
XML Query Languages

• The results retrieved by a query must be ranked by decreasing similarity to the query.
• The basic parameters of the similarity measure must be variable, so that they can be adapted to the characteristics of various document collections.
• A query must be evaluated in polynomial (typically sub linear) time with respect to the size of the collection. In particular, the best results must be retrieved efficiently.

Motivation

– A heterogeneous collection of data-centric documents
– Example
  • Assume that we need information about all books that have as author "Bradley".

• An XQL (XML Query Language) that exactly models our information need:
  
  \ `/book/author/text() = "Bradley"`
  
  Obviously, the only result of the query is document 1 -- despite the fact that at least the documents 2 and 3 exactly match our information need too. Moreover, the document 4 may also be of interest.
XML Query Languages

• Document (text)-centric
  - To search in Document (text)-centric XML documents, information retrieval techniques are appropriate.

XML Query Languages

• Data-centric
  - In homogeneous collections, some query languages have been proposed.
  - In heterogeneous collections, neither IR methods nor new query languages are appropriate.

XML Query Languages

• IR methods and heterogeneous XML
  - Ignoring the structure of XML doc and decreasing retrieval precision.
  - Using models based on term distributions that is not very applicable on data-centric XML doc.

XML Query Languages

• New XML Queries and heterogeneous XML doc
  - The results that do not fully match the query are not retrieved (rigid).
    (ignoring the similarity)
  - The user needs a thorough knowledge of the data structure to formulate queries.
XML Query Languages

• We need a query language for XML data-centric documents based on similarity.

Table of Contents

• XML background
• XML query languages
• ApproXQL
• The approach
• Conclusion

ApproXQL

• ApproXQL:
  - A simple pattern matching language inspired by XML Query Language (XQL proposed by W3C)
  - Based on query evaluation using evaluation of cost of embedding a query tree into data tree

Example

cd[title[“piano”] $and$ “concerto”] $and$
composer[“rachmaninov”]
ApproXQL

cd[title[“piano” $and$ “concerto”] $and$ composer[“rachmaninov”]]

Queries containing $or$ are decomposed to a disjunction normal form.

Example

cd[year[“2001”] $and$ (composer[“rachmaninov”] $or$ performer[“ashkenazy”])]

{cd[year[“2001”] $and$ composer[“rachmaninov”]],
  cd[year[“2001”] $and$ performer[“ashkenazy”]]}

Table of Contents

• XML background
• XML query languages
• ApproXQL
• The approach
• Conclusion

The approach

• Modeling data-centric XML document (XML data tree)
• Normalizing data tree
• Embedding a query tree into the data tree
• Evaluating embedding cost and query evaluation process
The approach

• Modeling
  - Model a collection of XML documents as a labeled tree with a single root.
  - Ignore ID-references and hyperlinks. (to simplify the model),
  - Use only a single node type.

• Normalization
  - Elements ➔ (are mapped) to nodes using the element name as label.
  - Attribute ➔ to two nodes, the first one gets the name as label, the second one gets the value of the attribute as label.
  - PCDATA sections ➔ to nodes by treating the whole text as label.

• Embedding
  - Mapping of a conjunctive query tree into the data tree
    - Embedding function
      • Label preserving
      • Node-type preserving
      • Parent-child preserving
    - Approximate tree embedding
      www.haifa.il.ibm.com/sigir00-xml/final-papers/Approximate.htm
The approach

• Embedding function
  - Example:

The approach

Embedding root: the data node that is mapped to the query root

Result: the sub tree that is anchored at the embedding root.

The approach

• Embedding query tree into data tree
  - Encoding and indexing XML tree
    • Text node (leaf node)
      - $(\text{pre}(u), \text{dist}(u))$
        » $\text{pre}(u)$: preorder enumeration of the data tree
        » $\text{dist}(u)$: the sum of the insert costs of all ancestors of $u$
    • Structural node:
      - $(\text{pre}(u), \text{dist}(u), \text{bound}(u))$
        » $\text{bound}(u)$: preorder number of the rightmost leaf node of the tree rooted at $u$. 
The approach

preorder enumeration of the data tree

preorder number of the rightmost leaf node

the sum of insert costs of all ancestors

Text index

Structural index

Postings
The approach

• Embedding function is not really a function because for a fixed query tree and a fixed data tree, several result may exist and for several embedding we may have the same result.

The approach

• Query evaluation process
  - Basic query transformation
    • Renaming: Changing the context and changing the search space of a query sub tree.
    • Insertion: Creating a query that expects the matches of a query sub tree in a more specific context.
    • Deletion: Moving the search space to a more general context.

The approach

• Evaluation of embedding cost
  - Each basic transformation has a cost
  - Embedding cost: the total cost of sequence of transformations
  - Procedure:
    • Decompose the user query into its conjunctive normal form
    • Create successively the closure of the set of conjunctive queries
    • Embed all queries into the data tree
    • Group the embedding according to the results
    • Find the embedding with lower cost in each group
    • Rank the results according to their embedding cost
The approach

Approximate embedding

↓

Infinite set of transformed queries

↓

Exponential runtime complexity

↑

Polynomial worst-case time complexity

Query evaluation is based on dynamic programming
- Processes the expanded query bottom-up.
- Computes the list of cost-minimal embedding roots of every query sub tree.
- Retrieves the list belonging to the query as list of evaluated results.

Expanding query representation
- Including all deletions and renamings of query (cost<infinite)
- Showing the embedding cost of a query node depends on its rename cost and on the embedding cost of its children

Example: Embedding cost of 
\[ \text{title["piano" and"concerto"]} \]
\[
\text{embedding\_cost("title")} = \\
\text{rename\_cost("title")} + \\
\text{embedding\_cost("piano")} + \\
\text{embedding\_cost("concerto")}
\]
### The approach

- Example: Embedding cost of

  \[
  \text{embedding\_cost}(\text{“title”}) = \\
  \text{rename\_cost}(\text{“title”}) + \\
  \text{Embedding\_cost}(\text{“piano”}) + \\
  \text{embedding\_cost}(\text{“concerto”})
  \]

  Min ‘delete cost’ and ‘sum of renaming the node and inserting node between title and concerto’

\[
\text{cd[track[title[“piano” $\text{and}$” concerto” $]}
\text{or$composer[“rachmaninov”]]}
\]

The total cost of deleting the query node and all its inner descendents

The cost for deleting of an inner node=3 (deletion of title)

Deletion of title and track

Cost of deleting the leaf node
The approach

- Time complexity of the query evaluation algorithm: $O(t \cdot n \cdot (i + r \cdot l))$
  - $t$: number of text selectors
  - $n$: number of name selectors
  - $i$: the time to access the index
  - $r$: maximal number of alternative labels of a query node
  - $l$: the number of entries of the largest posting

Conclusion

- An interesting idea in interpreting query as matching query tree to data tree (embedding paradigm).
- Because of approximate behavior (as an advantage) of embedding process, and several embedding possibilities, we need a ranking method to make a decision.
- Then ranking method based on embedding cost evaluation is proposed.

A bad advocate is the best attack!!
Don't you think so?