Keyword Search in Databases

Wei Wang

University of New South Wales, Australia
Outline

- Introduction
- IR Preliminaries
- Systems
- Open Issues
Search Without Boundary

- Example:
  - Fail to locate “World Wide Web” in the abstract of this talk
  - Should search within the scope of a “logical unit”

- Non-trivial to support such search
  - Regular expressions are hard to learn/use
  - Only works for linear layout of texts
  - Scale to disk-resident data

- Other issues that matter!
  - Ranking
  - Efficiency

Think of Google for examples …
Keyword Search in RDMBSs

What are the “logical units” in RDBMSs?

Query: “netvista maxtor”

<table>
<thead>
<tr>
<th>prodID</th>
<th>custID</th>
<th>Date</th>
<th>Comment</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>P121</td>
<td>C2332</td>
<td>6-30-2002</td>
<td>“disk crashed after ... use on an IBM Netvista X41”</td>
<td>“Change Maxtor HD to Seagate HD”</td>
</tr>
<tr>
<td>P131</td>
<td>C3131</td>
<td>7-3-2002</td>
<td>“lower-end IBM Netvista caught fire, ...”</td>
<td>“dial 911 ...”</td>
</tr>
<tr>
<td>P131</td>
<td>C3143</td>
<td>8-3-2002</td>
<td>“IBM Netvista unstable with Maxtor HD”</td>
<td>“yeah, we knew it, ...”</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>prodID</th>
<th>manufacturer</th>
<th>model</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>P121</td>
<td>IBM</td>
<td>“D540X”</td>
<td>“1.7G CPU, ..., Maxtor 80G HD”</td>
</tr>
<tr>
<td>P131</td>
<td>IBM</td>
<td>“D710P”</td>
<td>“1.5G CPU, ..., Maxtor 200G HD”</td>
</tr>
</tbody>
</table>

Top-5 Results: \{c3, c1, c3—p2, c1—p1, c2—p2\}
Roots of the Problem

- Normalization, physical design choices, etc., may break down a piece of information into tuples in different tables.
- In addition, “relationships” among “objects” are ubiquitous.

How are “belgium” and “netherlands” related?

- How are “belgium” and “netherlands” related?

Diagram:

```
+----------------+       +----------------+       +----------------+
| name           |       | country        |       | continent      |
| country        |       | continent      |       | country        |
| continent      |       | name           |       | name           |

Belgium

+----------------+       +----------------+       +----------------+
| name           |       | country        |       | city           |
| country        |       | continent      |       | country        |
| city           |       | hq             |       | country        |
| hq             |       | organization   |       | country        |
| organization   |       | country        |       | name           |

Belgium

+----------------+       +----------------+
| name           |       | country         |
| country         |       | gov             |
| country         |       | hq              |
| gov             |       | country         |

Belgium

Monarchy

+----------------+       +----------------+       +----------------+
| name           |       | country        |       | continent      |
| country        |       | continent      |       | country        |
| continent      |       | name           |       | name           |

Belgium

Monarchy

Netherlands

Netherlands
```
Applications

- **Keyword search in DB good for:**
  - Casual/Web Users
    - “*What’s SQL? ... I have 3 years database experience with MS Access.*”
    - “*I don’t know what’s the cryptic attribute names in your database, but can’t you just show me all you know about John?*”
    - “*I am looking for a movie, but I cannot remember its name, etc. All I know is that one actor is named Jim and the film was probably out in 1995*”
  - Exposing Database Content Online
  - Analytical applications
- **Typical applications**
  - Enterprise search
  - CRM
  - Content Management System
  - Forensic
  - Exploratory/interactive data analysis
  - Web accessible database
  - Other unseen examples to come
Challenges

- Related work is not immediate applicable
  - What’s the result of the query?
  - How to rank the “heterogeneous” results?
  - How to make it efficient?

DB – dealing with structured data, concerned with efficiency
IR – dealing with unstructured documents, concerned with effectiveness

Integration of DB and IR has begun …
What this tutorial is about

- Describing important work that supports keyword search for data stored in the DBMS
  - From a DB perspective
  - Focuses on relational data

- It is *not* about
  - A comprehensive list of *all* related work
    - See SIGMOD 05 panel report
  - Keyword search for semi-structured/XML data
    - See VLDB 05 tutorial
Outline

- Introduction
- IR Preliminaries
- Systems
- Open Issues

ACK
Part of the IR Preliminary slides are modified from Manning et al.
Information Retrieval: A Primer

- Systematic introduction of IR
  - John Sheperd
  - Jian Zhang

- Our goal here
  - Introduce a few IR concepts and techniques that will be used in keyword search in databases
Boolean Model

- Simple retrieval model based on set theory
  - Give me emails that contains “9314” and “timetable”

  “9314” AND “t…”

- If documents containing a keyword (term) are immediately available, we only need to do a merge
  - c.f., RID intersection to answer conjunctive predicates in RDBMS
Inverted index

- For each term $T$, we must store a list of all documents that contain $T$.
  - Within each posting, sort by docID
- Do we use an array or a list for this?

For example:

- For term $9314$, postings list: $2, 4, 8, 16, 32, 64, 128$
- For term $timetable$, postings list: $1, 2, 3, 5, 8, 13, 21, 34$

B+-tree

Postings lists
Query Processing: the Merge

- Walk through the two postings simultaneously, in time linear in the total number of postings entries.

If the list lengths are $x$ and $y$, the merge takes $O(x+y)$ operations.

Crucial: postings sorted by docID.
Inverted index construction

Documents to be indexed.

Tokenizer

Token stream.

Linguistic modules

Modified tokens.

Indexer

Inverted index.

More on these later.

Friends, Romans, countrymen.

friend

roman

countryman

2 4

1 2

13 16
### Term-document Matrix

<table>
<thead>
<tr>
<th></th>
<th>“9314”</th>
<th>“timetable”</th>
<th>“9318”</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>D1</strong></td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>...</td>
</tr>
<tr>
<td><strong>D2</strong></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>...</td>
</tr>
<tr>
<td><strong>D3</strong></td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>...</td>
</tr>
<tr>
<td><strong>D4</strong></td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>...</td>
</tr>
<tr>
<td><strong>...</strong></td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

#### Query: 9314 AND timetable

- Thus far, our queries have all been Boolean
  - Docs either match or not
- Good for expert users with precise understanding of their needs and the corpus
- Applications can consume 1000’s of results
  - What about most users?

#### Answer: \{D2, D3\}
### Term-document Matrix

<table>
<thead>
<tr>
<th></th>
<th>“9314”</th>
<th>“timetable”</th>
<th>“9318”</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>...</td>
</tr>
<tr>
<td>D2</td>
<td>3</td>
<td>2</td>
<td>100</td>
<td>...</td>
</tr>
<tr>
<td>D3</td>
<td>1000</td>
<td>10</td>
<td>6</td>
<td>...</td>
</tr>
<tr>
<td>D4</td>
<td>0</td>
<td>0</td>
<td>7</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

**Query:** 9314 AND timetable

**Answer:**
1) D3
2) D2

- Consider the number of occurrences of a term in a document \( \rightarrow \) (raw) term frequency (tf)
- tf is often attenuated:
  - \( ntf = 1 + \log(tf) \), if \( tf > 0 \)
  - \( ntf = 0 \), otherwise
- \( s(D, Q) = \sum_{t \in Q} s(D, t) \) \( \rightarrow \) conjunctive query only
  - \( = \sum_{t \in Q} tf_{t,D} \) \( \rightarrow \) to be refined ...
### Term-document Matrix

<table>
<thead>
<tr>
<th></th>
<th>“9314”</th>
<th>“timetable”</th>
<th>“9318”</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>...</td>
</tr>
<tr>
<td>D2</td>
<td>3</td>
<td>20</td>
<td>100</td>
<td>...</td>
</tr>
<tr>
<td>D3</td>
<td>1000</td>
<td>10</td>
<td>6</td>
<td>...</td>
</tr>
<tr>
<td>D4</td>
<td>0</td>
<td>0</td>
<td>7</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

- **Query**: 9314 AND timetable
- **Answer**:
  1) ???
  2) ???

- Forced to cmp “apples” with “oranges” “fairly”
  - Intuition: frequently appearing words are less important
- **idf** (inverse document frequency) = \( \frac{N}{df} \)
  - \( \text{idf}_{9318} = \frac{4}{3}, \text{idf}_{\text{timetable}} = \frac{4}{2} \)
  - often attentuated by \( \log() \) too.
  - **idf** is a collection-wide/specific statistics
State-of-the-art

- Other factors
  - Doc length normalization
  - tf in the query $\rightarrow$ qtf
    - cosine similarity: measure angles of the doc vec and the query vec

- Modern document scoring schemes
  - Okapi
    - $k_1$: [1.0, 2.0]
    - $b$: $\sim$0.75
    - $k_3$: [0, 1000]
  - Pivoted normalization weighting
    - $s$: $\sim$ 0.20

<table>
<thead>
<tr>
<th>$tf_w$</th>
<th>document freq</th>
</tr>
</thead>
<tbody>
<tr>
<td>$qtf_w$</td>
<td>w’s freq in the query</td>
</tr>
<tr>
<td>$N$</td>
<td>#docs in the collection</td>
</tr>
<tr>
<td>$df_w$</td>
<td>#docs containing w</td>
</tr>
<tr>
<td>$dl$</td>
<td>document length (in bytes)</td>
</tr>
<tr>
<td>$avdl$</td>
<td>avg doc length</td>
</tr>
</tbody>
</table>

\[
\sum_{t \in Q \cap D} \ln \frac{N - df + 0.5}{df + 0.5} \cdot \frac{(k_1 + 1)tf}{(k_1(1-b) + b \frac{dl}{avdl}) + tf} \cdot \frac{(k_3 + 1)qtf}{k_3 + qtf}
\]

\[
\sum_{t \in Q \cap D} \ln \frac{1 + \ln(1 + \ln(tf))}{(1 - s) + s \frac{dl}{avdl}} \cdot qtf \cdot \ln \frac{N + 1}{df}
\]
Precision & Recall

- **Precision**: fraction of retrieved docs that are relevant
  \[ P = \frac{tp}{tp + fp} \]
- **Recall**: fraction of relevant docs that are retrieved
  \[ R = \frac{tp}{tp + fn} \]

- Relevant
- Retrieved
- Not Retrieved
- tp
- fp
- fn
- tn

\[ tp = \text{true positive} \]
\[ fp = \text{false positive} \]
\[ fn = \text{false negative} \]
\[ tn = \text{true negative} \]

\[ fp = \text{false positive} \]

\[ f_p = \text{false positive} \]
Precision vs. Recall

- How to design an algorithm that has
  - 99.9999% precision?
  - 99.9999% recall?

- Recall is a non-decreasing function of the number of docs retrieved

- In a good system, precision decreases as either number of docs retrieved or recall increases
  - A fact with strong empirical confirmation
  - F-measure or 11-MAP can be used to show the trade-off.
Outline

- Introduction
- IR Preliminaries
- Systems
- Open Issues
## List of Systems

<table>
<thead>
<tr>
<th>Dimensions</th>
<th>Year</th>
<th>Type</th>
<th>Data model</th>
<th>Query model</th>
<th>Generations</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1st generation</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2nd generation</td>
</tr>
<tr>
<td><strong>Search for information unit</strong></td>
<td>1998</td>
<td></td>
<td></td>
<td>Web Pages</td>
<td>2005</td>
</tr>
<tr>
<td><strong>Proximity Search</strong></td>
<td>2001</td>
<td></td>
<td></td>
<td>Graph-structured Data</td>
<td>2006</td>
</tr>
<tr>
<td><strong>BANKS 1</strong></td>
<td>2002</td>
<td></td>
<td></td>
<td>Relational Data</td>
<td>2007</td>
</tr>
<tr>
<td><strong>DBXplorer</strong></td>
<td>2003</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>DISCOVER 1</strong></td>
<td>2004</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>DISCOVER 2</strong></td>
<td>2005</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Liu et al 06</strong></td>
<td>2006</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>SPARK</strong></td>
<td>2007</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Other details**
- AND/OR
- Top-$k$ ranking
Proximity Search

“FIND movies NEAR Cage and Travolta”
- 1st result:

Info:
- Stanford, VLDB 1998
- Graph-based
- Emphasizes on supporting efficient shortest path queries
Data and Query Models

- **Data Model**
  - Undirected graph with weighted edges (representing referencing relationships)
  - Loosely based on the OEM model

- **Query Model**
  - Objects in the “Find” set

- **Ranking**
  - Rank each object in the “Find” set with regards to its proximity to objects in the “Near” set.
  - Let \( b(f, n) = r_F(f) * r_N(n) / \text{min\_dist}(f, n) \)
  - \( \text{score}(f) = \sum_{n \in N} b(f, n) \)

\[ \text{min\_dist}(f, n) = \min \text{ distance between } f \text{ and } n; \text{ or } \infty \text{ if } > \text{ threshold } K \]
“FIND movies NEAR Cage and Travolta”

- let $F := \{\text{all movies in the database}\}$
- let $N := \{\text{all nodes containing Cage}\} + \{\text{all nodes containing Travolta}\}$
- foreach $f$ in $F$
  - score($f$) = $\sum_{n \in N} b(f, n)$
  - $= \sum_{n \in N} (r_F(f) \times r_N(n) / \text{min\_dist}(f, n)^t)$
- Return those $f$ with the top-$k$ scores
Pre-calculation of min_dist

Scalability Issues for Existing Algorithms
  □ Dijkstra’s alg. / Floyd-Warshall’s alg.
  □ Transitive closure
  **Reason**: designed for in-memory data; will incur huge amount random I/Os ...

Solution: Matrix multiplication with Sorting
  □ Initially, let $A[][] = \text{the adjacency matrix of } G$
  □ Compute $A \rightarrow A^2 \rightarrow A^4 \rightarrow \ldots \rightarrow A^K$
    ■ $A'[i][j] = \min_k (A[i][k] + A[k][j])$  // min dist between $i$ and $j$ thru $k$
    ■ Finally, $A'[i][j] = \min (A[i][j], A'[i][j])$  // cmp with old min dist
  □ To minimize # of random seek
    ■ Loop thru $k$, then “match” $i$ and $j$’s in $k$’s “neighborhood”
    ■ Cluster “neighborhood” of every nodes by sorting
Hub Index

- It works but $A^k$ is usu. large
- Observation
  - Size of min_dist[][]
    - Before, $O(|A| \times |B|)$
    - Afterwards, $O(|A| \times |H| + |B| \times |H| + |H| \times |H|)$
- Can be generalized even if H is not “separators”
  - Select a subset of nodes as hubs (H)  
    *How: select modes with large degrees*
  - min_dist*[i][j] records min distance between i and j without cross any objects in H
    
    \[ A'[i][j] = \min_k (A[i][k] + A[k][j]), \text{ if } k \notin H \]
  - Query need to combine both information (see next page)
- Benefits
  - Smaller min_dist index: scale linearly when 2.5% nodes selected as hubs

similar idea also exploited to find better in-memory shortest-path algorithms in the algorithm community
Query Processing with the Hub Index

- 3 cases
  - Both in H
    - \((p, q)\): return Hubs\([p, q]\)
  - One obj in H
    - \((q, x)\): \(\min ((x, a) + (a, q)), \forall a \in H\) and \(a\) in \(x\)’s neighborhood
  - Neither objs in H
    - \((x, y)\): \(\min ((x, y), (x, a) + (a, b) + (b, y)), \forall a, b \in H\) and \(a(b)\) in \(x(y)\)’s neighborhood
Cost

“FIND movies NEAR Cage and Travolta”

- let $F := \{\text{all movies in the database}\}$
- let $N := \{\text{all nodes containing Cage}\} + \{\text{all nodes containing Travolta}\}$
- foreach $f$ in $F$
  - $score(f) = \sum_{n \in N} b(f, n)$
    - $= \sum_{n \in N} (r_F(f) * r_N(n) / \min\_dist(f, n))$
- Return those $f$ with the top-$k$ scores

- Worst case: $2^*|F|^*|N|$, as 2 I/Os for each $f$
- Cost can be greatly reduced by clustering objects with the same labels