Search Engines

Information Retrieval in Practice

Annotations by Michael L. Nelson

All slides ©Addison Wesley, 2008
Web Crawler

• Finds and downloads web pages automatically
  – provides the collection for searching
• Web is huge and constantly growing
• Web is not under the control of search engine providers
• Web pages are constantly changing
• Crawlers also used for other types of data
Retrieving Web Pages

• Every page has a unique *uniform resource locator* (URL)

• Web pages are stored on web servers that use HTTP to exchange information with client software

• e.g.,

```
http://www.cs.umass.edu/csinfo/people.html
```

Retrieving Web Pages

• Web crawler client program connects to a domain name system (DNS) server
• DNS server translates the hostname into an internet protocol (IP) address
• Crawler then attempts to connect to server host using specific port
• After connection, crawler sends an HTTP request to the web server to request a page — usually a GET request
Web Crawler

• Starts with a set of *seeds*, which are a set of URLs given to it as parameters
• Seeds are added to a URL request queue
• Crawler starts fetching pages from the request queue
• Downloaded pages are parsed to find link tags that might contain other useful URLs to fetch
• New URLs added to the crawler’s request queue, or *frontier*
• Continue until no more new URLs or disk full
Web Crawling

• Web crawlers spend a lot of time waiting for responses to requests
• To reduce this inefficiency, web crawlers use threads and fetch hundreds of pages at once
• Crawlers could potentially flood sites with requests for pages
• To avoid this problem, web crawlers use \textit{politeness policies}
  – e.g., delay between requests to same web server
Controlling Crawling

• Even crawling a site slowly will anger some web server administrators, who object to any copying of their data

• Robots.txt file can be used to control crawlers

User-agent: *
Disallow: /private/
Disallow: /confidential/
Disallow: /other/
Allow: /other/public/

User-agent: FavoredCrawler
Disallow:

Sitemap: http://mysite.com/sitemap.xml.gz
Simple Crawler Thread

```plaintext
procedure CRAWLER_THREAD(frontier)
    while not frontier.done() do
        website ← frontier.nextSite()
        url ← website.nextURL()
        if website.permitsCrawl(url) then
            text ← retrieveURL(url)
            storeDocument(url, text)
            for each url in parse(text) do
                frontier.addURL(url)
            end for
        end if
    end while
end procedure
```
Freshness

• Web pages are constantly being added, deleted, and modified
• Web crawler must continually revisit pages it has already crawled to see if they have changed in order to maintain the *freshness* of the document collection
  – *stale* copies no longer reflect the real contents of the web pages
Freshness

- HTTP protocol has a special request type called HEAD that makes it easy to check for page changes
  - returns information about page, not page itself

Client request: HEAD /csinfo/people.html HTTP/1.1
Host: www.cs.umass.edu

HTTP/1.1 200 OK
Date: Thu, 03 Apr 2008 05:17:54 GMT
Server: Apache/2.0.52 (CentOS)
Last-Modified: Fri, 04 Jan 2008 15:28:39 GMT

Server response: ETag: "239c33-2576-2a2837c0"
Accept-Ranges: bytes
Content-Length: 9590
Connection: close
Content-Type: text/html; charset=ISO-8859-1

See also: http://ws-dl.blogspot.com/2014/05/2014-05-08-support-for-various-http.html
Freshness

• Not possible to constantly check all pages
  – must check important pages and pages that change frequently
• Freshness is the proportion of pages that are fresh
• Optimizing for this metric can lead to bad decisions, such as not crawling popular sites
• *Age* is a better metric
Freshness vs. Age

missed update!
if we're a SE index,
we don't care
if we're an archive,
we care
Age

• Expected age of a page $t$ days after it was last crawled:

$$\text{Age}(\lambda, t) = \int_0^t P(\text{page changed at time } x)(t - x)dx$$

• Web page updates follow the Poisson distribution on average
  – time until the next update is governed by an exponential distribution

$$\text{Age}(\lambda, t) = \int_0^t \lambda e^{-\lambda x} (t - x)dx$$
Age

- Older a page gets, the more it costs not to crawl it
  - e.g., expected age with mean change frequency $\lambda = 1/7$ (one change per week)

\[
\text{wait 7 days, expected age } \approx 2.6 \text{ days}
\]
Focused Crawling

• Attempts to download only those pages that are about a particular topic
  – used by *vertical search* applications

• Rely on the fact that pages about a topic tend to have links to other pages on the same topic
  – popular pages for a topic are typically used as seeds

• Crawler uses *text classifier* to decide whether a page is on topic
Deep Web

• Sites that are difficult for a crawler to find are collectively referred to as the *deep* (or *hidden*) *Web*
  – much larger than conventional Web

• Three broad categories:
  – private sites
    • no incoming links, or may require log in with a valid account
  – form results
    • sites that can be reached only after entering some data into a form
  – scripted pages
    • pages that use JavaScript, Flash, or another client-side language to generate links

Sitemaps

- Sitemaps contain lists of URLs and data about those URLs, such as modification time and modification frequency
- Generated by web server administrators
- Tells crawler about pages it might not otherwise find
- Gives crawler a hint about when to check a page for changes
Sitemap Example

<?xml version="1.0" encoding="UTF-8"?>
<urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">
  <url>
    <loc>http://www.company.com/</loc>
    <lastmod>2008-01-15</lastmod>
    <changefreq>monthly</changefreq>
    <priority>0.7</priority>
  </url>
  <url>
    <loc>http://www.company.com/items?item=truck</loc>
    <changefreq>weekly</changefreq>
  </url>
  <url>
    <loc>http://www.company.com/items?item=bicycle</loc>
    <changefreq>daily</changefreq>
  </url>
</urlset>
Distributed Crawling

• Three reasons to use multiple computers for crawling
  – Helps to put the crawler closer to the sites it crawls
  – Reduces the number of sites the crawler has to remember
  – Reduces computing resources required

• Distributed crawler uses a hash function to assign URLs to crawling computers
  – hash function should be computed on the host part of each URL
Desktop Crawls

• Used for desktop search and enterprise search
• Differences to web crawling:
  – Much easier to find the data
  – Responding quickly to updates is more important
  – Must be conservative in terms of disk and CPU usage
  – Many different document formats
  – Data privacy very important
Document Feeds

• Many documents are *published*
  – created at a fixed time and rarely updated again
  – e.g., news articles, blog posts, press releases, email

• Published documents from a single source can be ordered in a sequence called a *document feed*
  – new documents found by examining the end of the feed
Document Feeds

• Two types:
  – A *push feed* alerts the subscriber to new documents
  – A *pull feed* requires the subscriber to check periodically for new documents

• Most common format for pull feeds is called *RSS*
  – Really Simple Syndication, RDF Site Summary, Rich Site Summary, or ...
RSS Example

<?xml version="1.0"?>
<rss version="2.0">
  <channel>
    <title>Search Engine News</title>
    <link>http://www.search-engine-news.org/</link>
    <description>News about search engines.</description>
    <language>en-us</language>
    <pubDate>Tue, 19 Jun 2008 05:17:00 GMT</pubDate>
    <ttl>60</ttl>

    <item>
      <title>Upcoming SIGIR Conference</title>
      <link>http://www.sigir.org/conference</link>
      <description>The annual SIGIR conference is coming! Mark your calendars and check for cheap flights.</description>
      <pubDate>Tue, 05 Jun 2008 09:50:11 GMT</pubDate>
      <guid>http://search-engine-news.org#500</guid>
    </item>
  </channel>
</rss>
RSS Example

...<item>
<title>New Search Engine Textbook</title>
<link>http://www.cs.umass.edu/search-book</link>
<description>A new textbook about search engines will be published soon.</description>
<pubDate>Tue, 05 Jun 2008 09:33:01 GMT</pubDate>
<guid>http://search-engine-news.org#499</guid>
</item>
</channel>
</rss>
RSS

• **ttl** tag (time to live)
  – amount of time (in minutes) contents should be cached

• RSS feeds are accessed like web pages
  – using HTTP GET requests to web servers that host them

• Easy for crawlers to parse

• Easy to find new information
Conversion

• Text is stored in hundreds of incompatible file formats
  – e.g., raw text, RTF, HTML, XML, Microsoft Word, ODF, PDF
• Other types of files also important
  – e.g., PowerPoint, Excel
• Typically use a conversion tool
  – converts the document content into a tagged text format such as HTML or XML
  – retains some of the important formatting information
Character Encoding

• A character encoding is a mapping between bits and glyphs
  – i.e., getting from bits in a file to characters on a screen
  – Can be a major source of incompatibility

• ASCII is basic character encoding scheme for English
  – encodes 128 letters, numbers, special characters, and control characters in 7 bits, extended with an extra bit for storage in bytes
Character Encoding

• Other languages can have many more glyphs
  – e.g., Chinese has more than 40,000 characters, with over 3,000 in common use

• Many languages have multiple encoding schemes
  – e.g., CJK (Chinese-Japanese-Korean) family of East Asian languages, Hindi, Arabic
    – must specify encoding
    – can’t have multiple languages in one file

• Unicode developed to address encoding problems
Unicode

• Single mapping from numbers to glyphs that attempts to include all glyphs in common use in all known languages

• Unicode is a mapping between numbers and glyphs
  – does not uniquely specify bits to glyph mapping!
  – e.g., UTF-8, UTF-16, UTF-32
Unicode

• Proliferation of encodings comes from a need for compatibility and to save space
  – UTF-8 uses one byte for English (ASCII), as many as 4 bytes for some traditional Chinese characters
  – variable length encoding, more difficult to do string operations
  – UTF-32 uses 4 bytes for every character

• Many applications use UTF-32 for internal text encoding (fast random lookup) and UTF-8 for disk storage (less space)
Unicode

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Hexadecimal</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>0–127</td>
<td>0–7F</td>
<td>0xxxxxxx</td>
</tr>
<tr>
<td>128–2047</td>
<td>80–7FF</td>
<td>110xxxxx 10xxxxx</td>
</tr>
<tr>
<td>2048–55295</td>
<td>800–D7FF</td>
<td>1110xxxx 10xxxxx 10xxxxx</td>
</tr>
<tr>
<td>55296–57343</td>
<td>D800–DFFF</td>
<td>Undefined</td>
</tr>
<tr>
<td>57344–65535</td>
<td>E000–FFFF</td>
<td>1110xxxx 10xxxxx 10xxxxx</td>
</tr>
<tr>
<td>65536–1114111</td>
<td>10000–10FFFF</td>
<td>11110xxx 10xxxxx 10xxxxx 10xxxxx</td>
</tr>
</tbody>
</table>

- e.g., Greek letter pi (π) is Unicode symbol number 960
- In binary, 00000011 11000000 (3C0 in hexadecimal)
- Final encoding is 11001111 10000000 (CF80 in hexadecimal)
Storing the Documents

• Many reasons to store converted document text
  – saves crawling time when page is not updated
  – provides efficient access to text for snippet generation, information extraction, etc.

• Database systems can provide document storage for some applications
  – web search engines use customized document storage systems
Storing the Documents

• Requirements for document storage system:
  – Random access
    • request the content of a document based on its URL
    • hash function based on URL is typical
  – Compression and large files
    • reducing storage requirements and efficient access
  – Update
    • handling large volumes of new and modified documents
    • adding new anchor text
Large Files

• Store many documents in large files, rather than each document in a file
  – avoids overhead in opening and closing files
  – reduces seek time relative to read time

• Compound documents formats
  – used to store multiple documents in a file
  – e.g., TREC Web
<DOC>
<DOCN>WTX001-B01-10</DOCN>
<DOCHDR>
http://www.example.com/test.html 204.244.59.33 19970101013145 text/html 440
HTTP/1.0 200 OK
Date: Wed, 01 Jan 1997 01:21:13 GMT
Server: Apache/1.0.3
Content-type: text/html
Content-length: 270
Last-modified: Mon, 25 Nov 1996 05:31:24 GMT
</DOCHDR>
<HTML>
<TITLE>Tropical Fish Store</TITLE>
Coming soon!
</HTML>
</DOC>

<DOC>
<DOCN>WTX001-B01-109</DOCN>
<DOCHDR>
http://www.example.com/fish.html 204.244.59.33 19970101013149 text/html 440
HTTP/1.0 200 OK
Date: Wed, 01 Jan 1997 01:21:19 GMT
Server: Apache/1.0.3
Content-type: text/html
Content-length: 270
Last-modified: Mon, 25 Nov 1996 05:31:24 GMT
</DOCHDR>
<HTML>
<TITLE>Fish Information</TITLE>
This page will soon contain interesting information about tropical fish.
</HTML>
</DOC>
Compression

• Text is highly redundant (or predictable)
• Compression techniques exploit this redundancy to make files smaller without losing any of the content
• Compression of indexes covered later
• Popular algorithms can compress HTML and XML text by 80%
  – e.g., DEFLATE (zip, gzip) and LZW (UNIX compress, PDF)
  – may compress large files in blocks to make access faster
BigTable

- Google’s document storage system
  - Customized for storing, finding, and updating web pages
  - Handles large collection sizes using inexpensive computers
BigTable

- No query language, no complex queries to optimize
- Only row-level transactions
- Tablets are stored in a replicated file system that is accessible by all BigTable servers
- Any changes to a BigTable tablet are recorded to a transaction log, which is also stored in a shared file system
- If any tablet server crashes, another server can immediately read the tablet data and transaction log from the file system and take over
BigTable

• Logically organized into rows
• A row stores data for a single web page

• Combination of a row key, a column key, and a timestamp point to a single cell in the row
• **BigTable**

  - **BigTable** can have a huge number of columns per row
    - all rows have the same column groups
    - not all rows have the same columns
    - important for reducing disk reads to access document data

• **Rows** are partitioned into tablets based on their row keys
  - simplifies determining which server is appropriate
Detecting Duplicates

• Duplicate and near-duplicate documents occur in many situations
  – Copies, versions, plagiarism, spam, mirror sites
  – 30% of the web pages in a large crawl are exact or near duplicates of pages in the other 70%

• Duplicates consume significant resources during crawling, indexing, and search
  – Little value to most users
  – **MLN edit:** (near-)duplicates can replace 404s

Duplicate Detection

• *Exact* duplicate detection is relatively easy

• *Checksum* techniques
  
  – A checksum is a value that is computed based on the content of the document
    
    • e.g., sum of the bytes in the document file

    | Tropical fish | fish | Sum |
    |---------------|------|-----|
    | 54 72 6F 70 69 63 61 6C 20 66 69 73 68 | 508 |

  – Possible for files with different text to have same checksum

• Functions such as a *cyclic redundancy check* (CRC), have been developed that consider the positions of the bytes
Near-Duplicate Detection

• More challenging task
  – Are web pages with same text context but different advertising or format near-duplicates?

• A near-duplicate document is defined using a threshold value for some similarity measure between pairs of documents
  – e.g., document $D1$ is a near-duplicate of document $D2$ if more than 90% of the words in the documents are the same

Opposite but related problem: discovering when pages go “off-topic”
Near-Duplicate Detection

• **Search:**
  – find near-duplicates of a document $D$
  – $O(N)$ comparisons required

• **Discovery:**
  – find all pairs of near-duplicate documents in the collection
  – $O(N^2)$ comparisons

• IR techniques are effective for search scenario

• For discovery, other techniques used to generate compact representations
Fingerprints

1. The document is parsed into words. Non-word content, such as punctuation, HTML tags, and additional whitespace, is removed.

2. The words are grouped into contiguous $n$-grams for some $n$. These are usually overlapping sequences of words, although some techniques use non-overlapping sequences.

3. Some of the n-grams are selected to represent the document.

4. The selected n-grams are hashed to improve retrieval efficiency and further reduce the size of the representation.

5. The hash values are stored, typically in an inverted index.

6. Documents are compared using overlap of fingerprints
Fingerprint Example

Tropical fish include fish found in tropical environments around the world, including both freshwater and salt water species.

(a) Original text

tropical fish include, fish include fish, include fish found, fish found in, found in tropical, in tropical environments, tropical environments around, environments around the, around the world, the world including, world including both, including both freshwater, both freshwater and, freshwater and salt, and salt water, salt water species

(b) 3-grams

938 664 463 822 492 798 78 969 143 236 913 908 694 553 870 779

(c) Hash values

664 492 236 908

(d) Selected hash values using $0 \mod 4$
Simhash

• Similarity comparisons using word-based representations more effective at finding near-duplicates
  – Problem is efficiency
• Simhash combines the advantages of the word-based similarity measures with the efficiency of fingerprints based on hashing
• Similarity of two pages as measured by the cosine correlation measure is proportional to the number of bits that are the same in the simhash fingerprints
Simhash

1. Process the document into a set of features with associated weights. We will assume the simple case where the features are words weighted by their frequency.

2. Generate a hash value with $b$ bits (the desired size of the fingerprint) for each word. The hash value should be unique for each word.

3. In $b$-dimensional vector $V$, update the components of the vector by adding the weight for a word to every component for which the corresponding bit in the word’s hash value is 1, and subtracting the weight if the value is 0.

4. After all words have been processed, generate a $b$-bit fingerprint by setting the $i$th bit to 1 if the $i$th component of $V$ is positive, or 0 otherwise.
Simhash Example

Tropical fish include fish found in tropical environments around the world, including both freshwater and salt water species.

(a) Original text

tropical 2  fish 2  include 1  found 1  environments 1  around 1  world 1  including 1  both 1  freshwater 1  salt 1  water 1  species 1

(b) Words with weights

first position:
-2 tropical 01100001 fish 210101011 include 111001110
-1 found 000111110 environments 100101101 around 110001011
-1 world 001010110 including 111000000 both 101011110
-1 freshwater 001111111 salt 110110101 water -100100101
1 species 111011111 3 3

(c) 8 bit hash values

-4+3+2 =1  1 -5 9 -9 3 1 3 3

(d) Vector $V$ formed by summing weights

1 0 1 0 1 1 1 1

(e) 8-bit fingerprint formed from $V$
Removing Noise

• Many web pages contain text, links, and pictures that are not directly related to the main content of the page.

• This additional material is mostly noise that could negatively affect the ranking of the page.

• Techniques have been developed to detect the content blocks in a web page.
  – Non-content material is either ignored or reduced in importance in the indexing process.
See also: https://code.google.com/p/boilerpipe/
Finding Content Blocks

• Cumulative distribution of tags in the example web page

– Main text content of the page corresponds to the “plateau” in the middle of the distribution
Finding Content Blocks

• Represent a web page as a sequence of bits, where $b_n = 1$ indicates that the $n$th token is a tag

• Optimization problem where we find values of $i$ and $j$ to maximize both the number of tags below $i$ and above $j$ and the number of non-tag tokens between $i$ and $j$

• i.e., maximize

$$\sum_{n=0}^{i-1} b_n + \sum_{n=i}^{j} (1 - b_n) + \sum_{n=j+1}^{N-1} b_n$$

left-hand part of the curve  flat part of the curve  right-hand part of the curve
Finding Content Blocks

• Other approaches use DOM structure and visual (layout) features