Web Search

20 January 2008

With a constantly increasing size of billions of freely accessible documents, one of the major issues raised by the World Wide Web is that of searching in an effective and efficient way through these documents to find these that best suit a user's need. The purpose of the chapter is to describe the techniques that are at the core of today's search engines (such as Google[1], Yahoo![2], Microsoft Live Search[3] or Exalead[4]), that is, mostly keyword search in collections of text documents. We also briefly touch upon other techniques and research issues that may be of importance in next-generation search engines.

This chapter is organized as follows: in Section 1, we discuss the Web and the languages and protocols it relies upon. We then present in Section 2 the techniques that can be used to retrieve pages from the Web, that is, to *crawl* it. First-generation search engines, exemplified by Altavista[5] relied mostly on the classical information retrieval (IR) techniques that are described in Section 3. With the advent of Google, other techniques that make use of the graph structure of the Web (see Section 4) have very effectively complemented text IR. We then proceed to a brief discussion of currently active research topics about the Web in Section 5.

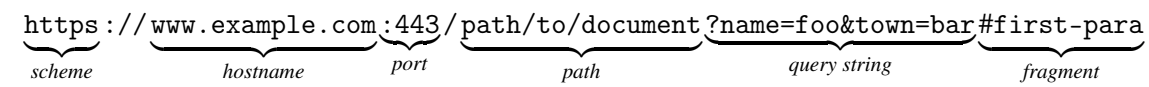
1 The World Wide Web

Whereas the Internet is a physical network of computers (or *hosts*) connected to each other from all around the world, the World Wide Web, WWW or Web in short, is a logical collection of hyperlinked documents shared by the hosts of this network. An hyperlinked document is just a document with references to other documents of the same collection. Note that documents of the Web may both refer to static documents stored on the hard drive of some host of the Internet, and to dynamic documents that are generated on the fly when accessing the document. This means that there is a virtually unlimited number of documents on the Web, since dynamic documents can change on each request. When one speaks of the Web, it is mostly about the *public* part of the Web, which is freely accessible, but there are also various *private* Webs that are restricted to some community or company, either on private *Intranets* or on the Internet, with password-protected pages.

Documents, and, more generally, *resources* on the Web, are identified by a *URL* (*Uniform Resource Locator*) which is a character string that follows a fixed format that is illustrated on the imagi-

[1] `http://www.google.com/`
[2] `http://www.yahoo.com/`
[3] `http://www.live.com/`
[4] `http://www.exalead.com/`
[5] `http://www.altavista.com/`

nary URL below, with basic components described next.



scheme: describes the way the resource can be accessed; on the Web, it is generally one of the Web *protocol* (`http`, `https`) that is described below.

hostname: this is the domain name of a host, as given by the domain name system (DNS, [IET99a]). Frequently on the Web, the hostname of a website will start with `www.`, but this is not a rule.

port: TCP port where the server listens on the host; it defaults to 80 for the `http` scheme and 443 for the `https` scheme and is rarely present.

path: the logical path of the document; for simple cases, this may correspond to some part of the file path of the static document to be retrieved on the host.

query string: additional parameters identifying the resource, mostly used with dynamic documents.

fragment: identifies the part of the document that the URL identifies.

Note that query strings and fragments are optional (and, most of the time, absent) and that the path can be omitted to refer to the *root* of the Web host. URLs can also be relative (by opposition to the *absolute* URL above), in which case both the scheme and hostname portions are omitted. A relative URL is to be interpreted in a given URL *context* (for instance, the URL of the current document) and is resolved in a straightforward way: if the context is that of the URL above, the relative URLs `/titi`[6] and `tata` would be resolved, respectively, as `https://www.example.com/titi` and `https://www.example.com/path/to/tata` in a way similar to (Unix) relative paths resolution.

The choice format for documents (or, in this case, *pages*) on the Web is HTML (the HyperText Markup Language), though a minority of documents, including hyperlinked documents, are in other formats (mostly PDF or documents from word-processing software). HTML is originally a dialect of SGML, the ancestor of XML, but is hardly ever parsed as such. The most common version, HTML 4.01 [W3C99] is described by a recommendation of the World Wide Web Consortium (or W3C), an organism that regroups academics and industrials for the development of standards about the World Wide Web. XHTML 1.0 [W3C02] is a direct XMLization of HTML 4.01, with minor differences. An example XHTML document is given in Figure 1. As it is an SGML or XML file, an (X)HTML document is made out of elements, attributes and text content. Elements carry, between other things, meta-information about the document (e.g., `<meta>`, `<title>`), structural information at the document level (e.g., `<table>`, `<ul>`, `<p>`), structural information at the character level (e.g., `<strong>`, `<em>`) or references to other media (e.g., `<img>`, `<object>`). An element of importance is `<a>`, which defines a hyperlink to another resource on the Web identified by the URL given as the `href` attribute. Both relative and absolute links are allowed here. The context of resolution is the URL of the current page, unless it contains a `<base>` element that indicates another context. HTML pages can also contain other *disguised* hyperlinks in the form of JavaScript code that loads other URLs, of redirection after a timeout with some specific use of the `<meta>` element, or of Flash or Java applets; all these links are less easy to identify and then less accessible to users and Web robots.

[6]Note that here `/titi` is considered as a relative URL, because it lacks the scheme and hostname part; the path `/titi`, however, is an absolute path.

```
<!DOCTYPE html PUBLIC
"-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml"
      lang="en" xml:lang="en">
  <head>
    <meta http-equiv="Content-Type"
          content="text/html; charset=utf-8" />
    <title>Example XHTML document</title>
  </head>
  <body>
    <p>This is a
      <a href="http://www.w3.org/">link to the
      <strong>W3C</strong>!</a></p>
  </body>
</html>
```

Figure 1: Example XHTML Document

Request	`GET /myResource HTTP/1.1` `Host: www.example.com`

Response	`HTTP/1.1 200 OK`
	`Content-Type: text/html; charset=ISO-8859-1`
	
	`<html>`
	`  <head><title>myResource</title></head>`
	`  <body><p>Hello world!</p></body>`
	`</html>`

Figure 2: HTTP example request and response

Although HTML pages are primarily seen thanks to a browser (and, most of the time, a graphical browser as Microsoft Internet Explorer or Mozilla Firefox), the HTML code is not supposed to describe the way the page will appear in a browser (it is the role of styling languages like CSS or XSL) but the core structure and content of the document in a way accessible to all kind of browsers and a wide variety of *user agents* such as the crawlers that we describe in Section 2. For this reason, it is important that HTML documents be valid against the W3C specifications; tools like the W3C validator available at `http://validator.w3.org/` can be of help. Sadly, because of a history of browser wars, browser limitations, browser permissiveness and author laziness, most of the (X)HTML pages on the Web are far from being valid, or even well-formed in the sense of XML well-formedness, accounting for what has been called *tag soup*.

Pages of the Web are accessed using the usual client-server architecture of the Internet: a *Web server* on a remote host accepts requests from a client for a given resource, and provides it to him. Two communication protocols are mainly used for this exchange: HTTP and HTTPS. The latter allows for encryption, authentication and advanced features such as session tracking; it is essential for e-commerce on the Web and all other sensitive applications, but rarely used for regular documents

and will not be discussed further. HTTP [IET99b], or *HyperText Transfer Protocol*, is a quite simple protocol built on top of the Internet protocols IP (*Internet Protocol*, for addressing) and TCP (*Transmission Control Protocol*, for transportation) that is widely used on the World Wide Web. Figure 2 shows an example request and response from, respectively, a Web client and a Web server: the request asks for the resource identified by the path `/myResource` on hostname `www.example.com` and the server answers that the document was found (code `200 OK`; other common codes are `404 NOT FOUND` or `500 SERVER ERROR`) and provides it. The reason why the hostname is given, whereas the server has already been contacted, is that a given server may have several different domain names, with different content (thus, `www.google.com` and `www.google.fr` point to the same machine). This *virtual hosting* is one of the novel feature of HTTP/1.1 with respect to previous versions. Other features of the HTTP protocol include login/password protection, content negotiation (the content served for a given URL is not fixed and depends on the preference indicated by the client), cookies (persistent chunks of information that are stored on the client, e.g., for session management purpose), *keep-alive* requests (several requests can be made to the same server without closing the connexion), and more.

2 Web Crawling

The first task to build a search engine over the Web is to retrieve and index a significant portion of it. This crawling is done by user agents that are called *crawlers*, *(Web) spiders* or *(Web) robots*. Their design raises a number of important engineering issues that will be discussed here.

Crawling the Web is basically just starting from a given URL or set of URLs, retrieving and indexing this page, discovering hyperlinks (mostly from the `<a>` elements of the HTML pages) on the page and repeat the process on each found link. There is no real termination condition here, as it is vain to try and retrieve the entire Web (which is actually virtually infinite, as already discussed), but the crawl can be terminated after some delay or after some number of URLs have been discovered or indexed. This is essentially a graph browsing problem, which can obviously approached by either *breadth-first* (all pages pointed by a page are indexed before the links they contain are analyzed) or *depth-first* (a page is indexed, and its links are extracted, as soon as a link to it is discovered) techniques. Obviously, because of the non-existence of termination conditions, and the possibility of being lost in *robot traps* (infinite paths in the graph), a breadth-first approach is more adapted here; actually, a mix of a breadth-first browsing with depth-first browsing of limited depth of each discovered site can be a good compromise.

An important subtask is the identification of duplicate pages in the Web, in order to avoid browsing them multiple times. Trivial duplicates are documents that share the same URL, though it can be written in slightly different ways: this means that a *canonization* of URLs has to be performed, to detect for instance that `http://example.com:80/toto` and `http://example.com/titi/../toto` are actually the same resource. The identification of other kind of duplicates, that do not have the same URL, is more intricate but also crucial, since it would not be very interesting for a user to get a list of identical pages as a result to a search engine query. Identical duplicates are easy to identify by hashing, but there are often some little differences between two pages (for instance, a date that is automatically generated at each request by the server, or random content such as *Tips of the day*) that are essentially duplicates. A way to discover such near-duplicates is by hashing each set of n consecutive tokens appearing in the pages for some fixed n: two pages that contain almost the same sets of such n-grams are likely to be near-duplicates.

There are also some *crawling ethics* to abide to. A *Standard for robot exclusion* [Kos94] have been proposed to allow webmasters to specify some pages not to be crawled by Web spiders (the reasons

```
User-agent: *
Allow: /searchhistory/
Disallow: /search
```

Figure 3: Example `robots.txt` robot exclusion file

can be varied: for confidentiality purposes, in order not to put too heavy a load on a resource-intensive Web application, to help robots not to fall into robot traps, etc.). This standard is followed by all major search engines, and consist in a `/robots.txt` file, that can be put at the root of every Web server and contain restrictions on what part of the website spiders are allowed to crawl. An example of such a file is given in Figure 3. It disallows the crawling of all URLs starting whose path starts with `/search`, with the exception of those starting with `/searchhistory/`, to any robots. Another way of expressing such limitations, at the level of a HTML page this time (which can be useful if a webmaster does not have control over the document root), is through a `<meta name="ROBOTS">` directive in the header of the document, such as

```
<meta name="ROBOTS" content="NOINDEX,NOFOLLOW">
```

which disallows robots to either index or follow links from the current Web page. Available keywords are `INDEX`, `FOLLOW`, `NOINDEX`, `NOFOLLOW`, with a default of `INDEX,FOLLOW`. Yet another way of influencing robot crawling and indexing is discussed in Section 4.3. A last rule that a spider programmer should respect is to avoid too many requests in a short time to a given host, that could result in DOS (Denial Of Service) from the host. A good rule of thumb is to wait between 100ms and 1s between two successive requests to the same Web server.

Because of this last rule, and because network delays are typically much higher than the time needed to process a Web page, it is crucial to send in parallel a large number of requests to different hosts; this also means that a per-host queue of URLs to be processed has to be managed. This parallel processing is typically performed using a multi-threaded environment, but asynchronous input and outputs (with for instance the `select` POSIX C function) provide the same functionality without the overhead introduced by threads. The *keep-alive* feature of HTTP/1.1 can also be used to chain requests (after some delay) to the same host. In large-scale crawlers for general Web search, the crawling itself will be run in parallel on a number of machines, that have to be regularly synchronized, which raises further issued not discussed here.

Another aspect of crawling is the refreshing of URLs: though we stated earlier that we did not want to crawl twice the same URL, it can be very important to do so in the context of perpetually changing Web content. Furthermore, it is also important to identify frequently changing Web pages in order to crawl them more often than others. Thus, the main page of an online newspaper should probably be crawled every day, while it may take up to a month to a large-scale crawler to crawl a significant portion of the Web. HTTP proposes a way to ask for a document if and only if it has been modified since some given date (`If-Modified-Since` header) but this is often unreliable, even more so in the context of dynamic documents which are regenerated at each request. Changes in Web pages have then to be identified by the crawler, without taking into account minor changes, for instance using techniques described above for identifying near-duplicates and crawling strategies have to be adapted accordingly.

We describe next the indexing of retrieved HTML documents, for keyword-based query purposes.

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 US $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.

Figure 4: Example set of documents

d_1 the_1 jaguar_2 is_3 a_4 new_5 world_6 mammal_7 of_8 the_9 felidae_{10} family_{11}
d_2 jaguar_1 has_2 designed_3 four_4 new_5 engines_6
d_3 for_1 jaguar_2 atari_3 was_4 keen_5 to_6 use_7 a_8 68k_9 family_{10} device_{11}
d_4 the_1 jacksonville_2 jaguars_3 are_4 a_5 professional_6 us_7 football_8 team_9
d_5 mac_1 os_2 x_3 jaguar_4 is_5 available_6 at_7 a_8 price_9 of_{10} us_{11} $\$199_{12}$ for_{13} apple's_{14} new_{15} family_{16} pack_{17}
d_6 one_1 such_2 ruling_3 family_4 to_5 incorporate_6 the_7 jaguar_8 into_9 their_{10} name_{11} is_{12} jaguar_{13} paw_{14}
d_7 it_1 is_2 a_3 big_4 cat_5

Figure 5: Tokenization of document set of Figure 4

3 Web Information Retrieval

Let us consider the set of 7 (one-sentence) documents represented in Figure 4. We describe next and illustrate on this particular example how to index a collection of text documents in a suitable way to efficiently answer keyword queries. This problem and related ones are known as *information retrieval* or, simply, *search* problems. We first describe general text preprocessing techniques that can be applied in such a context in Section 3.1 and then present the *inverted index* model in Section 3.2. We then proceed to the problem of answering keyword queries using such an index. Finally, we discuss in Section 3.5 what changes arise when other media than pure text are considered, especially text enriched with structural information as in HTML, and also multimedia content. Note that we do not use here the hyperlinked structure of the Web, which will be the topic of Section 4.

3.1 Text Preprocessing

The techniques described here are general techniques for dealing with text corpora. Depending upon the application, some variant or other has to be applied. Furthermore, the original language or languages of the document have a major impact on the preprocessing made. Some of the choices relevant to the context of Web indexing are discussed in the following sections.

3.1.1 Tokenization

The first step is to tokenize the initial documents into sequences or *tokens*, or simply words. This is illustrated on our example document set in Figure 5. At this stage, inter-word punctuation is generally removed and case is normalized (unless, obviously, the application requires differently, as may be the case in a search engine dedicated to linguists researching the usage of punctuations). This may seem like a very simple step where it is sufficient to replace whitespace and punctuation by token separators, but the problem is actually deeper than this:

- Whereas words are immediately visible in a language such as English, other languages (notably, Chinese or Japanese) do not use whitespace to mark word boundaries. Tokenization of such languages require much more complex procedures that typically use both advanced linguistic routines and dictionaries. Specifically for Chinese, an alternative is to use individual ideograms as tokens, but this may lead to invalid matches in query answering.

- Some care has to be taken for a number of textual oddities, such as acronyms, elisions, numbers, units, URLs, e-mail addresses, etc. They should probably be preserved as single tokens in most applications, and in any case should be dealt with consistently.

- Even in English, tokenization is not always obvious, especially with respect to compound words. An immediate question is whether to consider intra-word hyphens as token separator, but the problem is broader. Consider for instance the term *hostname* that can be found in corpus in three variant forms: *hostname*, *host-name* and *host name*. If we want to be able to use any of these terms to query all three variants, some analysis has to be performed, probably with the help of a lexicon, either to consider *host name* as a single compound word, or to break *host-name* into two tokens. The latter solution will also allow searching for *host* and *name*, which may be appropriate, depending on the context. Note that in languages where compounds are even more easily produced than in English, e.g., in German, such an analysis is indispensable.

3.1.2 Stemming

Once tokens are identified, an optional step is to perform some *stemming* in order to remove morphological markers from inflected words, or, more generally, to merge several lexically related tokens into a single *stem*. Such a step is often needed, for instance to be able to retrieve documents containing *geese* where *goose* is queried, but the degree of stemming varies widely depending upon the application (and upon the considered language, obviously: the notion of stemming does not make much sense in a language without any morphological variations like Chinese). Here is a scale of possible stemming schemes, from finest to coarsest:

Morphological stemming. This consists in the sole removal of bound morphemes (such as plural, gender, tense or mood inflections) from words. Note that this can be a very complex task in morphologically rich languages such as Turkish, or, in a lesser way, French, which require advanced linguistic processing for resolutions of *homographs* (different words that are written in the same way). Consider for instance the famous sentence in French: “Les poules du couvent couvent.” (The hens of the monastery brood.) Here, the first *couvent* [monastery] is an uninflected noun, which should stay as is, whereas the second *couvent* [brood] is an inflected form of the verb *couver* [to brood], which should be stemmed respectively. In English, the situation is simpler and plain procedures that remove final -s, -ed, -ing, etc., with a few adaptations for semi-regular (-y/-ies) or irregular (mouse/mice) inflections, can be enough. Note that some ambiguities remain, as illustrated with the word *stocking*, which can be either an uninflected noun, or an inflected form of the verb *stock*. Depending upon the application, one may choose either a cautious stemming (that do not remove all morphological markers, and will then fail to retrieve some query matches) or a more aggressive one (that will retrieve invalid query matches). A morphological stemming has been applied on our running example in Figure 6.

Lexical stemming. Stemming can be pushed further to merge lexically related words from different parts of speech, such as *policy*, *politics*, *political* or *politician*. An effective algorithm for such

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$_5$

Figure 6: Document set of Figure 4, after tokenization and stemming

d_1 jaguar$_2$ new$_5$ world$_6$ mammal$_7$ felidae$_{10}$ family$_{11}$
d_2 jaguar$_1$ design$_3$ four$_4$ new$_5$ engine$_6$
d_3 jaguar$_2$ atari$_3$ keen$_5$ 68k$_9$ family$_{10}$ device$_{11}$
d_4 jacksonville$_2$ jaguar$_3$ professional$_6$ us$_7$ football$_8$ team$_9$
d_5 mac$_1$ os$_2$ x$_3$ jaguar$_4$ available$_6$ price$_9$ us$_{11}$ \$199$_{12}$ apple$_{14}$ new$_{15}$ family$_{16}$ pack$_{17}$
d_6 one$_1$ such$_2$ rule$_3$ family$_4$ incorporate$_6$ jaguar$_8$ their$_{10}$ name$_{11}$ jaguar$_{13}$ paw$_{14}$
d_7 big$_4$ cat$_5$

Figure 7: Document set of Figure 4, after tokenization, stemming and stop-word removal

a stemming in English, Porter's stemming [Por80] and has been widely used. Note that further ambiguities arise, with for instance *university* and *universal* both stemmed to *univers*. This kind of stemming can also be coupled to the use of lexicons in order to merge synonyms or near-synonyms.

Phonetic stemming. The purpose of phonetic stemming is to retrieve words despite spelling variations or errors. Soundex [US 07] is a widely used loose phonetic stemming for English, originally used in U.S. censuses, that stems for instance both *Robert* and *Rupert* to *R163*. As can be seen from this example, it is a very coarse form of stemming, and should probably not be used in contexts where the precision of matches is important.

Note that in some circumstances, it can be useful to produce different indexes that use different form of stemming, to support both exact and approximate queries.

3.1.3 Stop-word removal

The presence of some words in documents, such as determiners (*the*, *a*, *this*, etc.), function verbs (*be*, *have*, *make*, etc.), conjunctions (*that*, *and*, etc.) is often not informative. Furthermore such words are very common in documents and indexing them often costly increase storage size. It is then common to remove them from documents at that point (or at least not to index them in the next stage), as illustrated on Figure 7 (to be compared with Figure 6).

3.2 Inverted Index

Once all needed preprocessing has been performed, the document set can be indexed in an *inverted index* that will make possible to answer keyword queries efficiently. An inverted index is just an index of all documents where each term (that is, each stemmed token that is not a stop word) occur. A (partial) example of inverted index for our example is given in Figure 8. For small-scale applications

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
...	

Figure 8: Partial inverted index for document set of Figure 4

family	$d_1/11, d_3/10, d_5/16, d_6/4$
football	$d_4/8$
jaguar	$d_1/2, d_2/1, d_3/2, d_4/3, d_5/4, d_6/8+13$
new	$d_1/5, d_2/5, d_5/15$
rule	$d_6/3$
us	$d_4/7, d_5/11$
world	$d_1/6$
...	

Figure 9: Partial inverted index for document set of Figure 4, with positions

where the index fits on a single machine, lists of occurrences of documents are usually stored, packed, for each given term, in a file that is then mapped to memory (using the POSIX system call `mmap`), while a secondary index gives the offset of each term in the index; database management systems can also be used, although they are kind of overkill in such a case. For large-scale applications, such as search engines for the Web, however, the index is distributed on a cluster of machines; a hashing function usually associates a given machine to a term, for indexing and index interrogation purposes. Because updating the index is typically quite costly (since it requires updating the document list of various terms, changing the structure of the index), index building is usually done once when all terms have been extracted, by sorting and merging term occurrence lists from each document. Besides, in the case where the index needs to be constantly updated because the crawling runs in parallel with the indexing, index updates are typically kept in memory and incorporated to the index in a batch processing at regular times.

Applications sometimes require to keep in the index information about the position of each term in the original document. This is for instance the case when phrases can be searched (this is usually done in search engine interfaces by enclosing phrases between quotes), or when the search engine allows operators that need this position information (e.g., the NEAR operator of Altavista, that requires two terms to be close to each other). This information can easily be stored in the index, by simple addition of the positions (as integers) next to the document the term occurs in, see Figure 9 for an illustration.

A last refinement that is often made on the inverted index is to assign some *weight* to the occurrence of a term in a document, depending on the relevance and informativeness of the term. A common weighting scheme is *tf-idf*, or *term frequency—inverse document frequency*: this scheme assigns a weight to a term that is proportional to its number of occurrences in the document, as well as raising the weight of terms that are present in few documents, being thus particularly informative and characteristic of this document. The mathematical definition for the weight $\text{tfidf}(t,d)$ of term t in

family	$d_1/11/.13$, $d_3/10/.13$, $d_6/4/.08$, $d_5/16/.07$
football	$d_4/8/.47$
jaguar	$d_1/2/.04$, $d_2/1/.04$, $d_3/2/.04$, $d_4/3/.04$, $d_6/8+13/.04$, $d_5/4/.02$
new	$d_2/5/.24$, $d_1/5/.20$, $d_5/15/.10$
rule	$d_6/3/.28$
us	$d_4/7/.30$, $d_5/11/.15$
world	$d_1/6/.47$
...	

Figure 10: Partial inverted index for document set of Figure 4, with positions and tf-idf weighting

document d is as follows:

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

where $n_{t',d'}$ is the number of occurrences of t' in d' and D is the set of all documents; the first term raises the weight of frequently occurring terms in the given document, while the second term depends negatively of the global frequency of the term in the document set. This weighting scheme can then be added too to the index, as shown on Figure 10; documents are usually stored in decreasing weight order, to improve the efficiency of top-k retrieval, as will be seen in the next section.

3.3 Answering Keyword Queries

Consider now the problem of answering a keyword query given an inverted index built as described in the previous section. If we want to retrieve all documents containing a given keyword, we just need to look up the (stemmed) keyword in the index and display the corresponding list of documents; associated weights give an indication of the relevance of each result to the keyword query. Consider now arbitrary multi-keyword boolean queries (containing AND, OR, NOT) operators, such as:

(*jaguar* AND *new* AND NOT *family*) OR *cat*.

They can be answered in the same way, by retrieving the document lists from all keywords appearing in the query and applying the set operations corresponding to the boolean operators (respectively, intersection, union and difference for AND, OR and AND NOT). Assigning a score to each document retrieved by the query is not completely straightforward, especially in the presence of OR or NOT operators. For queries that only contain the NOT operator, some monotonous functions of the scores (or weights) of all matched terms can be used; we discuss in Section 3.4 the cosine similarity, that can also be used here, but a simple and effective way to compute the global score of the document is just to add all scores of matched terms. Queries that precise the location of terms relatively to each other (phrase queries or queries with a NEAR operator) can be answered in the same way, by retrieving from the index all matching documents with the associated positions, and checking whether the conditions imposed by the query (such as, position of keyword t should be that of keyword t' minus one for the phrase query *"t t′"*) are satisfied.

In most applications, it is often desirable to return only a subset of the documents that match a query, since a user cannot be expected to browse through thousands or even millions of documents. Consider a conjunctive keyword query:

$$t_1 \text{ AND } \ldots \text{ AND } t_n$$

and let k be a fixed number of documents to be retrieved (for instance, 10 or 50). We describe here an efficient algorithm, known as Fagin's threshold algorithm [FLN01], to answer top-k queries (without having to retrieve and compute the intersection of all documents where each term occurs). We also make the assumption that, in addition to an inverted index that return documents containing a given term in decreasing weight order, we have another inverted index that can be used to directly check the weight of a term in a document (this can be done, for instance, with a dichotomic search of a document in an inverted index where documents are sorted by identifiers). We denote by $s(t,d)$ the weight of t in d (e.g., tfidf) and $g(s_1,\ldots,s_n)$ the monotonous function that computes the global score of a document given the weight of each term in the document (e.g., addition).

1. Let R be the empty list, and $m = +\infty$.
2. For each $1 \leq i \leq n$:
   (a) Retrieve the document $d^{(i)}$ containing term t_i that has the next largest $s(t_i,d^{(i)})$.
   (b) 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 $j \neq i$.
   (c) 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.

3.4 Clustering

If one wants to search on the Web information about the jaguar animal, one is probably not interested in the other meanings of the word *jaguar*, such as the car make or the version of Mac OS X. *Clustering* can be used in such contexts to partition a set of documents (the result of the keyword query) into a set of homogeneous document collections. The result of a clustered search for *jaguar* on the Clusty[7] search engine is shown on Figure 11.

One way to achieve such a clustering is the following. Start from some document set that is to be clustered. We shall see this document set in a *document vector space* model, that is the dual of the inverted index model: documents are described by the term that occur in them, with associated weighting, and each term is seen as a dimension of a vector space documents live in. The coordinate of a document d in this vector space, along the dimension corresponding t, will be the weight of t in d (say, tfidf(t,d)). We then consider the *cosine* similarity between two documents d and d', seen as vectors:

$$\cos(d,d') = \frac{d \cdot d'}{\|d\| \times \|d'\|}$$

where $d \cdot d'$ is the scalar product of d and d' and $\|d\|$ the norm of vector d. With this definition (which is a simple extension of the usual cosine function in the Euclidean plane), $\cos(d,d) = 1$ and $\cos(d,d') = 0$ if d and d' are orthogonal, that is, if they do not share any common term.

This definition of similarity is all that we need to apply standard clustering algorithms, for instance the following simple agglomerative clustering:

[7] `http://clusty.com/`

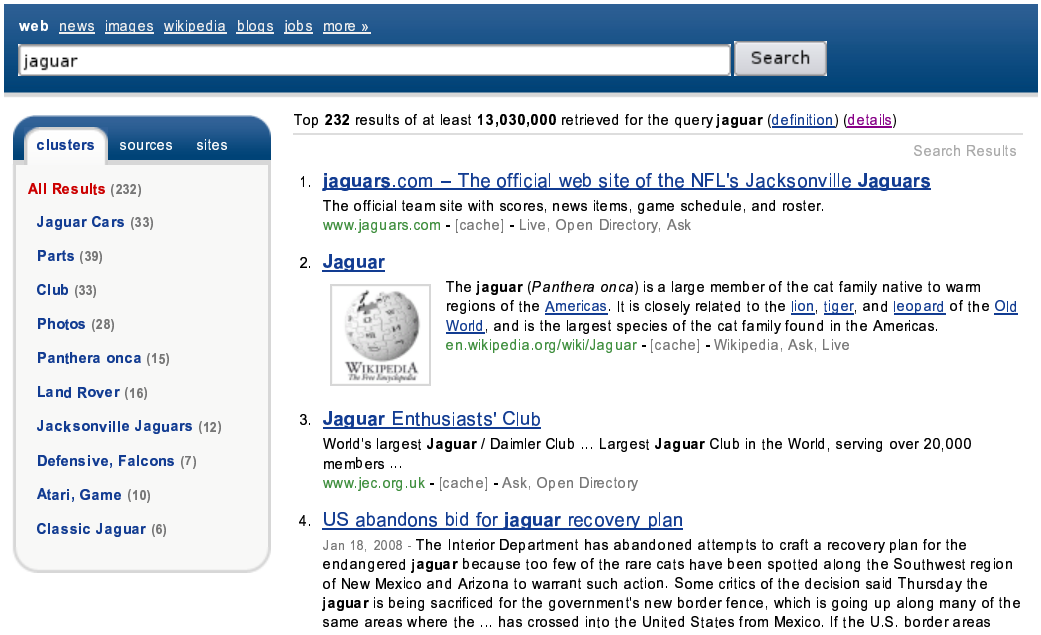


Figure 11: Example clustering from Clusty of the results of the query *jaguar*

1. Initially, each document forms its own cluster.
2. The similarity between two clusters is defined as the maximal similarity between elements of each cluster.
3. Find the two clusters whose mutual similarity is highest. If it is lower than a given threshold, end the clustering. Otherwise, regroup these clusters. Repeat.

Note that many other more refined algorithms for clustering exist.

3.5 Beyond Text Indexing

HTML Web pages are not just text, but text enriched with meta-information and document-level and character-level structure. This enrichment can be used in different ways: a separate index for the title or other meta-information of a page can be built and independently queried, or the tokens of a document that are emphasized can be given a higher weight in the inverted index. For some applications, the tree structure of Web pages can be stored and queried with languages such as XPath or XQuery (cf. the corresponding chapters); because most Web pages, even when they are well-formed and valid, do not really use HTML structural elements in a meaningful and consistent way, this approach is not very useful on the Web as a whole (see Section 5.1 for a discussion of how the semantic Web aims to change this).

Indexing of multimedia content on the Web (images, sound, music, videos) is still addressed as a text indexing problem, where the text comes from the context of the document: surrounding text, text in or around the links pointing to the content, filenames, or, in the case of videos, associated subtitles (this is especially used for indexing television broadcasts, which often include a subtitle track for people with hearing disabilities). More elaborate indexing of multimedia content is currently researched

on, with the help of speech recognition and sound, image, and video analysis techniques. An existing application of sound analysis is Musipedia[8] which proposes to find the partition corresponding to the melody a user is whistling. Another promising research application is image search from a similar image (e.g., to retrieve all pictures of a given person).

4 Web Graph Mining

As all hyperlinked environments, the World Wide Web can be seen as a directed graph in the following way: Web pages are vertices of the graph, while hyperlinks between vertices are edges. This viewpoint has led to major advances in Web search, notably with the PageRank and HITS algorithms that are presented below.

Extraction of knowledge from graphs, or *graph mining* has been used on other graph structures than the Web, for instance on the graph of publications, where edges are the citation links between publications; *cocitation analysis* relies on the observation that two papers that are cited by about the same set of papers are similar. Other graphs susceptible to this kind of analysis include graphs of dictionaries or encyclopedias or graphs of social networks.

4.1 PageRank

Though tf-idf weighting adds some relevance score to a document matching a keyword, it does not distinguish between reference documents that are highly trusted and obscure documents containing erroneous information. The idea of using the graph structure of the Web to assign some score to a document relies in the following idea or variants of it: if a document is linked by a large number of *important* documents, it is itself *important*.

PageRank [BP98], which was introduced with much success by the founders of the Google search engine, is a formalization of this idea. Let $G = (g_{ij})$ be the transition matrix of the Web graph (or a large part of it), that we assumed to be normalized in the following way:

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

This matrix describes a *random walk* on the pages of the Web: a *random surfer* goes from page to page, choosing with uniform probability any outgoing link. The *PageRank* of a page i can now be defined informally as the probability that the surfer that follows the random walk has arrived on page i at some distant given point in the future. Observe now that if v denotes the initial position as a column vector (say, a uniform column vector would mean that the random surfer start with uniform probability on each page), $(G^T)v$ is a column vector indicating the position after one step of the random walk. The PageRank can then be defined as the limit of this process, that is the PageRank of page i is the i-th component of the column vector:

$$\lim_{k \to +\infty} (G^T)^k v$$

if such a limit exists. Some problems arise with this definition. The limit (if it exists) could be dependent on the initial position of the random surfer, which is kind of disappointing from a robustness point of view. Besides, some pages may have no outgoing links (they are called *sinks*) which means that the random surfer will eventually be blocked on these pages. Note that one can show that neither

[8] `http://www.musipedia.org/`

of this problem occurs when the graph is strongly connected (i.e., there is a path in the graph from every node to every node), but the Web graph as a whole cannot be assumed to be strongly connected.

For these reasons, we introduce some change in our random surfer model: at each step of the random walk, with some fixed probability d (typically around 15%; $1-d$ is called the *damping factor*), the surfer goes to an arbitrary uniformly chosen page of the Web; otherwise, it follows the outgoing links of the page with uniform probability as before (and if there are no outgoing links, the surfer goes in all cases to an arbitrary uniformly chosen page of the Web). With these modifications, the PageRank of page i is defined as the i-th component of the column vector:

$$\lim_{k \to +\infty} ((1-d)G^T + dU)^k v$$

where G has been modified so that sink pages are replaced by pages with outgoing links to any page of the Web, and U is the matrix with all $\frac{1}{N}$ values where N is the number of vertices. One can show that this limit indeed exists, whenever $d > 0$ (Perron–Frobenius theorem) and is independent of the choice of the vector v, whenever $\|v\| = 1$. This formula can be used to compute the PageRank scores of all pages in an iterative way: starting from, say, the uniform column vector v, $((1-d)G^T + dU)v$ is computed by simple matrix multiplication, then $((1-d)G^T + dU)^2 v$ by another matrix multiplication, $((1-d)G^T + dU)^3 v$, etc., until convergence. This computation is illustrated in Figures 12 (initial uniform column vector) and 13 (limit of the iterative process)[9].

It is important to understand that PageRank assigns a *global* importance score to every page of the Web graph. This score is independent of any query. Then, PageRank can be used to improve scoring of query results in the following way: weights of documents in the inverted index are updated by a monotonous function of the previous weight and of the PageRank, say,

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

thus raising the weight (and therefore their order in query results) of important documents.

4.2 HITS

The *HITS* algorithm (Hypertext Induced Topic Selection) is another approach proposed by Kleinberg in [Kle99]. The main idea is to distinguish two kinds of Web pages: *hubs* and *authorities*. Hubs are pages that point to good authorities, whereas authorities are pages that are pointed to by good hubs. Note that, as with PageRank, we use again a mutually recursive definition that will lead to an iterative fixpoint computation. For example, in the domain of Web pages about automobiles, good hubs will probably be portals linking to the main Web page of car makes, that will be good authorities.

More formally, let G' be the transition matrix (this time, not normalized, i.e., with boolean 0 and 1 entries) of a graph (say, a subgraph of the Web graph). We consider the following iterative process, where a and h are column vectors, initially of norm 1:

$$\begin{cases} a := \frac{1}{\|G'^T h\|} G'^T h \\ h := \frac{1}{\|G' a\|} G' a \end{cases}$$

If some basic technical conditions on G' hold, we can show that this iterative process converges, to column vectors a and h which represent respectively the authority and hub scores of vertices of the graph.

[9] In this case, the damping factor was set to 1 but convergence was all the same guaranteed by the strong connectivity of the graph.

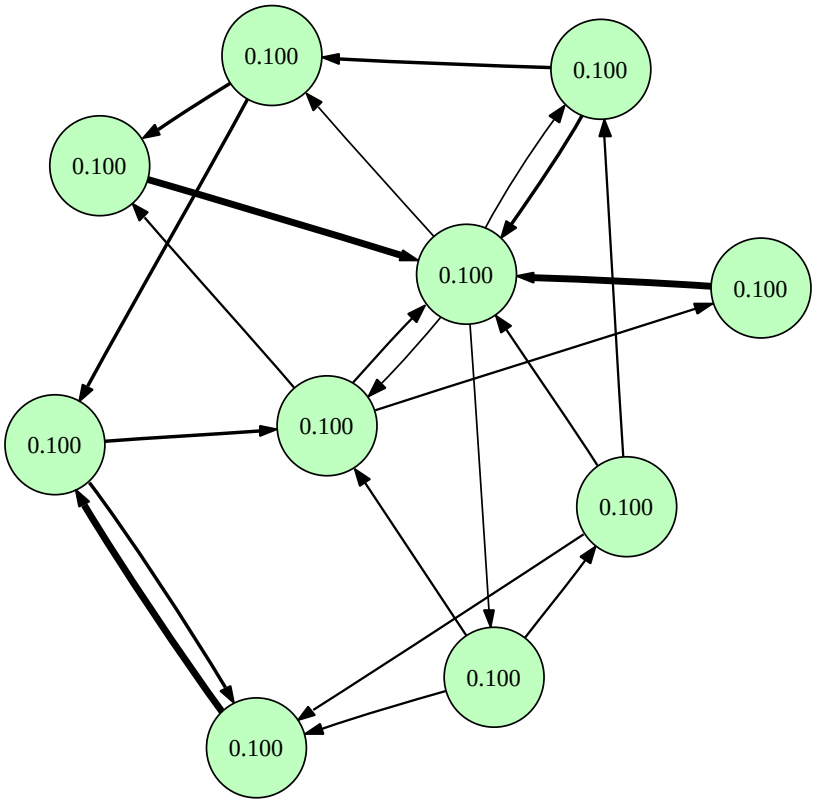


Figure 12: Example graph

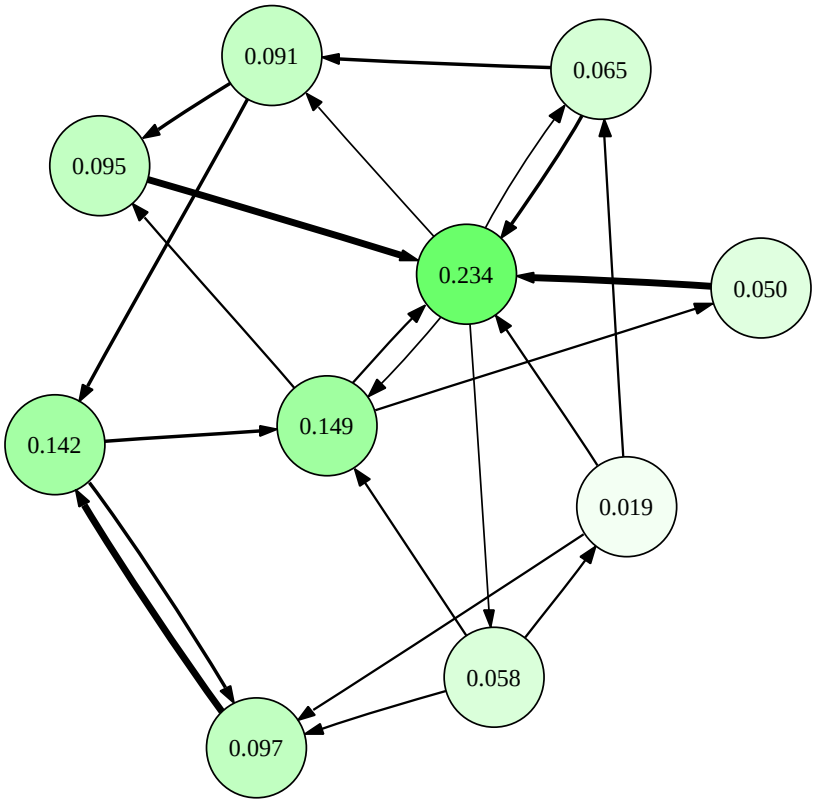


Figure 13: PageRank (damping factor of 1) for graph of Figure 12

Kleinberg propose then the following way of using authority scores to order query results from the Web:

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.

The process is here very different from PageRank, as authority scores are computed for each request (on a subgraph kind of centered around the original query). For this reason, and although HITS give interesting results, it is not as efficient as PageRank, for which all scores can be precomputed and top-k optimization is possible.

4.3 Spamdexing

The term *spamdexing* describes all fraudulent techniques that are used by unscrupulous webmasters to artificially raise the visibility of their website to users of search engines. As with virus and antivirus, or spam and spam fighting, spamdexing and the fight against it is an unceasing series of techniques implemented by spamdexers, closely followed by countertechniques deployed by search engines. The motivation of spamdexers is to bring users to their webpages so as to generate revenue from pay-per-view or pay-per-use content (especially in the industries of online gambling and online pornography), or from advertising.

A first set of techniques consists in lying about the content of a Web page, by adding keywords to a page that are unrelated to its content; this may be done either as text present in the page but invisible to users through the use of CSS, JavaScript or HTML presentational elements, or in the meta-information about the page that can be provided in the `<meta name="description">` or `<meta name="keywords">` tags in the header. As a result, current search engines tend not to give a strong importance to this kind of meta-information, or even to ignore them altogether. Furthermore, they implement automatic methods to find text hidden to a regular user and ignore it. In some cases, this is even used as a reason to lower the importance of the Web page.

PageRank and similar techniques are subject to *link farm* attacks, where a huge number of hosts on the Internet are 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. Countermeasures by search engines include detection of websites with empty or duplicate content, and the use of heuristics to discover subgraphs that look like link farms.

An assumption made by the graph mining techniques described earlier is that the addition of a link to a Web page is a form of approval of the content of the linked page, thus raising its importance. While this is mostly true when Web pages are written by a single individual or entity, this does not hold with user-editable content, such as wikis, guestbooks, blogs with comment systems, and so on. Spamdexers have an incentive to use these platforms to add links to their website. They can also make use of security exploits in Web application to the same effect. While most webmasters take care to control whatever is added to their websites and to remove spam content, this cannot be assumed on a global level. A partial solution to this is the possibility of adding a `rel="nofollow"` attribute

to all <a> links that have not been validated or are not approved by the webmaster (some content management systems and blog platforms automatically add this attribute to any link inside content provided by users). Most current-day Web spiders recognize this attribute and ignore this link.

5 Hot Topics

We briefly describe in this section research topics related to the search of information on the Web that are particularly active at the moment of writing. More information on some of these, as well as in-depth coverage of some other parts of this chapter, can be found in [Cha03].

5.1 Semantic Web

The content of the Web as it is now is essentially expressed in natural language and thus essentially unintelligible to machines, without resorting to imprecise information techniques (see Section 5.4). The *semantic Web* can be seen as an extension of the current Web, where human-readable content is annotated with machine-readable descriptions. It relies upon standards from the W3C such as RDF [W3C04b] for describing objects and relationships between objects in a machine-readable way, and RDFS [W3C04a] and OWL[W3C04c] for expressing the *schema* of the data model. RDF allows the description of graph structures of relations between concepts.

The semantic Web, if it becomes widespread, could be queried in much more precise a way than the current Web, since semantic content could be indexed. A language, SPARQL [W3C08], has been recently proposed for querying semantic Web sources, and one can imagine that a semantic Web search engine would allow the user to express complex semantic queries, such as asking for the birth date of a given individual. An important issue in the semantic Web community is that schemata, or ontologies, expressed in RDFS or in OWL, are not standardized across the Web. Semantic Web search would then also require to integrate data sources using different schemata.

5.2 Web 2.0

Web 2.0 is a *buzzword* that has appeared recently to refer to recent changes in the Web, notably:

- Web applications with rich dynamic interfaces, especially with the help of AJAX technologies (AJAX stands for *Asynchronous JavaScript And XML* and is a way for a browser to exchange data with a Web server without requiring a reload of a Web page); it is exemplified by GMail[10] or Google Suggest[11];
- user-editable content, collaborative work and social networks, e.g. in blogs, wikis such as Wikipedia[12], and social network websites like MySpace[13] and Facebook[14];
- aggregation of content from multiple sources (e.g., from RSS feeds) and personalization, that is proposed for instance by Netvibes[15] or Yahoo! Pipes[16].

[10] http://mail.google.com/
[11] http://www.google.com/webhp?complete=1
[12] http://www.wikipedia.org/
[13] http://www.myspace.com
[14] http://www.facebook.com/
[15] http://www.netvibes.com/
[16] http://pipes.yahoo.com/pipes/

Though Web 2.0 is more used in marketing contexts than in the research community, some interesting research problems are related to these technologies, especially in the application of graph mining techniques similar to those employed on the graph of the Web to the graph of social network websites, and in the works about *mashups* for aggregating content from multiple sources on the Web.

5.3 Deep Web

The *deep Web* (also known as *hidden Web* or *invisible Web*) is the part of Web content that lies in online databases, typically queried through HTML forms, and is not usually accessible by following hyperlinks. As classical crawlers only follow these hyperlinks, they do not index the content that is behind forms. A 2001 study [Bri00] estimated that there could be as much as 500 times more content on the deep Web than on the *surface Web* that is indexed by search engines. This number may sound surprising, but there is no doubt that much high-quality information is present on the deep Web: all *Yellow pages* directories, information from the US *Census bureau*, weather or geolocation services, etc.

There are two approaches to the indexing of the deep Web. A first possibility is an *extensional* approach, where content from the deep Web is generated by submitting data into forms, and the resulting Web pages are stored in an index, as with classical Web content. A more ambitious *intensional* approach is to try and understand the structure and semantics of a service of the deep Web, and to store this semantic description in an index. A semantic query from a user would then be dispatched to all relevant services, and the information retrieved from them. Whatever the method, searching the deep Web requires first discovering all relevant forms, and some analysis to understand what data to submit to a form. In the *intensional* approach, deep web search is also needed to extract information from the pages resulting from the submission of a form, which is the topic of the next section.

5.4 Information Extraction

Classical search engines do not try to extract information from the content of Web pages, they only store and index them as they are. This means that the only possible kind of queries that can be asked is keyword queries, and provided results are complete Web pages. The purpose of Web *information extraction* is to provide means to extract structured data and information from Web pages, so as to be able to answer more complex queries. For instance, an information extractor could extract phone numbers from Web pages, as well as the name of their owner, and provide an automatically built directory service. Information extraction is facilitated by very structured Web pages, such as those that are dynamically generated on response to the submission of an HTML form (e.g., Figure 5.4); a *wrapper* for this kind of dynamic site can be generated, in order to abstract away its interface.

A survey of existing information extraction techniques on the Web can be found in [CKGS06]. Most works are in a supervised or semi-supervised context, where humans pre-annotate Web pages whose content is to be extracted, or where human give some feedback on automatic wrapper construction. Unsupervised approaches rely either on the detection of linguistic or sentence-level patterns that express some concept or relation between concepts (e.g., addresses usually follow some kind of fixed format that can be discovered in corpus; textual patterns like *was born in* `year` can be found to automatically extract birth dates of individuals), or the detection of structural patterns in the Web page (repetitive structures such as tables or lists, for instance).

Showing results 1 through 25 (of 94 total) for all:xml

1. cs.LO/0601085 [abs, ps, pdf, other] :
Title: **A Formal Foundation for ODRL**
Authors: Riccardo Pucella, Vicky Weissman
Comments: 30 pgs, preliminary version presented at WITS-04 (Workshop on Issues in the Theory of Security), 2004
Subj-class: Logic in Computer Science; Cryptography and Security
ACM-class: H.2.7; K.4.4

2. astro-ph/0512493 [abs, pdf] :
Title: **VOFilter, Bridging Virtual Observatory and Industrial Office Applications**
Authors: Chen-zhou Cui (1), Markus Dolensky (2), Peter Quinn (2), Yong-heng Zhao (1), Francoise Genova (3) ((1)NAO China, (2) ESO, (3) CDS)
Comments: Accepted for publication in ChJAA (9 pages, 2 figures, 185KB)

3. cs.DS/0512061 [abs, ps, pdf, other] :
Title: **Matching Subsequences in Trees**
Authors: Philip Bille, Inge Li Goertz
Subj-class: Data Structures and Algorithms

4. cs.IR/0510025 [abs, ps, pdf, other] :
Title: **Practical Semantic Analysis of Web Sites and Documents**
Authors: Thierry Despeyroux (INRIA Rocquencourt / INRIA Sophia Antipolis)
Subj-class: Information Retrieval

5. cs.CR/0510013 [abs, pdf] :
Title: **Safe Data Sharing and Data Dissemination on Smart Devices**
Authors: Luc Bouganim (INRIA Rocquencourt), Cosmin Cremarenco (INRIA Rocquencourt), François Dang Ngoc (INRIA Rocquencourt, PRISM - UVSQ), Nicolas Dieu (INRIA Rocquencourt), Philippe Pucheral (INRIA Rocquencourt, PRISM - UVSQ)
Subj-class: Cryptography and Security; Databases

Figure 14: Example pages resulting from the submission of a HTML form

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.

Exercises

Exercise 1 *(1) Use Google. For each query, note the number of answers. Query "Bonnie and Clyde", "bonnie clyde", "bonny and Clyde", "Bonnie or Clyde", "bonnieclyde", "Bonnie and Bonnie". (2) Analyze your results. (3) Consider the same queries with AltaVista, Ask Jeeves, Yahoo! and MSN Search. Compare.*

Exercise 2 *Consider the document set example from Figure 4. Suppose that we want to index the term* be *(we consider therefore that it is not a stopword). Compute the line of the inverted index for term* be*, with positions and tf-idf weighting.*

Exercise 3 *Use Fagin's threshold algorithm to compute the top-*2 *result of the query:*

jaguar AND *new*

on the inverted index of Figure 10.

Exercise 4 *Implement the PageRank algorithm as described in the section for a graph of up to one thousand pages. Then:*

1. *Test it on the graph of Figure 12.*
2. *Add a few sinks and test your algorithm again.*
3. *Pick the page p with the least PageRank. Add some new nodes to simulate a link farm. How many pages do you need to introduce to promote p as the most popular.*

Exercise 5 *Find a free crawler on the Web and play with it.*

Exercise 6 *Write a "mini" crawler. Its input is a few words and the URL of a "seed page", e.g., your homepage. The crawler should crawl, say one hundred pages, and sort the words based on their number of occurrences. Try different crawling strategies: depth first, breadth first, popular first. Bonus: use some stemming.*

Exercise 7 *Choose a few interesting pages, e.g., news, music. Try to find metadata for these pages: author, date, purpose, citation of sources, copyright, etc. How much metadata could be found inside the pages?*

Exercise 8 *When ask a keyword query, a metasearch engine queries several search engines and aggregate their answers. Find some on the Web, e.g., metacrawler, and test them.*

Exercise 9 *Find the homepages of the authors of this book. Add pointers to these pages from your own homepage. This will be tested using Google "Find pages that link to the page: ".*

References

[BP98] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks*, 30(1–7):107–117, April 1998.

[Bri00] BrightPlanet. The deep Web: Surfacing hidden value. White Paper, July 2000.

[Cha03] Soumen Chakrabarti. *Mining the Web: Discovering Knowledge from Hypertext Data*. Morgan Kaufmann, San Fransisco, USA, 2003.

[CKGS06] Chia-Hui Chang, Mohammed Kayed, Mohem Ramzy Girgis, and Khaled F. Shaalan. A survey of Web information extraction systems. *IEEE Transactions on Knowledge and Data Engineering*, 18(10):1411–1428, October 2006.

[FLN01] Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal aggregation algorithms for middleware. In *Proc. ACM Symp. on Principles of Database Systems (PODS)*, Santa Barbara, USA, May 2001.

[IET99a] IETF. Request For Comments 1034. Domain names—concepts and facilities. `http://www.ietf.org/rfc/rfc1034.txt`, June 1999.

[IET99b] IETF. Request For Comments 2616. Hypertext transfer protocol—HTTP/1.1. `http://www.ietf.org/rfc/rfc2616.txt`, June 1999.

[Kle99] Jon M. Kleinberg. Authoritative Sources in a Hyperlinked Environment. *Journal of the ACM*, 46(5):604–632, 1999.

[Kos94] Martijn Koster. A standard for robot exclusion. `http://www.robotstxt.org/orig.html`, June 1994.

[Por80] Martin F. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, July 1980.

[US 07] US National Archives and Records Administration. The Soundex indexing system. `http://www.archives.gov/genealogy/census/soundex.html`, May 2007.

[W3C99] W3C. HTML 4.01 specification, September 1999. http://www.w3.org/TR/REC-html40/.

[W3C02] W3C. XHTML 1.0: The extensible hypertext markup language (second edition). http://www.w3.org/TR/xhtml1/, August 2002.

[W3C04a] W3C. RDF vocabulary description language 1.0: RDF schema. http://www.w3.org/TR/rdf-schema/, February 2004.

[W3C04b] W3C. RDF/XML syntax specification. http://www.w3.org/TR/rdf-syntax-grammar/, February 2004.

[W3C04c] W3C. Web Ontology Language (OWL). http://www.w3.org/2004/OWL/, 2004.

[W3C08] W3C. SPARQL query language for RDF. http://www.w3.org/TR/rdf-sparql-query/, January 2008.