Elementary IR: Scalable Boolean Text Search

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
    - Libraries, government, law, etc.
  - Renaissance due to web search and advertising
    - Still a small market in "Enterprise search"

Plan of Attack

- Start with naïve Boolean Search on keywords
  - With unordered answer sets
- Later:
  - Intelligent result ranking
- We'll skip:
  - Text-oriented index compression
  - 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

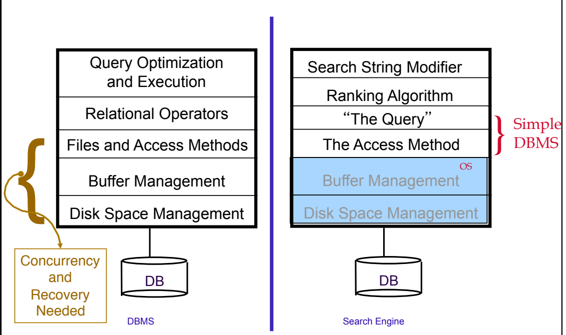
IR vs. DBMS

- Seem like very different beasts

IR	DBMS
Imprecise Semantics	Precise Semantics
Keyword search	SQL
Unstructured data format	Structured data
Read-Mostly. Add docs occasionally	Expect reasonable number of updates
Page through top *k* results	Generate full answer

- Under the hood, not as different as they might seem
  - In practice, no product does both well (yet)
- IR engines more "custom" than most DBMSs

Recall From the First Lecture



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!
  - Lots of open source libraries for this

Boolean Text Search

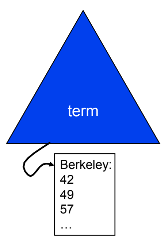
- Find all docs matching a Boolean expression:
  - "Windows" AND ("Glass" OR "Door") AND NOT "Microsoft"
- Note: query terms are stemmed/stopped
- When web search engines say "10,000,000 documents found", that's the Boolean search result size
  - More or less ;-)

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

Simple Relational Text Index

- Given a *corpus* of text files
  - Files(<u>docID</u> text, content text)
- Create and populate a table
  InvertedFile(term text, docID text)
- Build a B+-tree or Hash index on InvertedFile.term
  - Use something like "Alternative 3" index
    - Keep lists at the bottom sorted by docID
    - Typically called a "postings list"
- Now you can do single-word queries.



Inverted File

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

Term	docID
data	http://www-inst.eecs.berkeley.edu/~cs186
database	http://www-inst.eecs.berkeley.edu/~cs186
date	http://www-inst.eecs.berkeley.edu/~cs186
day	http://www-inst.eecs.berkeley.edu/~cs186
dbms	http://www-inst.eecs.berkeley.edu/~cs186
decision	http://www-inst.eecs.berkeley.edu/~cs186
demonstrate	http://www-inst.eecs.berkeley.edu/~cs186
description	http://www-inst.eecs.berkeley.edu/~cs186
design	http://www-inst.eecs.berkeley.edu/~cs186
desire	http://www-inst.eecs.berkeley.edu/~cs186
developer	http://www.microsoft.com
differ	http://www-inst.eecs.berkeley.edu/~cs186
disability	http://www.microsoft.com
discussion	http://www-inst.eecs.berkeley.edu/~cs186
division	http://www-inst.eecs.berkeley.edu/~cs186
do	http://www-inst.eecs.berkeley.edu/~cs186
document	http://www-inst.eecs.berkeley.edu/~cs186
document	http://www.microsoft.com
microsoft	http://www.microsoft.com
microsoft	http://www-inst.eecs.berkeley.edu/~cs186
midnight	http://www-inst.eecs.berkeley.edu/~cs186
midterm	http://www-inst.eecs.berkeley.edu/~cs186
minibase	http://www-inst.eecs.berkeley.edu/~cs186
million	http://www.microsoft.com
monday	http://www.microsoft.com
more	http://www.microsoft.com
most	http://www-inst.eecs.berkeley.edu/~cs186
ms	http://www-inst.eecs.berkeley.edu/~cs186
msn	http://www.microsoft.com
must	http://www-inst.eecs.berkeley.edu/~cs186
necessary	http://www-inst.eecs.berkeley.edu/~cs186
need	http://www-inst.eecs.berkeley.edu/~cs186

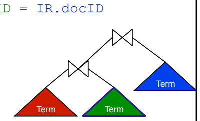
Handling Boolean Logic

- "term1" OR "term2":
  - Union of two postings lists (docID sets)!
- "term1" AND "term2":
  - Intersection of two postings lists!
    - merge of postings lists (already sorted by docID!)
- "term1" AND NOT "term2":
  - Set subtraction
    - merge again!
- "term1" OR NOT "term2":
  - Union of "term1" with "NOT term2".
    - "Not term2" = all docs not containing term2. Yuck!
  - Usually not allowed!

Boolean Search in SQL

"Berkeley Database Research"

```
SELECT IB.docID
  FROM InvertedFile IB, InvertedFile ID, InvertedFile IR
 WHERE IB.docID = ID.docID AND ID.docID = IR.docID
   AND IB.term = "Berkeley"
   AND ID.term = "Database"
   AND IR.term = "Research"
ORDER BY magic_rank()
```



- Note: joins here instead of intersect
  - Why is that equivalent?
- Simple query plan
  - An indexscan on each `Ix.term` "instance" in FROM clause
  - A merge-join of the 3 indexscans (ordered by docID)

Boolean Search in SQL

"Windows" AND ("Glass" OR "Door") AND NOT "Microsoft"

```
(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()
```

- Basically still a bunch of merge joins on index scans
- Only one SQL query (template) in Boolean Search
  - Single-table selects, UNION, INTERSECT, EXCEPT
  - Customize everything for this!
- magic_rank() is the "secret sauce" in the search engines
  - Combos of statistics, linguistics, and graph theory tricks

A bit fancier: Phrases and "Near"

- Suppose you want a phrase
  - E.g. "Happy Days"
- Augment the schema:
  - InvertedFile (term string, docID string, position int)
  - Index on term, Alternative 3 style
  - 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 the merging of the 2 lists during AND!
- Can do a similar thing for "term1" NEAR "term2"
  - Position < k off
  - Think about the refinement to merge…

Getting the document content?

- InvertedFile (term string, position int, docID int)
  - IDs smaller, compress better than URLS
- Files(docID int, URL string, snippet string, …)
  - and possibly a cache file ID
- Btree on InvertedFile.term
- Btree on Files.docID

- Requires a final "join" step between typical query result and Files.docID
  - Do this *lazily*: one results page at a time!

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)
    - "Log-Structured Merge" 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
- See why text search engines and DBMSs are separate?
  - Also, text-search engines tune that one SQL query to death!
  - The benefits of a special-case workload.

Tons more tricks

- How to "rank" the output?
  - Coming soon
- Document "clustering" and other visualization ideas
- How to use compression for better I/O performance?
  - E.g. making postings lists smaller
  - Try to make things fit in RAM, processor cache
- How to deal with synonyms, misspelling, abbreviations?
- How to write a good web crawler?
- Dealing with SEO (a.k.a. web spam)
- User Customization

You Already 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 pre-sorting and merge
  - Or can be done with hash or index joins
- Much of the other stuff is custom to text & web
  - Linguistics and statistics (more the latter!)
  - Exploiting graph structure of the web
  - Understanding content types, user desires, etc.

IR Buzzwords to Know (so far!)

- I taught this w.r.t. relational foundations
- 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

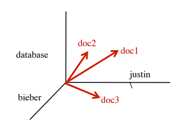
Summary

- IR & Relational systems share building blocks for scalability
  - IR internal representation is relational
  - Equality indexes (B-trees)
  - Dataflow (iterators) and parallel dataflow
  - "Join" algorithms, esp. merge-join
- 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 specifics of the domain
  - But don't tend to change the scalability infrastructure

Text/Web Search II:
Ranking & Crawling

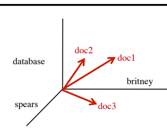
Berkeley cs186

Classical IR Ranking



- Abstraction: Vector space model
  - We'll think of every document as a "vector"
    - Imagine there are 10,000 possible terms
    - Each document (bag of words) can be represented as an array of 10,000 counts
    - I.e. a point in 10,000-d space
  - "similarity" of two documents: "distance" in 10,000d
- A query is just a short document
  - Rank all docs by their distance to the query

Classical IR Ranking



- What's the right distance metric?
  - Problem 1: two long docs seem more similar to each other than to short docs
    - Solution: normalize each dimension by vector's (Euclidean) length
    - Now every doc is a point on the unit sphere
  - Now: the cosine of the angle between two normalized vectors happens to be their dot product $A \cdot B = (x_A x_B + y_A y_B)$!
    - from law of cosines
    - http://en.wikipedia.org/wiki/Dot_product#Proof_of_the_geometric_interpretation
  - BTW: for normalized vectors, cosine *ranking* is the same as *ranking* by Euclidean distance

TF × IDF

log damps out idf. What is the idf of a term that occurs in all of the docs? In almost no docs?

- Counting occurrences isn't a good way to weight each term
  - Want to favor repeated terms in this doc
  - Want to favor unusual words in this doc
- TF × IDF (Term Frequency × Inverse Doc Frequency)
  - For each doc d
    - DocTermRank =
      #occurrences of t in d
      TF
      × log((total #docs)/(#docs with this term))
      × **IDF**

Indexing TF × IDF

- Let's add some more to our schema
  - TermInfo(term string, numDocs int).

    Denominator of IDF
    - This is a "materialized" view on the invertedFile table.
    - Write down the SQL for the view!
  - InvertedFile(term string, docID int64, DocTermRank float)
    - i.e. store TFxIDF with string per doc

In SQL Again…

–InvertedFile (term string, docID int64, DocTermRank float)

Simple Boolean Search

```
CREATE VIEW BooleanResult AS (
SELECT IB.docID, IB.DocTermRank as bTFIDF,
       ID.DocTermRank as dTFIDF,
       IR.DocTermRank as rTFIDF,
  FROM InvertedFile IB, InvertedFile ID, InvertedFile IR
 WHERE IB.docID = ID.docID AND ID.docID = IR.docID
   AND IB.term = "Berkeley"
   AND ID.term = "Database"
   AND IR.term = "Research");

SELECT docID,
       (<Berkeley-querytermrank>*bTFIDF +
        <Database-querytermrank>*dTFIDF +
        <Research-querytermrank>*rTFIDF>) AS magic_rank
  FROM BooleanResult
ORDER BY magic_rank;
```

Cosine similarity. Note that the query "doc" vector is a constant, can be computed

Ranking

Sort

Σ_i qTermRank$_i$*DocTermRank$_i$

Berkeley

docID	DTRank
42	0.361
49	0.126
57	0.111

Database

docID	DTRank
16	0.137
49	0.654
57	0.321

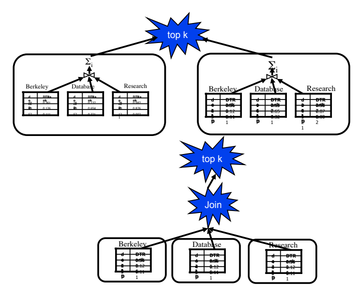
Research

docID	DTRank
29	0.987
49	0.876
121	0.002

- We'll only rank Boolean results
  - Note: this is just a heuristic! (Why?)
    - What's a fix? Is it feasible?
  - Recall: merge-join the postings-lists from each term, sort by docID
- While merging postings lists…
  - For each docID that matches
    - Compute cosine distance to query
      - I.e. For all terms, Sum of (product of query-term-rank and DocTermRank)
    - This collapses the view in the previous slide

Parallelizing (!!)

- Partition InvertedFile by DocID
  - Parallel "top k"
- Partition InvertedFile by term
  - Distributed Join
  - top k: parallel or not?
- Pros/cons?
  - What are the relevant metrics?



There's usually one more join stage

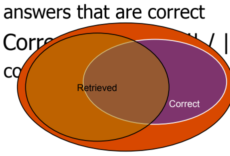
- Docs(docID, title, URL, crawldate, snippet)

```
SELECT D.title, D.URL, D.crawldate, D.snippet
       (<Berkeley-tfidf>*B.bTFIDF +
        <Database-tfidf>*B.dTFIDF +
        <Research-TFIDF>*B.rTFIDF>) AS magic_rank
  FROM BooleanResult AS B, Docs as D
 WHERE B.docID = D.docID
ORDER BY magic_rank;
```

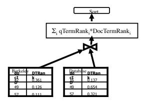
- Typically rank *before* the join with Docs
  - Join done via a parallel "fetch matches"
    - A la index NL
  - Or ensure Docs is replicated everywhere

Quality of a non-Boolean Answer

- Suppose only top k answers are retrieved
- Two common metrics:
  - Precision: |Correct ∩ Retrieved| / |Retrieved|
    - i.e. % answers that are correct
  - Recall: |Corr… | / |Correct|
    - i.e. % co…



Phrase & Proximity Ranking

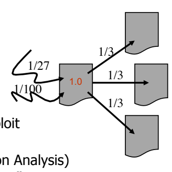


- Query: “The Who”
  - How many matches?
    - Our previous query plan?
  - Ranking quality?
- One idea: index all 2-word runs in a doc
  - “bigrams”, can generalize to “n-grams”
  - give higher rank to bigram matches
- More generally, proximity matching
  - how many words/characters apart?
    - add a *position* field to the inverted index as above
    - use proximity to boost overall rank

Some Additional Ranking Tricks

- Query expansion, suggestions
  - Can do similarity lookups on terms, expand/modify people's queries
- Fix misspellings
  - E.g. via an inverted index on q-grams of letters
  - Trigrams for “misspelling” are:
    {mis, iss, ssp, spe, pel, ell, lli, lin, ing}
- Document expansion
  - Can add terms to a doc before inserting into inverted file
    - E.g. in “anchor text” of refs to the doc
    - E.g. by classifying docs (e.g. “english”, “japanese”, “adult”)
- Not all occurrences are created equal
  - Mess with DocTermRank based on:
    - Fonts, position in doc (title, etc.)

Hypertext Ranking



- On the web, we have more information to exploit
  - The hyperlinks (and their anchor text)
  - Ideas from Social Network Theory (Citation Analysis)
  - “Hubs and Authorities” (Clever), “PageRank” (Google)
- Intuition (Google's PageRank)
  - If you are important, and you link to me, then I'm important
  - Recursive definition --> recursive computation
    1. Everybody starts with weight 1.0
    2. Share your weight among all your outlinks (and yourself, a damping factor)
    3. Repeat (2) until things converge
  - Note: computes the first eigenvector of the adjacency matrix
    - And you thought linear algebra was boring :-)
  - Leaving out some details here …
- PageRank sure seems to help
  - But rumor says that other factors matter as much or more
    - Anchor text, title/bold text, etc. --> much tweaking over time

Random Notes from the Real World

- The web's dictionary of terms is HUGE. Includes:
  - numerals: “1”, “2”, “3”, … “987364903”, …
  - codes: “SCOOT_TRIGRAM_RIGHT”, “cmptrgm”, …
  - misspellings: “teh”, “quik”, “browne”, “focs”
  - multiple languages: “hola”, “bonjour”, “ここんんににちちはは” (Japanese), etc.
- Web spam
  - Try to get top-rated. Seach Engine Optimization (SEO)
  - Imagine how to spam TF x IDF
    - “Stanford Stanford Stanford Stanford Stanford Stanford Stanford Stanford Stanford … Stanford lost The Big Game”
    - And use white text on a white background :-)
  - Imagine spamming PageRank…?!
- Some “real world” stuff makes life easier
  - Terms in queries are Zipfian! Can cache answers in memory effectively.
  - Queries are usually little (1-2 words)
  - Users don't notice minor inconsistencies in answers
- Big challenges in running thousands of machines, 24x7 service!

Building a Crawler

- Duh! This is graph traversal.

```
crawl(URL) {
    doc = fetch(URL);
    foreach href in the URL
        crawl(*href);
}
```

- Well yes, but:
  - better not sit around waiting on each fetch
  - better run in parallel on many machines
  - better be “polite”
  - probably won't “finish” before the docs change
    - need a “revisit policy”
  - all sorts of yucky URL details
    - dynamic HTML, “spider traps”
    - different URLs for the same data (mirrors, .. in paths, etc.)

Single-Site Crawler

- multiple outstanding fetches
  - each with a modest timeout
    - don't let the remote site choose it!
  - typically a multithreaded component
    - but can typically scale to more fetches/machine via a single-threaded “event-driven” approach
- a set of pending fetches
  - this is your crawl “frontier”
  - can grow to be quite big!
  - need to manage this wisely to pick next sites to fetch
  - what traversal would a simple FIFO queue for fetches give you? Is that good?

Crawl ordering

- What do you think?
  - Breadth first vs. Depth first?
  - Content driven? What metric would you use?
- What are our goals
  - Find good pages soon (may not finish before restart)
  - Politeness

Crawl Ordering, cont.

- Good to find high PageRank pages, right?
  - Could prioritize based on knowledge of P.R.
    - E.g. from earlier crawls
  - Research sez: breadth-first actually finds high P.R. pages pretty well though
    - Random doesn't do badly either
  - Other research ideas to kind of approximate P.R. online
  - Have to be at the search engines to really know how this is best done
    - Part of the secret sauce!
    - Hard to recreate without a *big* cluster and *lots* of NW

Scaling up

- How do you parallelize a crawler?
  - Roughly, you need to partition the frontier a la parallel join or map/reduce
  - Load balancing requires some thought
    - partition by URL prefix (domain name)? by entire URL?
- DNS lookup overhead can be a substantial bottleneck
  - E.g. the mapping from www.cs.berkeley.edu to 169.229.60.105
  - Pays to maintain local DNS caches at each

More on web crawlers?

- There is a quite detailed Wikipedia page
  - Focus on academic research, unfortunately
  - Still, a lot of this stuff came out of universities
    - Wisconsin (webcrawler '94), Berkeley (inktomi '96), Stanford (google '99)

Resources

- Textbooks
  - *Managing Gigabytes*, Witten/Moffat/Bell
  - *Modern Information Retrieval,* Baeza-Yates/Ribeiro-Neto
  - *Introduction to Information Retrieval,* Manning/ Raghavan/Schütze (free online!)
- Lecture Notes
  - Manning/Raghavan/Schütze notes to go with text
    - Source of some material in these slides