Outline

1 Web crawling
   - Discovering new URLs
   - Identifying duplicates
   - Crawling architecture

2 Web Information Retrieval

3 Web Graph Mining

4 Conclusion
Web Crawlers

- **Crawlers, (Web) spiders, (Web) robots**: autonomous user agents that retrieve pages from the Web.
- **Basics of crawling**:
  1. Start from a given URL or set of URLs
  2. Retrieve and process the corresponding page
  3. Discover new URLs (cf. next slide)
  4. Repeat on each found URL
- No real termination condition (virtual unlimited number of Web pages!)
- **Graph-browsing problem**
  - **Deep-first**: not very adapted, possibility of being lost in robot traps
  - **Breadth-first**
  - **Combination of both**: breadth-first with limited-depth deep-first on each discovered website
Sources of new URLs

- From HTML pages:
  - hyperlinks `<a href="...">...</a>`
  - media `<img src="...">` `<embed src="...">` `<object data="...">`
  - frames `<frame src="...">` `<iframe src="...">`
  - JavaScript links `window.open("...")`
  - etc.

- Other hyperlinked content (e.g., PDF files)

- Non-hyperlinked URLs that appear anywhere on the Web (in HTML text, text files, etc.): use regular expressions to extract them

- Referrer URLs

- Sitemaps [sit08]
Scope of a crawler

- **Web-scale**
  - The Web is infinite! Avoid robot traps by putting depth or page number limits on each Web server
  - Focus on important pages [APC03] (cf. lecture on the Web graph)

- Web servers under a list of **DNS domains**: easy filtering of URLs

- A given topic: **focused crawling techniques** [CvdBD99, DCL+00] based on classifiers of Web page content and predictors of the interest of a link.

- The national Web (cf. **public deposit**, national libraries): what is this? [ACMS02]

- A given Web site: what is a Web site? [Sen05]
A word about hashing

**Definition**

A **hash function** is a deterministic mathematical function transforming objects (numbers, character strings, binary...) into fixed-size, seemingly random, numbers. The more random the transformation is, the better.

**Example**

Java hash function for the `String` class:

\[
\sum_{i=0}^{n-1} s_i \times 31^{n-i-1} \mod 2^{32}
\]

where \( s_i \) is the (Unicode) code of character \( i \) of a string \( s \).
Identification of duplicate Web pages

Problem

Identifying duplicates or near-duplicates on the Web to prevent multiple indexing

trivial duplicates: same resource at the same canonized URL:

http://example.com:80/toto
http://example.com/titi/../toto

exact duplicates: identification by hashing

near-duplicates: (timestamps, tip of the day, etc.) more complex!
Near-duplicate detection

**Edit distance.** Count the *minimum number of basic modifications* (additions or deletions of characters or words, etc.) to obtain a document from another one. Good measure of similarity, and can be computed in $O(mn)$ where $m$ and $n$ are the size of the documents. But: does not scale to a large collection of documents (unreasonable to compute the edit distance for every pair!).

**Shingles.** Idea: two documents similar if they mostly share the same *succession of $k$-grams* (succession of tokens of length $k$).

**Example**

I like to watch the sun set with my friend.
My friend and I like to watch the sun set.

$S = \{i \text{ like, like to, my friend, set with, sun set, the sun, to watch, watch the, with my}\}$

$T = \{\text{and i, friend and, i like, like to, my friend, sun set, the sun, to watch, watch the}\}$
Hashing shingles to detect duplicates [BGMZ97]

- Similarity: *Jaccard coefficient* on the set of shingles:
  \[
  J(S, T) = \frac{|S \cap T|}{|S \cup T|}
  \]

- Still *costly to compute!* But can be approximated as follows:
  1. Choose \( N \) different hash functions
  2. For each hash function \( h_i \) and each set of shingles \( S_k = \{s_{k1} \ldots s_{kn}\} \), store \( \phi_{ik} = \min_j h_i(s_{kj}) \)
  3. Approximate \( J(S_k, S_l) \) as the proportion of \( \phi_{ik} \) and \( \phi_{il} \) that are equal

- Possibly to repeat in a hierarchical way with super-shingles (we are only interested in *very* similar documents)
Crawling ethics

- Standard for robot exclusion: `robots.txt` at the root of a Web server [Kos94].
  
  **User-agent:** *
  **Allow:** /searchhistory/
  **Disallow:** /search

- Per-page exclusion (*de facto* standard).
  
  `<meta name="ROBOTS" content="NOINDEX, NOFOLLOW">`

- Per-link exclusion (*de facto* standard).
  
  `<a href="toto.html" rel="nofollow">Toto</a>`

- Avoid Denial Of Service (DOS), wait 100ms/1s between two repeated requests to the same Web server
Parallel processing

Network delays, waits between requests:

- **Per-server queue** of URLs
- **Parallel processing of requests to different hosts:**
  - multi-threaded programming
  - asynchronous inputs and outputs (*select*, classes from `java.util.concurrent`): less overhead
- Use of *keep-alive* to reduce connexion overheads
Refreshing URLs

- Content on the Web changes
- Different change rates:
  - online newspaper main page: every hour or so
  - published article: virtually no change
- Continuous crawling, and identification of change rates for adaptive crawling:
  - If-Last-Modified HTTP feature (not reliable)
  - Identification of duplicates in successive request
Outline

1. Web crawling

2. Web Information Retrieval
   - Text Preprocessing
   - Inverted Index
   - Answering Keyword Queries

3. Web Graph Mining

4. Conclusion
Information Retrieval, Search

Problem

*How to index Web content so as to answer (keyword-based) queries efficiently?*

Context: set of text documents

1. $d_1$ The jaguar is a New World mammal of the Felidae family.
2. $d_2$ Jaguar has designed four new engines.
3. $d_3$ For Jaguar, Atari was keen to use a 68K family device.
4. $d_4$ The Jacksonville Jaguars are a professional US football team.
5. $d_5$ Mac OS X Jaguar is available at a price of US $199 for Apple’s new “family pack”.
6. $d_6$ One such ruling family to incorporate the jaguar into their name is Jaguar Paw.
7. $d_7$ It is a big cat.
Text Preprocessing

Initial text preprocessing steps

- Number of optional steps
- Highly depends on the application
- Highly depends on the document language (illustrated with English)
Language Identification

How to find the language used in a document?

- Meta-information about the document: often not reliable!
- **Unambiguous** scripts or letters: not very common!

한글
カタカナ
Għarbi
porn
Language Identification

How to find the language used in a document?

- Meta-information about the document: often not reliable!
- **Unambiguous** scripts or letters: not very common!
  - 한글
  - カタカナ
  - 괌
  - Għarbi
  - þorn

**Respectively:** Korean Hangul, Japanese Katakana, Maldivian Dhivehi, Maltese, Icelandic

- Extension of this: frequent characters, or, better, frequent *k*-grams
- Use standard machine learning techniques (**classifiers**)
Tokenization

Principle
Separate text into tokens (words)

Not so easy!

- In some languages (Chinese, Japanese), words not separated by whitespace
- Deal consistently with acronyms, elisions, numbers, units, URLs, emails, etc.
- **Compound words:** `hostname`, `host-name` and `host name`. Break into two tokens or regroup them as one token? In any case, lexicon and linguistic analysis needed! Even more so in other languages as German.

Usually, remove punctuation and normalize case at this point
Tokenization: Example

\[ d_1 \] the jaguar is a new world mammal of the felidae family.

\[ d_2 \] jaguar has designed four new engines.

\[ d_3 \] for jaguar atari was keen to use a 68k family device.

\[ d_4 \] the jacksonville jaguars are a professional us football team.

\[ d_5 \] mac os x jaguar is available at a price of $199 for apple’s new family pack.

\[ d_6 \] one such ruling family to incorporate the jaguar into their name is jaguar paw.

\[ d_7 \] it is a big cat.
Stemming

**Principle**

- Merge different forms of the same word, or of closely related words, into a single stem

- Not in all applications!
- Useful for retrieving documents containing *geese* when searching for *goose*
- Various degrees of stemming
- Possibility of building different indexes, with different stemming
Stemming schemes (1/2)

Morphological stemming.

- Remove **bound morphemes** from words:
  - plural markers
  - gender markers
  - tense or mood inflections
  - etc.

- Can be linguistically **very complex**, cf:
  *Les poules du couvent couvent.*
  [The hens of the monastery brood.]

- In English, somewhat **easy**:
  - Remove final -s, ’s, -ed, -ing, -er, -est
  - Take care of semiregular forms (e.g., -y/-ies)
  - Take care of irregular forms (mouse/mice)

- But still some **ambiguities**: cf stocking, rose
Stemming schemes (2/2)

Lexical stemming.

- Merge lexically related terms of various parts of speech, such as policy, politics, political or politician
- For English, Porter’s stemming [Por80]; stem university and universal to univers: not perfect!
- Possibility of coupling this with lexicons to merge (near-)synonyms

Phonetic stemming.

- Merge phonetically related words: search despite spelling errors!
- For English, Soundex [US 07] stems Robert and Rupert to R163. Very coarse!
Stemming Example

d_1 the_1 jaguar_2 be_3 a_4 new_5 world_6 mammal_7 of_8 the_9 felidae_10 family_11
d_2 jaguar_1 have_2 design_3 four_4 new_5 engine_6
d_3 for_1 jaguar_2 atari_3 be_4 keen_5 to_6 use_7 a_8 68k_9 family_10 device_11
d_4 the_1 jacksonville_2 jaguar_3 be_4 a_5 professional_6 us_7 football_8 team_9
d_5 mac_1 os_2 x_3 jaguar_4 be_5 available_6 at_7 a_8 price_9 of_10 us_11 $199_12 for_13 apple_14 new_15 family_16 pack_17
d_6 one_1 such_2 rule_3 family_4 to_5 incorporate_6 the_7 jaguar_8 into_9 their_10 name_11 be_12 jaguar_13 paw_14
d_7 it_1 be_2 a_3 big_4 cat_5
Stop Word Removal

Principle

Remove uninformative words from documents, in particular to lower the cost of storing the index.

determiners:  a, the, this, etc.
function verbs:  be, have, make, etc.
conjunctions:  that, and, etc.
   etc.
Stop Word Removal Example

\[\begin{align*}
\vec{d}_1 & \text{ jaguar}_2 \text{ new}_5 \text{ world}_6 \text{ mammal}_7 \text{ felidae}_10 \text{ family}_11 \\
\vec{d}_2 & \text{ jaguar}_1 \text{ design}_3 \text{ four}_4 \text{ new}_5 \text{ engine}_6 \\
\vec{d}_3 & \text{ jaguar}_2 \text{ atari}_3 \text{ keen}_5 \text{ 68k}_9 \text{ family}_10 \text{ device}_11 \\
\vec{d}_4 & \text{ jacksonville}_2 \text{ jaguar}_3 \text{ professional}_6 \text{ us}_7 \text{ football}_8 \text{ team}_9 \\
\vec{d}_5 & \text{ mac}_1 \text{ os}_2 \text{ x}_3 \text{ jaguar}_4 \text{ available}_6 \text{ price}_9 \text{ us}_11 \text{ $199}_12 \text{ apple}_14 \\
& \text{ new}_15 \text{ family}_16 \text{ pack}_17 \\
\vec{d}_6 & \text{ one}_1 \text{ such}_2 \text{ rule}_3 \text{ family}_4 \text{ incorporate}_6 \text{ jaguar}_8 \text{ their}_10 \text{ name}_11 \\
& \text{ jaguar}_13 \text{ paw}_14 \\
\vec{d}_7 & \text{ big}_4 \text{ cat}_5
\end{align*}\]
Inverted Index construction

After all preprocessing, construction of an inverted index:

- Index of **all terms**, with the list of documents where this term occurs
- Small scale: disk storage, with memory mapping (cf. `mmap`) techniques; secondary index for offset of each term in main index
- Large scale: distributed on a cluster of machines; hashing gives the machine responsible for a given term
- Updating the index is costly, so only batch operations (not one-by-one addition of term occurrences)
Inverted Index Example

family  \( d_1, d_3, d_5, d_6 \)
football  \( d_4 \)
jaguar  \( d_1, d_2, d_3, d_4, d_5, d_6 \)
new  \( d_1, d_2, d_5 \)
rule  \( d_6 \)
us  \( d_4, d_5 \)
world  \( d_1 \)

... 

Note:

- the length of an inverted (posting) list is highly variable – scanning short lists first is an important optimization.
- *entries* are homogeneous: this gives much room for compression.
Storing positions in the index

- phrase queries, NEAR operator: need to keep position information in the index
- just add it in the document list!

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Positions</th>
</tr>
</thead>
<tbody>
<tr>
<td>family</td>
<td>$d_1/11$, $d_3/10$, $d_5/16$, $d_6/4$</td>
</tr>
<tr>
<td>football</td>
<td>$d_4/8$</td>
</tr>
<tr>
<td>jaguar</td>
<td>$d_1/2$, $d_2/1$, $d_3/2$, $d_4/3$, $d_5/4$, $d_6/8 + 13$</td>
</tr>
<tr>
<td>new</td>
<td>$d_1/5$, $d_2/5$, $d_5/15$</td>
</tr>
<tr>
<td>rule</td>
<td>$d_6/3$</td>
</tr>
<tr>
<td>us</td>
<td>$d_4/7$, $d_5/11$</td>
</tr>
<tr>
<td>world</td>
<td>$d_1/6$</td>
</tr>
</tbody>
</table>

...  

⇒ so far, ok for **Boolean** queries: find the documents that contain a set of keywords; reject the other.
**TF-IDF Weighting**

The inverted is extended by adding Term Frequency—Inverse Document Frequency weighting

\[
\text{tfidf}(t, d) = \frac{n_{t,d}}{\sum_{t'} n_{t',d}} \cdot \log \frac{|D|}{|\{d' \in D | n_{t,d'} > 0\}|}
\]

- \(n_{t,d}\) number of occurrences of \(t\) in \(d\)
- \(D\) set of all documents

Documents (along with weight) are stored in decreasing weight order in the index
TF-IDF Weighting Example

<table>
<thead>
<tr>
<th>Term</th>
<th>TF-IDF Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>family</td>
<td>$d_1/11/.13$, $d_3/10/.13$, $d_6/4/.08$, $d_5/16/.07$</td>
</tr>
<tr>
<td>football</td>
<td>$d_4/8/.47$</td>
</tr>
<tr>
<td>jaguar</td>
<td>$d_1/2/.04$, $d_2/1/.04$, $d_3/2/.04$, $d_4/3/.04$, $d_6/8+13/.04$, $d_5/4/.02$</td>
</tr>
<tr>
<td>new</td>
<td>$d_2/5/.24$, $d_1/5/.20$, $d_5/15/.10$</td>
</tr>
<tr>
<td>rule</td>
<td>$d_6/3/.28$</td>
</tr>
<tr>
<td>us</td>
<td>$d_4/7/.30$, $d_5/11/.15$</td>
</tr>
<tr>
<td>world</td>
<td>$d_1/6/.47$</td>
</tr>
</tbody>
</table>

Exercise: take an entry, and check that the tf/idf value is indeed correct (take documents after stop-word removal).
Answering Boolean Queries

- **Single keyword query**: just consult the index and return the documents in index order.

- **Boolean multi-keyword query**

  \[(jaguar \text{ AND } new \text{ AND NOT } family) \text{ OR } cat\]

  Same way! Retrieve document lists from all keywords and apply adequate set operations:

  \[
  \begin{align*}
  &\text{AND } \text{ intersection} \\
  &\text{OR } \text{ union} \\
  &\text{AND NOT } \text{ difference}
  \end{align*}
  \]

- **Global score**: some function of the individual weight (e.g., addition for conjunctive queries)

- **Position queries**: consult the index, and filter by appropriate condition
Answering Top-$k$ Queries

\[ t_1 \text{ AND } \ldots \text{ AND } t_n \]

Problem

Find the top-$k$ results (for some given $k$) to the query, without retrieving all documents matching it.

Notations:

\[ s(t, d) \] weight of $t$ in $d$ (e.g., tfidf)

\[ g(s_1, \ldots, s_n) \] monotonous function that computes the global score (e.g., addition)
The Threshold Algorithm

1. Let $R$ be the empty list, and $m = +\infty$.
2. For each $1 \leq i \leq n$:
   1. Retrieve the document $d^{(i)}$ containing term $t_i$ that has the next largest $s(t_i, d^{(i)})$.
   2. Compute its global score $g_{d^{(i)}} = g(s(t_1, d^{(i)}), \ldots, s(t_n, d^{(i)}))$ by retrieving all $s(t_j, d^{(i)})$ with $j \neq i$.
   3. If $R$ contains less than $k$ documents, or if $g_{d^{(i)}}$ is greater than the minimum of the score of documents in $R$, add $d^{(i)}$ to $R$.
3. Let $m = g(s(t_1, d^{(1)}), s(t_2, d^{(2)}), \ldots, s(t_n, d^{(n)}))$.
4. If $R$ contains more than $k$ documents, and the minimum of the score of the documents in $R$ is greater than or equal to $m$, return $R$.
5. Redo step 2.
The TA, by example

$q = "new OR family"$, and $k = 3$. We use inverted lists sorted on the weight.

- **family**: $d_1/11/.13$, $d_3/10/.13$, $d_6/4/.08$, $d_5/16/.07$
- **new**: $d_2/5/.24$, $d_1/5/.20$, $d_5/15/.10$

... Initially, $R = \emptyset$ and $\tau = +\infty$.

1. $d^{(1)}$ is the first entry in $L_{\text{family}}$, one finds $s(\text{new}, d_1) = .20$; the global score for $d_1$ is $.13 + .20 = .33$.
2. Next, $i = 2$, and one finds that the global score for $d_2$ is .24.
3. The algorithm quits the loop on $i$ with $R = \langle [d_1, .33], [d_2, .24] \rangle$ and $\tau = .13 + .24 = .37$.
4. We proceed with the loop again, taking $d_3$ with score .13 and $d_5$ with score .17. $[d_5, .17]$ is added to $R$ (at the end) and $\tau$ is now $.10 + .13 = .23$. A last loop concludes that the next candidate is $d_6$, with a global score of .08. Then we are done.
Outline

1. Web crawling

2. Web Information Retrieval

3. Web Graph Mining
   - PageRank
   - HITS
   - Spamdexing

4. Conclusion
The Web Graph

The World Wide Web seen as a (directed) graph:

**Vertices**: Web pages

**Edges**: hyperlinks

Same for other interlinked environments:

- dictionaries
- encyclopedias
- scientific publications
- social networks
The transition matrix

\[
\begin{cases}
    g_{ij} = 0 & \text{if there is no link between page } i \text{ and } j; \\
    g_{ij} = \frac{1}{n_i} & \text{otherwise, with } n_i \text{ the number of outgoing links of page } i.
\end{cases}
\]

\[
G = \begin{pmatrix}
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & \frac{1}{4} & 0 & 0 & \frac{1}{4} & \frac{1}{4} & 0 & \frac{1}{4} & 0 \\
0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & \frac{1}{2} \\
\frac{1}{3} & \frac{1}{3} & 0 & \frac{1}{3} & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & \frac{1}{3} & 0 & \frac{1}{3} & 0 & \frac{1}{3} \\
0 & \frac{1}{3} & 0 & 0 & 0 & 0 & 0 & \frac{1}{3} & \frac{1}{3} & 0 \\
0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}
\]
PageRank (Google’s Ranking [BP98])

Idea

Important pages are pages pointed to by important pages.

PageRank simulates a random walk by iteratively computing the PR of each page, represented as a vector \( \mathbf{v} \).

Initially, \( \mathbf{v} \) is set using a uniform distribution (\( \mathbf{v}[i] = \frac{1}{|\mathbf{v}|} \)).

Definition (Tentative)

Probability that the surfer following the random walk in \( G \) has arrived on page \( i \) at some distant given point in the future.

\[
pr(i) = \left( \lim_{k \to +\infty} (G^T)^k \mathbf{v} \right)_i,
\]

where \( \mathbf{v} \) is some initial column vector.
PageRank Iterative Computation
PageRank Iterative Computation
PageRank Iterative Computation
PageRank Iterative Computation
PageRank Iterative Computation
PageRank Iterative Computation
PageRank Iterative Computation
PageRank Iterative Computation
PageRank Iterative Computation
PageRank Iterative Computation
PageRank Iterative Computation
PageRank Iterative Computation
PageRank Iterative Computation
PageRank Iterative Computation
PageRank With Damping

May not always converge, or convergence may not be unique. To fix this, the random surfer can at each step randomly jump to any page of the Web with some probability $d$ ($1 - d$: damping factor).

$$pr(i) = \left( \lim_{k \to +\infty} \left( (1 - d) G^T + d U \right)^k v \right)_i$$

where $U$ is the matrix with all $\frac{1}{N}$ values with $N$ the number of vertices.
Using PageRank to Score Query Results

- PageRank: *global* score, independent of the query
- Can be used to raise the weight of **important** pages:

  \[
  \text{weight}(t, d) = \text{tfidf}(t, d) \times \text{pr}(d),
  \]

- This can be directly incorporated **in the index**.
HITS (Kleinberg, [Kle99])

Idea

Two kinds of important pages: hubs and authorities. Hubs are pages that point to good authorities, whereas authorities are pages that are pointed to by good hubs.

$G'$ transition matrix (with 0 and 1 values) of a subgraph of the Web. We use the following iterative process (starting with $a$ and $h$ vectors of norm 1):

\[
\begin{align*}
    a & := \frac{1}{\|G'T h\|} \quad G'T h \\
    h & := \frac{1}{\|G'a\|} \quad G'a
\end{align*}
\]

Converges under some technical assumptions to authority and hub scores.
Using HITS to Order Web Query Results

1. Retrieve the set $D$ of Web pages matching a keyword query.
2. Retrieve the set $D^*$ of Web pages obtained from $D$ by adding all linked pages, as well as all pages linking to pages of $D$.
3. Build from $D^*$ the corresponding subgraph $G'$ of the Web graph.
4. Compute iteratively hubs and authority scores.
5. Sort documents from $D$ by authority scores.

Less efficient than PageRank, because local scores.
Spamdexing

Definition

Fraudulent techniques that are used by unscrupulous webmasters to artificially raise the visibility of their website to users of search engines.

Purpose: attracting visitors to websites to make profit.

Unceasing war between spamdexers and search engines.
Spamdexing: Lying about the Content

**Technique**

Put unrelated terms in:
- **meta-information** (`<meta name="description">`, `<meta name="keywords">`)
- text content hidden to the user with JavaScript, CSS, or HTML presentational elements

**Countertechnique**

- **Ignore** meta-information
- Try and **detect** invisible text
Link Farm Attacks

**Technique**
Huge number of hosts on the Internet used for the sole purpose of referencing each other, without any content in themselves, to *raise the importance* of a given website or set of websites.

**Countertechique**
- Detection of websites with *empty* or *duplicate* content
- Use of heuristics to discover *subgraphs* that look like link farms
Link Pollution

**Technique**

Pollute *user-editable* websites (blogs, wikis) or exploit security bugs to add *artificial* links to websites, in order to raise its importance.

**Countertechnique**

`rel="nofollow"` attribute to `<a>` links not validated by a page’s owner
Outline

1. Web crawling
2. Web Information Retrieval
3. Web Graph Mining
4. Conclusion
What you should remember

- The **inverted index** model for efficient answers of keyword-based queries.
- The **threshold algorithm** for retrieving top-$k$ results.
- **PageRank** and its iterative computation.
References

- **Specifications**
  - HTML 4.01, [http://www.w3.org/TR/REC-html40/](http://www.w3.org/TR/REC-html40/)

- **A book**
  
  *Mining the Web: Discovering Knowledge from Hypertext Data*, Soumen Chakrabarti, Morgan Kaufmann
Serge Abiteboul, Grégory Cobena, Julien Masanès, and Gerald Sedrati.  
A first experience in archiving the French Web.  

Serge Abiteboul, Mihai Preda, and Grégory Cobena.  
Adaptive on-line page importance computation.  

Andrei Z. Broder, Steven C. Glassman, Mark S. Manasse, and Geoffrey Zweig.  
Syntactic clustering of the Web.  

Sergey Brin and Lawrence Page.  
The anatomy of a large-scale hypertextual Web search engine.  
Soumen Chakrabarti.  
*Mining the Web: Discovering Knowledge from Hypertext Data.* Morgan Kaufmann, 2003.

Soumen Chakrabarti, Martin van den Berg, and Byron Dom.  
Focused crawling: A new approach to topic-specific Web resource discovery.  

Michelangelo Diligenti, Frans Coetzee, Steve Lawrence, C. Lee Giles, and Marco Gori.  
Focused crawling using context graphs.  
In *Proc. VLDB*, Cairo, Egypt, September 2000.

Jon M. Kleinberg.  
Authoritative Sources in a Hyperlinked Environment.  
Martijn Koster.  
A standard for robot exclusion.  

Martin F. Porter.  
An algorithm for suffix stripping.  

Pierre Senellart.  
Identifying Websites with flow simulation.  

sitemaps.org.  
Sitemaps XML format.  
US National Archives and Records Administration. The Soundex indexing system.