Language and Computers

Searching

L245
(Based on Dickinson, Brew, & Meurers (2013))

Spring 2017

Searching

- A breathtaking number of information resources are available: books, databases, the web, newspapers, ...
- To locate relevant information, we need to be able to search these resources, which often are **written texts**:
  - Searching in a library catalog (e.g., using IUCAT)
  - Searching the web (e.g., using Google)
  - Advanced searching in text corpora (e.g., using regular expressions)

Searching in speech

- One might also want to search for **speech**, e.g., to find a particular sentence spoken in an interview one only has a recording (audio file) of.
- This type of searching is generally not readily available with current technology.
- It is, however, possible to
  - detect the language of a spoken conversation, e.g., when listening in to a telephone conversation
  - detect a new topic being started in a conversation
- In the following, we focus on searching in text.

Types of data

1. **Structured data**: organized & searchable by categories: author, title, subject, and so forth.
   - Useful when the searcher knows the general topic that they are searching through
     - e.g., for *duck-billed platypuses*, look through *zoology* and *animal* topics
   - Problem: someone has to structure it
2. **Unstructured data**: much more available (e.g., the internet)
   - Keyword search can be highly effective
3. **Semi-structured data**: contains some categorization, but lacks much in the way of structure
   - Structure and format are often inconsistent, even for the same type of document (e.g., blogs, web reviews)

Structured data

Searching in a library catalog

- To find articles, books, and other library holdings, a library generally provides:
  - a **database** containing information on its holdings, and
  - a **database frontend** for users to interact with the database.
    - e.g., IUCAT, WorldCat
- Users search for the occurrence of **literal strings** occurring in the author, title, keywords, call number, etc. associated with an item held by the library.

Basic searching with IUCAT

- Literal strings are composed of characters which naturally must be in the same character encoding system (e.g. ASCII, ISO8859-1, UTF-8) as the strings encoded in the database.
- For literal strings, the search engine generally does not distinguish between upper and lower-case letters
- Adjacent words are both required in the results:
  - `art therapy`
  - `vitamin c`
- For some types of engines, **stop words** are ignored in searches, unless enclosed in double quotes (*a, an, as, at, be, but, by, do, for, if, in, is, it, of, on, the, to*)
  - IUCAT gives different results, though, for `art of therapy`

Special characters and operators

In addition to **querying** literal strings, the **query language** also supports the use of

- special **boolean expression** operations for combining two query strings
  - Use `AND` or `OR` to specify multiple words in any field, any order.
    - `art OR therapy`
    - Some engines offer `XOR`: e.g., `art XOR therapy` excludes “art therapy”
  - In principle: use parentheses to group words together when using more than one operator.
    - `art therapy NOT (music OR dance) therapy`
- Some engines offer special characters, like wildcards or truncation opterators (IUCAT no longer does)
  - IUCAT automatically checks variant stems

See https://kb.iu.edu/d/agya

Unstructured data

No explicit categorization of the documents to be retrieved

- Related to doing a keyword search in structured data
- Scale of the data is different: e.g., billions of webpages to search through
  - Types of search operators & ways to improve searches can differ from structured data

Some “unstructured” data contains hidden structure

- e.g., webpages with Chinese-English translations

By *unstructured*, we mean:

- The structure is not predetermined
- It is not uniformly applied or standardized
- Queries cannot be formulated on that particular type of structure

Information need

Searching involves *information need*: the information a searcher is seeking

- Information need gets translated into a query, hoping to capture that information need
- This is an imperfect process

(1) a. Information need: one or more Russian translations of the English word *table*

b. Possible query: `russian translation table`

Information need is unambiguous; query is ambiguous

- Could be looking for a table/chart of Russian translations (which may not include the word *table*)

Evaluating search results

Use of information need can be seen in the evaluations for the Text REtrieval Conference (TREC, http://trec.nist.gov/)

```
<top>

<num> Number: 303
<title> Hubble Telescope Achievements

<desc> Description:
Identify positive accomplishments of the Hubble telescope since
it was launched in 1991.

<narr> Narrative:
Documents are relevant that show the Hubble telescope has produced
new data, better quality data than previously available, data that
has increased human knowledge of the universe, or data that has led
to disproving previously existing theories or hypotheses.  Documents
limited to the shortcomings of the telescope would be irrelevant.
Details of repairs or modifications to the telescope without
reference to positive achievements would not be relevant.

</top>
```

Information need & evaluation

To evaluate search technology, TREC expresses information needs in natural language

- Evaluation: judge particular documents as to whether they meet information need in such descriptions

More specifically, TREC defines “right answers” as:

> *If you were writing a report on the subject of the topic and would use the information contained in the document in the report, then the document is relevant.*
> *(http://trec.nist.gov/data/reljudge_eng.html)*

Searching the web

A computer user

- wants to find something on “the web”, i.e., in files accessible via the hypertext transfer protocol (http) protocol on the internet
- goes to a **search engine** = program that matches documents to a user’s search requests
- enters a **query** = request for information
- gets a list of websites that might be relevant to the query
- **evaluates the results**: either picks a website with the information looked for or reformulates the query

The nature of the web

- Web pages are generally less structured than a record in a library database (with title, author, subject, and other fields).
- One generally searches for words found anywhere in the document.
- It is possible to include **meta data** in a web page.
  - Meta data is additional, structured information that is not shown in the web page itself
    - e.g., language a web page is in, its character encoding, author, keywords, etc.
  - Example for a **meta tag**: `<META name="keywords" lang="en-us" content="vacation, Greece">`

Search engines

Some ways in which search engines can differ:

- Treatment of word tokens:
  - **stemming**: treat *bird* and *birds* as the same or not
  - **capitalization**: treat *trip* and *Trip* the same or not
- Options for searching: use of **operators** or special interface for advanced searching
- How search results are **ranked**
  - Potentially also **clustered** (grouped into similar results), e.g., http://yippy.com

Search engines

How they work (roughly)

Search engines (e.g., Google)

- Store a copy of all web pages
- Create an **index** to provide efficient access to this large number of pages (e.g., Google currently searches over 1 trillion pages)
- Compute a rank for each web page to be able to rank the query results

Search engine indexing

Manning et al (2008)

As a search engine crawls the web, it builds a **term-by-document matrix**

- shows which terms (i.e., words) appear in which documents
- e.g., for some mystery novels:

	Affair at Styles	Secret Adversary	Sherlock Holmes
Poirot	1	0	0
Sherlock	0	0	1
adventure	1	1	1
exceedingly	1	0	1
strychnine	1	0	0
subsided	1	0	1

- 1 denotes that the word appears in that document, and a 0 denotes that it does not

Inverted indexing

Matrix-building is done offline, i.e., before a search engine is queried

- We derive a representation which is faster for page look-up, namely an **inverted index**
- e.g., assuming every document has a unique ID:

Poirot	→1, 4, 13, 15, 45, ...
Sherlock	→3, 111, ...
adventure	→1, 2, 3, 4, 5, 9, 15, ...
exceedingly	→1, 3, 11, 25, ...
strychnine	→1, 15, 60, ...
subsided	→1, 3, 12, 13, 25, ...

- Each term now points to a list of documents that it appears in
- To search for, e.g., *strychnine*, we have an immediate list of documents that it appears in

Ranking of results

- Ideally, the webpages matching a query are returned as an ordered list based on a page's **relevance**.
- How can a search engine, which does not understand language, determine the relevance of a particular page?

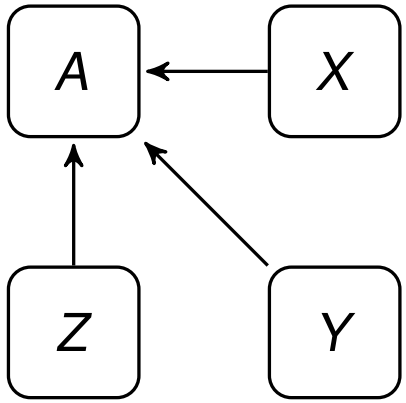
Information used to rank results

- Counting up the number of times a term is used
  - i.e., replace earlier 1s/0s with counts
  - can also upweight words in titles or metadata
- Paying attention to user behavior
  - e.g., counting how often a web result was clicked on by a user (**click-through measurement**)
  - can be for users in general or for specific users
- Paying attention to external information
  - e.g., bonuses/penalties for known sites of high/low quality
- Counting the number of links to and from a page, to determine how popular a page is.
  - As a result, unpopular or new pages require a more specific query to be found.
  - So, we turn to this **weblinking** ...

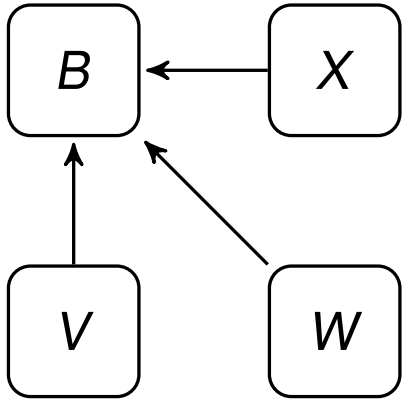
Weblinking

Example of how pages link

In this example, pages X, Y, and Z all link to page A.



Are these links better or worse than the links to page B?



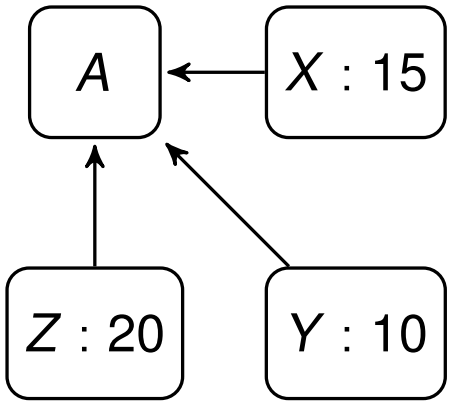
That depends on how popular, or authoritative, the links are.

Weblinking

Figuring popularity

- In order to compare how popular website A is as compared to how popular website B is, we can add up how popular each incoming site is.
- It's like each site that links to A gets to vote for A, but they get so many votes based on how popular each one of them is.

e.g., X casts 15 votes for A, Y casts 10, and Z casts 20:

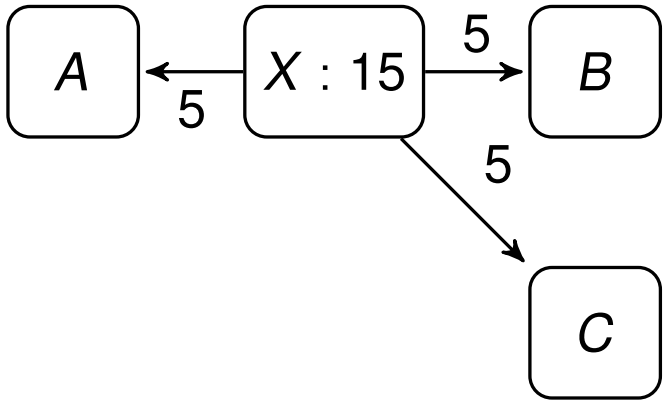


But now A has 45 votes. That's too many!

Weblinking

Factoring out outgoing links

The solution Google uses (called PageRank) is to spread out each page's votes through all the pages it links to.



So, after spreading votes out among their different webpages, let's say A's final score is: 12.

- On its own, the number means nothing.
- But if we compare the number with website B, which we'll say has a score of 10, A is more "authoritative"

Weblinking

Ranking with weblinks

To sum, there are two main things to consider when calculating a ranking for a website based on its weblinks:

- Links coming in
- Links going out

The formula (for Google) is as follows, where R(A) means *rank of page A*; C(X) means *number of pages going out of X*

$$R(A) = \frac{R(X)}{C(X)} + \frac{R(Y)}{C(Y)} + \frac{R(Z)}{C(Z)} \ldots$$

Weblinking

Explanation of weblinks ranking formula

1. We add up all the pages *coming into* page A because to know how popular A is, we need to know how popular everyone else thinks it is.
2. We divide by the pages *going out* of X, Y, and Z because we're spreading out its weight among all the pages they link to.
   - If we didn't divide, page A would have a huge ranking.

This tells us how "popular" a site is, which is one factor used in ranking results

Semi-structured data

Semi-structured data contains some categorization, but is not fully structured

- e.g., Wikipedia entries, reddit, Internet Movie Database (http://www.imdb.com)
  - Since users add much of the content, the way it is structured and categorized varies from user to user
- Crucially, even with some structure, the information one may wish to search for is unstructured

Compare pages of two actors on IMDB (as of February 25, 2015):

- *Nancy Cartwright (I)*, but *Yeardley Smith* (no *(I)*)
- Bio pages list salaries, but only for some works

Semi-structured example

IMDB

Some snippets of trivia about Yeardley Smith:

Spouse (2)

Daniel Erickson (18 May 2001 - 8 September 2008) (divorced)
Christopher Grove (1990 - 1992) (divorced)

Trade Mark (2)

Best known as the voice of ”Lisa Simpson” on the TV show “The Simpsons” (1989)
High pitched, nasal voice

- To search for dates, they come in different formats with different information: *1989*, *8 September 2008*
  - Likely also dates listed on IMDB in the format *September 8, 2008* or *September 8, 2008*
- No field for “best-known-as”, yet information is there

How would we do a search for actor “best known as”?

Semi-structured example

IMDB

Consider this snippet of trivia about Nancy Cartwright:

- "Attended Ohio University from 1976-1978 as an interpersonal communication major and ..."

What if we want to find where actors went to school?

- We have to find patterns like `X University`, `Y College`, `University of Z`, & misspelled variants

Consider Parker Posey's bio:

- "Parker attended high school at R. H. Watkins High School in Laurel, Mississippi, and college at the prestigious SUNY Purchase."
- University appears as *U* within an abbreviation.

⇒ We are describing a search for, not just specific strings, but for *patterns* in the data

Motivating regular expressions

If one wants to be able to describe more complex patterns of words and text, sometimes boolean expressions aren't enough:

- In a large document I want to find addresses with a zip code starting with 911 (around Pasadena, CA); but clearly we would not want to report back all occurrences of emergency phone numbers in the document.
- I want to find all Indiana email addresses which occur in a long text.

Anything where you have to match a complex pattern so-called **regular expressions** are useful.

Regular expressions: What they are

- A regular expression is a compact description of a set of strings, i.e., a language
  - They can be used to search for occurrences of these strings
- Regular expressions can only describe so-called **regular languages**.
  - This means that some patterns cannot be specified using regular expressions
- Note that just like any other formalism, regular expressions as such have no linguistic contents, but they can be used to refer to strings encoding a **natural language** text.

Regular expressions: Tools that use them

- A variety of unix tools (grep, sed, . . .), editors (emacs, jEdit, . . .), and programming languages (perl, python, Java, . . .) incorporate regular expressions.
- Implementations are very efficient so that large text files can be searched quickly; but still becoming efficient enough for web searching
- The various tools and languages differ w.r.t. the exact syntax of the regular expressions they allow.

The syntax of regular expressions (I)

Regular expressions consist of

- strings of literal characters: `f`, `dude2`, `30 years!`, `natural language`
- disjunction:
  - ordinary disjunction: `devoured|ate`, `famil(y|ies)`
  - character classes: `[Tt]he`, `bec[oa]me`
  - ranges: `[A-Z]` (any capital letter)
- negation:
  `[^a]` (any symbol but `a`)
  `[^A-Z0-9]` (not an uppercase letter or number)

The syntax of regular expressions (II)

- counters
  - optionality: `?`
    `colou?r`
  - any number of occurrences: `*` (Kleene star)
    `[0-9]* years`
  - at least one occurrence: `+`
    `[0-9]+ dollars`
- wildcard for any character: `.`
  `beg.n` for any character in between `beg` and `n`

The syntax of regular expressions (III)

- Escaped characters: to specify a character with a special meaning (*, +, ?, (,), |, [,]) it is preceded by a backslash (\)
  e.g., a period is expressed as \.
- Operator precedence, from highest to lowest:

  parentheses ()
  counters * + ?
  character sequences
  disjunction |

Grep

- grep is a powerful and efficient program for searching in text files using regular expressions.
- It is standard on Unix, Linux, and Mac OSX, and there also are various ports to Windows (e.g., http://gnuwin32.sourceforge.net/packages/grep.htm, http://www.interlog.com/~tcharron/grep.html or http://www.wingrep.com/).
- The version of grep that supports the full set of operators mentioned above is generally called egrep (for extended grep).

Grep: Examples for using regular expressions (I)

In the following, we assume a text file `f.txt` containing, among others, the strings that we mention as matching.

- Strings of literal characters:
  `egrep 'and' f.txt` matches and, `Ayn Rand`, `Candy` and so on
- Character classes:
  `egrep 'the year [0-9][0-9][0-9][0-9]' f.txt` matches `the year 1776`, `the year 1812`, `the year 2001`, and so on
- Escaped characters:
  `egrep 'why\?' f.txt` matches `why?`, whereas
  `egrep 'why?' f.txt` matches `why` and `wh`

Grep: Examples for using regular expressions (II)

- disjunction (|): `egrep 'couch|sofa' f.txt` matches `couch` or `sofa`
- grouping with parentheses:
  `egrep 'un(interest|excit)ing' f.txt` matches `uninteresting` or `unexciting`.
- Any character (.):
  `egrep 'o.e' f.txt` matches `ore`, `one`, `ole`

Grep: Examples for using regular expressions (III)

- Kleene star (*):
  `egrep 'a*rgh' f.txt` matches `argh`, `aargh`, `aaargh`
  `egrep 'sha(la)*' f.txt` matches `sha`, `shala`, `shalala`, or `shalalalalalalalala`
- One or more (+):
  `egrep 'john+y' f.txt` matches `johny`, `johnny`, ..., but not `johy`
- Optionality (?):
  `egrep 'joh?n' f.txt` matches `jon` and `john`

Corpora

- A **corpus** is a collection of text.
- Corpora with the works of various writers, newspaper texts, etc. have been collected and electronically encoded.
- Corpora can be quite large
- The British National Corpus is a 100 million word collection representing a wide cross-section of current written and spoken British English.
- Another example is the European Parliament Proceedings Parallel Corpus 1996–2003.

Corpora sometimes have interfaces allowing for regular expression searching

- Good (non-RE) search interface: http://corpus.byu.edu