Selected aspects of Web Search and Web Crawling
Motivation

The search engines have to bring some sort of semantics to the semantic-free structure of the web in order to deliver desired results in an efficient manner.

- Billions of interconnected pages
- Everybody can publish
- Various formats and languages
- A lot of low-quality content
- A lot of dynamic content

World Wide Web

Search engine logic and algorithms

→ The search engines have to bring some sort of semantics to the semantic-free structure of the web in order to deliver desired results in an efficient manner.
General search engine architecture

World Wide Web

Crawler module

Page repository

Indexing Module

Indexes

Content index

Structure index

Query-independent part

Query module

Ranking module

Queries

Results

user
Outline

- **Web crawling**
  - Web crawler architecture
  - General crawler functionalities
  - Web data extraction
  - Robots Exclusion Standard
  - Sitemaps

- **Web search**
  - Preprocessing for index generation
  - Content based indexes
  - Structure based indexes
    - PageRank
    - HITS
Crawler

• Due to the self-organized nature of the web, there is no central collection or categorization organization, thus search engines have to carry out the collection and categorization of data on their own.

• This task is done by the web crawler module (sometimes also referred to as web robot or web scutter) which tries, after initialization with a starting set of URIs, to detect further pages.

• Beside their usage by search engines they are deployed in many other application domains where data extraction (information retrieval) from the web is necessary (including negative ones such as spam bots searching for mail addresses).

• Crawling process is often based on conventions between crawler developer and site admin due to the facts that:
  – regular crawlers should not annoy site admins and
  – site admins often have a high interest in providing special information to a crawler to support the crawling effectiveness or efficiency.
Universal Crawler architecture

Start

Initialize Frontier

Seed URIs

Dequeue URI from Frontier

List of unvisited URIs

Frontier

Fetch page

Web

(Evaluate content)

Extract URIs and add new ones to Frontier

Stored for further processing

Store page

Status is reached if e.g. a specified number of pages has been crawled or if the frontier is empty

Done?

yes

Stop

no
Fetching of pages

- Fetching is realized by regular HTTP client
- Many aspects for increasing efficiency during this task can be considered:
  - Management of a timeout mechanism to avoid wasting valuable time
  - More efficient socket communication should be used instead of predefined libraries/classes (such as the java.net.HttpURLConnection class)
  - Parallelizing the access to the frontier
  - Pre-fetching of Domain Name System requests (domain names of hyperlink targets have to be transferred to IP addresses)
  - Parallelizing download via multithreaded algorithm
General crawling strategies

**Breadth-first**
- Always follow all links to adjacent pages of the current page first
- Add all found URIs of adjacent pages to a queue
- If all links of the current page have been visited, take the first page of the queue and start over

**Depth-first**
- Always follow the first unvisited link of the current page
- Declare this visited page as the current page and start over
- Realize backtracking at dead ends or after reaching a predefined depth
Link extraction

1. Find all anchor (<a>) tags and grab the value of all associated "href" attributes
2. Perform filtering to exclude certain file types that are not crawled (e.g. PDFs)
3. URIs have to be transformed to a uniform ("canonical" or "normalized"), crawler-dependent representation
   • Transformations during this canonicalization process may include
     • Converting relative URIs to absolute ones
       e.g. /WORLD \rightarrow http://www.cnn.com/WORLD
     • Adding or removing port numbers
     • Remove fragment identifiers
       e.g. http://www.cnn.com/1.html#3 \rightarrow http://www.cnn.com/1.html
4. If resulting representation is not in the frontier already, add it

\rightarrow Proceed till a given number of links has been extracted or if no new links are available
Web page parsing

- Due to the fact that it is usually necessary to access and extract more information than just hyperlinks via a sequential scan, a comfortable access to unstructured information within (X)HTML has to be realized.
- Many implementations use a transformation to a Document Object Model (DOM) representation for this purpose.
- Access to unstructured web data is called **web scraping**.
- Major problem: Not well-formed HTML pages that increase difficulties to access content.
- Transformation from badly written (X)HTML pages to well-formed one (**tidying**) should be done before accessing page.

Uncomfortable access → (X)HTML representation → DOM representation → Easy access methods
Focused crawlers

• Besides general purpose web robots, crawlers focusing on special areas of the web are widely used.
• To identify relevant subsections of the web a classification component is used that analyses the found URIs and pages by aspects such as:
  – Avoidance of given top-level-domains (e.g. no " .com")
  – Existence of specific terms (crawlers are often called **topical crawlers** e.g. used by crawlers to find content for **scraper sites** = sites that bundle content from different other sites)
  – Existence of a specific structure (e.g. only web pages that conform to the general weblog structure should be downloaded)
Robots Exclusion Standard

- Convention to prevent cooperating web crawlers from accessing specified parts of a website (no real standard – defined by consensus)

- Possible reasons for its use:
  - Privacy aspects (content should not be available via search engines)
  - Content of subpages is assumed to be irrelevant

- A crawler should first check the existence of a file called `robots.txt` in the website’s root directory that describes general access rules

- Various extensions / improvements available such as Automated Content Access Protocol (ACAP)
Robots Exclusion Standard

- Disallows all cooperative web crawlers to visit the three mentioned subdirectories
  
<table>
<thead>
<tr>
<th>User-agent: *</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disallow: /cgi-bin/</td>
</tr>
<tr>
<td>Disallow: /tmp/</td>
</tr>
<tr>
<td>Disallow: /private/</td>
</tr>
</tbody>
</table>

- Disallows the two mentioned web crawlers to access the whole website
  
Google image search crawler

<table>
<thead>
<tr>
<th>User-agent: Googlebot-Image</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disallow: /fotos</td>
</tr>
</tbody>
</table>

Yahoo blog search crawler

<table>
<thead>
<tr>
<th>User-agent: yahoo-blogs/v3.9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disallow: /blog</td>
</tr>
<tr>
<td>User-agent: *</td>
</tr>
<tr>
<td>Disallow: /cgi-bin/</td>
</tr>
<tr>
<td>Crawl-delay: 60</td>
</tr>
<tr>
<td>Sitemap: <a href="http://www.example.com/sitemap.xml.gz">http://www.example.com/sitemap.xml.gz</a></td>
</tr>
</tbody>
</table>

- Minimum of 60 seconds should be between two visits + the sitemap of this website is available at the given URI
Motivation for sitemaps

- If this link section is downloaded via XMLHttpRequest and included into the page dynamically, a web crawler would have to execute the JavaScript statement to get the links to subpages

- **BUT**: How should the web crawler know that hyperlinks are returned by the XMLHttpRequest and not useless content?

- **Alternative**: It should be possible to inform crawlers about further subpages of a website that are not directly accessible
Sitemaps protocol

- Standardized way that allows to inform web crawlers about available URIs and some of their properties on a website
- Useful if some pages are not well linked or e.g. are only reachable via AJAX mechanism

```xml
<?xml version="1.0" encoding="UTF-8"?>
<urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">
  <url>
    <loc>http://www.example.com/index.html</loc>
    <lastmod>1970-06-07</lastmod>
    <changefreq>monthly</changefreq>
    <priority>0.8</priority>
  </url>
  <url>
    <loc>http://www.example.com/fotos.html</loc>
    <changefreq>weekly</changefreq>
  </url>
  ...
</urlset>
```
Indexing

User expects efficient delivery of results

Query: "Friedrich Schiller"

In large document collections the process of parsing known pages to search terms for every query would be very inefficient. How can pages be represented internally for efficient information retrieval?

Document 1

Quote from Friedrich Schiller:
Appearance rules the world.

Document 2

Quote from Arthur Schopenhauer:
The world is nothing but will and idea.

Document 3

Quote from Friedrich Nietzsche:
In his writings Schiller did not create humans, but puppets.
Content Index

• An information basis that can be built from the document collection is the **content index**
• Stores textual information for each page in compressed form
• Many preprocessing steps are often taken to fill this index:
  – Removal of unnecessary formatting/structure information such as HTML tags
  – During the **tokenization process** punctuation marks are removed and character strings without spaces are considered as tokens (= **terms**)
  – Conversion of all characters to upper or lower case
  – Reduction of terms to their canonical form ("**stemming**")
    • E.g. "programs" and "programming" are transformed to "program"
  – Removing of unnecessary and common words that do not bring any meaning
    • E.g. a, an, the, on, in, ...
• Simple but efficient type of content index: **Inverted Index**
  – Attaches each distinctive term to a list of documents that contain the term
Simple Inverted Index structure

<table>
<thead>
<tr>
<th>term</th>
<th>document</th>
</tr>
</thead>
<tbody>
<tr>
<td>quote</td>
<td>D1, D2, D3</td>
</tr>
<tr>
<td>friedrich</td>
<td>D1, D3</td>
</tr>
<tr>
<td>schiller</td>
<td>D1, D3</td>
</tr>
<tr>
<td>appearance</td>
<td>D1</td>
</tr>
<tr>
<td>rule</td>
<td>D1</td>
</tr>
<tr>
<td>world</td>
<td>D1, D2</td>
</tr>
<tr>
<td>nietzsche</td>
<td>D3</td>
</tr>
<tr>
<td>writing</td>
<td>D3</td>
</tr>
<tr>
<td>human</td>
<td>D3</td>
</tr>
<tr>
<td>puppet</td>
<td>D3</td>
</tr>
<tr>
<td>arthur</td>
<td>D2</td>
</tr>
<tr>
<td>schopenhauer</td>
<td>D2</td>
</tr>
<tr>
<td>nothing</td>
<td>D2</td>
</tr>
<tr>
<td>will</td>
<td>D2</td>
</tr>
<tr>
<td>idea</td>
<td>D2</td>
</tr>
<tr>
<td>create</td>
<td>D3</td>
</tr>
</tbody>
</table>

Due to intersection of the document sets for the terms "friedrich" and "schiller" document 1 and document 3 are part of the result set for the query "friedrich schiller"

What if the user only expects results that contain the **phrase** "friedrich schiller"?

→ This information can not be expressed by the simple inverted index structure

D1 = Document 1  
D2 = Document 2  
D3 = Document 3
Inverted Index structure with term positions

<table>
<thead>
<tr>
<th>term</th>
<th>document</th>
</tr>
</thead>
<tbody>
<tr>
<td>quote</td>
<td>(D1,1), (D2,1), (D3,1)</td>
</tr>
<tr>
<td>friedrich</td>
<td>(D1,3), (D3,3)</td>
</tr>
<tr>
<td>schiller</td>
<td>(D1,4), (D3,8)</td>
</tr>
<tr>
<td>appearance</td>
<td>(D1,5)</td>
</tr>
<tr>
<td>rule</td>
<td>(D1,6)</td>
</tr>
<tr>
<td>world</td>
<td>(D1,8), (D2,6)</td>
</tr>
<tr>
<td>nietzsche</td>
<td>(D3,4)</td>
</tr>
<tr>
<td>writing</td>
<td>(D3,7)</td>
</tr>
<tr>
<td>human</td>
<td>(D3,12)</td>
</tr>
<tr>
<td>puppet</td>
<td>(D3,14)</td>
</tr>
<tr>
<td>arthur</td>
<td>(D2,3)</td>
</tr>
<tr>
<td>schopenhauer</td>
<td>(D2,4)</td>
</tr>
<tr>
<td>nothing</td>
<td>(D2,8)</td>
</tr>
<tr>
<td>will</td>
<td>(D2,10)</td>
</tr>
<tr>
<td>idea</td>
<td>(D2, 12)</td>
</tr>
<tr>
<td>create</td>
<td>(D3, 11)</td>
</tr>
</tbody>
</table>

By extending the stored information with term position data the query for the phrase "Friedrich Schiller" can be processed properly.

Besides regular content information, the inverted index can be extended by document structure information such as directly surrounding HTML tags of a term (e.g. storage of <h1> makes it possible to search for headings etc.)

**Open question:**
How are documents sorted if more then one document fits for a query?

D1 = Document 1  
D2 = Document 2  
D3 = Document 3
Term Frequency Inverse Document Frequency

- The TFIDF is a weight to describe the importance of a term in a document within a document collection.
- Considers
  - Term Frequency (TF): How important is a term in a document?
  - Inverse Document Frequency (IDF): How important is a term regarding the whole document collection?
- There exist different concrete definitions for this weight but the principle is always similar.
- An example formula is:

\[
\text{TFIDF}_t = \text{TF}_t \times \text{IDF}_t
\]

\[
\text{TF}_t = \frac{n_{t, j}}{N_j}
\]

\[
\text{IDF}_t = \frac{D}{D_t}
\]

If the document is very large the probability that it contains term t increases, thus making large documents that contain t not as important as short ones containing it.

If a term is very rare, a document that contains it is more important.

Number of occurrences of term t in document j

Number of all terms in document j

Number of documents in the document set

Number of documents that contain term t
Term Frequency Inverse Document Frequency

Regarding TFIDF this document would be ranked on the first position for the query *quote friedrich*

→ TF(quote) = 1/8
→ TF(friedrich) = 1/8
→ IDF(quote) = 1
→ IDF(friedrich) = 1.5

Sum of both TFIDF for the query: 1*1/8 + 1.5*1/8 = 0.313

→ TF(quote) = 1/14
→ TF(friedrich) = 1/14
→ IDF(quote) = 1
→ IDF(friedrich) = 1.5

Sum of both TFIDF for the query: 1*1/14 + 1.5*1/14 = 0.179

Rank is higher and thus is presented on first position of result set to the user

- The TFIDF can be calculated query-independent on a set of documents
- when the query occurs the resulting value regarding the specific query can be calculated (by *ranking module*)
Relevance ranking

- Classical information retrieval:
  → Ranking is a function of query term frequency within a document (TF) and across all documents (IDF)
- Central assumptions that count for classical document collections (libraries, ...) cannot be applied to the web - typical aspects that have to be considered for the web include:
  - Queries in the web are very short (average: 2.35 terms)
  - There is a enormous variety of documents (language, quality, duplication) and a very huge vocabulary
  - Numerous misinformation is on many pages (everybody can publish)
- This leads to the central requirement for not only evaluating the content but particularly factor the structure of the web
- Linkage intensity of a page can be seen as its relevance and popularity
  - Important algorithms that take this into account: PageRank and HITS

![Diagram showing Content Score, Structure score, and Overall Score]
PageRank overview

- PageRank is a way to prioritize results of web keyword search based on evaluation of the link structure.
- Invented by Google founders Brin and Page and applied as part of Google’s ranking algorithm.
- Simplified assumption: A hyperlink from page A to page B is a recommendation of the content of page B by the author of A.
  - Quality of a page is related to its in-degree (number of incoming links).
- Recursion: Quality of a page is related to
  - its in-degree and to
  - the quality of pages linking to it.

Diagram: PageRank? → PageRanks? → PageRanks?
PageRank details

- Hyperlinks can be seen as paths along which users travel from one page to another.
- In this respect the popularity of a web page can be measured in terms of how often an average web user visits it (random walk model).

- If the current page has got $N$ outgoing links, then the user moves to one of the successors with a probability of $1/N$.
- This can be interpreted as forwarding $N$ identical fractions of the own prestige to the linked pages:
  \[ \text{Prestige(currentPage)} / N \]

Open question: How is the calculation of prestige initiated?
PageRank details

- Prestige calculation can be initiated by assigning all pages a prestige of e.g. 1 or of \((1/\text{NUMBER\_OF\_NODES})\)
- Then: Iterative calculation of prestige that converges against the final values
  → The prestige of a node is given by the sum of the portion of prestige all linking nodes assign to it

\[
Prestige(p) = \sum \left( \frac{\text{Prestige}(q)}{N} \right)
\]

<table>
<thead>
<tr>
<th>Current node</th>
<th>Nodes linking to p</th>
<th>Prestige(p) = \sum(\text{Prestige}(q)/N)</th>
<th>Number of all outgoing links of q (out-degree of q)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2, 3, 6</td>
<td>1/6</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1, 3, 5</td>
<td>1/18</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>1, 2, 5</td>
<td>1/12</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>1, 5</td>
<td>1/4</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>1, 2, 6</td>
<td>1/36</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>1, 2, 3</td>
<td>1/6</td>
<td>6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Iteration 0</th>
<th>Iteration 1</th>
<th>Iteration 2</th>
<th>Rank at Iteration 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>P(1)=1/6</td>
<td>P(1)=1/18</td>
<td>P(1)=1/36</td>
<td>5</td>
</tr>
<tr>
<td>P(2)=1/6</td>
<td>P(2)=5/36</td>
<td>P(2)=1/18</td>
<td>4</td>
</tr>
<tr>
<td>P(3)=1/6</td>
<td>P(3)=1/12</td>
<td>P(3)=1/36</td>
<td>5</td>
</tr>
<tr>
<td>P(4)=1/6</td>
<td>P(4)=1/4</td>
<td>P(4)=17/72</td>
<td>1</td>
</tr>
<tr>
<td>P(5)=1/6</td>
<td>P(5)=5/36</td>
<td>P(5)=11/72</td>
<td>3</td>
</tr>
<tr>
<td>P(6)=1/6</td>
<td>P(6)=1/6</td>
<td>P(6)=14/72</td>
<td>2</td>
</tr>
</tbody>
</table>

values are not normalized
The previously mentioned simple random walk model has some major problems, such as:

- What if the user is caught in a cyclic link graph?

- What if the user enters a **dangling node** (a node with no outgoing links)?

After just 13 iterations, the prestige distribution for the graph is nearly:

| Iteration 13 | 0 | 0 | 0 | 2/3 | 1/3 | 1/5 |

Continuously reduces prestige of nodes of 1 and 3 without redistributing the prestige to the graph.

→ Adjustment is needed
PageRank details

- To extend the behaviour of a regular walk through the web and to avoid problems, the average probability of leaving the current page not via an outgoing link but by entering an arbitrary other page of the overall web graph (=adjustment factor) is considered.

- Adding this factor leads to a realistic model for web surfing:
  - A user enters an arbitrary page by specifying the URI in the web browser or moves from page to page by accessing available hyperlinks.

Probability that random user enters a new URI is denoted by d (thus the probability for entering a specific node in the total node set T is $d/|T|$).

Remaining probability for entering one of the available links is $(1-d)$. 

slide 27
• The extended random walk model is expressed within the formula of PageRank:

\[
\text{PageRank}\ (p) = \frac{d}{|T|} + (1 - d) \sum_{(q,p) \in E} (\text{PageRank}\ (q) / \text{outdegree}\ (q))
\]

- Probability with which the random user enters node \( p \) by typing URI into the browser
- Remaining probability for moving to the next node by entering an outgoing link
- There exists a link from \( q \) to \( p \) (\( E = \text{set of all links} \))
- Number of all outgoing links of \( q \)

• Instead of using an iteration, the PageRank problem can be stated in two ways:
  - As an eigenvector problem
  - As a linear homogeneous system

Out of the scope of this lecture
HITS - Hubs and Authorities

- HITS (Hyperlinked Induced Topic Search) differentiates between **Hubs** and **Authorities** and delivers result lists for both categories.

**Authority:**
- Page with many inlinks
- The idea has been mentioned before:
  - A page has good content on some topic and thus many people trust it and link to it
- BUT: How can it be assured that the linking pages are of high quality and thus their “vote” is really valuable?

**Hub:**
- Page with many out-links
- Central idea:
  - A hub serves as an organizer of information on a particular topic and points to many pages connected to this topic
- BUT: How can it be assured that it really points to “good” pages?
HITS - Hubs and Authorities

- **Authority** comes from **in-edges** - being a **hub** comes from **out-edges**

- **Better authority** comes from in-edges from **good hubs** - being a **better hub** comes from out-edges to **good authorities**

- Circular theses:
  "A page is a good hub (and, therefore, deserves a high hub score) if it points to good authorities, and a page is a good authority (and, therefore, deserves a high authority score) if it is pointed to by good hubs"
  → **Mutual reinforcement**
HITS

- **HITS is query-dependent**
  - Initially a set of documents is determined that fits a given user query
  - Based on this set HITS is used to determine the hub and authority value for each included document → two ranking lists are finally presented to the user
  - Due to the definition of hubs and authorities the calculation can be carried out in an iterative manner (comparable to PageRank)

\[
\text{Repeat until HUB and AUTH converge:}
\]

\[
\text{HUB}[x] := \sum \text{AUTH}[r_i] \quad \text{for all } r_i \text{ with } (x, r_i) \text{ in } E
\]

\[
\text{AUTH}[x] := \sum \text{HUB}[q_i] \quad \text{for all } q_i \text{ with } (q_i, x) \text{ in } E
\]

Page x that is part of the result set

Authority score

Hub score

\begin{align*}
\text{There exists a link from } x \text{ to } r
\end{align*}

\begin{align*}
\text{Set of hyperlinks}
\end{align*}

\begin{align*}
\text{There exists a link from } q \text{ to } x
\end{align*}
Summary

World Wide Web

Document collection with very specific characteristics

Web crawler

- Crawling strategies
- Link extraction
- Parsing
- Robots Exclusion standard / Sitemaps

Indexing

1010 0101 1100

- Crawling strategies
- Link extraction
- Parsing
- Robots Exclusion standard / Sitemaps

Ranking

- Query-dependent
- Hubs + Authorities

Content Score

Structure score = Popularity Score

Overall Score

HITS

- Query-dependent
- Hubs + Authorities

PageRank

- Random walk model

Queries

User interface

Good results sorted by relevance
References

Robots Exclusion Standard  http://www.robotstxt.org/robotstxt.html
Sitemaps protocol        http://www.sitemaps.org/
HTML tidying             http://www.w3.org/People/Raggett/tidy/

HITS publication          http://www.cs.cornell.edu/home/kleinber/auth.pdf

Book about search engines and ranking
“Google’s PageRank and Beyond – The science of search engine rankings”
Author: Langville and Meyer ISBN: 978-0-691-12202-1

Fundamental paper about “Web crawling ethics”:
Web crawling ethics revisited: Cost, privacy, and denial of service by Mike Thelwall and David Stuart, 2006