Text Search for Fine-grained Semi-structured Data

Soumen Chakrabarti
Indian Institute of Technology, Bombay
www.cse.iitb.ac.in/~soumen/

Acknowledgments

<table>
<thead>
<tr>
<th>Acknowledgments</th>
</tr>
</thead>
<tbody>
<tr>
<td>S. Sudarshan</td>
</tr>
<tr>
<td>B. Aditya</td>
</tr>
</tbody>
</table>

Two extreme search paradigms

Searching a RDBMS

- Complex data model: tables, rows, columns, data types
- Expressive, powerful query language
- Need to know schema to query
- Answer = unordered set of rows
- Ranking: afterthought

Information Retrieval

- Collection = set of documents, document = sequence of terms
- Terms and phrases present or absent
- No (nontrivial) schema to learn
- Answer = sequence of documents
- Ranking: central to IR
Convergence?

SQL→XML search
- Trees, reference links
- Labeled edges
- Nodes may contain
  - Structured data
  - Free text fields
- Data vs. document
- Query involves node data and edge labels
  - Partial knowledge of schema ok
- Answer = set of paths

Web search←IR
- Documents are nodes in a graph
- Hyperlink edges have important but unspecified semantics
  - Google, HITS
- Query language remains primitive
  - No data types
  - No use of tag-tree
- Answer = URL list

Outline of this tutorial

- Review of text indexing and information retrieval (IR)
- Support for text search and similarity join in relational databases with text columns
- Text search features in major XML query languages (and what’s missing)
- A graph model for semi-structured data with “free-form” text in nodes
- Proximity search formulations and techniques; how to rank responses
- Folding in user feedback
- Trends and research problems
Text indexing basics

- “Inverted index” maps from term to document IDs
- Term offset info enables phrase and proximity (“near”) searches
- Document boundary and limitations of “near” queries
- Can extend inverted index to map terms to
  - Table names, column names
  - Primary keys, RIDs
  - XML DOM node IDs

Information retrieval basics

- Stopwords and stemming
- Each term $t$ in lexicon gets a dimension in vector space
- Documents and the query are vectors in term space
- Component of $d$ along axis $t$ is $\text{TF}(d,t)$
  - Absolute term count or scaled by max term count
- Downplay frequent terms: $\text{IDF}(t) = \log(1 + |D|/|D_t|)$
  - Better model: document vector $d$ has component $\text{TF}(d,t) \times \text{IDF}(t)$ for term $t$
- Query is like another “document”; documents ranked by cosine similarity with query
<table>
<thead>
<tr>
<th>IR support</th>
<th>Data model</th>
<th>SQL, Datalog</th>
<th>XML-QL, Xquery</th>
<th>WHIRL</th>
<th>ELIXIR, XIRQL</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>Relational</td>
<td>XML-like</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Schema</td>
<td>WHIRL</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No schema</td>
<td>DBXplorer, BANKS, DISCOVER</td>
<td>EasyAsk, Mercado, DataSpot, BANKS</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- “None” = nothing more than string equality, containment (substring), and perhaps lexicographic ordering
- “Schema”: Extensions to query languages, user needs to know data schema, IR-like ranking schemes, no implicit joins
- “No schema”: Keyword queries, implicit joins

**WHIRL (Cohen 1998)**

- `place(univ,state) and job(univ,dept)`
- Ranked retrieval from a RDBMS:
  - `select univ from job where dept ~ 'Civil'`
- Ranked similarity join on text columns:
  - `select state, dept from place, job where place.univ ~ job.univ`
- Limit answer to best $k$ matches only
- Avoid evaluating full Cartesian product
  - “Iceberg” query
- Useful for data cleaning and integration
WHIRL scoring function

A where-clause in WHIRL is a

- Boolean predicate as in SQL (age=35)
  - Score for such clauses are 0/1
- Similarity predicate (job ~ 'Web design')
  - Score = cosine(job, 'Web design')
- Conjunction or disjunction of clauses
  - Sub-clause scores interpreted as probabilities
  - $\text{score}(B_1 \land ... \land B_m; \theta) = \Pi_{1 \leq i \leq m} \text{score}(B_i; \theta)$
  - $\text{score}(B_1 \lor ... \lor B_m; \theta) = 1 - \Pi_{1 \leq i \leq m} (1 - \text{score}(B_i; \theta))$

Query execution strategy

```
select state, dept from place, job
where place.univ ~ job.univ
```

- Start with place(U1,S) and job(U2,D)
  where U1, U2, S and D are “free”
  - Any binding of these variables to constants is associated with a score
- Greedily extend the current bindings for maximum gain in score
- Backtrack to find more solutions
XQuery

- Quilt + Lorel + YATL + XML-QL
- Path expressions

```
<dishes_with_flour> { FOR $r IN document("recipes.xml")
    //recipe[//ingredient[@name="flour"]]
    RETURN <dish>{$r/title/text()}</dish> }
</dishes_with_flour>
```

Early text support in XQuery

- Title of books containing some para mentioning both “sailing” and “windsurfing”

  ```
  FOR $b IN document("bib.xml")//book
  WHERE SOME $p IN $b//paragraph SATISFIES
  (contains($p,"sailing") AND
   contains($p,"windsurfing"))
  RETURN $b/title
  ```

- Title and text of documents containing at least three occurrences of “stocks”

  ```
  FOR $a IN view("text_table") WHERE
  numMatches($a/text_document,"stocks") > 3
  RETURN
  <text>{$a/text_title}{$a/text_document}</text>
  ```
## Tutorial outline

<table>
<thead>
<tr>
<th>IR support</th>
<th>Data model</th>
<th>XML-like</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>SQL, Datalog</td>
<td>XML-QL, XQuery</td>
</tr>
<tr>
<td>Schema</td>
<td>WHIRL</td>
<td>ELIXIR, XIRQL</td>
</tr>
<tr>
<td>No schema</td>
<td>DBXplorer, BANKS, DISCOVER</td>
<td>EasyAsk, Mercado, DataSpot, BANKS</td>
</tr>
</tbody>
</table>

- Review of text indexing and information retrieval
- Support for text search and similarity join in relational databases with text columns (WHIRL)
- Adding IR-like text search features to XML query languages (Chinenyanga et al. Führ et al. 2001)

### ELIXIR: Adding IR to XQuery

- Ranked select
  
  ```xml
  for $t in document("db.xml")/items/(book|cd) where $t/text() ~ "Ukrainian recipe"
  return <dish>$t</dish>
  ```

- Ranked similarity join: find titles in recent VLDB proceedings similar to speeches in Macbeth
  
  ```xml
  for $vi in document("vldb.xml")/issue[@volume>24], $si in document("macbeth.xml")/speech
  where $vi//article/title ~ $si
  return <similar><title>$vi//article/title</title>
  <speech>$si</speech></similar>
  ```
How ELIXIR works

ELIXIR query

VLDB.xml

Macbeth.xml

Base XML documents

XQuery filters/
transformers

ELIXIR Compiler

Flatten to WHIRL

WHIRL select/join filters

Rewrite to XML

Result

A more detailed view

VLDB.xml

<issue><volume>10</volume>
<article>...</article>
<issue><volume>23</volume>
<article><title:size separation spatial join</title></article></issue>

Macbeth.xml

<act number="...">
<scene number="...">
<speech>To Ireland, I; our separated fortune.</speech>
</scene>
</act>

q21.xml

<q21> { for $t in document("VLDB.xml")//issue [volume > 24]/title return <tuple:title>{$t}; </tuple> }</q21>

q22.xml

<q22> { for $a in document("Macbeth.xml")//act/scene/speech return <tuple;line>{$a}; </tuple> }</q22>

q21:<tuple><title:size separation spatial join</title></tuple>

q22:<tuple><line>To Ireland, I; our separated fortune.
</line></tuple>

WHIRL query

<q3($title,$line) := q21($title), q22($line), $title ~ $line>

<similar { for $row in q3/tuple return $row }/>

Result
Observations

- SQL/XQuery + IR-like result ranking
- Schema knowledge remains essential
  - “Free-form” text vs. tagged, typed field
  - Element hierarchy, element names, IDREFs
- Typical Web search is two words long
  - End-users don’t type SQL or XQuery
  - Possible remedy: HTML form access
  - Limitation: restricted views and queries

Using proximity without schema

- General, detailed representation: XML
- Lowest common representation
  - Collection, document, terms
  - Document = node, hyperlink = edge
- Middle ground
  - Graph with text (or structured data) in nodes
  - Links: element, subpart, IDREF, foreign keys
  - All links hint at unspecified notion of proximity

Exploit structure where available, but do not impose structure by fiat
Two paradigms of proximity search

- A single node as query response
  - Find node that matches query terms...
  - ...or is “near” nodes matching query terms
    (Goldman et al., 1998)
- A connected subgraph as query response
  - Single node may not match all keywords
  - No natural “page boundary”

Single-node response examples

- Travolta, Cage
  - Actor, Face/Off
- Travolta, Cage, Movie
  - Face/Off
- Kleiser, Movie
  - Gathering, Grease
- Kleiser, Woo, Actor
  - Travolta
Basic search strategy

- Node subset $A$ activated because they match query keyword(s)
- Look for node near nodes that are activated
- Goodness of response node depends
  - Directly on degree of activation
  - Inversely on distance from activated node(s)

Ranking a single node response

- Activated node set $A$
- Rank node $r$ in “response set” $R$ based on proximity to nodes $a$ in $A$
  - Nodes have relevance $\rho_R$ and $\rho_A$ in $[0,1]$
  - Edge costs are “specified by the system”
- $d(a,r) =$ cost of shortest path from $a$ to $r$
- Bond between $a$ and $r$
  $$b(a,r) = \frac{\rho_A(a)\rho_R(r)}{d(a,r)^t}$$
- Parameter $t$ tunes relative emphasis on distance and relevance score
- Several ad-hoc choices
Scoring single response nodes

- **Additive**
  \[ \text{score}(r) = \sum_{a \in A} b(a, r) \]

- **Belief**
  \[ \text{score}(r) = 1 - \prod_{a \in A} (1 - b(a, r)) \]

- **Goal:** list a limited number of find nodes with the largest scores

- **Performance issues**
  - Assume the graph is in memory?
  - Precompute all-pairs shortest path (\(|V|^3\))?  
  - Prune unpromising candidates?

---

Hub indexing

- Decompose APSP problem using sparse vertex cuts  
  - \(|A|+|B|\) shortest paths to \(p\)
  - \(|A|+|B|\) shortest paths to \(q\)
  - \(d(p, q)\)

- To find \(d(a, b)\) compare
  - \(d(a \rightarrow p \rightarrow b)\) not through \(q\)
  - \(d(a \rightarrow q \rightarrow b)\) not through \(p\)
  - \(d(a \rightarrow p \rightarrow q \rightarrow b)\)
  - \(d(a \rightarrow q \rightarrow p \rightarrow b)\)

- Greatest savings when \(|A| \approx |B|\)

- Heuristics to find cuts, e.g. large-degree nodes
Connected subgraph as response

- Single node may not match all keywords
- No natural “page boundary”
- Two scenarios
  - Keyword search on relational data
    - Keywords spread among normalized relations
  - Keyword search on XML-like or Web data
    - Keywords spread among DOM nodes and subtrees

### Tutorial outline

<table>
<thead>
<tr>
<th>IR support</th>
<th>Data model</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Relational</td>
</tr>
<tr>
<td>None</td>
<td>SQL, Datalog</td>
</tr>
<tr>
<td>Schema</td>
<td>WHIRI</td>
</tr>
<tr>
<td>No schema</td>
<td>DBXplorer,</td>
</tr>
<tr>
<td></td>
<td>BANKS, DISCOVER</td>
</tr>
</tbody>
</table>

- Adding IR-like text search features to XML query languages
- A graph model for relational data with “free-form” text search and implicit joins
- Generalizing to graph models for XML
Keyword search on relational data

- Tuple = node
- Some columns have text
- Foreign key constraints = edges in schema graph
- Query = set of terms
- No natural notion of a document
  - Normalization
  - Join may be needed to generate results
  - Cycles may exist in schema graph: ‘Cites’

DBXplorer and DISCOVER

- Enumerate subsets of relations in schema graph which, when joined, may contain rows which have all keywords in the query
  - “Join trees” derived from schema graph
- Output SQL query for each join tree
- Generate joins, checking rows for matches
  (Agrawal et al. 2001, Hristidis et al. 2002)
Discussion

- Exploits relational schema information to contain search
- Pushes final extraction of joined tuples into RDBMS
- Faster than dealing with full data graph directly
- Coarse-grained ranking based on schema tree
- Does not model proximity or (dis) similarity of individual tuples
- No recipe for data with less regular (e.g. XML) or ill-defined schema

Generalized graph proximity

- General data graph
  - Nodes have text, can be scored against query
  - Edge weights express dissimilarity
- Query is a set of keywords as before
- Response is a connected subgraph of the database
- Each response graph is scored using
  - Node weights which reflect match, maximize
  - Edge weights which reflect lack of proximity, minimize
Motivation from Web search

- “Linux modem driver for a Thinkpad A22p”
  - Hyperlink path matches query collectively
  - Conjunction query would fail
- Projects where X and P work together
  - Conjunction may retrieve wrong page
- General notion of graph proximity

“Information unit” (Lee et al., 2001)

- Generalizes join trees to arbitrary graph data
- Connected subgraph of data without cycles
- Includes at least one node containing each query keyword
- Edge weights represent price to pay to connect all keyword-matching nodes together
- May have to include non-matching nodes
Setting edge weights

- Edges are generally directed
  - Foreign to primary key in relational data
  - Containing to contained element in XML
  - IDREFs have clear source and target
- Consider the RDMS scenario
- Forward edge weight for edge \((u, v)\)
  - \(u, v\) are tuples in tables \(R(u), R(v)\)
  - Weight \(s(R(u), R(v))\) between tables
    - Configured heuristically based on semantics
    - \(w_p(u, v) = s(R(u), R(v))\) all such tuple pairs \(u, v\)
- Proximity search must traverse edges in both directions ... what should \(w_B(u, v)\) be?

Backward edge weights

- “Distance” between a pair of nodes is asymmetric in general
  - Ted Raymond acted only in The Truman Show, which is 1 of 55 movies for Jim Carrey
  - \(w(e_1)\) should be larger than \(w(e_2)\) (think “resistance” on the edge)
- For every edge \((u, v)\) that exists, 
  \(w_B(u, v) = s(R(v), R(u)) \cdot \text{IN}_v(u)\)
  - \(\text{IN}_v(u)\) is the #edges from \(R(v)\) to \(u\)
- \(w(u, v) = \min\{w_F(u, v), w_B(u, v)\}\)
- More general edge weight models possible, e.g., \(R \rightarrow S \rightarrow T\) relation path-based weights
Node weight = relevance + prestige

- Relevance w.r.t. keyword(s)
  - 0/1: node contains term or it does not
  - Cosine score in [0,1] as in IR
  - Uniform model: a node for each keyword (e.g. DataSpot)

- Popularity or prestige
  - E.g. “mohan transaction”
  - Indegree
  - PageRank

\[ p(v) = \frac{d}{N} + (1 - d) \sum_{u \rightarrow v} \text{OutDegree}(u) \]

Trading off node and edge weights

- A high-scoring answer \( A \) should have
  - Large node weight
  - Small edge weight

- Weights must be normalized to extreme values
- \( N(v) \) = node weight of \( v \)
- Overall NodeScore = \( \sum_{v \in A} \log\left(1 + \frac{N(v)}{N_{\text{max}}}ight) \)

- Overall EdgeScore = \( \frac{1}{1 + \sum_{e \in A} \log\left(1 + \frac{w(e)}{w_{\text{min}}}ight)} \)

- Overall score = \( \text{EdgeScore} \times \text{NodeScore}^\lambda \)
  - \( \lambda \) tunes relative contribution of nodes and edges

- Ad-hoc, but guided by heuristic choices in IR
Data structures for search

- Answer = tree with at least one leaf containing each keyword in query
  - Group Steiner tree problem, NP-hard
- Query term $t$ found in source nodes $S_t$
- Single-source-shortest-path SSSP iterator
  - Initialize with a source (near-) node
  - Consider edges backwards
  - getNext() returns next nearest node
- For each iterator, each visited node $\nu$ maintains for each $t$ a set $\nu.R_t$ of nodes in $S_t$ which have reached $\nu$

Generic expanding search

- Near node sets $S_t$ with $S = \bigcup_t S_t$
- For all source nodes $\sigma \in S$
  - create a SSSP iterator with source $\sigma$
- While more results required
  - Get next iterator and its next-nearest node $\nu$
  - Let $t$ be the term for the iterator’s source $s$
  - $\text{crossProduct} = \{s\} \times \Pi_{t \neq t} \nu.R_t$
  - For each tuple of nodes in $\text{crossProduct}$
    - Create an answer tree rooted at $\nu$ with paths to each source node in the tuple
  - Add $s$ to $\nu.R_t$
Search example ("Vu Kleinberg")

First response
Folding in user feedback

- As in IR systems, results may be imperfect
  - Unlike SQL or XQuery, no exact control over matching, ranking and answer graph form
  - Ad-hoc choices for node and edge weights

- Per-user and/or per-session
  - By graph/path/node type, e.g. “want author citing author,” not “author coauthoring with author”

- Across users
  - Modifying edge costs to favor nodes (or node types) liked by users

Random walk formulations

- Generalize PageRank to treat outlinks differently
  - $\tau(u,v)$ is the “conductance” of edge $u \rightarrow v$

- $p(v)$ is a function of $\tau(u,v)$ for all in-neighbors $u$ of $v$
  - $p_{\text{guess}}(v)$ ... at convergence
  - $p_{\text{user}}(v)$ ... user feedback

Gradient ascent/descent:

- For each $u \rightarrow v$, set (with learning rate $\eta$):
  $$\tau(u,v) \leftarrow \tau(u,v) + \eta \text{sgn}(p_{\text{user}}(v) - p_{\text{guess}}(v)) \frac{p(u)}{\sum_{u' \rightarrow v} p(u')}$$

- Re-iterate to convergence
Prototypes and products

- DTL DataSpot → Mercado Intuifind
  www.mercado.com/
- EasyAsk www.easyask.com/
- ELIXIR www.smi.ucd.ie/elixir/
- XIRQL ls6-www.informatik.uni-dortmund.de/ir/projects/hyrex/
- Microsoft DBXplorer
- BANKS www.cse.iitb.ac.in/banks/

Summary

- Confluence of structured and free-format, keyword-based search
  - Extend SQL, XQuery, Web search, IR
  - Many useful applications: product catalogs, software libraries, Web search
- Key idiom: proximity in a graph representation of textual data
  - Implicit joins on foreign keys
  - Proximity via IDREF and other links
- Several working systems
- Not enough consensus on clean models
Open problems

- Simple, clean principles for setting weights
  - Node/edge scoring ad-hoc
  - Contrast with classification and distillation
- Iceberg queries
  - Incremental answer generation heuristics do not capture bicriteria nature of cost
- Aggregation: how to express / execute
- User interaction and query refinement
- Advanced applications
  - Web query, multipage knowledge extraction
  - Linguistic connections through WordNet

Selected references

Selected references