8 Ranked Retrieval of XML Data

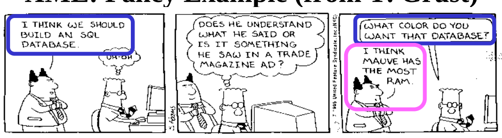
8.1 Basic Concepts of XML

- (Freely definable) tags: book, title, author
  - with start tag: <book> etc.
  - and end tag: </book> etc.
- **Elements**: <book> ... </book>
  Elements have a name (book) and a content (...)
  Elements may be nested.
- Each XML document has a root element and forms a tree.

Element content may be typed (mostly PCDATA – parsed character data, i.e., strings, possibly with nested elements).
Elements may have **attributes** that have a name and a value (content), e.g. <article year=1999>.
Elements optionally have id attributes (element ids) from which references within a document can be constructed via idref attributes.
Elements may have outgoing hyperlinks via href attributes.

Elements with a common parent are ordered.

XML: Fancy Example (from T. Grust)




```
<strip copyright=„...“ year=2000>
  <panel no=„1“>
      <speech speaker=„Pointy-Haired Boss“ to=„Dilbert“>
          <character> Pointy-Haired Boss </character>
          <bubble> I think we should build an SQL database. </bubble>
      </speech> </panel>
  <panel no=„2“>
      <speech mode=„thinking“>
          <character> Dilbert </character>
          <bubble> Does he understand what he said or ... </bubble>
      </speech> </panel>
  <panel no=„3“>
      <speech speaker=„Dilbert“ mode=„question“>
          <bubble> What color do you want that database? </bubble>
      </speech>
      <speech speaker=„Pointy-Haired Boss“ tone=„self-confident“>
          <bubble> I think mauve has the most RAM.</bubble>
      </speech> </panel>
</strip>
```


IMDB Data in XML

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
- <movie id="62480">
   <title>Contact</title>
   <production_year>1997</production_year>
   <production_country>USA</production_country>
   <production_language>English</production_language>
   <production_language>Spanish</production_language>
   <production_location>Arecibo Observatory, Arecibo, Puerto Rico</production_location>
   <production_location>Canyon de Chelly National Monument, Chinle, Arizona, USA</prod
...
- <producer xmlns:xlink="http://www.w3.org/1999/xlink/namespace/" xlink:type="simple" x
   <name>Bradshaw, Joan</name>   <job>executive producer</job>
   </producer>
- <cast order="credits">
- <casting position="1">
   <actor xmlns:xlink="http://www.w3.org/1999/xlink/namespace/" xlink:type="simple" xlin
    Foster, Jodie</actor>
   <role>Dr. Eleanor Ann 'Ellie' Arroway</role>
   </casting>

   <plot author="Ed Howell <eon@combase.com>">Contact, based on the novel of the
     same name by Carl Sagan, is the story of a free thinking radio astronomer ... </plot>
...
   <goof type="factual error">: It is impossible for Dr. Arroway to have graduated from
     MIT Magna Cum Laude, as MIT gives no such distinctions.</goof>
...
```


XML-based Data Annotation Standards

XML is the basis for a variety of application-field-specific data annotation standards, e.g.:

- MathML: Mathematical Markup Language
- CML: Chemical Markup Language
- CellML for computer-based biological models
- ebXML for e-business data (e.g., invoices)
- WeatherML for weather observation data
- NITF for news agencies
- etc. etc.

> XML is mere syntax, but it creates a momentum towards standardized terminologies (ontologies), thus potentially enabling large-scale data exchange (and more effective information search)

CML Example

```
<cml title=„ethanol“ id=„cml_ethanol_karne“>
  <molecul title=„ethanol“ id=mol_ethanol_karne“>
    <formula> C2 H6 O </formula>
    <string title=„CAS“>64-17-5</string>
    <float title=„molecular weight“>46.07</float>
    <atomArray>
      <atom id=„ethanol_karne_a_1“>
        <float builtin=„x3“ units=„A“>1.0303</float>
        <float builtin=„y3“ units=„A“>0.8847</float>
        <float builtin=„z3“ units=„A“>0.9763</float>
        <string builtin=„elementType“>C</string>
      </atom>
      <atom id=„ethanol_karne_a_2“>
       ... </atom> ...
    </atomArray>
    <bondArray>
      <bond id=„ethanol_karne_b_1“>
        <string builtin=„atomRef“>ethanol_karne_a_1</string>
        <string builtin=„atomRef“>ethanol_karne_a_2</string>
        <string builtin=„order“ convention=„MDL“>1</string>
      </bond>
...
```

Boolean Retrieval with XPath and XQuery

XPath and XQuery are query languages for XML data, both standardized by the W3C and supported by various database products.
Their search capabilities include
- **logical conditions** over element and attribute content (first-order predicate logic a la SQL; simple conditions only in XPath)
- **regular expressions** for pattern matching of element names along paths or subtrees within XML data

+ joins, grouping, aggregation, transformation, etc. (XQuery only)

In contrast to database query languages like SQL an XML query does not necessarily (need to) know a fixed structural schema for the underlying data.
A **query result** is a set of qualifying nodes, paths, subtrees, or subgraphs from the underlying data graph, or a set of XML documents constructed from this raw result.

XPath by Examples

/movie/casting/actor	all actors in the castings of all movies
/movie[title=„Contact“]/casting	all people in the casting of a given movie
/movie[title=„Contact“]//actor /movie[title=„Contact“]/*/actor	all actors in a given movie
/movie/casting[@position=1]/actor	all stars
/movie[casting/actor = „Foster, Jodie]//title	titles of movies with a given actor
/movie[casting/actor = „Foster, Jodie]/casting[@position=1]/actor	stars of movies with a given actor
/movie[//goof]/casting/* /movie/casting/*[ancestor::*[name() = goof]]	casting details for movies with goofs

Semantics of XPath Queries

An XPath **path expression** (the core of a query) is a sequence of **location steps**, separated by /, each of which has
- a navigation axis (e.g., children denoted by /, descendants //, etc.) relative to a context node (i.e., a current node) and a
- condition to be matched, which may in turn be a path expression with a logical condition for the end node of a qualifying path

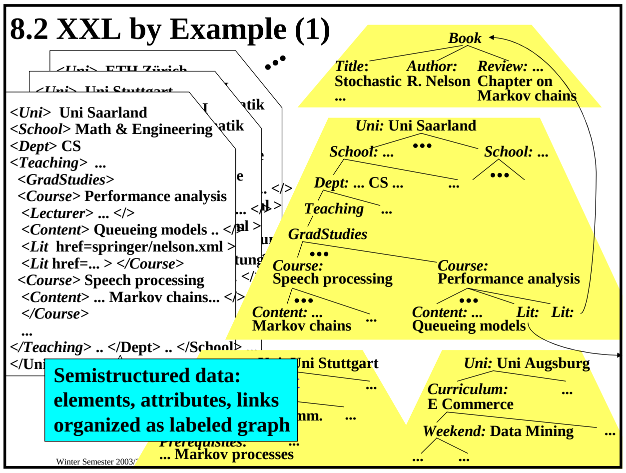
The evaluation of a path expression computes a function
nodes $\rightarrow 2^{\text{nodes}}$,
i.e., determines for a given initial context node the set of nodes that are reachable by the given sequence of location steps and whose paths satisfy all specified conditions.

XPath Location Axes

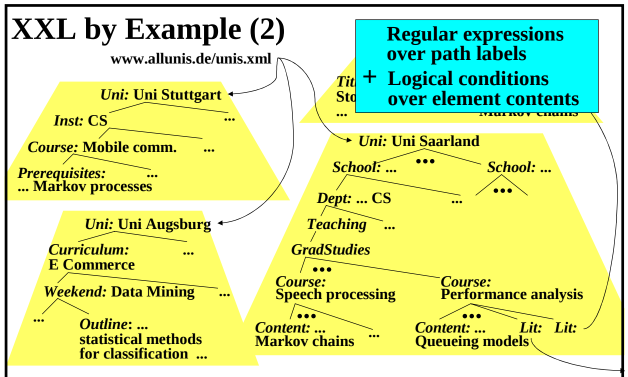
The general form of a location step is
axis::test[predicate]
where test is a function on a node (with Boolean result, e.g., referring to the element name or position)
and predicate is a function or a path expression

Axis	***Shortcut***	***Comment***
child::node()		node() is true for all nodes
descendant::node()	//	
descendant-or-self::node()		
self::node()	.	
parent::node()	..	
ancestor::node()		
ancestor-or-self::node()		
following::node()		in preorder traversal of doc tree
preceding::node()		
following-sibling::node()		siblings to the right
preceding-sibling::node()		siblings to the left
attribute::	@	

8.2 XXL by Example (1)

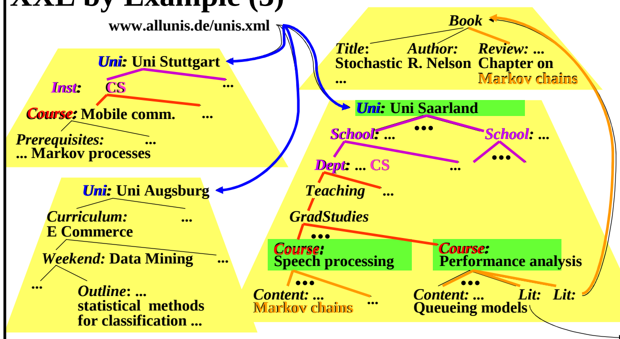


XXL by Example (2)



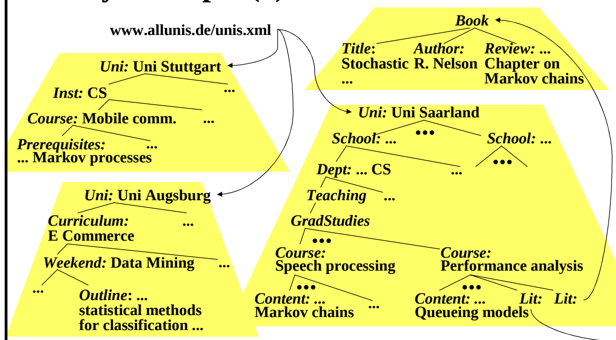
Select U, C From www.allunis.de/unis.xml Where Uni As U
And U.#.School?.#.(Inst | Dept)+ As D And D Like „%CS%“
And D.#.Course As C And C.# Like „%Markov chain%“

XXL by Example (3)



Select U, C From www.allunis.de/unis.xml Where Uni As U
And U.#.School?.#.(Inst | Dept)+ As D And D Like „%CS%“
And D.#.Course As C And C.# Like „%Markov chain%“

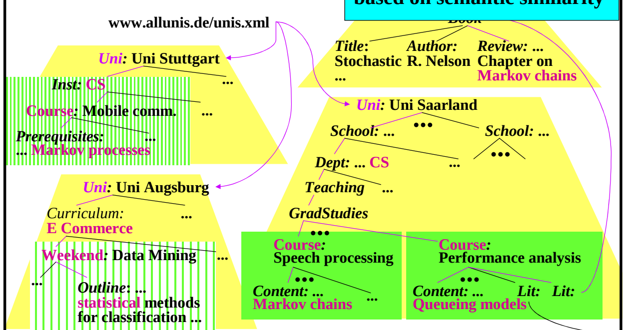
XXL by Example (4)



Select U, C From www.allunis.de/unis.xml Where Uni As U
And U.# As D And D ~ „CS“
And D.#.~Course As C And C.# ~ „Markov chain“

XXL by Example (5)

Result ranking of XML data based on semantic similarity



Select U, C From www.allunis.de/unis.xml Where Uni As U
And U.# As D And D ~ „CS“
And D.#.~Course As C And C.# ~ „Markov chain“

XXL Concepts

Extensible, simple core language

Where clause: conjunction of regular path expressions with binding of variables

Elementary conditions on element/attribute names and contents

Select *D, L, S* From *www.allunis.de/unis.xml*
Where *Uni.#.School?.#.(Inst|Dept) As D*
And *D.#.Lecturer As L And D.#.Student As S*
And *L.Name = S.Name And L.Area Like „%XML%“*

Semantic similarity conditions on names and contents
... D.#.~Lecturer As L And L.~Area ~ „XML“

Based on tf*idf similarity of contents,
ontological similarity of names,
probabilistic combination of conditions

Query Semantics:
- **query is a path/tree/graph pattern**
- **results are isomorphic paths/subtrees/subgraphs of the data graph**

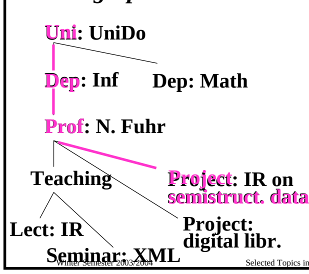
XXL Result Ranking

Query:

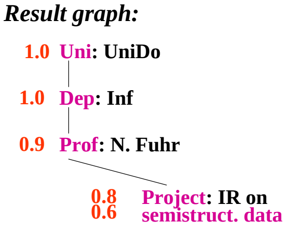
Where Uni.#.School?.#.(Inst | De[pt) ...]
And I.#.~Lecturer As L
And L.~Area ~ „XML“

Query Semantics:
- **query is a pattern with relaxable conditions**
- **results are approximate matches to query with similarity scores**

Data graph:



Result graph:



Relevance score: 0.432

XXL Semantics (1): Exact Queries and Variable-Free Path Expressions

Consider a data graph G=(V,E) with XML elements as nodes V and parent-child relationships or links as edges E.

An ***XXL query*** (actually its *Where* clause) is a conjunction of
- ***search conditions*** q1, ..., qm (m⩾1) where each condition is a *path expression*, which can optionally be bound to an *element variable*, and
- ***elementary content conditions*** c1, ..., ck (k⩾0) of the form *variable op constant* or *variable op variable* with comparison operator op ∈{=, !=, <, >, Like, ...}.

A ***path expression*** is a restricted regular expression over element labels: every label x is an expression, the wild card # is an expression, and for expressions x and y, x.y, x|y, x.#.y are expressions, too.

A path e1...en of the data graph satisfies a variable-free path expression q if the sequence of labels along the path is in the language defined by the regular expression q. $\mu[q] \subseteq V^+$ denotes the ***set of matching paths***.

XXL Semantics (2): Exact Queries with Variables

Every path expression can have an optional *As* clause which binds the end points of the qualifying paths to an element variable.

The „uses" relation between path expressions q1, ..., qm of the same query is defined as follows: qi < qj (qj uses qi) if qj contains a variable that is bound to the qualifying paths of qi, We restrict the uses relation such that its transitive closure is irreflexive and acyclic.

A path expression qi may contain element variables. The elements that are bound to the variables are substituted into p.
For a given variable binding ν: VAR $\rightarrow$ V, the ***result*** $\mu_\nu[qi] \subseteq V^+$ of qi containing variables x, y, ... is μ[qi[x/ ν(x).name, y/ ν(y).name, ...]].

A subgraph of G is a ***result of the query*** with path expressions q1, ..., qm and elementary content conditions c1, ..., ck with variables x, y, ...
if there is a (global) variable binding ν such that
the subgraph is the union of paths p1, ..., pm with pi $\in \mu_\nu$[qi] for all i
and cj[x/ ν(x).content, y/ ν(y).content, ...] evaluates to true for all j.

XXL Semantics (3): Queries with Similarity Conditions

Assume that similarity functions are defined between element names and between texts (and between dates, spatial names, etc.)

An element with label *l* ***approximately matches*** subcondition *~label* in path expression qi if the similarity *sim(l, label) > 0*.
An element with content *c* bound to variable *x* approximately matches elementary content condition *x ~const* if *sim(c,const)>0*, and two elements with contents *c, c'* bound to variables *x, x'* approximately satisfy elementary content condition *x~x'* if *sim(c,c')>0*.

A subgraph of G is an ***approximate result*** of query q
with path expressions q1, ..., qm and
elementary content conditions c1, ..., ck with variables x, y, ...
if there is a (global) variable binding ν such that
the subgraph is the union of paths p1, ..., pm such that
- pi is an approximate result in μ_ν[qi] for all i and
- cj[x/ ν(x).content, y/ ν(y).content, ...] is approximately satisfied for all j.

XXL Semantics (4): Query Result Scoring

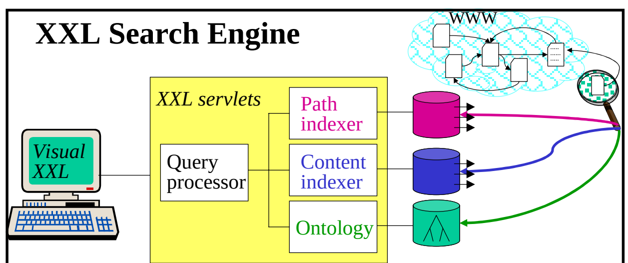
An ***element e or a path p is scored***
- with regard to a subcondition of the form *x, #, x|y, ~x* by the similarity with which it approximately matches the subcondition,

and an ***element e or a pair (e, e') of elements is scored***
- with regard to an elementary content condition *x ~ const* or *x ~ x'* by the similarity between e and the given constant or between e and e'.

An approximately matching ***subgraph is scored*** with regard to a query by the product of the scores of its components with regard to the underlying path subconditions and elementary content conditions.

The ***result of an XXL query with similarity conditions*** is a ranked list of approximately matching subgraphs in descending order of scores.

XXL Search Engine

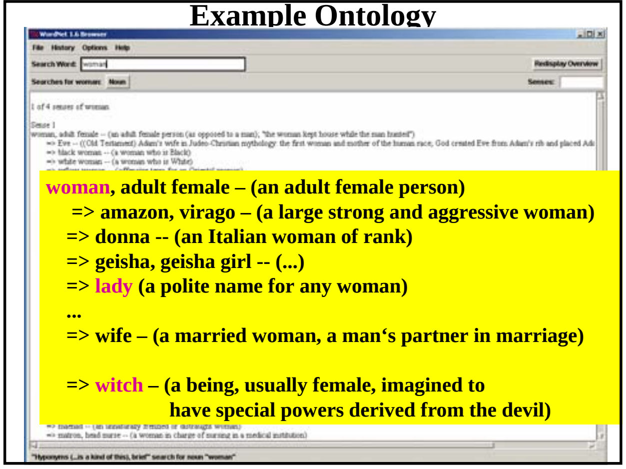


Select ... Where
Uni.#.(Inst|Dept) As F
And F ~ „Computer Science"
And F.#.~Course.#
~ „Markov Chains"

Uni.#.(Inst|Dept) As F
F ~ „Computer Science"
F.#.Course.#
~ „Markov Chains"
F.#.Seminar.#
~ „Markov Chains"

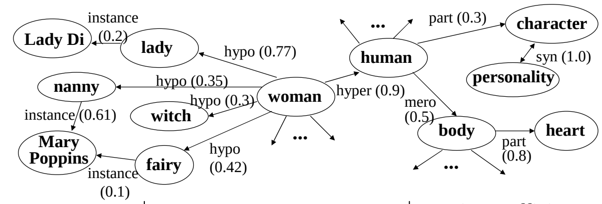
- **Query decomposition into index-supported subexpressions**
- **wide range of optimizations**

Example Ontology



Ontology Graph

An ontology graph is a directed graph with concepts (and their descriptions) as nodes and semantic relationships as edges (e.g., hypernyms).



$$\text{sim}(c1, c2) = \frac{2\,|\{docs\ with\ c1\} \cap \{docs\ with\ c2\}|}{|\{docs\ with\ c1\}| + |\{docs\ with\ c2\}|}$$

Dice coefficient from large corpus (e.g. focused crawl)

$$\text{sim*}(c1, cn) = \max\{\ \prod_{i=1..n-1} sim(c_i, c_{i+1})\ |\ all\ paths\ from\ c1\ to\ cn\}$$

Benefit from Ontology Service

Ontology service accessible via SOAP or RMI
Ontology filled with WordNet, geo gazetteer, focused crawl results, extracted tables & forms

Threshold-based query expansion:
substitute ~w by (c_1 | ... | c_k) with all c_i for which sim(w, c_i) ≥ δ

Support for automatic tagging of HTML data:
- heuristic rules (<homepage>, <publication>, table headings, etc.)
- named-entity recognition (persons, companies, cities, temporal phrases)

Query keyword disambiguation:
- contexts con(w) and con(ci) for word w and concepts $c_i \in \{c_1, ..., c_k\}$
- word-bag similarity sim(con(w), con(c)) based on cos or KL diff
- choose meaning argmax_c {sim(con(w), con(c))}

Mapping of concept-value query conditions onto Deep-Web portals:
~sheetmusic ~ „flute" → instrument = (flute | piccolo | recorder)
→ category = reeds
→ style = (classical | jazz | folk)

Benefit from Ontology Service

Ontology service accessible via SOAP or RMI
Ontology filled with WordNet, geo gazetteer, focused crawl results, extracted tables & forms

Threshold-based ...
substitute ~w by ...

Support for auto...
- heuristic rules (...
- named-entity re...

Query keyword d...
- contexts con(w) ...
- word-bag simila...
- choose meaning ...


```
<element name="WSF_Form0Select0_En
  <simpleType>
    <restriction base="string">
</restriction>
  </simpleType>
</element>
<simpleType
name="WSF_Form0Select1_Enum">
  <restriction base="string">
    <enumeration value=„Alternative"/
    <enumeration value=„Blues"/>
    <enumeration value=„Children's"/>
    <enumeration value=„Classical"/>
    <enumeration value=„Country"/>
    <enumeration value=„Folk"/>
    <enumeration value=„Jazz"/>
    ...
```

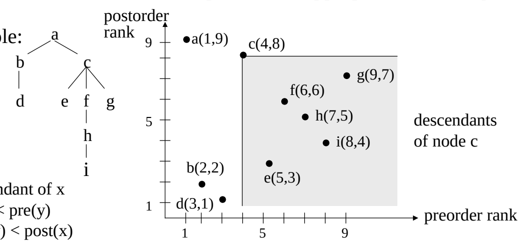

Mapping of concept-value query conditions onto Deep-Web portals:
~sheetmusic ~ „flute" → instrument = (flute | piccolo | recorder)
→ category = reeds
→ style = (classical | jazz | folk)

Index Structures for Path Expressions on Trees

Index based on Pre-/Postorder Numbering Scheme:
- Store with each element the
  - rank of the element in a preorder traversal of the doc tree and the
  - rank of the element in a postorder traversal.
- Build multidimensional index over pre-/postorder ranks (and element ids, element names, and nesting depths)

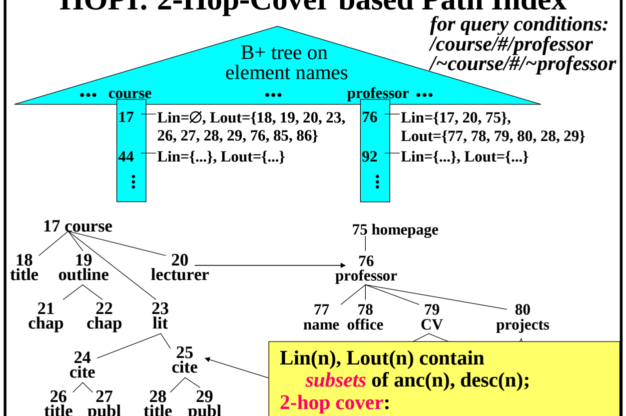
→ efficient eval. of XPath axes possible by appropriate window queries

Example:



y is descendant of x
iff pre(x) < pre(y)
and post(y) < post(x)

HOPI: 2-Hop-Cover based Path Index



Lin(n), Lout(n) contain
subsets **of anc(n), desc(n);**
2-hop cover:
m →* n ⇒ ∃x: x∈Lout(m) ∧ x ∈Lin(n)

Constructing a 2-Hop Cover

Definition (E. Cohen et al., SODA 2002):
a *2-hop cover* of a graph G=(V,E)
is a labeling (Lin, Lout) of all nodes where
1. **Lin(n) ⊆ {m | m→* n}, Lout(n) ⊆ {p | n→* p}, and**
2. **∀ (m,n) ∈E+ ∃ center node x with x∈Lout(m) ∧ x ∈Lin(n)**

Theorem (Cohen et al.):
The size of a 2-hop cover is $\Sigma_{n\in V}$ |Lin(n)| + |Lout(n)|.
Finding a minimal 2-hop cover is NP-complete.

Polynomial Algorithm with O(log ...
T' := E+ //uncovered connections;
while T' ≠∅ {
for each node n construct center graph
C(n) := {(m,p) | (m,n), (n,p) ∈ E+};
find node n with densest subgraph S(n) of C(n)∩T';
Lin(n) := sources of S(n); Lout(n) := sinks of S(n);
remove edges of S(n) from T' };

+ keep center graphs in priority queue
+ incrementally update center subgraphs
+ avoid complete transitive closure
+ support incremental updates

Efficient HOPI Construction

Divide-and-conquer:
- **Partition G by partitioning the XML document graph (using greedy heuristics) with node weights = #elems in doc & edge weights = #cross-doc links**
- **Compute 2-hop cover for each partition**
- **Merge covers:**
  for each cross-partition edge x → y with x∈P, y∈Q
  add x to Lout(a) for all a ∈ P with a →* x and
  to Lin(b) for all b ∈ P with y →* b

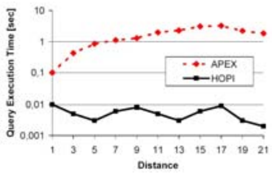
Implementation:
stores Lin and Lout sets in database tables
Lin (Id, InId) with indexes on *Id InId* and *InId Id*
Lout (Id, OutId) with indexes on *Id OutId* and *OutId Id*
Elems (Id, ElemName)
efficiently supports connection queries for all XPath axes on arbitrary XML data graphs

Experimental Results for HOPI

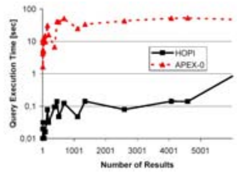
for DBLP data in XML:
419 000 docs with 5 Mio. elems
and 63 000 links
306 Mio. connections (2.4 GB)

with 53 partitions, HOPI has
27 Mio. connections (415 MB)

for query
//inproceedings[id=...]//author[id=...:]



for query
//inproceedings[id=...]//#



on synthetic data with Zipf-like indegrees and outdegrees
HOPI compresses TC by factor of 2 to 30

Towards more Efficient Query Processing for XXL

interpret conditions of the form
~course ~ „Internet“ and ~topic ~ „Performance“
as top-k queries with a score aggregation over
multiple index lists that come from
the ontology service and the content index

- apply Fagin-style top-k algorithms
  to retrieve best docs for similarity conditions
- then test exact-match conditions
  (needs pipelining for top-k evaluation)
- and identify result elements, paths, subgraphs

8.3 XSEarch

Data:
set of XML trees
with interior nodes having names and leaf nodes having contents
Queries:
set of generalized keywords of the form
name:content, name:, :content,
referring to element names and contents,
with each condition optionally having a + flag for mandatory matches

Key idea:
results should be semantically coherent tree fragments

Definition:
element e satisfies condition n:c if
e has label n and a descendant whose contents contains c

Example Data (1)

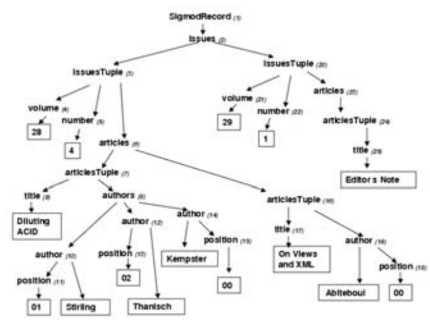


Figure 1: Part of the Sigmod Record document tree.

from: S. Cohen et al., XSEarch: a Semantic Search Engine for XML, VLDB Conference, 2003

Example Data (2)

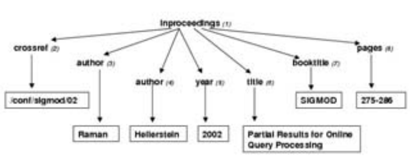


Figure 2: Part of the DBLP document tree.

from: S. Cohen et al., XSEarch: a Semantic Search Engine for XML, VLDB Conference, 2003

Query Answers

For two nodes n, n‘ in a tree the ***interconnection tree*** $T_{n,n'}$ consists of lca(n, n‘) as the root and the paths from lca(n,n‘) to n and n‘.

Nodes n, n‘ are ***meaningfully related,*** denoted n ≈ n‘, if $T_{n,n'}$
- does not contain two distinct nodes with the same name or
- the only two distinct nodes with the same name are n and n‘

A set N of nodes is
all-pairs related, denoted $\approx_a$(N), if n ≈ n‘ for all n,n‘∈N and
star-related, denoted $\approx_s$(N), if there is n*∈N s.t. n ≈ n* for all n ∈ N.

For a query with conditions c1 ... ck the sequence n1 ... nk of nodes and null values is an ***all-pairs answer*** if
- the non-null elements in {n1, ..., nk} are all-pairs related,
- ni is not the null-value if ci is a mandatory condition, and
- ni satisfies ci if it is not the null value.

A ***star-related answer*** is analogously defined.

An answer N‘ for a query q subsumes answer N
if N‘ is equal to N on all non-null elements.
N is a ***maximal answer*** if every N‘ that subsumes N is equal to N.

Query Answers

For two nodes n, n‘ in a tree the ***interconnection tree*** $T_{n,n'}$ consists of lca(n, n‘) as the root and the paths from lca(n,n‘) to n and n‘.

Nodes n, n‘ are ***meaningfully related,*** denoted n ≈ n‘, if $T_{n,n'}$
- does not contain two distinct nodes with the same name or
- the only two distinct nodes with the same name are n and n‘

A set N of nodes is
all-pairs related, denoted $\approx_a$(N), if n ≈ n‘ for all n,n‘∈N and
star-related, denoted $\approx_s$(N), if there is n*∈N s.t. n ≈ n* for all n ∈ N.

For a query with conditions c1 ... ck the sequence n1 ... nk of nodes and null values is an ***all-pairs answer*** if
- the non-null elements in {n1, ..., nk} are all-pairs related,
- ni is not the null-value if ci is a mandatory condition, and
- ni satisfies ci if it is not the null value.

A ***star-related answer*** is analogously defined.

An answer N‘ for a query q subsumes answer N if N‘ is equal to N on all non-null elements.
N is a ***maximal answer*** if every N‘ that subsumes N is equal to N.

Ranking Answers (1)

For query word w and leaf node n use tf*idf as a weight of node n.

For query word w and interior node n use the sum of weights over all leaf nodes below n.

For label l and node n use 1 as a weight if n‘s label is l, 0 otherwise.

Represent each node as an ***L*C-dimensional vector*** with the above weights as components, where L is # of all possible labels and C the # of distinct words.

For query word w and label l set ***query weight*** to tf*idf for condition l:w, 1.0 for condition :w, and user-specified (importance) weight for condition l: in all affected dimensions.

The ***similarity score for answer set N*** and query q, ***sim(q,N),*** is the sum, over all n∈N, of the cosine similarities between n and q.

Ranking Answers (2)

For answer set N to query q define
tsize(N) = # nodes in interconnection tree of N
ancdes(N) = # node pairs (n, n‘) in N
where n is ancestor of n‘ or vice versa

The total score of answer N to query q is:

$$\frac{sim(q,N)^{\alpha}}{tsize(N)^{\beta}} \cdot (1+\gamma\ ancdes(N))$$

with calibration parameters α, β, γ

Interrelationship Indexing

Goal: efficiently testing nodes n, n‘ if they are meaningfully related

Lemma:
If n is ancestor of n‘, then n ≈ n‘ iff
n ≈ parent(n‘) and label(n) ≠ label(parent(n‘)) and
child(n) ≈ n‘ and label(child(n)) ≠ label(n‘) .
If neither n is ancestor of n‘ nor vice versa, then n ≈ n‘ iff
n ≈ parent(n‘) and label(n) ≠ label(parent(n‘)) and
parent(n) ≈ n‘ and label(parent(n)) ≠ label(n‘).

Dynamic programming algorithm
on Boolean matrix interrel[1..#nodes] with depth-first node numbering:

```
for i := #nodes – 1 down to 0 { for j := i+1 to #nodes {
   if i is ancestor of j {
     let ch(i) be child of i on path to j and par(j) be parent of j;
     interrel[i,j] := interrel[ch(i),j] and label(ch(i)) != label(j) and
                      interrel[i,par(j)] and label(i) != label(par(j)); } } };
for i := 1 to #nodes – 1 { for j := i+1 to #nodes {
   if i is not an ancestor of j {
     let par(i) be parent of i and par(j) be parent of j;
     interrel[i,j] := interrel[par(i),j] and label(par(i)) != label(j) and
                      interrel[i,par(j)] and label(i) != label(par(j)); } } };
```


Experimental Results

on DBLP sample + SIGMOD Record data

queries: 1) q_{kw} = {+:Buneman, +:database}
2) q_{kw} = {+author:, +title:}
3) q_{kw+tag} = {+author:Buneman, +title:database}

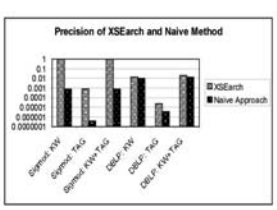


Figure 7: Precision of Q_{kw}, Q_{tag} and Q_{kw+tag} for Sigmod Record and DBLP.

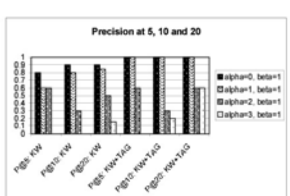


Figure 8: P@5, P@10 and P@20 of Q_{kw} and Q_{kw+tag} for DBLP.

8.4 XRank

Data: interlinked XML documents
Queries: simple keyword queries
Query results: ranked lists of elements

Key ideas:
- result ranking should consider
  - element-wise PageRank-style authorities
  - tree-node proximity of keyword-matching nodes
- query results are the most specific elements that have children with all keywords present

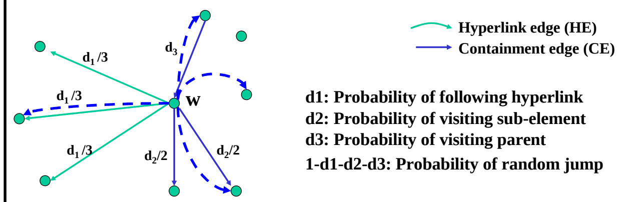
Node Proximity Aware Scoring

For query keyword w consider element e such that e' contains w and the path (within the tree) between e and e' has length t.
We define ***score(e, e', w) = score (e',w) * decay (e,e')***
with ***score(e',w) = ElemRank(e')*** if e' contains w, 0 otherwise,
decay(e, e') = δ^{t-1}, and calibration parameter δ, $0<\delta<1$.
If multiple e' exist that contain w then ***score(e,w) = max{score(e,e',w)}.***

For query with keywords w1, ..., wk
score(e, w1, ..., wk) $= (\sum_{i=1..k}$ ***score(e, wi)) * prox(e, w1, ..., wk)***
where
prox(e, w1, ..., wk) = size(smallest text window containing w1, ..., wk) $^{-1}$
measures the proximity of keywords in the linearized text below e

PageRank-Style Authority for Elements

ElemRank: define random walk on element graph and
compute stationary authorities using standard power iteration

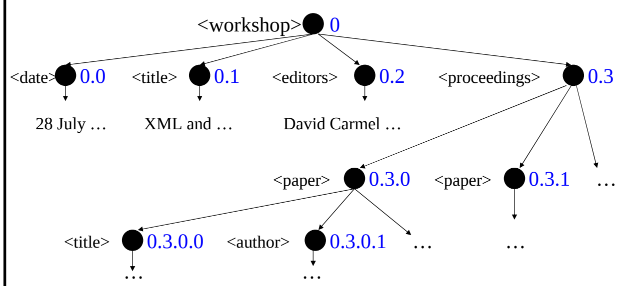


$$e(v) = d_1 \sum_{(u,v)\in HE} \frac{e(u)}{N_h(u)} + d_2 \sum_{(u,v)\in CE} \frac{e(u)}{N_c(u)} + d_3 \sum_{(u,v)\in CE^{-1}} e(u) + \frac{1-d_1-d_2-d_3}{N_d \times N_{de}(v)}$$

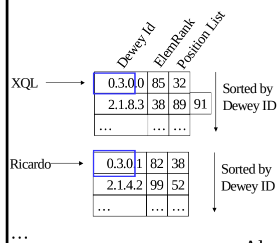
from: Lin Guo's presentation of Guo et al.: XRANK: Ranked Keyword Search over XML Documents, SIGMOD 2003

Dewey Numbering Scheme

Dewey decimal numbering for elements:
ancestor's ID is prefix of descendant's ID



Dewey Inverted Lists (DIL) Indexing



Challenges for Query Processing:

- Merge multiple inverted lists
  - Not equality merge-join
- How to suppress spurious results
  - Most specific results
- Capture the 2-dimensional proximity

Algorithm that addresses above issues
in a ***single scan*** over inverted lists

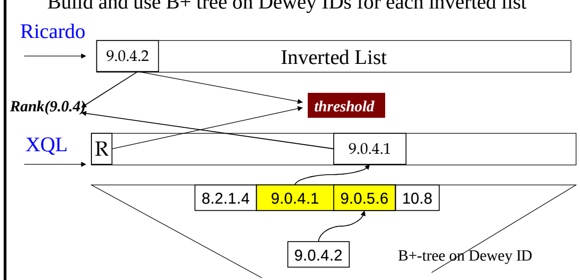
Query Processing with DIL

- Dewey Inverted Lists (DIL) are sorted by Dewey ID order
- Goal: find top *k* results by a ***single scan*** over inverted lists
- Algorithm
- Merge query keyword inverted lists in Dewey ID order
  - Ensures that entries with common prefixes are processed together
- Compute Longest Common Prefix (LCP) of Dewey IDs
  - LCP ensures most specific results

alternative: sort index lists by ElemRank
→ ***Ranked Dewey Inverted Lists (RDIL)***

Query Processing with RDIL

1. How to find the most specific result?
   Build and use B+ tree on Dewey IDs for each inverted list



2. When to stop scanning inverted lists?
   Extend Fagin's TA:
   threshold = aggregate score upper bounds
   with decay and prox set to 1.0.

Literature

- Anja Theobald, Gerhard Weikum: Adding Relevance to XML, 3rd International Workshop on Web and Databases (WebDB), 2000, LNCS 1997, Springer.
- Ralf Schenkel, Anja Theobald, Gerhard Weikum: Ontology-enabled XML Search, in: H. Blanken et al. (Eds.), Intelligent Search on XML Data, LNCS 2818, Springer, 2003.
- Ralf Schenkel, Anja Theobald, Gerhard Weikum: HOPI: An Efficient Connection Index for Complex XML Document Collections, EDBT Conference, 2004.
- Sara Cohen, Jonathan Mamou, Yaron Kanza, Yehoshua Sagiv: XSEarch: A Semantic Search Engine for XML, VLDB Conf., 2003.
- Lin Guo, Feng Shao, Chavdar Botev, Jayavel Shanmugasundaram: XRank: Ranked Keyword Search over XML Documents, SIGMOD Conf., 2003.
- Sihem Amer-Yahia, SungRan Cho, Divesh Srivastava: Tree Pattern Relaxation, EDBT Conference, 2002.