

Integrating Document and Data Retrieval Based on XML

By

Jan Marco Bremer
Michael Gertz

Presented by

E. Cem Sozgen

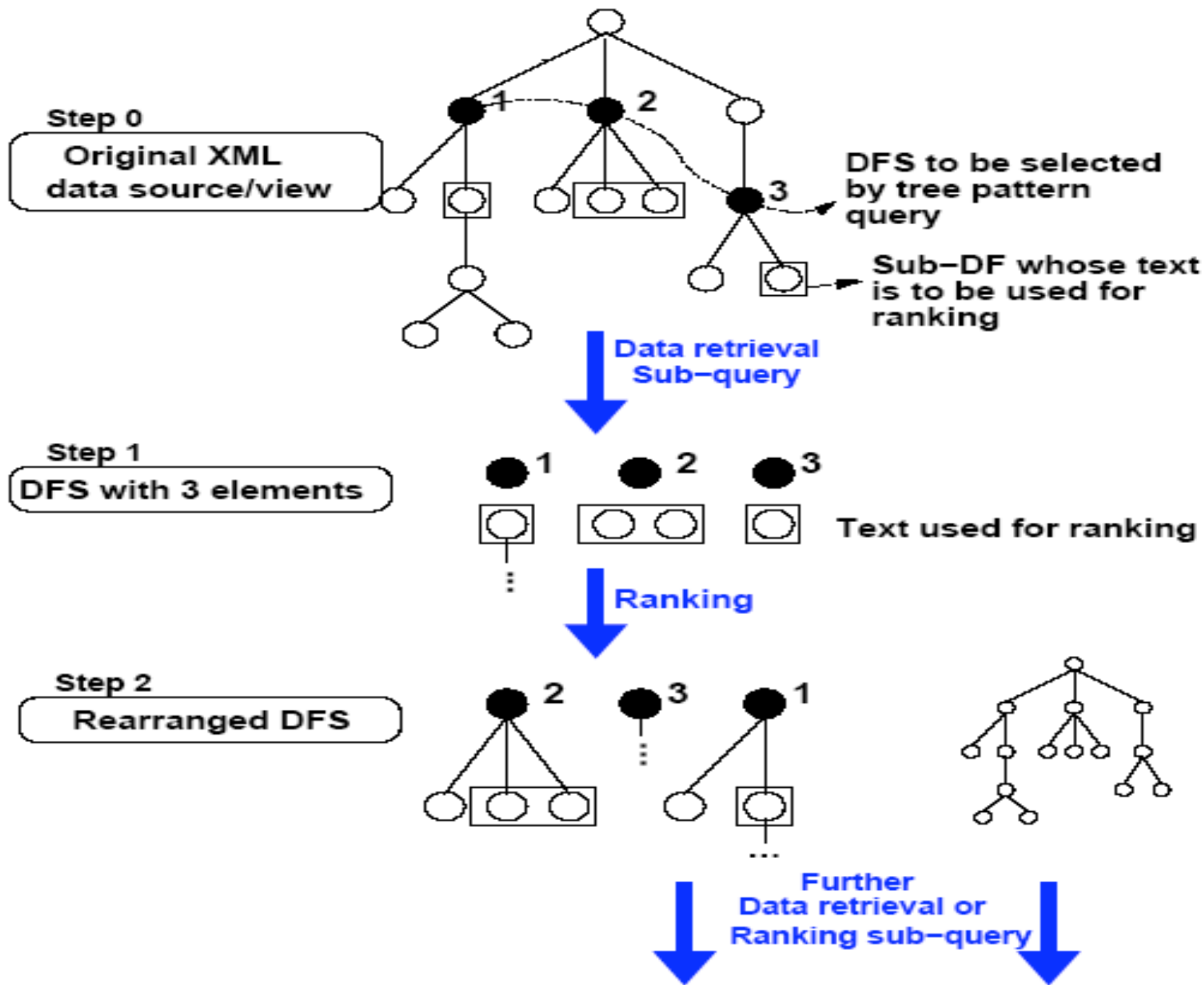
Outline

- Introduction
- Foundations of IIR
- XQuery/IR
- Efficient Data Retrieval
- Adding Support for Document Retrieval
- References
- Comments

Outline

- Introduction
- Foundations of IIR
- XQuery/IR
- Efficient Data Retrieval
- Adding Support for Document Retrieval
- References
- Comments

- Two types of query techniques:
 - Data retrieval:
 - Exact queries that allow a user to specify a subset of a data source
 - e.g. XQuery, SQL
 - Document Retrieval:
 - Ranking documents by relevance to a query of typically only a few terms
 - Relevance is based on term distribution statistics
 - A standard approach for weighting terms is the term frequency-inverse document frequency (tf-idf) approach
 - The standard index format for storing term frequencies is the *inverted file* format
- They are complementary. Also it is expensive to maintain two types of systems. So the idea is to integrate them into a single framework.








Dynamic Ranking

- Only term distribution statistics local to the current, intermediate document fragment sequences are used to derive relevance weights in an embedded document retrieval sub-query.

e.g. certain tree growth patterns that have applications in bridge construction

```
FOR $fragment IN document("library.xml")//article[contains(/abstract, "bridge construction")]  
RANK BY "tree", "growth" LIMIT 100  
RETURN $fragment/author
```

Summary of Contributions

-  A conceptually new approach to integrated data and document retrieval in general, and relevance-based document retrieval in particular
-  Syntax and semantics of XQuery/IR, a language that, based on XML and XQuery, implements the new approach
-  A new node identification scheme for tree-structured, node-labeled data sources with applications in many existing indexing approaches
-  Storage-efficient index structures that are based on the new node identification scheme, support data and document retrieval on XML, and also incorporate a mapping from logical node identifiers to their physical counterparts
-  Detailed statistics for structure and content of many large, real-world XML sources

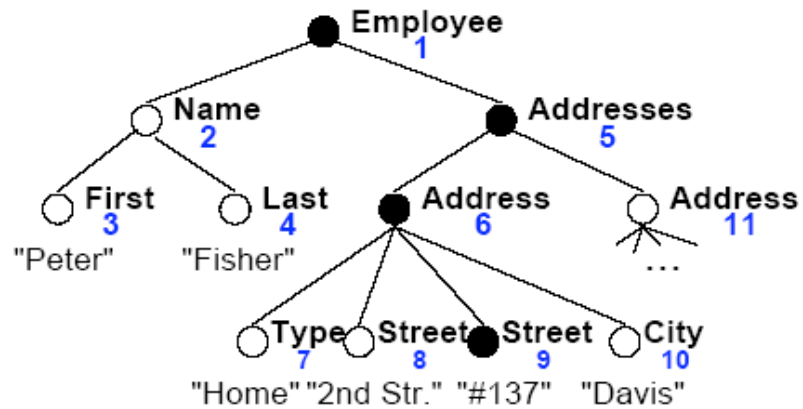
Outline

- Introduction
- Foundations of IIR
- XQuery/IR
- Efficient Data Retrieval
- Adding Support for Document Retrieval
- References
- Comments

Data Model

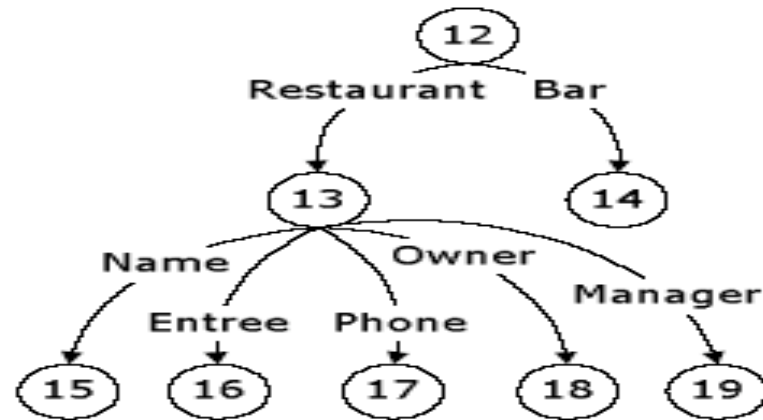
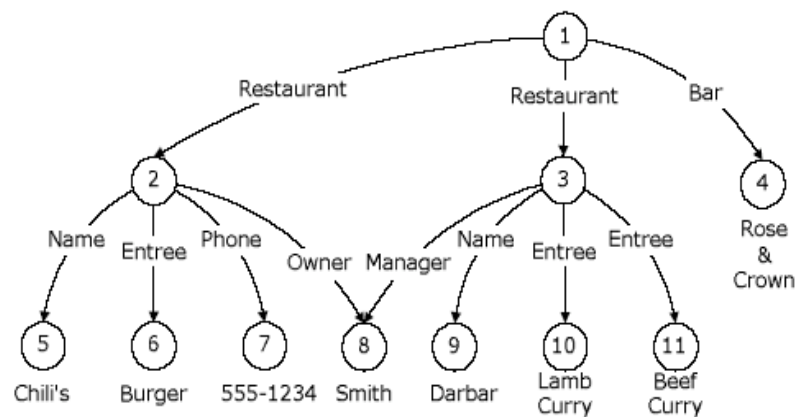
- Just a single, ordered, node labeled tree structure (an artificial root node is added to obtain a single tree view)
- Document Fragment (DF) F (corresponds to document)
 - L : set of node labels
 - T : set of text string values
 - F is $(V, \text{root}, \text{children}, \text{parent}, \text{label}, \text{text})$
- Documents or data sources or sources (all three are used in the same meaning) represent whole semi-structured database
- A document Fragment Sequence (DFS) is a sequence of document fragments

Data Model



- Employee/Addresses/Address/Street
rooted label path
- /Employee/Addresses/Address[1]/Street[2]
rooted data path

What is a DataGuide?



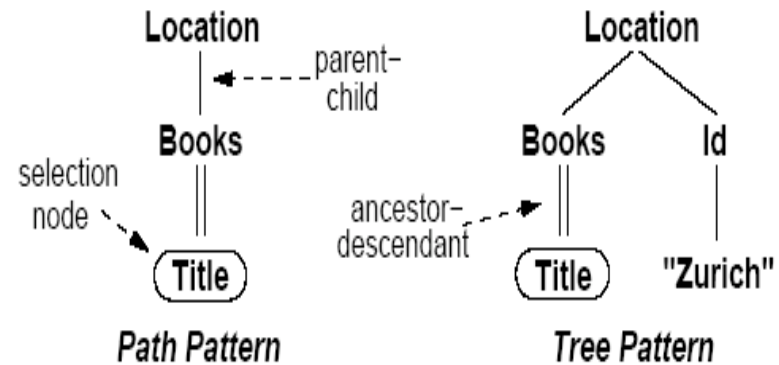
- Lore is a DBMS for semistructured data developed at Stanford University
- DataGuides are introduced for the Lore project
- Concise and accurate structural summaries of semistructured databases
- Enumerates all rooted label paths
- Eases query formulation and optimization

Schema

- A schema
 - defines the structure of the data
 - enables users to understand the structure of the database and form meaningful queries over it.
 - A query processor relies on the schema to devise efficient plans for computing query results.
- Extended DataGuides (XDG) are used in this paper
- Let $D = (V, \text{root}, \text{children}, \text{parent}, \text{label}, \text{text})$ be a data source. XDG for D is a triple $(F, \text{maxpos}, \text{nodeno})$ where
 - F is a document representing the DataGuide
 - maxpos is a function that returns the max number of siblings
 - nodeno is a function that assigns a unique node number to each node in F

Data Retrieval

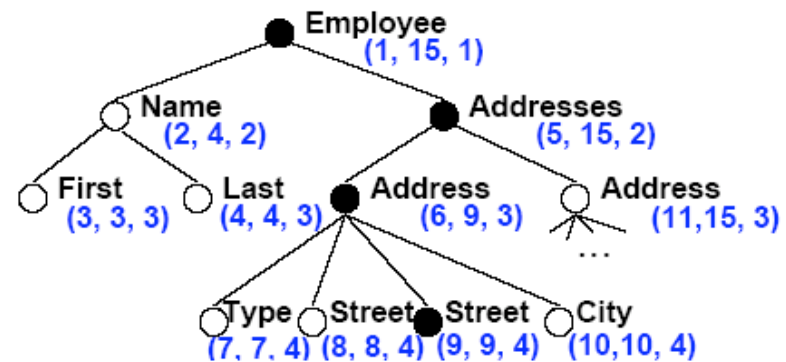
- XQuery is the standard XML query language
- XQuery is widely regarded as the future to access data on the web



- `//Location/Books//Title` (left)
- `//Location[./Id/text() = "Zurich"]/Books //Title` (right)

Query Processing Through Structural Joins

- Path and tree pattern queries are at the core of XML query languages. Structural joins are the most prominent approach to process such queries.
- A logical node id is assigned to every node in the data source. Every id of a node encodes enough information to determine the node's basic relationship to other nodes.
- Index structures that store lists of such node ids build the most important component of SJs.
- e.g. Books//Title (compare pairs of node ids in both lists)
- In existing approaches, multiple joins, one for each pairwise relationships, are required.



- Interval node identification scheme is used in present SJ approaches.
- Interval node ids (lval ids) is a triple (id, maxid, depth) of integers.
- Relationships can be determined based on [id, maxid] intervals

Outline

- Introduction
- Foundations of IIR
- XQuery/IR
- Efficient Data Retrieval
- Adding Support for Document Retrieval
- References
- Comments

Requirements

- Total order
- Local context
- Closure
- Transparency
- Exchangeability
- Visibility

Syntax and Semantics

- RankByClause ::= (<"rank" "by"> | <"stable" "rank" "by">) QuerySpecList
(<"based" "on"> TargetSpecList)?
("limit" n ("%")?)?
("ascending" | "descending")?
("using" MethodFuncCall)?

QuerySpecList ::= Expr ("," QuerySpecList)?

TargetSpecList ::= PathExpr ("," TargetSpecList)?

- Weighting algorithm:
 - Let F be the set of all document fragments and $\text{seq}(F)$ be the set of sequences of all document fragments.
 - Let $weight$ be a special node name in the label set L .
 - Then $W: \text{seq}(F) \times Q \rightarrow \text{seq}(F)$,
 $W(S, Q) = S'$
such that S' is equal to S except that each DF in S' has an additional node labeled $weight$ at the root.
- Rank operator:
 - $\text{rank}: \text{seq}(F) \times P \times N \times Q \times W \rightarrow \text{seq}(F)$
 $\text{rank}(S, \pi, k, q, W) = \text{first}_k(\text{sort}_{weight}(W(\pi(S), q)))$
where P is a set of path expressions.
 N is the set of natural numbers.
 Sort_{weight} refers to the sorting of DFs within S based on the $weight$ element.
 First_k eliminates all but the first k elements from S .

Examples

1. Retrieve a maximum of 100 paragraphs with relevant information about New York from a news source.

```
FOR $d IN
  document("news.xml")//article//paragraph
RANK BY "new", "york" LIMIT 100
RETURN $d
```

```
<paragraph weight=0.96>
  New York fire fighters. . .
</paragraph>
<paragraph weight=0.81>
  In New York, Wall Street
  closed at a record. . .
</paragraph>
<paragraph weight=0.79>
...

```

Examples

2. List the company's ten most experienced sales person in swim wear and pool accessory. The experience of a sales person is assumed to be higher the more sales transactions involving respective products the person was involved in.

```
LET $input:=
  {FOR $emp IN document("humanresources.xml")//employee
    RETURN
    <employee>
      <empid> $emp/@id </empid>
      {$emp/name}
      <products_sold>
        {FOR $prod IN document("sales.xml")//transaction[./sellerid = $emp/@id]/product
          RETURN
          <product>
            {$prod/name}
            {$prod/description}
          </product>
        }
      </products_sold>
    </employee>
  }
RETURN
<swim_wear_experts>
  FOR $emp IN $input/employee
  RANK BY "swim", "pool", "water", "aquatic", "suit", "accessory"
  BASED ON $emp/products_sold LIMIT 10
</swim_wear_experts>
```

Outline

- Introduction
- Foundations of IIR
- XQuery/IR
- Efficient Data Retrieval
- Adding Support for Document Retrieval
- References
- Comments

Which approach?

- Tree pattern queries are assumed to be the query models underlying data retrieval in XQuery. Therefore, the focus is how tree pattern queries can be efficiently processed based on suitable index structures.
- The choice of an index structure for processing pattern queries highly depends on how suitable the structure is for a document retrieval extension.
- Structural joins provide a unified framework to index structure and values of a data source based on inverted files. Structural joins are a well-known and efficient approach to process tree pattern queries. Inverted files are the core index structures for classical document retrieval. Therefore, structural joins are an obvious choice to implement IIR.

Modifications and Extensions to Basic Structural Join Approach

1. A new logical node identification scheme that encodes complete rooted data paths is used. The encoding is context-independent, space efficient, and fast to decode. No decoding is required to determine parent-child and ancestor-descendant relationships between node ids.
2. In all index structures, node ids are grouped by common, rooted label path. In combination with an Extended DataGuide (XDG), this provides for efficient index access and storage space utilization.
3. The main path index structure employs a sparse storage of node ids that has no negative impact on reconstructing index lists, but further reduces the index size.
4. We extend the standard set of index structures for structural joins by a physical address index. This index maps logical node ids to their physical addresses at no overhead beyond storing just the addresses.

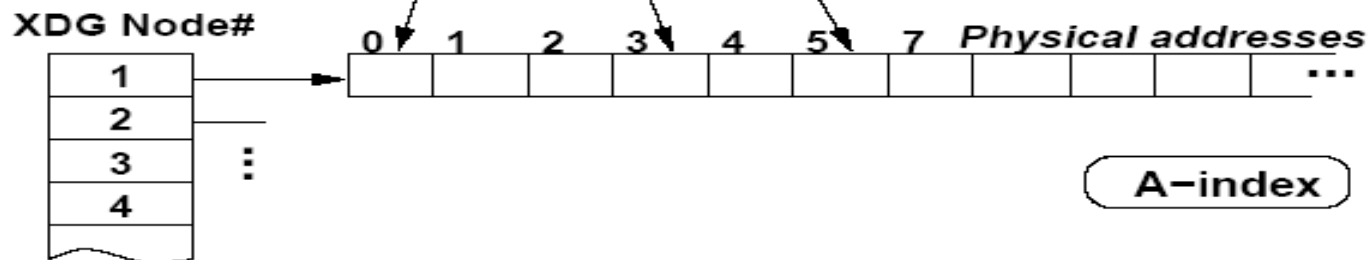
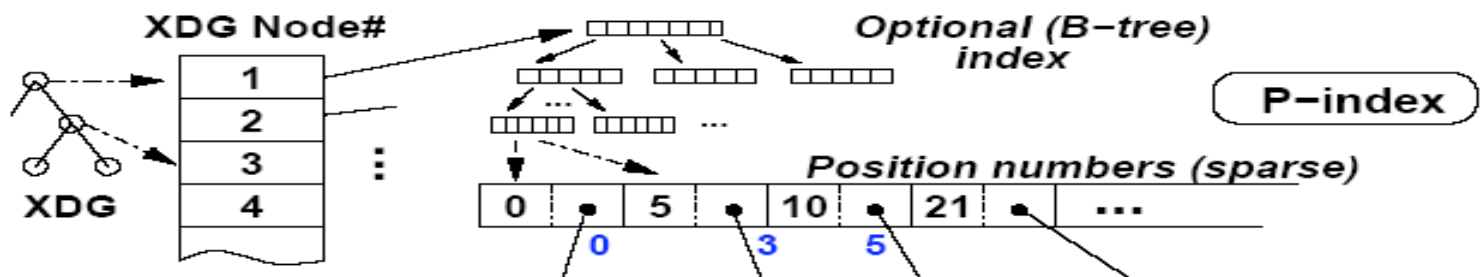
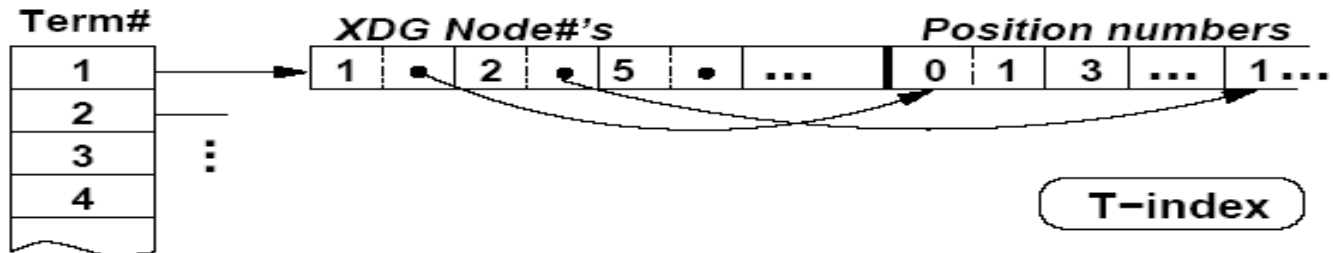
minPID and μ PID

- μ PID node identifiers consist of a pair of integers, which encode each of the two parts of a minPID. The first integer is the *node number* that is assigned to each node in the XDG and represents the minPID's rooted label path. Each node in the XDG corresponds to a rooted label path as found in a source. The second number in a μ PID is called *position number* and contains the encoded sibling position sequence.
- Position numbers are constructed by appending bits related to single sibling positions within a rooted data path. Therefore, all position numbers related to a certain node number (rooted label path) have the same bit length. Hence, they can be interpreted as k-bit integers.
- Node relationships:
 - Node $b = (n_2, N_2)$ is a child node of node $a = (n_1, N_1)$ iff
$$n_2 \in \text{children}(n_1) \text{ in XDG} \wedge N_1 = \text{prefix}_{\text{numlen}(n_1)}(N_2)$$
 - Node b is a descendant of node a iff
$$n_2 \in \text{descendant}(n_1) \text{ in XDG} \wedge N_1 = \text{prefix}_{\text{numlen}(n_1)}(N_2)$$




Index Structures

- μ PIDs are grouped by their XDG node numbers. So
 - Only position numbers need to be stored repeatedly.
 - Within each group, the number of bits of position numbers is constant.
 - Furthermore, the grouping provides for a more efficient processing of pattern queries.
- Sparse Storage:
 - 72, 73, <gap>, 76, 77, 78 will be stored as (72,0), (76,2)
 - In real world data, gaps are relatively rare.

Index Structures

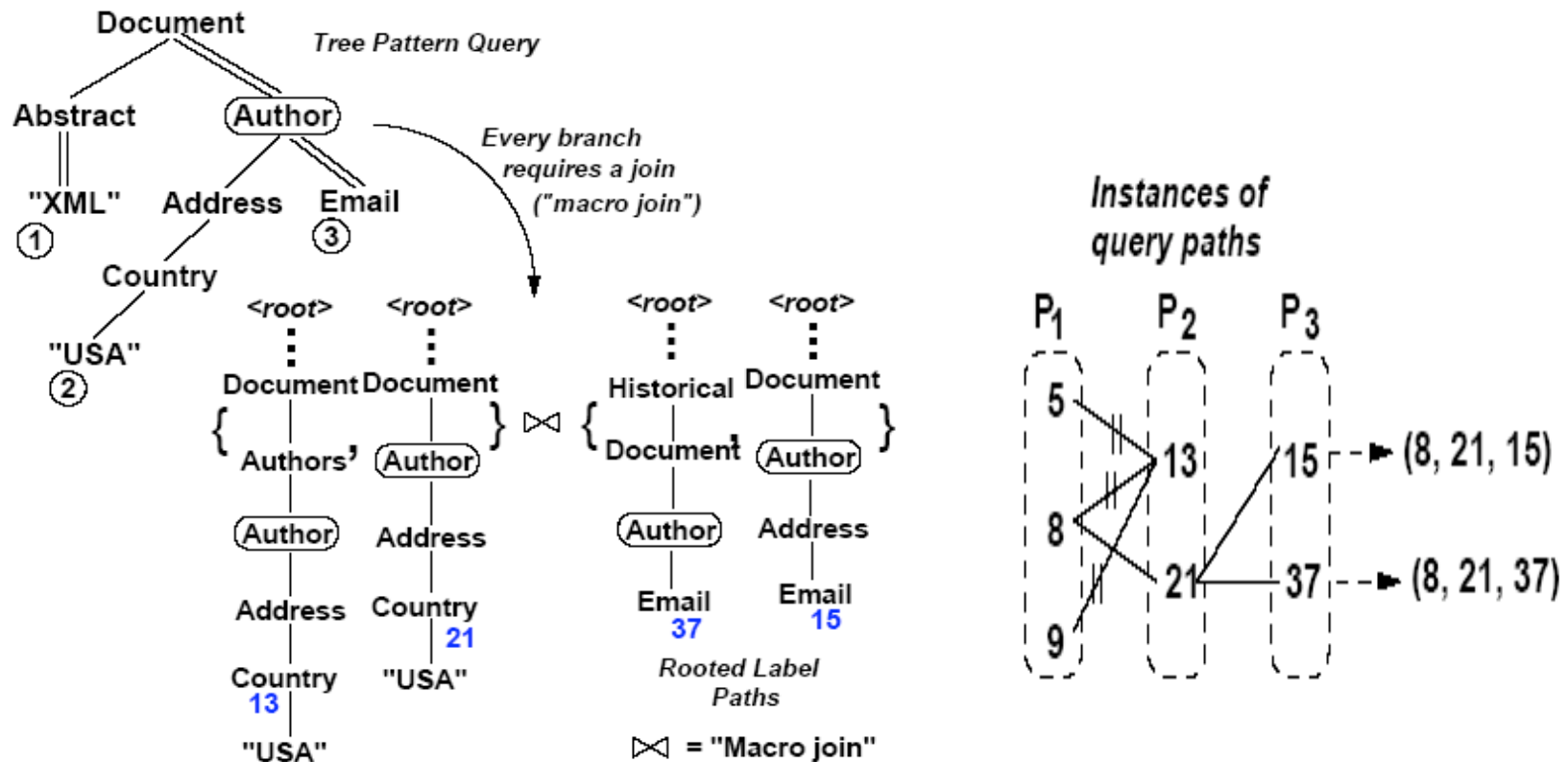


Query Processing

- Processing path pattern queries
 - Simple path patterns
 - e.g. //Location/Books//Title
 -  Find all matches of the pattern in the XDG.
 -  For each node number n_i , retrieve the index lists $a\text{-seq}(n_i)$ from the A-index.
 -  For each physical address in these lists, obtain the DF from the source.
 - General path patterns
 - e.g. //Location/Books[//Title]
 - $p\text{-seq}(\text{nodeno}(\text{LP1})) \bowtie p\text{-seq}(\text{nodeno}(\text{LP2}))$
 - LP1 is /DigitalLibrary/Location/Books
 - LP2 is /DigitalLibrary/Location/Books/Bk/Title
 - Path patterns with term condition
 - e.g. //Location/Books//Title[contains(text(), "XML")]
 - for all terms t_1, \dots, t_m in the containment condition, node id lists for all matching node numbers n_1, \dots, n_r are iteratively accessed.
 - If there are multiple terms involved $\bigcup t\text{-seq}(t, n_i)$
 - If the term condition is with respect to an inner node within a source
$$\bigcup_{j=1, \dots, m} \bigcup_{d \in \text{des}(n_i)} t\text{-seq}(t, n_i)$$
 - $p\text{-seq}(\text{nodeno}(\text{LP1})) \bowtie t\text{-seq}(t, \text{nodeno}(\text{LP1}))$

Query Processing

- Processing tree pattern queries



Evaluation

Name	Size [Mb]	Nodes (attribs.)	Terms (unique)	Avg. terms	Paths (labels)	Depth
Big10	1243.1	16,235,910 (20%)	177.7m (772,336)	13.2 (1.9)	946 (350)	9
Reuters	1354.0	37,864,292 (51%)	174.5m (313,755)	6.9 (2.4)	27 (26)	7
XMark 1Gb	1118.0	20,532,978 (19%)	126.0m (47,537)	7.0 (2.5)	548 (83)	12
XMark 500	469.4	8,631,135 (19%)	52.5m (47,464)	7.0 (2.2)	548 (83)	12
XMark 100	111.1	2,048,193 (19%)	12.4m (46,235)	7.0 (2.2)	548 (83)	12
Fin'l Times	564.1	2,847,870 (0%)	92.0m (396,571)	32.3	17 (17)	3
LA Times	475.3	5,472,913 (10%)	72.9m (250,560)	14.7 (1.2)	71 (28)	7
DBLP	127.7	3,736,406 (12%)	12.9m (412,267)	3.4 (3.7)	145 (40)	6
SwissProt	109.5	5,166,890 (42%)	8.6m (136,950)	1.7 (1.7)	264 (99)	5
Shakespeare	7.3	179,609 (0%)	0.9m (23,076)	5.1	58 (22)	5

Name	Path index [Mb]		Term index [Mb]		Addr. index [Mb]	Node id length [bits]			
	Ival	μ PID (%Ival)	Ival	μ PID (%Ival)		Ival	μ Pmax (P#)	Pos#avg (P/T)	
Big10	106.5	23.1 (22%)	747.5	347.1 (46%)	51.9	52	51 (41)	22.5	24.4
Reuters	261.8	55.9 (21%)	1,113.8	519.1 (47%)	70.2	55	34 (29)	25.7	26.9
XMark 1Gb	139.5	24.2 (17%)	844.8	306.4 (36%)	63.8	54	39 (29)	21.1	19.8
XMark 500	56.6	9.5 (17%)	339.7	127.2 (37%)	25.1	52	38 (28)	20.0	18.8
XMark 100	12.0	2.0 (17%)	72.0	31.9 (44%)	5.6	46	36 (26)	17.9	16.6
Financial Times	16.6	0.9 (5%)	305.0	115.0 (38%)	10.5	46	26 (21)	20.1	18.0
LA Times	33.9	5.7 (17%)	386.2	216.5 (56%)	18.2	49	48 (41)	27.3	29.1
DBLP	22.3	5.1 (23%)	78.5	35.7 (45%)	11.4	35	34 (26)	20.2	19.2
SwissProt	32.0	10.3 (32%)	54.0	25.8 (48%)	9.9	49	39 (30)	21.5	22.8
Shakespeare	0.9	0.3 (37%)	4.6	3.2 (70%)	0.5	39	35 (29)	26.2	28.2

Evaluation

Name	P-index		T-index		A-index
	Ival	μ PID	Ival	μ PID	
Big10	8.6	1.9	60.1	27.9	4.0
Reuters	19.3	4.1	82.3	38.3	5.2
XMark 1Gb	12.5	2.2	75.6	27.4	5.7
XMark 500	12.1	2.0	72.4	27.1	5.3
XMark 100	10.8	1.8	64.8	28.7	5.0
Fin'l Times	2.9	0.2	54.1	20.4	1.9
LA Times	7.1	1.2	81.3	45.6	3.8
DBLP	17.5	4.0	61.5	28.0	8.7
SwissProt	29.2	9.4	49.3	23.6	9.0
Shakespeare	12.3	4.1	63.1	43.8	6.8
<i>Average</i>	<i>13.2</i>	<i>3.1</i>	<i>66.5</i>	<i>31.1</i>	<i>5.5</i>

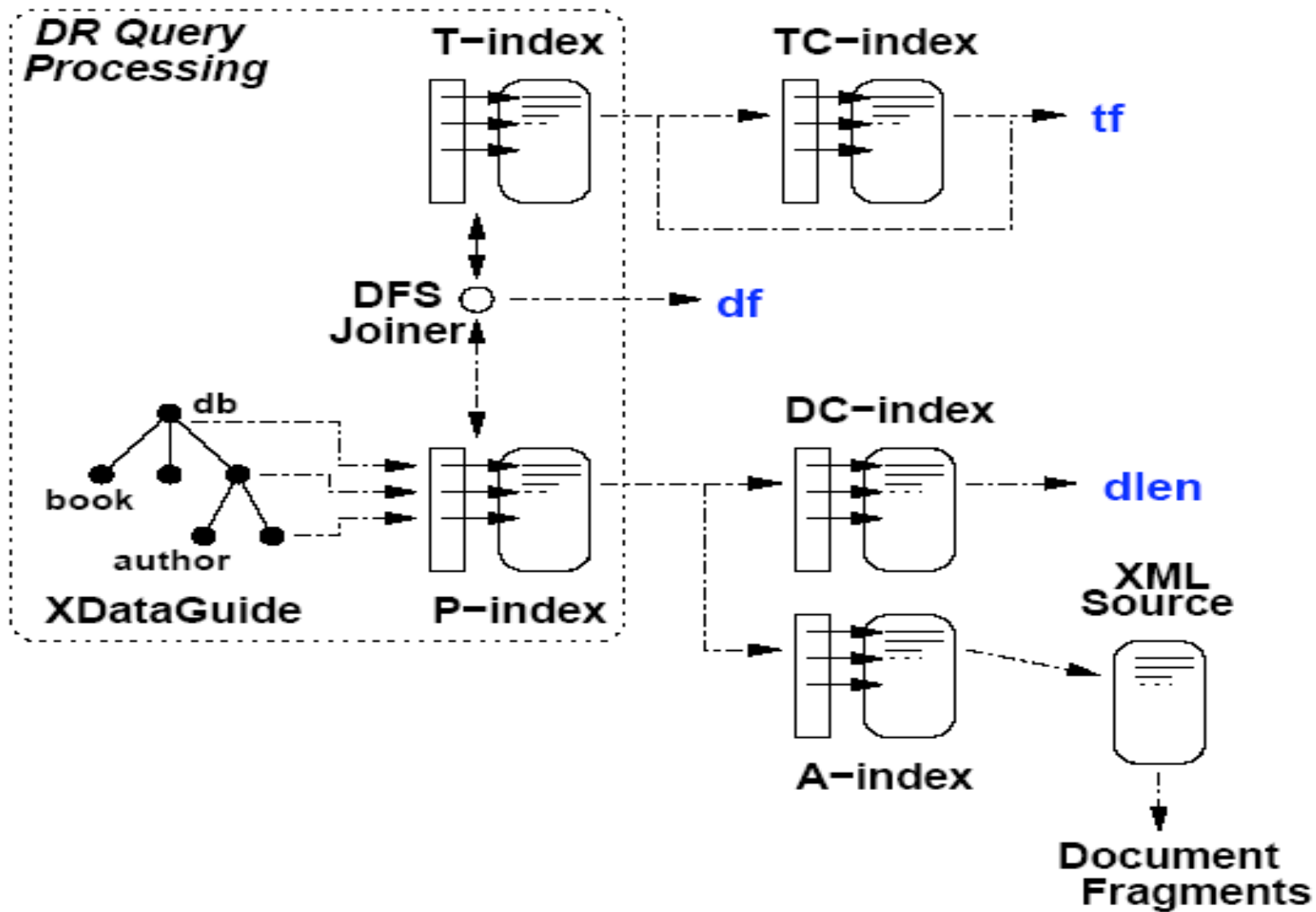
Outline

- Introduction
- Foundations of IIR
- XQuery/IR
- Efficient Data Retrieval
- Adding Support for Document Retrieval
- References
- Comments

Options to Incorporate parameters

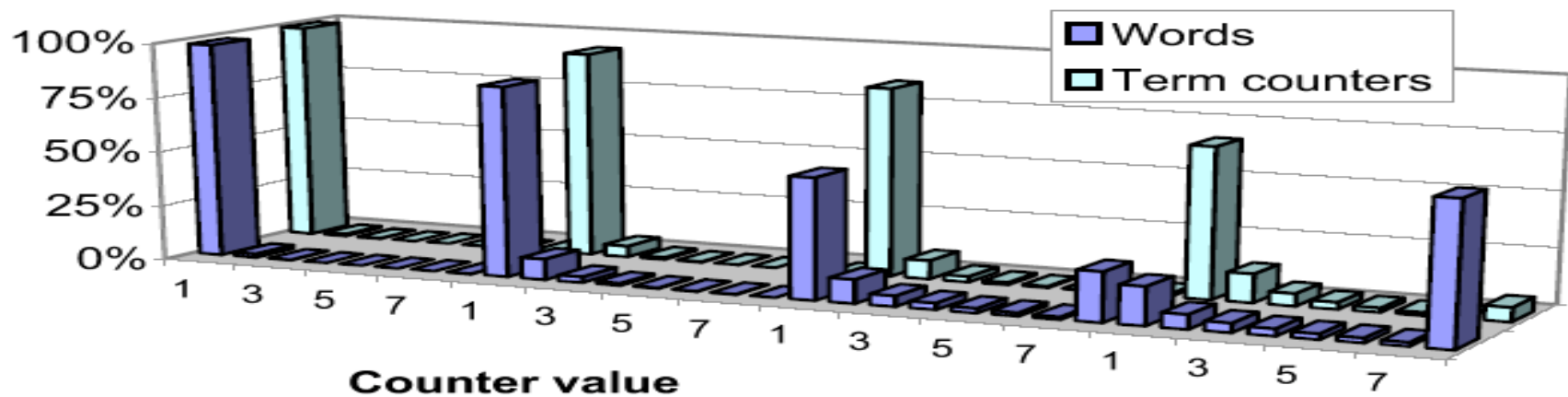
- Three parameters
 - *tf*: the number of times a certain term occurs in a document fragment
 - *df*: the number of fragments in a document fragment sequence that contain the term, and
 - *dlen*: the total number of non-unique terms found in a document fragment.
- The *df* parameter is a by-product of data retrieval or can be obtained through an additional structural join.
- Options for placing *tf* and *dlen* counters into the core index framework, counters can be
 1. repeatedly stored or accumulated at runtime,
 2. stored directly within the existing core indexes, or in additional indexes, or in both,
 3. variable-length or fixed-length as approximations. Variable-length storage within core indexes is not advisable, but an option when using an additional (*split*) index.

Resulting Index Framework



Evaluation

Name	P-index	A-index	DC-index	T-index _{1bit}	TC-index _{max}	Total
Big10	23.1 (2%)	51.9 (4%)	9.5 (1%)	360.6 (29%)	83.3 (7%)	533.7 (43%)
Reuters	55.9 (4%)	70.2 (5%)	16.2 (1%)	538.3 (40%)	46.9 (3%)	680.6 (50%)
XMark 1Gb	24.2 (2%)	63.8 (6%)	8.6 (1%)	325.4 (29%)	23.0 (2%)	422.0 (38%)
XMark 500	9.5 (2%)	25.1 (5%)	3.5 (1%)	137.0 (29%)	17.2 (4%)	175.1 (37%)
XMark 100	2.0 (2%)	5.6 (5%)	0.8 (1%)	35.7 (32%)	10.5 (9%)	44.1 (40%)
Fin'l Times	0.9 (0%)	10.5 (2%)	2.0 (0%)	121.1 (22%)	41.4 (7%)	134.5 (24%)
LA Times	5.7 (1%)	18.2 (4%)	4.6 (1%)	224.7 (47%)	34.5 (7%)	253.2 (53%)
DBLP	5.1 (4%)	11.4 (9%)	1.9 (1%)	37.2 (29%)	7.7 (6%)	55.6 (44%)
SwissProt	10.3 (9%)	9.9 (9%)	1.5 (1%)	26.8 (25%)	2.9 (3%)	48.5 (44%)
Shakespeare	0.3 (4%)	0.5 (7%)	0.1 (1%)	3.3 (45%)	0.4 (5%)	4.2 (58%)



Outline

- Introduction
- Foundations of IIR
- XQuery/IR
- Efficient Data Retrieval
- Adding Support for Document Retrieval
- References
- Comments

References

- *Next-Generation Information Retrieval: Integrating Document and Data Retrieval Based on XML*
Jan-Marco Bremer. Ph.D. Thesis, Technical Report CSE-2003-16. Department of Computer Science, University of California at Davis, September 2003.
- R. Goldman and J. Widom. DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases. Proceedings of the Twenty-Third International Conference on Very Large Data Bases, pages 436-445, Athens, Greece, August 1997.
- Amer-Yahi, S., Botev, C., Shanmugasundaram, J.: TeXQuery: A full-text search extension to XQuery. In: Proceedings of the 13th World Wide Web Conference. (2004) 583–594
- Fuhr, N., Grossjohann, K.: XIRQL: A query language for information retrieval in XML documents. In: Proceedings of 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. (2001) 172–180

Outline

- Introduction
- Foundations of IIR
- XQuery/IR
- Efficient Data Retrieval
- Adding Support for Document Retrieval
- References
- Comments

Comments...