Elementary IR: Scalable Boolean Text Search

(Compare with R & G 27.1-3)
Information Retrieval: History

- **A research field traditionally separate from Databases**
  - Hans P. Luhn, IBM, 1959: “Keyword in Context (KWIC)”
  - G. Salton at Cornell in the 60’s/70’s: SMART
    - Around the same time as relational DB revolution
  - Tons of research since then
    - Especially in the web era

- **Products traditionally separate**
  - Originally, document management systems for libraries, government, law, etc.
  - Gained prominence in recent years due to web search
    - Still used for non-web document management. (“Enterprise search”).
Today: Simple (naïve!) IR

- **Boolean Search on keywords**
- **Goal:**
  - Show that you already have the tools to do this from your study of relational DBs
- **We’ll skip:**
  - Text-oriented storage formats
  - Intelligent result ranking (hopefully later!)
  - Parallelism
    - Critical for modern relational DBs too
  - Various bells and whistles (lots of little ones!)
    - Engineering the specifics of (written) human language
      - E.g. dealing with tense and plurals
      - E.g. identifying synonyms and related words
      - E.g. disambiguating multiple meanings of a word
      - E.g. clustering output
IR vs. DBMS

- Seem like very different beasts

<table>
<thead>
<tr>
<th>IR</th>
<th>DBMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Imprecise Semantics</td>
<td>Precise Semantics</td>
</tr>
<tr>
<td>Keyword search</td>
<td>SQL</td>
</tr>
<tr>
<td>Unstructured data format</td>
<td>Structured data</td>
</tr>
<tr>
<td>Read-Mostly. Add docs occasionally</td>
<td>Expect reasonable number of updates</td>
</tr>
<tr>
<td>Page through top $k$ results</td>
<td>Generate full answer</td>
</tr>
</tbody>
</table>

- Under the hood, not as different as they might seem
  - But in practice, you have to choose between the 2 today
IR’s “Bag of Words” Model

• **Typical IR data model:**
  – Each document is just a bag of words (“terms”)

• **Detail 1: “Stop Words”**
  – Certain words are not helpful, so not placed in the bag
  – e.g. real words like “the”
  – e.g. HTML tags like <H1>

• **Detail 2: “Stemming”**
  – Using language-specific rules, convert words to basic form
  – e.g. “surfing”, “surfed” --> “surf”
  – Unfortunately have to do this for each language
    • Yuck!
Boolean Text Search

- Find all documents that match a Boolean containment expression:
  - “Windows”
    - AND (“Glass” OR “Door”)
    - AND NOT “Microsoft”

- Note: query terms are also filtered via stemming and stop words

- When web search engines say “10,000 documents found”, that’s the Boolean search result size
  - More or less ;-)
Text “Indexes”

- **When IR folks say “text index”...**
  - usually mean more than what DB people mean

- **In our terms, both “tables” and indexes**
  - Really a logical schema (i.e. tables)
  - With a physical schema (i.e. indexes)
  - Usually not stored in a DBMS
    - Tables implemented as files in a file system
    - We’ll talk more about this decision soon
A Simple Relational Text Index

• **Given:** a corpus of text files
  – Files(docID string, content string)

• **Create and populate a table**
  InvertedFile(term string, docID string)

• **Build a B+-tree or Hash index on** InvertedFile.term
  – Something like “Alternative 3” critical here!!
    • Keep lists of dup keys *sorted by docID*
      – Will provide “interesting orders” later on!
    • Fancy list compression important, too
    • Typically called a *postings list* by IR people
  – Note: URL instead of RID, the web is your “heap file”!
    • Can also *cache* pages and use RIDs

• **This is often called an “inverted file” or “inverted index”**
  – Maps from words -> docs, rather than docs -> words

• **Given this, you can now do single-word text search queries!**
An Inverted File

- **Snippets from:**
  - Old class web page
  - Old microsoft.com home page
- **Search for**
  - databases
  - microsoft

<table>
<thead>
<tr>
<th>Term</th>
<th>docID</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td><a href="http://www-inst.eecs.berkeley.edu/~cs186">http://www-inst.eecs.berkeley.edu/~cs186</a></td>
</tr>
<tr>
<td>database</td>
<td><a href="http://www-inst.eecs.berkeley.edu/~cs186">http://www-inst.eecs.berkeley.edu/~cs186</a></td>
</tr>
<tr>
<td>date</td>
<td><a href="http://www-inst.eecs.berkeley.edu/~cs186">http://www-inst.eecs.berkeley.edu/~cs186</a></td>
</tr>
<tr>
<td>day</td>
<td><a href="http://www-inst.eecs.berkeley.edu/~cs186">http://www-inst.eecs.berkeley.edu/~cs186</a></td>
</tr>
<tr>
<td>dbms</td>
<td><a href="http://www-inst.eecs.berkeley.edu/~cs186">http://www-inst.eecs.berkeley.edu/~cs186</a></td>
</tr>
<tr>
<td>decision</td>
<td><a href="http://www-inst.eecs.berkeley.edu/~cs186">http://www-inst.eecs.berkeley.edu/~cs186</a></td>
</tr>
<tr>
<td>demonstrate</td>
<td><a href="http://www-inst.eecs.berkeley.edu/~cs186">http://www-inst.eecs.berkeley.edu/~cs186</a></td>
</tr>
<tr>
<td>description</td>
<td><a href="http://www-inst.eecs.berkeley.edu/~cs186">http://www-inst.eecs.berkeley.edu/~cs186</a></td>
</tr>
<tr>
<td>design</td>
<td><a href="http://www-inst.eecs.berkeley.edu/~cs186">http://www-inst.eecs.berkeley.edu/~cs186</a></td>
</tr>
<tr>
<td>desire</td>
<td><a href="http://www-inst.eecs.berkeley.edu/~cs186">http://www-inst.eecs.berkeley.edu/~cs186</a></td>
</tr>
<tr>
<td>developer</td>
<td><a href="http://www.microsoft.com">http://www.microsoft.com</a></td>
</tr>
<tr>
<td>differ</td>
<td><a href="http://www-inst.eecs.berkeley.edu/~cs186">http://www-inst.eecs.berkeley.edu/~cs186</a></td>
</tr>
<tr>
<td>disability</td>
<td><a href="http://www.microsoft.com">http://www.microsoft.com</a></td>
</tr>
<tr>
<td>discussion</td>
<td><a href="http://www-inst.eecs.berkeley.edu/~cs186">http://www-inst.eecs.berkeley.edu/~cs186</a></td>
</tr>
<tr>
<td>division</td>
<td><a href="http://www-inst.eecs.berkeley.edu/~cs186">http://www-inst.eecs.berkeley.edu/~cs186</a></td>
</tr>
<tr>
<td>do</td>
<td><a href="http://www-inst.eecs.berkeley.edu/~cs186">http://www-inst.eecs.berkeley.edu/~cs186</a></td>
</tr>
<tr>
<td>document</td>
<td><a href="http://www-inst.eecs.berkeley.edu/~cs186">http://www-inst.eecs.berkeley.edu/~cs186</a></td>
</tr>
<tr>
<td>document</td>
<td><a href="http://www.microsoft.com">http://www.microsoft.com</a></td>
</tr>
<tr>
<td>microsoft</td>
<td><a href="http://www.microsoft.com">http://www.microsoft.com</a></td>
</tr>
<tr>
<td>microsoft</td>
<td><a href="http://www-inst.eecs.berkeley.edu/~cs186">http://www-inst.eecs.berkeley.edu/~cs186</a></td>
</tr>
<tr>
<td>midnight</td>
<td><a href="http://www-inst.eecs.berkeley.edu/~cs186">http://www-inst.eecs.berkeley.edu/~cs186</a></td>
</tr>
<tr>
<td>midterm</td>
<td><a href="http://www-inst.eecs.berkeley.edu/~cs186">http://www-inst.eecs.berkeley.edu/~cs186</a></td>
</tr>
<tr>
<td>minibase</td>
<td><a href="http://www-inst.eecs.berkeley.edu/~cs186">http://www-inst.eecs.berkeley.edu/~cs186</a></td>
</tr>
<tr>
<td>million</td>
<td><a href="http://www.microsoft.com">http://www.microsoft.com</a></td>
</tr>
<tr>
<td>monday</td>
<td><a href="http://www.microsoft.com">http://www.microsoft.com</a></td>
</tr>
<tr>
<td>more</td>
<td><a href="http://www.microsoft.com">http://www.microsoft.com</a></td>
</tr>
<tr>
<td>most</td>
<td><a href="http://www-inst.eecs.berkeley.edu/~cs186">http://www-inst.eecs.berkeley.edu/~cs186</a></td>
</tr>
<tr>
<td>ms</td>
<td><a href="http://www-inst.eecs.berkeley.edu/~cs186">http://www-inst.eecs.berkeley.edu/~cs186</a></td>
</tr>
<tr>
<td>msn</td>
<td><a href="http://www.microsoft.com">http://www.microsoft.com</a></td>
</tr>
<tr>
<td>must</td>
<td><a href="http://www-inst.eecs.berkeley.edu/~cs186">http://www-inst.eecs.berkeley.edu/~cs186</a></td>
</tr>
<tr>
<td>necessary</td>
<td><a href="http://www-inst.eecs.berkeley.edu/~cs186">http://www-inst.eecs.berkeley.edu/~cs186</a></td>
</tr>
<tr>
<td>need</td>
<td><a href="http://www-inst.eecs.berkeley.edu/~cs186">http://www-inst.eecs.berkeley.edu/~cs186</a></td>
</tr>
</tbody>
</table>
Handling Boolean Logic

- **How to do “term1” OR “term2”?**
  - Union of two postings lists (docID sets)!

- **How to do “term1” AND “term2”?**
  - Intersection of two postings lists!
    - Can be done via merge-join over postings lists
    - Remember: postings list per key sorted by docID in index

- **How to do “term1” AND NOT “term2”?**
  - Set subtraction
    - Also easy because sorted (basically merge join logic again)

- **How to do “term1” OR NOT “term2”**
  - Union of “term1” and “NOT term2”.
    - “Not term2” = all docs not containing term2. Yuck!
  - Usually not allowed!

- **Query Optimization: what order to handle terms if you have many ANDs?**
Boolean Search in SQL

- `(SELECT docID FROM InvertedFile
  WHERE word = "window"
INTERSECT
SELECT docID FROM InvertedFile
WHERE word = "glass" OR word = "door")
EXCEPT
SELECT docID FROM InvertedFile
WHERE word="Microsoft"
ORDER BY magic_rank()`

- There’s only one SQL query template in Boolean Search
  - Single-table selects, UNION, INTERSECT, EXCEPT
- `magic_rank()` is the “secret sauce” in the search engines
  - Hopefully we’ll study this later in the semester
  - Combos of statistics, linguistics, and graph theory tricks!
One step fancier: Phrases and “Near”

- **Suppose you want a phrase**
  - E.g. “Happy Days”
- **Different schema:**
  - InvertedFile (term string, position int, docID string)
  - Alternative 3 index on term
  - Postings lists sorted by (docID, position)
- **Post-process the results**
  - Find “Happy” AND “Days”
  - Keep results where positions are 1 off
    - Can be done during merge-join to AND the 2 lists!
- **Can do a similar thing for “term1” NEAR “term2”**
  - Position < \( k \) off
  - Think about refinement to merge-join...
Somewhat better compression

- InvertedFile (term string, position int, docID int)
- Files(docID int, docID string, snippet string, ...)
- Btree on InvertedFile.term
- Btree on Docs.docID

- Requires a final join step between typical query result and Files.docID
  - Can do this lazily: cursor to generate a page full of results
Updates and Text Search

- **Text search engines are designed to be query-mostly**
  - Deletes and modifications are rare
  - Can postpone updates (nobody notices, no transactions!)
    - Can work off a union of indexes
    - Merge them in batch (typically re-bulk-load a new index)
  - Can’t afford to go offline for an update?
    - Create a 2nd index on a separate machine
    - Replace the 1st index with the 2nd!
  - So no concurrency control problems
  - Can compress to search-friendly, update-unfriendly format
  - Can keep postings lists sorted
- **For these reasons, text search engines and DBMSs are usually separate products**
  - Also, text-search engines tune that one SQL query to death!
  - The benefits of a special-case workload.
Lots more tricks in IR

- **How to “rank” the output?**
  - A mix of simple tricks works well
  - Some fancier tricks can help (use hyperlink graph)
- **Other ways to help users paw through the output?**
  - Document “clustering” (e.g. Clusty.com)
  - Document visualization
- **How to use compression for better I/O performance?**
  - E.g. making postings lists smaller
  - Try to make things fit in RAM
    - Or in processor caches
- **How to deal with synonyms, misspelling, abbreviations?**
- **How to write a good web crawler?**

- **We’ll return to some of these later**
  - The book *Managing Gigabytes* covers some of the details
Recall From the First Lecture

Query Optimization and Execution
Relational Operators
Files and Access Methods
Buffer Management
Disk Space Management

Concurrency and Recovery Needed

DBMS

Search String Modifier
Ranking Algorithm
“The Query”
The Access Method
Buffer Management
Disk Space Management

DB

Search Engine

Simple DBMS

DBMS

Concurrency and Recovery Needed

Concurrence and Recovery Needed
You Know The Basics!

• “Inverted files” are the workhorses of all text search engines
  – Just B+-tree or Hash indexes on bag-of-words
• Intersect, Union and Set Difference (Except)
  – Usually implemented via sorting
  – Or can be done with hash or index joins
• Most of the other stuff is not “systems” work
  – A lot of it is cleverness in dealing with language
  – Both linguistics and statistics (more the latter!)
Revisiting Our IR/DBMS Distinctions

- **Semantic Guarantees on Storage**
  - DBMS guarantees transactional semantics
    - If an inserting transaction commits, a subsequent query *will see* the update
    - Handles multiple concurrent updates correctly
  - IR systems do not do this; nobody notices!
    - Postpone insertions until convenient
    - No model of correct concurrency.
    - Can even return incorrect answers for various reasons!

- **Data Modeling & Query Complexity**
  - DBMS supports any schema & queries
    - But requires you to define schema
    - And SQL is hard to figure out for the average citizen
  - IR supports only one schema & query
    - No schema design required (unstructured text)
    - Trivial (natural?) query language for simple tasks
    - No data correlation or analysis capabilities -- “search” only
Revisiting Distinctions, Cont.

- **Performance goals**
  - DBMS supports general SELECT
    - plus mix of INSERT, UPDATE, DELETE
    - general purpose engine must always perform “well”
  - IR systems expect only one stylized SELECT
    - plus delayed INSERT, unusual DELETE, no UPDATE.
    - special purpose, must run super-fast on “The Query”
    - users rarely look at the full answer in Boolean Search
      - Postpone any work you can to subsequent index joins
      - But make sure you can rank!
Summary

• **IR & Relational systems share basic building blocks for scalability**
  – IR internal representation is relational!
  – Equality indexes (B-trees)
  – Iterators
  – “Join” algorithms, esp. merge-join
  – “Join” ordering and selectivity estimation

• **IR constrains queries, schema, promises on semantics**
  – Affects storage format, indexing and concurrency control
  – Affects join algorithms & selectivity estimation

• **IR has different performance goals**
  – Ranking and best answers **fast**

• **Many challenges in IR related to “text engineering”**
  – But don’t tend to change the scalability infrastructure
IR Buzzwords to Know (so far!)

- Learning this in the context of relational foundations is fine, but you need to know the IR lingo!
  - Corpus: a collection of documents
  - Term: an isolated string (searchable unit)
  - Index: a mechanism mapping terms to documents
  - Inverted File (= Postings File): a file containing terms and associated postings lists
  - Postings List: a list of pointers ("postings") to documents
Exercise!

- **Implement Boolean search directly in Postgres**
  - Using the schemas and indexes here.
    - Write a simple script to load files.
    - You can ignore stemming and stop-words.
  - Run the SQL versions of Boolean queries
    - Measure how slow search is in Postgres
  - Identify contributing factors in performance
    - E.g. how much disk space does the postgres version use (including indexes) vs. the raw documents vs. the documents gzip’ed
    - E.g. is PG identifying the “interesting orders” in the postings lists? (use EXPLAIN) If not, can you force it to do so?

- **Compare PG to an idealized implementation**
  - Calculate the idealized size of the InvertedFile table for your data
  - Use the cost models for IndexScan and MergeJoin to calculate the expected number of IOs. Distinguish sequential and random Ios.
  - Why is PG slow? Storage overhead? Optimizer smarts?