Crawling

June 10, 2015

Crawling is one of the most important tasks of a search engine. The breadth, depth, and freshness of the search results depend crucially on the quality of crawling. As the number of pages and sites on the web increases rapidly, deploying an effective and efficient crawling strategy becomes critical for a search engine.

1 A basic crawler

A basic crawler maintains a frontier a collection of pages to be crawled and iteratively selects and crawls pages from it.

- The frontier is initialized with a list of seed pages.
- The next page is selected carefully, for politeness and performance reasons.
- New URLs are processed and filtered before being added to the frontier.

1.1 Working of a basic crawler

To start the web crawling, one needs a set of some URLs. These initially selected URLs are called seed URLs or seed pages. Selection of seed pages is dealt in section 2.3. The frontier is initialized with a list of seed pages. The next page is selected carefully, for politeness and performance reasons. The page is parsed to get the outlinks. These links are processed and filtered before being added to the frontier. This process continues until frontier becomes empty. Below is the pseudocode for a basic crawler

```
def crawl(seeds):

    # The frontier is initially the seed set
    frontier.add_pages(seeds)

    # Iteratively crawl the next item in the frontier
    while not frontier.is_empty():

        # Crawl the next URL and extract anchor tags from it
        url = frontier.choose_next()
        page = crawl_url(url)
        urls = parse_page(page)

        # Update the frontier and send the page to the indexer
        frontier.add_pages(urls)
        send_to_indexer(page)
```

Figure 1: Pseudocode for a basic crawler

1.2 HTTP Fetching

Requesting and downloading a URL involves several steps:

1. A DNS server is asked to translate the domain into an IP address.
2. (optional) An HTTP HEAD request is made at the IP address to determine the page type, and whether the page contents have changed since the last crawl.
3. An HTTP GET request is made to retrieve the new page contents.

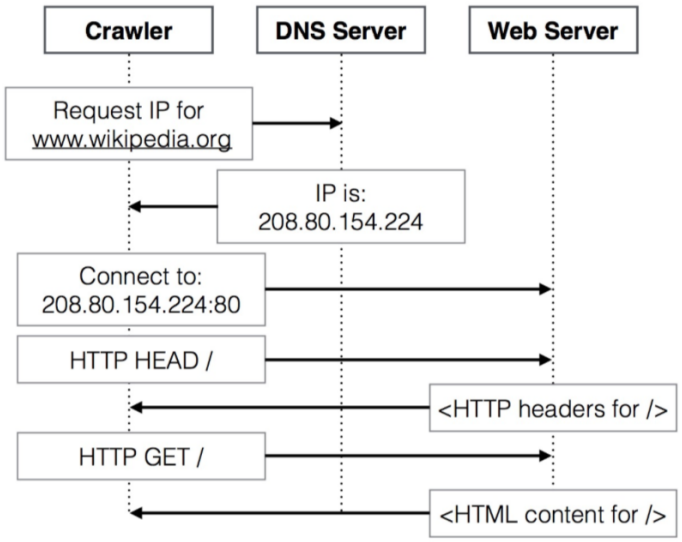


Figure 2: Request process for http://www.wikipedia.org/

1.3 HTTP Requests

The HTTP request and response take the following form:

(a) HTTP Request: $\langle method \rangle\ \langle url \rangle\ \langle HTTP\ version \rangle\ \left[\ \langle optional\ headers \rangle\ \right]$

(b) Response Status and Headers: $\langle HTTP\ version \rangle\ \langle code \rangle\ \langle status \rangle\ \left[\ \langle optional\ headers \rangle\ \right]$

(c) Response Body

The following figure shows a typical HTTP request:

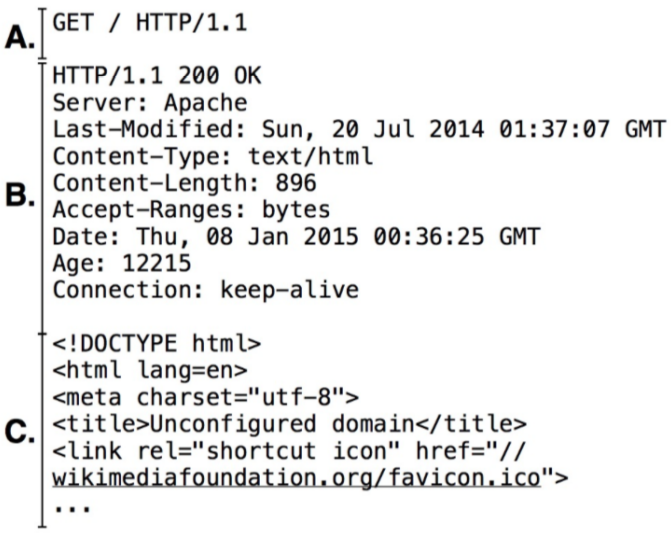


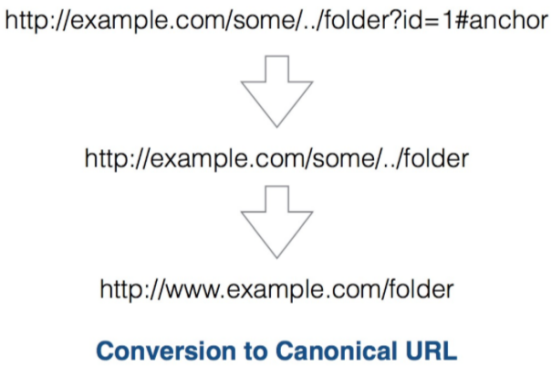
Figure 3: HTTP Request and Response

1.4 URL Extraction

Once the crawler downloades the files, it must parse them according to their content type (usually available in the Content-Type header), and extract URLs on the pages for adding them to the frontier. HTML documents in the wild often have formatting errors which the parser must address. Other document formats have their own issues. URLs may be embedded in PDFs, Word documents, etc. Many URLs are missed, especially due to dynamic URL schemes and web pages generated by JavaScript and AJAX calls. This is part of the so-called "dark web".

1.5 URL Canonicalization

Many possible URLs can refer to the same resource. It is important for the crawler (and index!) to use a canonical, or normalized, version of the URLs to avoid repeated requests. Many rules have been used; some are guaranteed to only rewrite URLs to refer to the same resource, and others can make mistakes. It can also be worthwhile to create specific normalization rules for important web domains, e.g. by encoding which URL parameters result in different web content. Below is the example of canonicalization of a URL:



1.5.1 Rules for Canonicalization

Here are a few possible URL canonicalization rules

Rule	Safe?	Example
Remove default port	Always	http://example.com:80 → http://example.com
Decoding octets for unreserved characters	Always	http://example.com/%7Ehome → http://example.com/~home
Remove . and ..	Usually	http://example.com/a/./b/../c → http://example.com/a/c
Force trailing slash for directories	Usually	http://example.com/a/b → http://example.com/a/b/
Remove default index pages	Sometimes	http://example.com/index.html → http://example.com
Removing the fragment	Sometimes	http://example.com/a#b/c → http://example.com/a

Figure 4: URL canonicalization rule

In summary, Web crawling requires attending to many details. DNS responses should be cached, HTTP HEAD requests should generally be sent before GET requests, and so on. Extracting and normalizing URLs is important, because it dramatically affects your coverage and the time wasted on crawling, indexing, and ultimately retrieving duplicate content.

2 Coverage

The Internet is too large and changes too rapidly for any crawler to be able to crawl and index it all. Instead, a crawler should focus on strategic crawling to balance coverage and freshness.
A crawler should prioritize crawling high-quality content to better answer user queries. The Internet contains a lot of spam, redundant information, and pages which are not likely to be relevant to users' information needs.

2.1 Selection Policies

A selection policy is an algorithm used to select the next page to crawl. Standard approaches include:

- Breadth-first search: This distributes requests across domains relatively well and tends to download high-PageRank pages early.
- Backlink count: Prioritize pages with more in-links from already-crawled pages.
- Larger sites first: Prioritize pages on domains with many pages in the frontier.
- Partial PageRank: Approximate PageRank scores are calculated based on already-crawled pages. There are also approaches which estimate page quality based on a prior crawl.

2.2 Comparing Approaches

Baeza-Yates et al compare these approaches to find out which fraction of high quality pages in a collection is crawled by each strategy at various points in a crawl. Breadth-first search does relatively poorly. Larger sites first is among the best approaches, along with "historical" approaches which take PageRank scores from a prior crawl into account. OPIC, a fast approximation to PageRank which can be calculated on the fly, is another good choice. The "omniscient" baseline always fetches the highest PR page in the frontier.

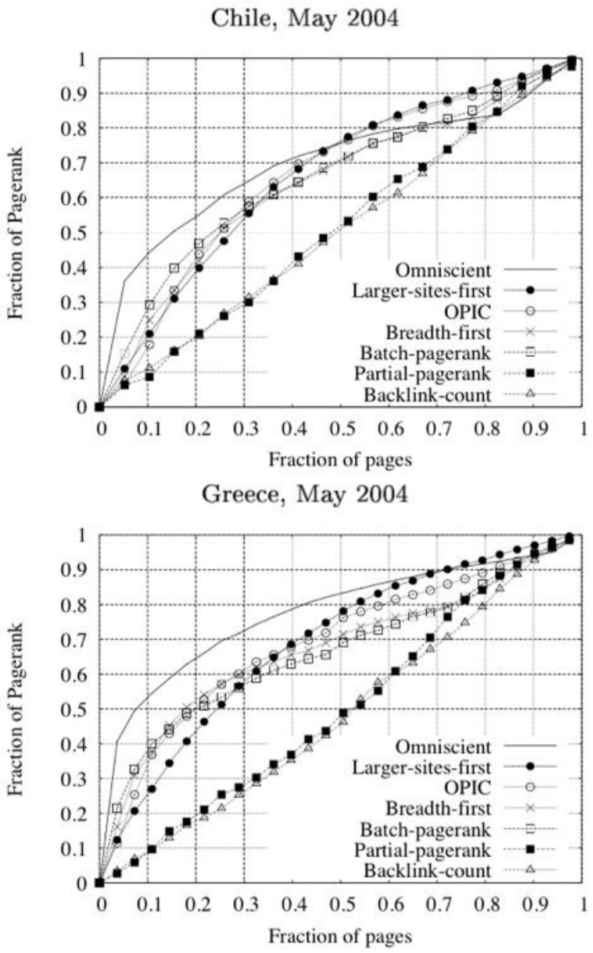


Figure 5: Ricardo Baeza-Yates, Carlos Castillo, Mauricio Marin, and Andrea Rodriguez. 2005. Crawling a country: better strategies than breadth-first for web page ordering.

2.3 Seed URL selection

It is important to choose the right seed URLs to initialize your frontier. A simple baseline approach is to start with the URLs in an Internet directory, such as http://www.dmoz.org. In general, good hubs tend to lead to many high-quality web pages. These hubs can be identified with a careful analysis of a prior crawl. A good seed url may be judged based upon the measures of:

- Quality
- Importance
- Potential yield

2.3.1 Quality

Quality (or lack of quality) of a url as a potential seed url could be based upon things such as number of outlinks on the seed url page, being a spam page or having outlinks pointing to spam pages. A good seed url should have large number of outlinks and no or few spam outlinks. High quality urls are chosen as potential seed urls since, as the starting point of a crawl, it?s likely that starting with a low quality domain would likely result in many more low quality pages being crawled.

2.3.2 Importance

Importance of a seed url can be judged based on the factors such as popularity,trustworthines, reliability, quality, page rank of the seed page.

2.3.3 Potential yield

Potential yield of documents , or the potential for the discovery of new URLs, for a url could be calculated based upon statistics gathered from past crawls from that seed page.

3 Freshness

The web is constantly changing, and re-crawling the latest changes quickly can be challenging. It turns out that aggressively re-crawling as soon as a page changes is sometimes the wrong approach: it is better to use a cost function associated with the expected age of the content, and tolerate a small delay between re-crawls.

The web is constantly changing as content is added, deleted, and modified. In order for a crawler to reflect the web as users will encounter it, it needs to re-crawl content soon after it changes. This need for freshness is key to providing a good search engine experience. For instance, when breaking news develops, users will rely on your search engine to stay updated. It's also important to refresh less time-sensitive documents so the results list does not contain spurious links to deleted or modified data.

HTTP HEAD Requests

A crawler can determine whether a page has changed by making an HTTP HEAD request. The response provides the HTTP status code and headers, but not the document body. The headers include information about when the content was last updated as shown in Figure 6. However, it is not feasible to constantly send HEAD requests, so this is not an adequate strategy for freshness.

Request

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

Response

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

Figure 6: HTTP Response

Freshness vs Age

It turns out that optimizing to minimize freshness is a poor strategy: it can lead the crawler to ignore important sites. Instead, it is better to re-crawl pages when the age of the last crawled version exceeds some limit. The age of a page is the elapsed time since the first update after the most recent crawl. See Figure 7.

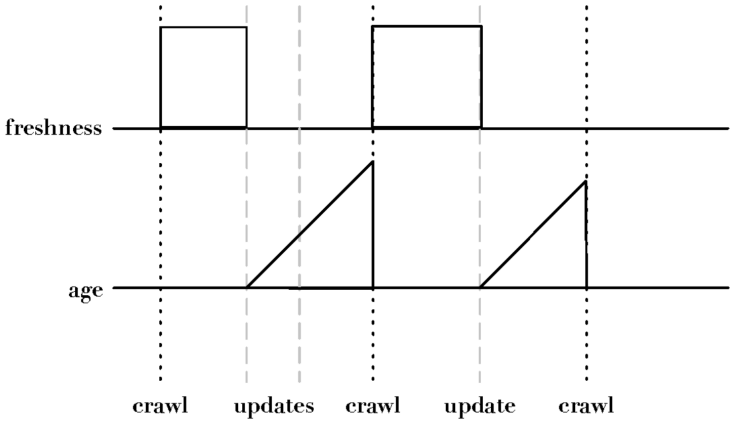


Figure 7: Freshness is binary, age is continuous.

The expected age of a page t days after it was crawled depends on its update probability:

$$age(\lambda, t) = \int_0^t P(page\ changed\ at\ time\ x)(t-x)dx$$

On average, page updates follow a Poisson distribution - the time until the next update is governed by an exponential distribution. This makes the expected age:

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

Freshness vs Coverage

The opposing needs of freshness and coverage need to be balanced in the scoring function used to select the next page to crawl. Finding an optimal balance is still an open question. Fairly recent studies have shown that even large name-brand search engines only do a modest job at finding the most recent content. However, a reasonable approach is to include a term in the page priority function for the expected age of the page content. For important domains, you can track the site-wide update frequency λ.

4 Storing Crawled Content

We need to normalize and store the contents of web documents so they can be indexed, so snippets can be generated, and so on. Online documents have many formats and encoding schemes. There are hundreds of character encoding systems we have not mentioned here. A good document storage system should support efficient random access for lookups, updates, and content retrieval. Often, a distributed storage system like Big Table is used.

Content Conversion

Downloaded page content generally needs to be converted into a stream of tokens before it can be indexed. Content arrives in hundreds of incompatible formats: Word documents, PowerPoint, RTF, OTF, PDF, etc. Conversion tools are generally used to transform them into HTML or XML(Figure 8). Depending on your needs, the crawler may store the raw document content and/or normalized content output from a converter.

Figure 8: Content Conversion.

Character Encodings

Crawled content will be represented with many different character encodings, which can easily confuse text processors. A character encoding is a map from bits in a file to glyphs on a screen. In English, the basic encoding is ASCII, but also UTF-8 and UTF-32 (Example in Figure **??**).

- ASCII uses 8 bits: 7 bits to represent 128 letters, numbers, punctuation and control characters and an extra bit for padding.
- UTF-8 uses on byte for ASCII characters, and more bytes for extended characters. It is often preferred for file storage.
- UTF-32 uses four bytes for every character, and is more convenient for use in memory.

5 Crawling Structured Data

In addition to the obvious content for human readers, the web contains a great deal of structured content for use in automated systems.

- Document feeds are an important way to manage freshness at some of the most frequently-updated web sites.
- Much of the structured data owned by various web entities is published in a structured format. This can provide signals for relevance, and can also aid in reconstructing structured databases.

In addition to unstructured document contents, a great deal of structured data exists on the web. We will focus here on two types:

- Document Feeds: sites use to announce their new content.
- Content metadata: used by web authors to publish structured properties of objects on their site.

Document Feeds

Site which post articles, such as blogs or news sites, typically offer a listing of their new content in the form of a document feed. Several common feed formats exist. One of the most popular is RSS, which stands for Really Simple Syndication.

RSS is an XML format for document listings. And RSS files are obtained just like web pages, with HTTP GET requests. The ttl filed(See in Figure 9) provides an amount of time(in minutes) that the contents should be cached. RSS feeds are very useful for efficiently managing freshness of news and blog content.

```
<?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>
    <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>
```

Figure 9: RSS example.

Structured Data

Many web pages are generated from structured data in databases, which can be useful for search engines and other crawled document collections. Several schemas exist for web authors to publish their structured data for these tools. The WHATWG web specification working group has produced several standard formats for this data, such as micro data embedded in HTML. See Figure 10.

```
<section itemscope itemtype="http://schema.org/Person">
    Hello, my name is
    <span itemprop="name">John Doe</span>,
    I am a
    <span itemprop="jobTitle">graduate research assistant</span>
    at the
    <span itemprop="affiliation">University of Dreams</span>.
    My friends call me
    <span itemprop="additionalName">Johnny</span>.
    You can visit my homepage at
    <a href="http://www.JohnnyD.com" itemprop="url">www.JohnnyD.com</a>.
    <section itemprop="address" itemscope itemtype="http://schema.org/PostalAddress">
        I live at
        <span itemprop="streetAddress">1234 Peach Drive</span>,
        <span itemprop="addressLocality">Warner Robins</span>,
        <span itemprop="addressRegion">Georgia</span>.
    </section>
</section>
```

Figure 10: Microdata example.

6 Politness

It is important to remember that we are providing web site owners with a service, and to be mindful of the resources we consume by crawling their sites. Following robots.txt and suing sitemaps.xml can also help the crawler avoid pitfalls particular to the web site.

Need for politeness

Web crawlers are generally distributed and multithreaded, and are capable of taxing the capabilities of most web servers. This is particularly true of the crawlers for major businesses, such as Bing and Google. In addition, some content should not be crawled at all.

- Some web content is considered private or under copyright, and its owners prefer that it not be crawled and indexed.
- Other URLs implement API calls and do not lead to indexable content.

This can easily create conflict between search providers and the web sites they are linking to. Politeness policies have been created as a way to mediate these issues.

Robots Exclusion Protocol

Web site administrators can express their crawling preferences by hosting a page at robots.txt. Every crawler should honor these preferences. These files indicate which files are permitted and disallowed for particular crawlers, identified by their user agents. They can also specify a preferred site mirror to crawl. More recently, sites have also begun requesting crawl interval delays in robots.txt.

Request Intervals

It is very important to limit the rate of requests your crawler makes to the same domain. Too-frequent requests are the main way a crawler can harm a web site. Typical crawler delays are in the range of 10-15 seconds per request. You should never crawl more than one page per second from the same domain. For large sites, it can take days or weeks to crawl the entire domain - this is preferable to overloading their site (and possibly getting your IP address blocked). If the sites' robots.txt file has a Crawl-delay directive, it should be honored. In a distributed crawler, all requests for the same domain are typically sent to the same crawler instance to easily throttle the rate of requests.