CS276B

Text Retrieval and Mining

Winter 2005

Lecture 15

Plan for today

- Vector space approaches to XML retrieval
- Evaluating text-centric retrieval

Text-centric XML retrieval

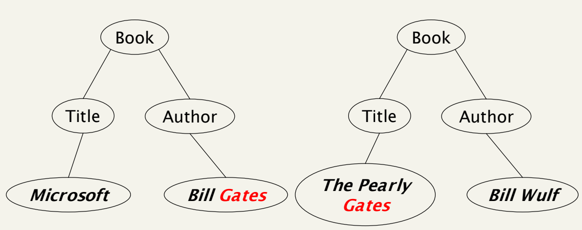
- Documents marked up as XML
  - E.g., assembly manuals, journal issues …
- Queries are user information needs
  - E.g., give me the Section (element) of the document that tells me how to change a brake light
- Different from well-structured XML queries where you tightly specify what you're looking for.

Vector spaces and XML

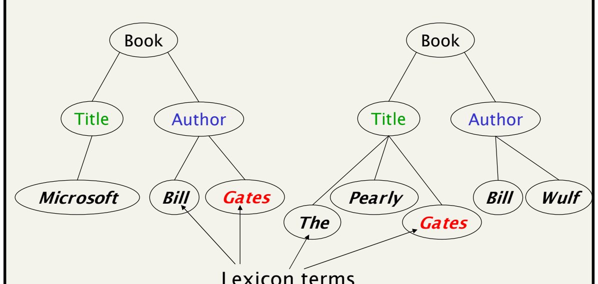
- Vector spaces – tried+tested framework for keyword retrieval
  - Other "bag of words" applications in text: classification, clustering …
- For text-centric XML retrieval, can we make use of vector space ideas?
- Challenge: capture the structure of an XML document in the vector space.

Vector spaces and XML

- For instance, distinguish between the following two cases



Content-rich XML: representation

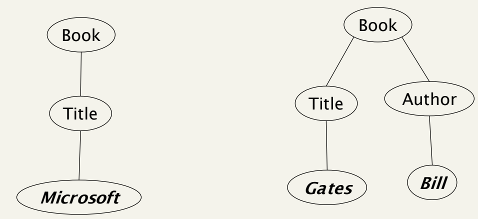


Encoding the Gates differently

- What are the axes of the vector space?
- In text retrieval, there would be a single axis for Gates
- Here we must separate out the two occurrences, under Author and Title
- Thus, axes must represent not only terms, but something about their position in an XML tree

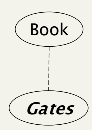
Queries

- Before addressing this, let us consider the kinds of queries we want to handle



Query types

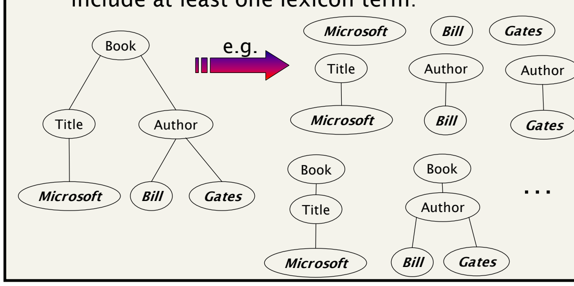
- The preceding examples can be viewed as subtrees of the document
- But what about?



- (***Gates*** somewhere underneath Book)
- This is harder and we will return to it later.

Subtrees and structure

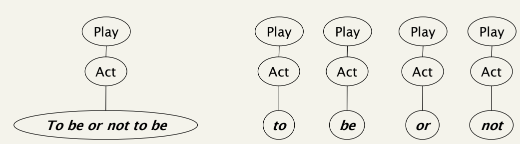
- Consider all subtrees of the document that include at least one lexicon term:



Structural terms

- Call each of the resulting (8+, in the previous slide) subtrees a *structural term*
- Note that structural terms might occur multiple times in a document
- Create one axis in the vector space for each distinct structural term
- Weights based on frequencies for number of occurrences (just as we had *tf*)
- All the usual issues with terms (stemming? Case folding?) remain

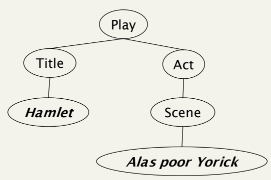
Example of *tf* weighting



- Here the structural terms containing ***to*** or ***be*** would have more weight than those that don't

Exercise: How many axes are there in this example?

Down-weighting



- For the doc on the left: in a structural term rooted at the node Play, shouldn't ***Hamlet*** have a higher *tf* weight than ***Yorick***?
- Idea: multiply *tf* contribution of a term to a node *k* levels up by γ^k, for some $\gamma < 1$.

Down-weighting example, γ=0.8

- For the doc on the previous slide, the *tf* of
  - ***Hamlet*** is multiplied by 0.8
  - ***Yorick*** is multiplied by 0.64

in any structural term rooted at Play.

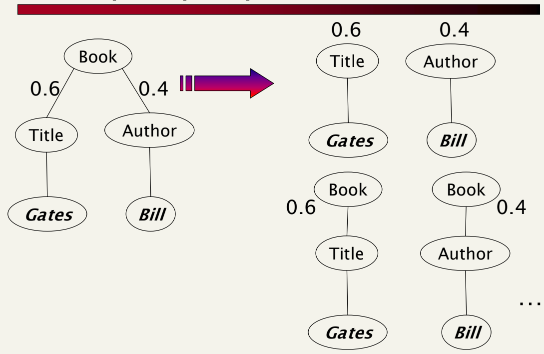
The number of structural terms

- Can be huge! ← Alright, how huge, really?
- Impractical to build a vector space index with so many dimensions
- Will examine pragmatic solutions to this shortly; for now, continue to believe …

Structural terms: docs+queries

- The notion of structural terms is independent of any schema/DTD for the XML documents
- Well-suited to a heterogeneous collection of XML documents
- Each document becomes a vector in the space of structural terms
- A query tree can likewise be factored into structural terms
  - And represented as a vector
  - Allows weighting portions of the query

Example query



Weight propagation

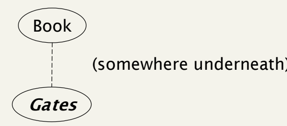
- The assignment of the weights 0.6 and 0.4 in the previous example to subtrees was simplistic
  - Can be more sophisticated
  - Think of it as generated by an application, not necessarily an end-user
- Queries, documents become normalized vectors
- Retrieval score computation "just" a matter of cosine similarity computation

Restrict structural terms?

- Depending on the application, we may restrict the structural terms
- E.g., may never want to return a Title node, only Book or Play nodes
- So don't enumerate/index/retrieve/score structural terms rooted at some nodes

The catch remains

- This is all very promising, but …
- How big is this vector space?
- Can be exponentially large in the size of the document
- Cannot hope to build such an index
- And in any case, still fails to answer queries like



Two solutions

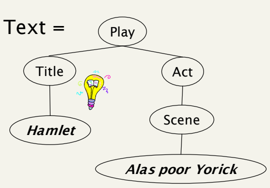
- Query-time materialization of axes
- Restrict the kinds of subtrees to a manageable set

Query-time materialization

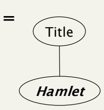
- Instead of enumerating all structural terms of all docs (and the query), enumerate only for the query
  - The latter is hopefully a small set
- Now, we're reduced to checking which structural term(s) from the query match a subtree of any document
- This is tree pattern matching: given a *text tree* and a *pattern tree*, find matches
  - Except we have many text trees
  - Our trees are labeled and weighted

Example

Text =



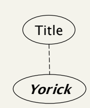
Query =



- Here we seek a doc with ***Hamlet*** in the title
- On finding the match we compute the cosine similarity score
- After all matches are found, rank by sorting

(Still infeasible)

- A doc with ***Yorick*** somewhere in it:
- Query =

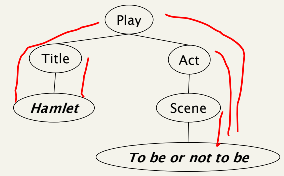


- Will get to it …

Restricting the subtrees

- Enumerating all structural terms (subtrees) is prohibitive, for indexing
  - Most subtrees may never be used in processing any query
- Can we get away with indexing a restricted class of subtrees
  - Ideally – focus on subtrees likely to arise in queries

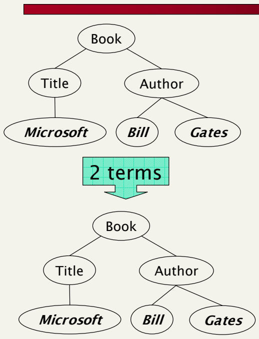
JuruXML (IBM Haifa)



- Only paths including a lexicon term
- In this example there are only 14 (why?) such paths
- Thus we have 14 structural terms in the index

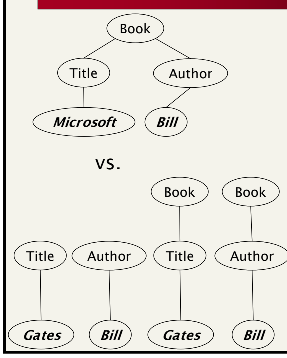
Why is this far more manageable?
How big can the index be as a function of the text?

Variations



- Could have used other subtrees – e.g., all subtrees with two siblings under a node
- Which subtrees get used: depends on the likely queries in the application
- Could be specified at index time – area with little research so far

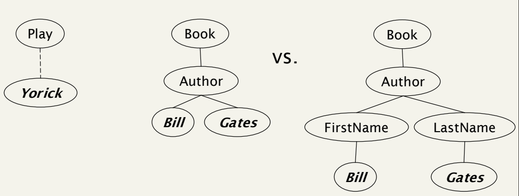
Variations



- Why would this be any different from just paths?
- Because we preserve more of the structure that a query may seek

Descendants

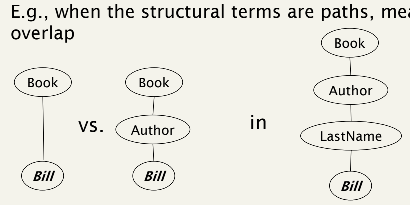
- Return to the descendant examples:



No known DTD.
Query seeks ***Gates*** under Author.

Handling descendants in the vector space

- Devise a *match* function that yields a score in [0,1] between structural terms
- E.g., when the structural terms are paths, measure overlap

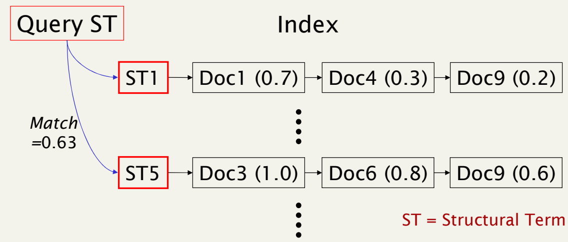


- The greater the overlap, the higher the *match* score
  - Can adjust *match* for where the overlap occurs

How do we use this in retrieval?

- First enumerate structural terms in the query
- Measure each for *match* against the dictionary of structural terms
  - Just like a postings lookup, except not Boolean (does the term exist)
  - Instead, produce a score that says "80% close to this structural term", etc.
- Then, retrieve docs with that structural term, compute cosine similarities, etc.

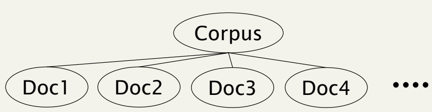
Example of a retrieval step



Now rank the Doc's by cosine similarity;
e.g., Doc9 scores 0.578.

Closing technicalities

- But what exactly is a Doc?
- In a sense, an entire corpus can be viewed as an XML document



What are the Doc's in the index?

- Anything we are prepared to return as an answer
- Could be nodes, some of their children …

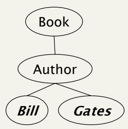
What are queries we can't handle using vector spaces?

- Find figures that describe the Corba architecture and the paragraphs that refer to those figures
  - Requires JOIN between 2 tables
- Retrieve the titles of articles published in the Special Feature section of the journal *IEEE Micro*
  - Depends on order of sibling nodes.

Can we do IDF?

- Yes, but doesn't make sense to do it corpus-wide
- Can do it, for instance, within all text under a certain element name say Chapter
- Yields a *tf-idf* weight for each lexicon term under an element
- Issues: how do we propagate contributions to higher level nodes.

Example



- Say ***Gates*** has high IDF under the Author element
- How should it be *tf-idf* weighted for the Book element?
- Should we use the *idf* for ***Gates*** in Author or that in Book?

INEX: a benchmark for text-centric XML retrieval

INEX

- Benchmark for the evaluation of XML retrieval
  - Analog of TREC (recall CS276A)
- Consists of:
  - Set of XML documents
  - Collection of retrieval tasks

INEX

- Each engine indexes docs
- Engine team converts retrieval tasks into queries
  - In XML query language understood by engine
- In response, the engine retrieves not docs, but <u>elements within docs</u>
  - Engine ranks retrieved elements

INEX assessment

- For each query, each retrieved element is human-assessed on two measures:
  - <u>Relevance</u> – how relevant is the retrieved element
  - <u>Coverage</u> – is the retrieved element too specific, too general, or just right
    - E.g., if the query seeks a definition of the Fast Fourier Transform, do I get the equation (too specific), the chapter containing the definition (too general) or the definition itself
- These assessments are turned into composite precision/recall measures

INEX corpus

- 12,107 articles from IEEE Computer Society publications
- 494 Megabytes
- Average article:1,532 XML nodes
  - Average node depth = 6.9

INEX topics

- Each topic is an information need, one of two kinds:
  - Content Only (CO) – free text queries
  - Content and Structure (CAS) – explicit structural constraints, e.g., containment conditions.

Sample INEX CO topic

<**Title**> computational biology </**Title**>

<**Keywords**> computational biology, bioinformatics, genome, genomics, proteomics, sequencing, protein folding </**Keywords**>

<**Description**> Challenges that arise, and approaches being explored, in the interdisciplinary field of computational biology</**Description**>

<**Narrative**> To be relevant, a document/component must either talk in general terms about the opportunities at the intersection of computer science and biology, or describe a particular problem and the ways it is being attacked. </**Narrative**>

INEX assessment

- Each engine formulates the topic as a query
  - E.g., use the keywords listed in the topic.
- Engine retrieves one or more elements and ranks them.
- Human evaluators assign to each retrieved element <u>relevance</u> and <u>coverage</u> scores.

Assessments

- Relevance assessed on a scale from Irrelevant (scoring 0) to Highly Relevant (scoring 3)
- Coverage assessed on a scale with four levels:
  - No Coverage (N: the query topic does not match anything in the element
  - Too Large (The topic is only a minor theme of the element retrieved)
  - Too Small (S: the element is too small to provide the information required)
  - Exact (E).
- So every element returned by each engine has ratings from {0,1,2,3} ¾{N,S,L,E}

Combining the assessments

- Define scores:

$$f_{strict}(rel, cov) = \begin{cases} 1 & \text{if } rel, cov = 3E \\ 0 & \text{otherwise} \end{cases}$$

$$f_{generalized}(rel, \text{cov}) = \begin{cases} 1.00 & \text{if } rel,cov = 3E \\ 0.75 & \text{if } rel,cov \in \{2E,3L,3S\} \\ 0.50 & \text{if } rel,cov \in \{1E,2L,2S\} \\ 0.25 & \text{if } rel,cov \in \{1S,1L\} \\ 0.00 & \text{if } rel,cov = 0N. \end{cases}$$

The *f*-values

- Scalar measure of goodness of a retrieved elements
- Can compute *f*-values for varying numbers of retrieved elements 10, 20 … etc.
  - Means for comparing engines.

From raw *f*-values to … ?

- INEX provides a method for turning these into precision-recall curves
- "Standard" issue: only elements returned by some participant engine are assessed
- Lots more commentary (and proceedings from previous INEX bakeoffs):
  - http://inex.is.informatik.uni-duisburg.de:2004/
  - See also previous years

Resources

- **Querying and Ranking XML Documents**
  - Torsten Schlieder, Holger Meuss
  - http://citeseer.ist.psu.edu/484073.html
- **Generating Vector Spaces On-the-fly** for Flexible XML Retrieval.
  - T. Grabs, H-J Schek
  - www.cs.huji.ac.il/course/2003/sdbi/Papers/ir-xml/xmlirws.pdf

Resources

- JuruXML - an XML retrieval system at INEX'02.
  - Y. Mass, M. Mandelbrod, E. Amitay, A. Soffer.
  - http://einat.webir.org/INEX02_p43_Mass_etal.pdf
- See also INEX proceedings online.