Kikori-KS: An Effective and Efficient Keyword Search System for Digital Libraries in XML

Toshiyuki Shimizu (Kyoto University)
Norimasa Terada (Nagoya University)
Masatoshi Yoshikawa (Kyoto University)

ICADL 2006 29th November

Outline

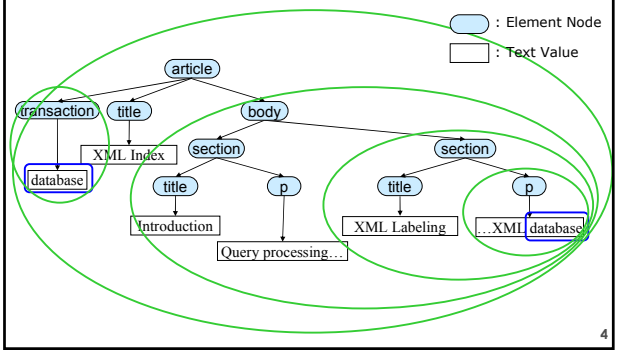
- Background
- Summary of our Contribution
- Kikori-KS
  - User Interfaces
  - Implementation of Keyword Search on Relational Databases
  - Ranking Model
- Experiments
- Conclusions, Future Works

Background (1/2)

- Large number of documents in digital libraries are now structured in XML
- Growing demand for XML Information Retrieval (XML-IR) Systems
  - We can identify meaningful document fragments by encoding documents in XML
    - ex) Sections, subsections and paragraphs in scholarly articles
    - Browsing only document fragments relevant to a certain topic
  - Keyword search on XML documents
    - Simple, intuitively understandable, yet useful form of queries, especially for unskilled end-users
    - We do not need to understand XML query languages and XML schema

Background (2/2)

- For the keyword "database"



Outline

- Background
- Summary of our Contribution
- Kikori-KS
  - User Interfaces
  - Implementation of Keyword Search on Relational Databases
  - Ranking Model
- Experiments
- Conclusions, Future Works

Summary of our Contribution

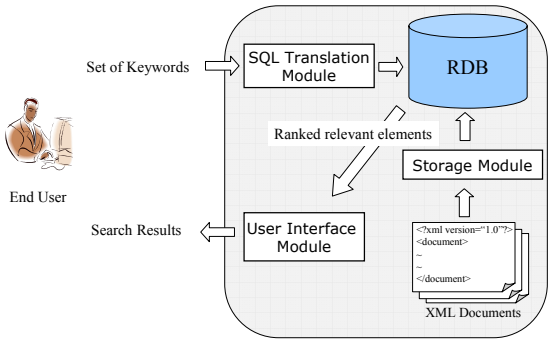
- We have developed Kikori-KS
  - A prototype system for XML-IR
  - Under Kikori Project
  - Accepts Keyword Set as a query
- User-friendly interface
  - *FetchHighlight* interface
- Storage schema on RDB
  - The database schema is carefully designed
  - Acceptable search time

Outline

- Background
- Summary of our Contribution
- Kikori-KS
  - User Interfaces
  - Implementation of Keyword Search on Relational Databases
  - Ranking Model
- Experiments
- Conclusions, Future Works

7

Overview of Kikori-KS



8

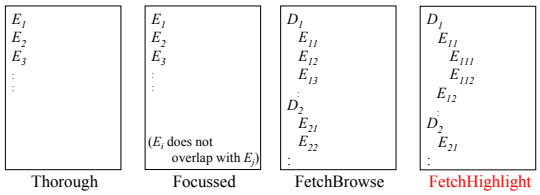
Outline

- Background
- Summary of our Contribution
- Kikori-KS
  - User Interfaces
  - Implementation of Keyword Search on Relational Databases
  - Ranking Model
- Experiments
- Conclusions, Future Works

9

User Interfaces

- Search results of XML-IR are document fragments, which may be nested
- INEX 2005 project* defined three strategies for element retrieval



- Three strategies of INEX are not necessarily intended to be used in designing user interfaces

*http://inex.is.informatik.uni-duisburg.de/2005/

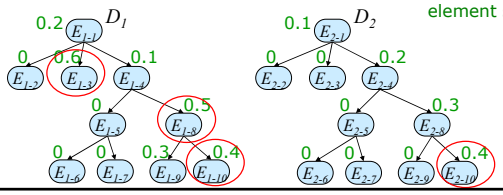
10

Retrieval Strategy of INEX (1/3)

- Thorough

$E_{1\text{-}3}$ (0.6)
$E_{1\text{-}8}$ (0.5)
$E_{1\text{-}10}$ (0.4)
$E_{2\text{-}10}$ (0.4)
:
:

- Relevant elements are retrieved in descending order of their scores



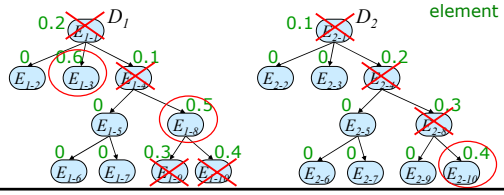
11

Retrieval Strategy of INEX (2/3)

- Focussed

$E_{1\text{-}3}$ (0.6)
$E_{1\text{-}8}$ (0.5)
$E_{2\text{-}10}$ (0.4)

- The system retrieves only focussed elements (i.e. non-overlapping elements)
- Ranked in relevance order

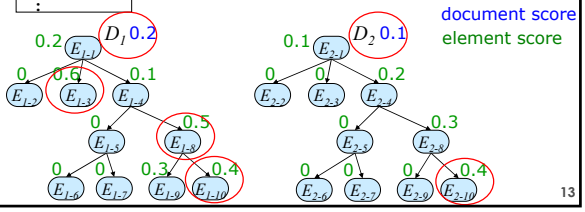


12

Retrieval Strategy of INEX (3/3)

- FetchBrowse
  - Fetching Phase
    - The system first identifies relevant documents and ranks them in relevance order
  - Browsing Phase
    - Within a fetched document, the system identifies relevant elements and ranks them in relevance order

D_1 (0.2)
$E_{1\text{-}3}$ (0.6)
$E_{1\text{-}8}$ (0.5)
$E_{1\text{-}10}$ (0.4)
:
D_2 (0.1)
$E_{2\text{-}10}$ (0.4)
:



FetchHighlight

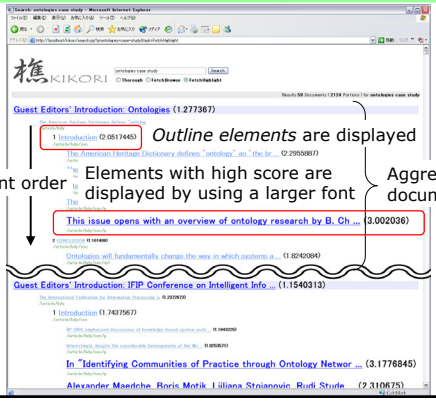
- Displaying search result elements aggregated by XML documents is effective
  - *FetchBrowse* is of that style
- Displaying search result elements in their document order is useful

D_1
E_{11}
E_{111}
E_{112}
E_{12}
:
D_2
E_{21}
:

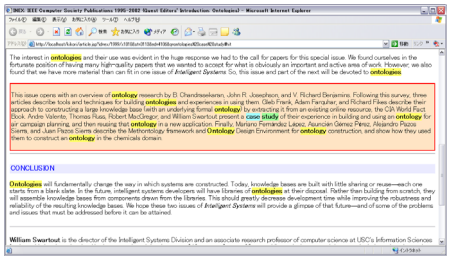
FetchHighlight

- XML documents are first sorted in their relevance order
- Relevant elements within the XML document are displayed in document order
- Elements are indented in accordance with their depth in the XML tree

FetchHighlight Interface



Browsing Document Fragment



* Selected document fragment is Highlighted
* Search words are Highlighted

The Feature of FetchHighlight Interface

- Focussed elements are easily identified
  - Users can also recognize the parts in the documents with many high relevant elements clustered
- *Outline elements*
  - Displayed even if the score is 0
  - The elements with particular structural information
    - ex) such as sections and subsections
  - Useful for browsing

Outline

- Background
- Summary of our Contribution
- Kikori-KS
  - User Interfaces
  - Implementation of Keyword Search on Relational Databases
  - Ranking Model
- Experiments
- Conclusions, Future Works

Storing XML documents into RDB

- A huge number of document fragments have to be handled efficiently
  - ex) There are 16,080,830 document fragments (elements) against 16,819 documents in the INEX 1.9 collection used in our experiments
- Storage schema based on XRel
  - Independent of the logical structure of XML documents.
- Conceptual Database Design
  - Element (docID, elemID, pathID, start, end, label)
  - Path (pathID, pathexp)
  - Term (term, docID, elemID, tfipf)

19

Conceptual Database Design

Element

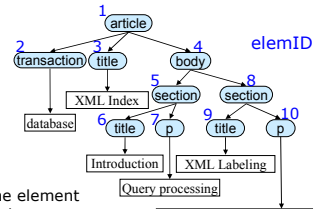
docID	elemID	pathID	start	end	label
1	1	1	1	236	XML Index
1	2	2	10	44	database
1	3	3	45	68	XML Index
:	:	:	:	:	:

Path

pathID	pathexp
1	/article
2	/article/transaction
3	/article/title
:	:

Term

term	docID	elemID	tfipf
database	1	1	0.3
database	1	2	0.1
:	:	:	:
XML	1	1	0.3
XML	1	3	0.4
:	:	:	:



* label: short text representing the element
* tfipf : term weight in the element

20

Schema Refinement (1/4)

- Materialized view
  - Join Element table, Path table, and Term table
- Partitioning the Term table with each term
  - Term_xyz (docID, elemID, tfipf, start, end, label, pathexp)
- Selecting *outline elements* and constructing an Outline table in advance
  - The system designer have to predefines outline elements
  - Outline (docID, elemID, start, end, label, pathexp)

21

Schema Refinement (2/4)

- Materialized view
  - Join Element table, Path table, and Term table

Term⋈ Element ⋈ Path

term	docID	elemID	tfipf	start	end	label	pathexp
database	1	1	0.3	1	236	XML Index	/article
database	1	2	0.1	10	44	database	/article/transaction
:	:	:	:	:	:	:	:
XML	1	1	0.3	1	236	XML Index	/article
XML	1	3	0.4	45	68	XML Index	/article/title
:	:	:	:	:	:	:	:

22

Schema Refinement (3/4)

- Partitioning the table by terms
  - Term_xyz (docID, elemID, tfipf, start, end, label, pathexp)

Term_database

docID	elemID	tfipf	start	end	label	pathexp
1	1	0.3	1	236	XML Index	/article
1	2	0.1	10	44	database	/article/transaction
:	:	:	:	:	:	:

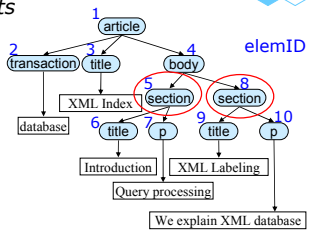
Term_XML

docID	elemID	tfipf	start	end	label	pathexp
1	1	0.3	1	236	XML Index	/article
1	3	0.4	45	68	XML Index	/article/title
:	:	:	:	:	:	:

23

Schema Refinement (4/4)

- Selecting *outline elements* and constructing an Outline table in advance
  - The system designer predefine outline elements



- Outline (docID, elemID, start, end, label, pathexp)

Outline

docID	elemID	start	end	label	pathexp
1	5	75	143	Introduction	/article/body/section
1	8	144	219	XML Labeling	/article/body/section

24

Query Translation

- The input keyword set is automatically translated into an SQL statement
  - Calculates the score of each relevant element
- Support for mandatory term (using a "+" sign) and a negation (using a "-" sign)

25

Outline

- Background
- Summary of our Contribution
- Kikori-KS
  - User Interfaces
  - Implementation of Keyword Search on Relational Databases
  - Ranking Model
- Experiments
- Conclusions, Future Works

26

Ranking Model (1/2)

- Vector Space Model
  - The score is the degree of similarity between the query vector and the element vector

$$Sim(Q,E) = \sum_{k \in Q,E} weight(k,Q) * weight(k,E)$$

Q : Query Vector

E : Element Vector

$$weight(k,Q) = qtf$$

We can use the term frequency of t in Q as the weight of the term in query

27

Ranking Model (2/2)

- Term weights in the element
  - Based on the formula in [Liu et al. 06] and [Grabs et al. 02]

$$weight(k,E) = \frac{ntf}{nel} * ipf$$

$$ntf = 1 + \ln(1 + \ln(tf))$$

Normalized term frequency (tf)

$$ipf = \ln \frac{N_p + 1}{ef_p}$$

Specificity of term t within elements that share path p

$$nel = \left((1-s) + s * \frac{el}{avgel_p} \right) * \left(1 + \ln(avgel_p)\right) \qquad s = 0.2$$

Normalization factor that reflects the element length (el) of E

28

Outline

- Background
- Summary of our Contribution
- Kikori-KS
  - User Interfaces
  - Implementation of Keyword Search on Relational Databases
  - Ranking Model
- Experiments
- Conclusions, Future Works

29

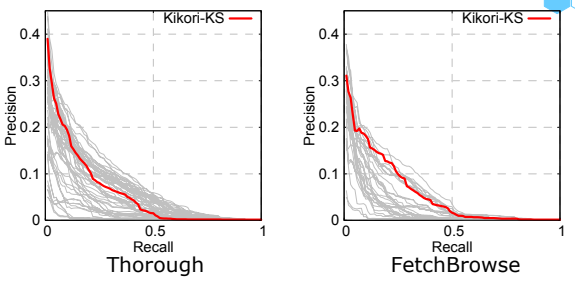
Experiments

- XML data set
  - INEX 1.9 (about 700 MB)
  - The articles of the IEEE Computer Society's publications are marked up in XML
- Query set
  - 40 queries of INEX 2005
  - INEX also provides relevance assessments
- Precision/Recall Graph for examining effectiveness
  - Precision/Recall Graph can be obtained using EvalJ*
- Query processing time for examining efficiency
- Experimental setup
  - CPU : Xeon 3.80 GHz (2 CPU)
  - RAM : 4.0 GB
  - OS : Miracle Linux 3.0
  - RDBMS : Oracle10g Release1

*http://sourceforge.net/projects/evalj/

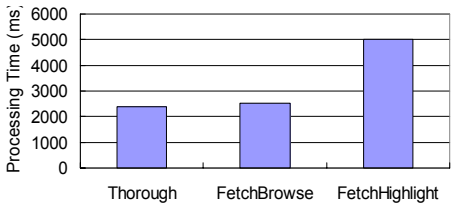
30

Precision/Recall



- The rank of Kikori-KS is relatively high especially in FetchBrowse

Processing Time



- Kikori-KS achieved acceptable search time

Outline

- Background
- Summary of our Contribution
- Kikori-KS
  - User Interfaces
  - Implementation of Keyword Search on Relational Databases
  - Ranking Model
- Experiments
- Conclusions, Future Works

Conclusions and Future Works

- Conclusions
  - Kikori-KS
    - A prototype system for XML-IR that accepts keyword set as a query
  - User-friendly FetchHighlight Interface
  - Storage schema on RDB
  - Experiments using INEX test collection
    - Kikori-KS can handle a keyword set query in an acceptable time and with relatively high precision
- Future Works
  - Developing storage schema and weighting methods for phrase searches
  - Introducing content and structure (CAS) searches

Thank you