Modern Information Retrieval

the concepts and technology behind search

Second edition

Ricardo Baeza-Yates
Berthier Ribeiro-Neto

Modern Information Retrieval

The Concepts and Technology behind Search

Ricardo Baeza-Yates
Berthier Ribeiro-Neto

Second edition

Addison-Wesley

Harlow, England • Reading, Massachusetts
Menlo Park, California • New York
Don Mills, Ontario • Amsterdam • Bonn
Sydney • Singapore • Tokyo • Madrid
San Juan • Milan • Mexico City • Seoul • Taipei

Chapter 11

Web Retrieval

with Yoelle Maarek

11.1 Introduction

Tim Berners-Lee conceived the conceptual Web in 1989, tested it successfully in December of 1990 [192], and released the first Web server early in 1991. It was called World Wide Web, but it is referred throughout this book as, simply, the *Web*. At that time, no one could have imagined the impact that the Web would have. The Web boom, characterized by exponential growth on the volume of data and information, imply that various daily tasks such as e-commerce, banking, research, entertainment, and personal communication can no longer be done outside the Web if convenience and low cost are to be granted.

The amount of textual data available on the Web is estimated in the order of petabytes. In addition, other media, such as images, audio, and video, are also available in even greater volumes. Thus, the Web can be seen as a very large, public and unstructured but ubiquitous data repository, which triggers the need for efficient tools to manage, retrieve, and filter information from the Web. As a result, Web search engines have become one of the most used tools in the Internet. Additionally, information finding is also becoming more important in large Intranets,[1] in which one might need to extract or infer new information to support a decision process, a task called data mining (or Web mining for the particular case of the Web).

The very large volume of data available, combined with the fast pace of change, make the retrieval of relevant information from the Web a really hard task. To cope with the fast pace of change, efficient crawling of the Web has become essential, as discussed in Chapter 12. In addition, crawling is also relevant to other problems such as information extraction and Web data mining.

[1] Intranet refers to the computer network built inside an organization, which may or may not be connected to the Internet itself.

In spite of recent progress in image and non-textual data search in general, the existing techniques do not scale up well on the Web (see Chapter 14). Thus, search on text continues to be the most popular and most studied alternative. While most search engines are based in the United States and focus on documents in English, there are important non-US search engines that have been designed for specific languages and can handle various scripts and alphabets, such as Kanji or Cyrillic. Examples include Baidu in China, Yandex in Russia, and Naver in South Korea.

We also notice that search continues to be dominated by a "syntactic" paradigm in which documents that contain user specified words or patterns are retrieved in response to a query. As discussed in Chapter 3, such words or patterns may or may not reflect the intrinsic semantics of the text. An alternative approach to syntactic search is to do a natural language analysis of the text. Techniques to preprocess natural language and extract the text semantics have been around for a while, yet they are not very effective. In fact, except for some fast entity extraction tools that have been devised recently [79], these techniques are too costly to be used with large amounts of data [86]. In addition, in most cases they are only effective with well written and structured text [1765], in combination with a thesaurus, or other contextual information such as a specific knowledge domain. For a more detailed discussion on structured data retrieval, see Chapter 13.

To simplify the problem, search engine designers make several assumptions about the Web that are not always valid. Regarding the data, they have implicitly assumed that a physical file, containing a Web page or another type of data, is a single logical document. However, we may have more than one logical document per page (*e.g.*, in a newspaper) or many files for one document (*e.g.*, a thesis). Regarding the users' needs, they have initially assumed that the main goal was straightforward information seeking and that those needs were less diverse than what they really are. However, as we discussed in Chapter 7, these two assumptions are not valid anymore.

There are basically two main forms of exploring the Web. The first one is to issue a word-based query to a search engine that indexes a portion of the Web documents. The second one is to browse the Web, which can be seen as a sequential search process of following hyperlinks, as embodied for example, in Web directories that classify selected Web documents by subject. Additional methods exist, such as taking advantage of the hyperlink[2] structure of the Web, yet they are not fully available, and likely less well known and also much more complex. We cover all these forms of Web search here, with more emphasis on the first two.

The rest of the chapter is organized as follows. We first discuss the challenges of searching the Web, followed by some Web characteristics so as to better allow an understanding of the complexity of the problem. Next, we discuss in detail the architecture of search engines, ranking models, and Web data management. Then we thoroughly cover all the issues related to how users typically interact with search engines. We continue with browsing, as compared to search, and also discuss methods beyond browsing. We conclude by discussing some problems and challenges related to Web search such as Web mining, computational advertising, metasearch, current trends, and research issues. As Web research is a very dynamic field, we may have missed some important work, for which we apologize in advance.

[2]We will use hyperlink or link to denote a pointer (anchor) from a Web page to another Web page.

The reader should be aware that by the time this book is in print, or simply picked up from a shelf and read in subsequent years, many of the features described here will have become obsolete,[3] and that our references to URLs might have turned into broken links or might have had their contents changed. In the extreme case, some key search engines might have entirely disappeared[4] or new ones might have been launched. Even so, we expect that the basic principles described here will remain stable enough in the upcoming years so as to provide the interested reader with an understanding of the basic science behind search engines.

11.2 A Challenging Problem

Let us now consider the main challenges posed by the Web with respect to search. We can divide them in two classes: those that relate to the data itself, which we refer to as data-centric, and those that relate to the users and their interaction with the data, which we refer as interaction-centric. Data-centric challenges are varied and include:

- Distributed data. Due to the intrinsic nature of the Web, data spans over a large number of computers and platforms. These computers are interconnected with no predefined topology and the available bandwidth and reliability on the network interconnections vary widely.

- High percentage of volatile data. Due to Internet dynamics, new computers and data can be added or removed easily. To illustrate, early estimations showed that 50% of the Web changes in a few months [859, 793, 257, 373, 561, 1215, 495]). Search engines are also confronted with dangling (or broken) links and relocation problems when domain or file names change or disappear.

- Large volume of data. The fast growth of the Web poses scaling issues that are difficult to cope with, as well as dynamic Web pages, which are in practice unbounded.

- Unstructured and redundant data. The Web is not a huge distributed hypertext system, as some might think, because it does not follow a strict underlying conceptual model that would guarantee consistency. Indeed, the Web is not well structured either at the global or at the individual HTML page level. HTML pages are considered by some as semi-structured data in the best case. Moreover, a great deal of Web data are duplicated either loosely (such as the case of news originating from the same news wire) or strictly, via mirrors or copies. Approximately 30% of Web pages are (near) duplicates [269, 1467, 278, 559, 560, 120]. Semantic redundancy is probably much larger.

- Quality of data. The Web can be considered as a new publishing media. However, there is, in most cases, no editorial process. So, data can be inaccurate, plain wrong, obsolete, invalid, poorly written or, as if often the case, full of errors, either innocent (typos, grammatical mistakes, OCR errors, etc.) or malicious. Typos and mistakes, specially in foreign names are pretty common.

[3] See our previous comment about the fast pace of change of the Web.
[4] Just consider the case of Excite when the first edition of this book was published.

- Heterogeneous data. Data not only originates from various media types, each coming in different formats, but it is also expressed in a variety of languages, with various alphabets and scripts (*e.g.* India), which can be pretty large (*e.g.* Chinese or Japanese Kanji).

Many of these challenges, such as the variety of data types and poor data quality, cannot be solved by devising better algorithms and software, and will remain a reality simply because they are problems and issues (consider, for instance, language diversity) that are intrinsic to human nature.

The second class of challenges are those faced by the user when interacting with the retrieval system. We distinguish two types of such interaction-centric challenges:

- Expressing a query. Human beings have needs or tasks to accomplish, which are frequently not easy to express as "queries". Queries, even when expressed in a more natural manner, are just a reflection of information needs and are thus, by definition, imperfect. This phenomenon could be compared to Plato's cave metaphor, where shadows are mistaken for reality.

- Interpreting results. Even if the user is able to perfectly express a query, the answer[5] might be split over thousands or millions of Web pages or not exist at all. In this context, numerous questions need to be addressed. Examples include: How do we handle a large answer? How do we rank results? How do we select the documents that really are of interest to the user? Even in the case of a single document candidate, the document itself could be large. How do we browse such documents efficiently?

Given the intrinsic problems posed by the Web, the key challenge for the user is to conceive a good query to be submitted to the search system, one that leads to a manageable and relevant answer. The key challenge for the retrieval system is to do a fast search and give back relevant answers, even for poorly formulated queries, as is the common case on the Web (see section 7.2.1).

In the current state of the Web, search engines need to deal with plain HTML and text, as well as with other data types, such as multimedia objects, XML data and associated semantic information, which can be dynamically generated and are inherently more complex. To make this distinction clear, in the rest of the chapter we use the term "*Web pages*" to denote HTML documents (see section 6.4.2) and the term "*Web documents*" to denote all available data types on the Web.

If the Semantic Web becomes a reality and overcomes all its inherent social issues, an XML-based Web, with standard semantic metadata and schema, might become available. In this hypothetical world, IR would become easier, and even multimedia search would be simplified. Spam would be much easier to avoid as well, as it would be easier to recognize good content. On the other hand, new retrieval problems would appear, such as XML processing and retrieval, and Web mining on structured data, both at a very large scale.

[5]We will sometimes use the term "answers" or "answer" to designate the set of results for a given query. The context will make it clear whether we are indeed considering a set rather than a specific answer to a natural-language question.

11.3 The Web

In this section we discuss the main characteristics of the Web, as well as mathematical models to describe its content and structure. There have been many studies investigating the Web in specific countries, which have shown that many properties and characteristics of a subset of the Web are also valid (and applicable) in the context of the global Web [97]. In spite of this, we still do not have a full understanding of the Web and its dynamics [794, 258, 1361, 98, 1011, 1010, 890].

11.3.1 Characteristics

Measuring the Internet and in particular the Web, is a difficult task due to its highly dynamic nature. At the time of writing (ISC survey of April 2010), there were more than 750 million computers in more than 200 countries directly connected to the Internet, many of them hosting Web servers [812]. Further, the estimated number of Web servers currently exceeds 206 million, according to the Netcraft Web survey of June, 2010 [1196]. Considering these numbers, we can say that there is about one Web server per every four computers directly connected to the Internet.

In two interesting articles, already (sadly) outdated, Bray [250] and Woodruff *et al.* [1719] studied different statistical measures of the early Web. According to the first, in November 1995 there were 11 million Web pages, while according to the second there were 2.6 Web million pages. A first question is how many different institutions (not Web servers) maintain Web data. This number is smaller than the number of servers, because many places have multiple servers. The exact number is unknown, but should be larger than 40% of the number of Web servers (this percentage was the value back in 1995). The size estimations at the beginning of 1998 ranged from 200 to 320 million, with 350 million as the best estimate in July of 1998 [198]. More recent studies on the size of search engines [135, 685] estimated that there were over 20 billion pages in 2005, and that the size of the static[6] Web is roughly doubling every eight months. The exact number of static Web pages was important before the use of dynamic pages[7] became popular. Nowadays, the Web is infinite for practical purposes, as we can generate an infinite number of dynamic pages (*e.g.* consider an on-line calendar). A taxonomy of Web pages is presented in section 12.3.1.

The most popular formats of Web documents are HTML, followed by GIF and JPG (both images), ASCII text, and PDF, in that order [97]. The most popular compression tools used are GNU zip, Zip, and Compress. With regard to HTML pages, there are many characteristics and statistics of interest, some of which are as follows. First, most HTML pages are not standard, meaning that they do not comply with all the HTML specifications. In fact, if browsers were behaving as strict HTML compilers, many pages would not be rendered. In addition, although HTML is an instance of SGML, HTML documents seldom start with a formal document type definition. Second, HTML pages are small (around 10KB) and usually contain few images. Most pages use images for presentation purposes such as colored bullets and lines. The average page contains between 5 and 15 hyperlinks (more than 8 links on average) and most of them are local, that is, they point to pages in their own Web

[6] Pages that are files in a Web server.

[7] A dynamic page is a HTML page that is built at the time the user interacts with a Web server.

server hierarchy. On average, the number of external pages pointing to any given page, is close to zero! Typically, only local links, *i.e.*, pages from the same domain, point to a given page. This is true even for homepages of large Web sites.

The most referenced[8] sites are the main Internet companies. On the other hand, the sites with most links to outside sites are directories such as the Yahoo![9] directory or the Open Directory Project[10] and Web 2.0 sites such as Wikipedia.[11] In some sense, these sites that aggregate links are "the glue" of the Web. Without them, we would have more isolated portions or "islands", as is the case with many personal Web pages.

Regarding the languages used in Web pages, there were three early studies. The first study was done by Funredes [1263] from 1996 to 1998, and later continued until 2005. It used the AltaVista search engine and was based on searching different words in different languages. Later, other search engines were used. This technique might not be significant statistically, but the results are consistent with the second study. This study was done by Alis Technology [28] and was based on an automatic software that detects the language used. One of the goals of the study was to test such software by running it in 8 thousand Web servers. The last early study was done by OCLC in June of 1998 [1230] by sampling Internet IP addresses and using the SILC language identification software. Newer studies of this kind are not available due to the size of the current Web.

In the year 2000, it was estimated that around 70% of the pages were written in English, and that the numbers of words available in other languages was growing faster than the number of words in English [676]. On January 2003, Google Zeitgeist[12] showed that around 50% of the queries to Google were using English, down from around 60% in 2001. Newer results are available only for specific countries [97].

11.3.2 Structure of the Web Graph

It is now commonly agreed that the Web can be viewed as a graph in which, in the simplest view, the nodes represent individual pages and the edges represent links between pages. The global structure of the Web graph has been extensively studied. The most complete study was conducted by Broder *et al.* in [271] and compared the topology of the Web graph to a *bow-tie*. This metaphor is schematically represented in Figure 11.1. With a bit of imagination, the reader can visualize a bow-tie where the largest strongly connected component (SCC) plays the role of the central knot of the tie.

Further refinements of this model identified areas inside the SCC or CORE component as described in [98, 506]. One limitation of this model lies in the fact that, as we already mentioned in the introduction, a page is not always a logical unit. The components identified in the graph were the following:

(a) CORE: sites that compose the strongly connected component of the graph. By definition, one can navigate from any site in CORE to any other site in CORE.

[8] We are talking here about sites being referenced by at least 1 million pages.
[9] yahoo.com/dir
[10] dmoz.org
[11] wikipedia.org
[12] URL: http://www.google.com/press/zeitgeist.html

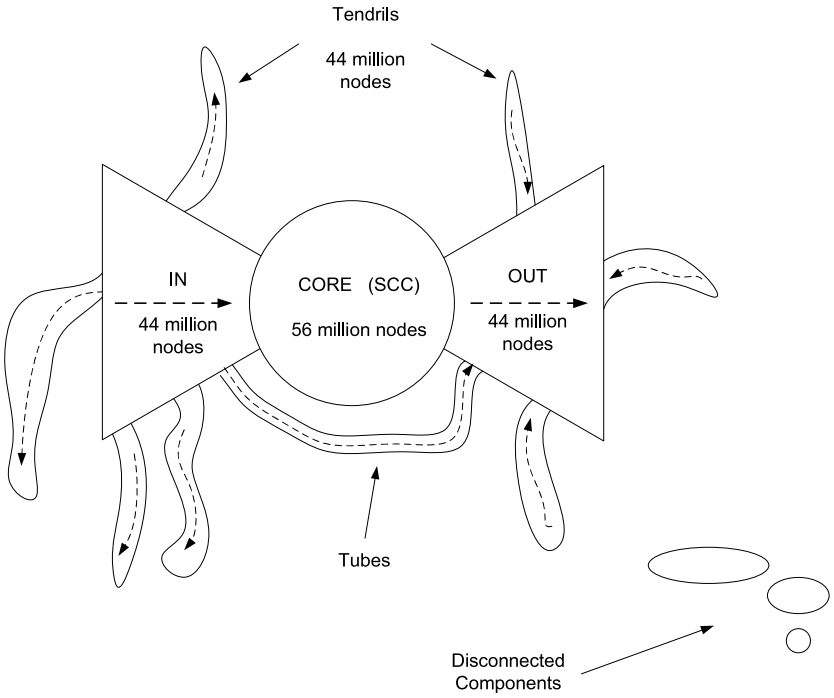

Figure 11.1: Original "bow-tie" structure of the Web (adapted from [271]).

(b) IN: sites that can reach sites in CORE but cannot be reached from sites in CORE.

(c) OUT: sites that can be reached from sites in CORE, but without a path to go back to CORE.

(d) TUBES: sites in paths that connect directly IN to OUT and that are outside CORE.

(e) TENTACLES or TENDRILS: sites that can be reached from sites in IN (T.IN) and sites that only reach sites in OUT (T.OUT), but that do not belong to the previous components.

(f) DISCONNECTED or ISLANDS: unconnected sites, whose connected component may have a structure similar to the whole Web.

In [94], this notation was extended by dividing the CORE component into four parts, as described below.

(a) Bridges: sites in CORE that can be reached directly from the IN component and that can reach directly the OUT component.

(b) Entry points: sites in CORE that can be reached directly from the IN component but are not in Bridges.

(c) Exit points: sites in CORE that reach the OUT component directly, but are not in Bridges.

(d) Normal: sites in CORE not belonging to the previously defined sub-components.

Figure 11.2 shows this more refined view of the bow-tie structure.

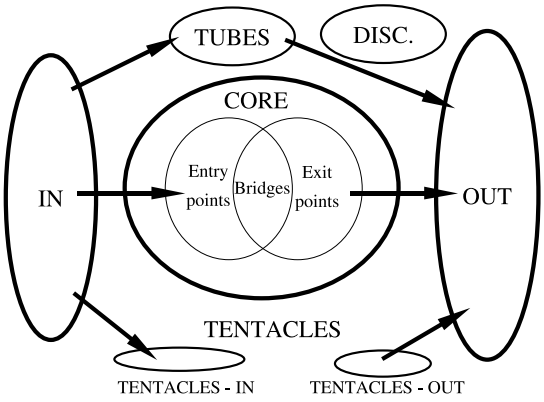

Figure 11.2: Schematic depiction of the macroscopic "bow-tie" structure of the Web.

In all the (bounded) studies of the Web [97], the CORE component is composed of a minority of the Web sites. On the other hand, it has a heavier density of Web pages. They also show that there is a correlation between structure and quality of the content, already known from link analysis (more connected, better quality). Some studies hint that the number of ISLANDS is much larger than we may think, as most of them are not connected to the Web and hence, not easy to find unless they are registered with the search engines [121].

11.3.3 Modeling the Web

All characterizations of the Web show that the quantitative properties of the CORE component follow a power law distribution. A general *power law* is a function that is invariant to scale changes. Its simplest form is:

$$f(x) = \frac{a}{x^{\alpha}} \quad \text{with} \quad \alpha > 0 \tag{11.1}$$

It is easy to verify that a scale change of the form $c \cdot x$, where c is a constant, just changes the constant a but not the shape of the function. This invariance is also called self-similarity. If $f(x)$ is a probability distribution, a must be set such that the sum of all probabilities is 1. Then a exists as a constant only if $\alpha > 1$. In that case, depending on the value of α, the moments of the distribution will be finite or not. When $\alpha \leq 2$, the average and all higher-order moments are infinite. When $2 < \alpha \leq 3$,

the mean exists, but the variance and higher-order moments are infinite, and so on. Examples of Web measures that follow a power law include:

- the number of pages per Web site and the number of Web sites per domain, as shown by studies of the content; and
- the incoming and outgoing link distributions, as well as the number of connected components, as shown by studies of the link structure.

This is true not only for the Web graph, but also for the host-graph, which is the connectivity graph at the level of Web sites. Table 11.1 shows a summary of the main findings adapted from [97]. For the page size, there are two exponents: one for pages with less than 20KB, and one for the rest. The same for the out-degree: one for pages with roughly less than 20 out-links, and one for pages with more out-links.

Region	Page Size		Pages	In-	Out-degree	
	Small	Large	per site	degree	Small	Large
Brazil	0.3	3.4	1.6	1.89	0.67	2.71
Chile	0.4	3.2	1.6	2.01	0.72	2.56
Greece	0.4	3.2	1.6	1.88	0.61	1.92
Indochina	n/a	n/a	1.2	1.63	0.66	2.62
Italy	n/a	n/a	1.3	1.76	0.68	2.52
South Korea	0.4	3.7	3.2	1.90	0.29	1.97
Spain	n/a	2.25	1.1	2.07	0.86	4.15
United Kingdom	n/a	n/a	1.3	1.77	0.65	3.61
World	n/a	n/a	n/a	2.1	n/a	2.7

Table 11.1: Summary of power-law exponents for the Web of various countries and regions.

We argue here that it is possible to model the document characteristics of the whole Web in a similar fashion as discussed in section 6.5. Fundamentally, the Heaps' and Zipf's laws are also valid at the scale of the Web, with a faster growing vocabulary (larger β) and a more biased word distribution (larger α).

In addition, the distribution of document sizes can be also be described by a mathematical model that states that the document sizes are self-similar [458], that is, they are invariant to scale changes (a similar behavior appears in Web traffic). This best model is based in the mix of two different distributions. The main body of the distribution follows a Logarithmic Normal distribution, such that the probability of finding a document of size x bytes is given by

$$p(x) = \frac{1}{x\sigma\sqrt{2\pi}} e^{-(\ln x - \mu)^2/2\sigma^2} \tag{11.2}$$

where the average (μ) and standard deviation (σ) are 9.357 and 1.318, respectively [146]. Figure 11.3 (left) shows the distribution of files sizes in a sample collection, where all logarithms are in base 10. Notice that the tail can be seen to the right of the value 4 in the horizontal axis as a line with negative slope.

The right tail of the distribution is "heavy-tailed". That is, the majority of documents are small, but there is a non trivial number of large documents. This is intuitive

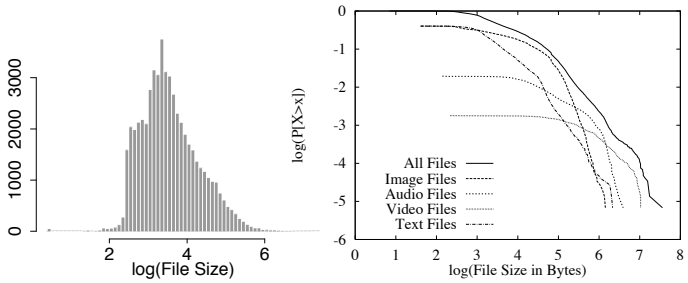

Figure 11.3: Left: Size distribution for all file sizes in a semi-log graph (courtesy of M. Crovella, 1998). Right: Tail of the probability distribution for different file size types in a log-log graph [458].

for image or video files, but it is also true for HTML pages. A good fit is obtained with the Pareto distribution (another power law function), namely,

$$p(x) = \frac{\alpha k^{\alpha}}{x^{1+\alpha}} \tag{11.3}$$

where x is measured in bytes and k and α are parameters of the distribution [146] (see Figure 11.3 (right)). The heavy tail appears in the fact that the curve decreases to the right as a line with a negative slope. For text files, the value of α is around 1.36, with smaller values for images and other binary formats [458, 1699]. Taking all Web documents into account, we get $\alpha = 1.1$ and $k = 9.3$KB. The cutting point between the Log Normal and the Pareto distribution is then larger than 9.3KB (that is, after 3.98 in the horizontal scale, such that the whole probability adds to 1) and 93% of all files have a size smaller than this cutting point. In fact, for file sizes smaller than 50KB, images are the typical files, from there to 300KB we have an increasing number of audio files, and over that to several megabytes, video files are more frequent. The parameters of these distributions were obtained from a sample of more than 46,000 Web pages requested by several users in a period of two months. Related information can be found on Web benchmarks such as WebSpec96 and the Sun/Inktomi Inkbench [811].

11.3.4 Link Analysis

On the Web, following [92], we distinguish three levels of link analysis:

- the *microscopic level*, which is related to the statistical properties of links and individual nodes.
- the *mesoscopic level* of link analysis, which is related to the properties of areas or regions of the Web.
- the *macroscopic level* of link analysis, which is related to the structure of the Web at large.

The *macroscopic level* of description of the Web started with the seminal "bow-tie" work by Broder*et al.* [271] already discussed in section 11.3.2. A related macroscopic description is the *Jellyfish structure* proposed in [1565], which can be applied to autonomous systems in the Internet. According to this view, we can identify a core portion surrounded by areas of decreasing link density, with many nodes forming long and loosely-connected chains or *tentacles*.

Mesoscopic link analysis is related to the properties of the neighborhood of a node, which is the context in which most of the link-based ranking functions work. A way of describing the neighborhood of a node is known as the "hop-plot": a plot of the number of different neighbors at different distances, such as the one depicted in Figure 11.4. The mesoscopic level is also the level of description at which local structures, such as communities or clusters of nodes, can be observed.

The *microscopic level* of description of the Web, which has been discussed by several authors [792, 139], is based on the observation that the distribution of the number of Web page links is very skewed, not showing the typical Poisson distribution observed in classic random graphs [536]. In scale-free networks, such as the Web, the distribution of the number of links of a page p follows a power-law:

$$Pr(\text{page } p \text{ has } k \text{ links}) \propto k^{-\alpha} \tag{11.4}$$

where usually $2 < \alpha < 3$. This implies a finite mean value (see section 11.3.3).

Scale-free networks have a few highly-connected links that act as "hubs" connecting many other nodes to the network. The connectivity of scale-free networks is resistant to random removal of edges [324] and can be explained in part by a "preferential attachment" process [140], also called a *rich-get-richer* phenomenon or Yule process. In this process, the probability of a new Web page v linking another Web page w is proportional to the number of incoming links to w.

Figure 11.5 shows a visual summary of the levels of link-based analysis we have described.

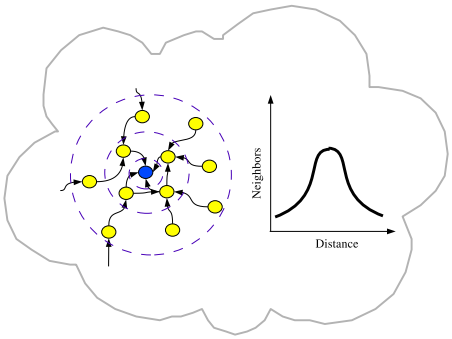

Figure 11.4: Schematic depiction of the "hop-plot": a plot of the number of neighbors at different distances [92].

Link analysis is an extremely rich source of information and can be used not only to infer relevance, as we discuss in section 11.5.2, but also to prioritize crawling (see section 12.5.1) or even to identify sub-structures such as communities on the Web

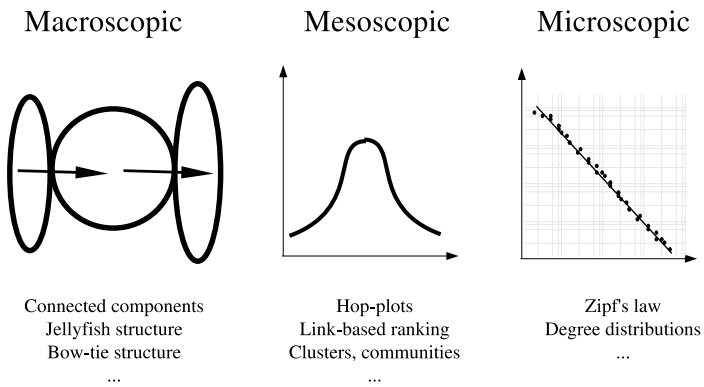

Figure 11.5: Levels of link-based analysis [92].

graph [625, 947]. In section 11.5.2 we use link analysis to define different ranking measures using links.

11.4 Search Engine Architectures

In this section, we cover different architectures of retrieval systems that model the Web as a full-text data repository. One main difference between standard IR systems and the Web is that, on the Web, all query processing and ranking must be done without accessing the source of the documents. This avoids accessing remote pages through the network at query time, which would be too slow. This fundamental difference has a direct impact on the indexing and searching algorithms, as well as on the complexity of query languages. Notice that, for the purpose of snippet generation, the source of the documents is used, but this is restricted to the top 10 results that are displayed to the users.

11.4.1 Basic Architecture

Most search engines use a *centralized crawler-indexer architecture*, as discussed in Chapter 12. Crawlers are programs (software agents) that traverse the Web sending new or updated pages to a main server where they are indexed. Crawlers are also called robots, spiders, wanderers, walkers, and knowbots. In spite of its name, a crawler is not actually sent to and executed on remote machines. Instead, it runs on a local system and sends requests to remote Web servers.

The index is used in centralized fashion to answer queries submitted from different places on the Web. Most search engines use variants of the inverted index (see section 9.2). In simplistic terms, an inverted index consists of a list of terms (vocabulary), where each term is associated with a list of pointers to the pages in which it occurs. It is important to remember that only a logical, rather than a literal, view of the text is indexed. Indeed, normalization operations are routinely conducted and may include removal of punctuation, substitution of multiple spaces between a pair of words by a

single space, and conversion of uppercase to lowercase letters (see Chapter 6). Some search engines eliminate stopwords to reduce the size of the index. However as they must handle hundreds of languages, stopwords are chosen statistically.

To make results richer and give users some insight on each answer in the result page, the index is complemented with metadata associated with each Web page such as its creation date, size, title, etc. Assuming that 500 bytes are required to store the URL and the metadata of each Web page, a total of 500GB is required to store descriptive information for 1 billion pages. This information can be compressed efficiently, so the actual size is significantly reduced in practice.

Given a query, the set of answers displayed is a subset of the complete result set (usually 10 results). If the user requests more results, the search engine can either recompute the query to generate the next 10 results or obtain them from a partial result set maintained in main memory. In any case, the search engine never computes the full answer for the whole Web collection because finding a few thousands of the most relevant results is generally enough. In fact, computing the whole answer set would be quite slow as some queries have a very large number of results.

State of the art indexing techniques can reduce the size of an inverted index to about 30% of its original size (and even less if stopwords are removed), as discussed in section 9.2.6 (see also [1703]). To illustrate, consider a Web collection of one billion pages requiring 1.5TB of disk space for proper storage. The uncompressed index requires roughly 60% of the overall space, that is 900GB. The compressed index, on the other hand, requires roughly less than half of that, say 400GB. As discussed in section 9.2.3, the index can be used to answer queries composed of multiple words by combining the list of documents for individual words to generate the final answer. Many search engines also support exact phrase and/or proximity search, which requires either additional information on the position of the terms in the documents or indexing frequent phrases as single indexing units (or both).

The search step can be conducted efficiently if each word is not too frequent. However, that is seldom the case on the Web. For this reason, as the number of results is potentially very large, all search engines use a lazy evaluation query processing scheme. That is, only the first results are computed and further results are computed on demand, when the user requests them after inspecting the first result page. More details on searching over an inverted index can be found in section 9.2.3.

Figure 11.6 (page 462) shows the schematic software architecture of an early search engine such as AltaVista [426]. It consists of two parts: one that handles the users' requests, composed of the user interface and the query engine component, and another that deals with data, composed of the crawler and indexer components. In 1998, the overall system ran on twenty multi-processor machines, all of them having a total of more than 130GB of RAM and over 500GB of disk. The query engine itself used more than 75% of these resources.

The main problem faced by this architecture, in addition to the gathering of the data, is its sheer volume. In fact, the crawler-indexer architecture was not able to cope with Web growth at the end of the 1990s. The solution was to distribute and parallelize computation, as explained next.

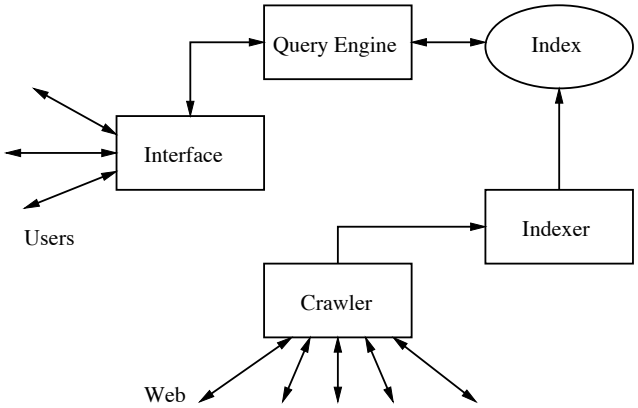

Figure 11.6: Typical early crawler-indexer architecture.

11.4.2 Cluster-based Architecture

Current search engines use a massive parallel and cluster-based architecture [151, 484, 99] (see Chapter 10). Due to the large size of the document collection, the inverted index does not fit in a single computer and must be distributed across the computers of a cluster. For this, as explained in section 10.3, document partitioning is used. The large volume of queries implies that the basic architecture must be replicated in order to handle the overall query load, and that each cluster must handle a subset of the query load. In addition, as queries originate from all around the world and Internet latency is appreciable across continents, cluster replicas are maintained in different geographical locations to decrease answer time. This allows search engines to be fault-tolerant in most typical worst-case scenarios, such as power outages or natural disasters. There are many crucial details to be carefully addressed in this type of architecture:

1. It is particularly important to achieve a good balance between the internal (answering queries and indexing) and external (crawling) activities of the search engine. This is achieved by assigning dedicated clusters to crawling, to document serving, to indexing, to user interaction, to query processing, and even to the generation of the result pages.

2. In addition, a good load balancing among the different clusters needs to be maintained. This is achieved by specialized servers called (quite trivially) *load balancers*.

3. Finally, since hardware breaks often, fault tolerance is handled at the software level. Queries are routed to the most adequate available cluster and CPUs and disks are routinely replaced upon failure, using inexpensive exchangeable hardware components.

Figure 11.7 shows a generic search cluster architecture with its key components. The front-end servers receive queries and process them right away if the answer is

already in the "answer cache" servers (see section 11.4.3). Otherwise they route the query to the search clusters through a hierarchical broker network. The exact topology of this network can vary but basically, it should be designed to balance traffic so as to reach the search clusters as fast as possible. Each search cluster includes a load balancing server (LB in the figure) that routes the query to all the servers in one replica of the search cluster. In this figure, we show an index partitioned into n clusters with m replicas. Although partitioning the index into a single cluster is conceivable, it is not recommended as the cluster would turn out to be very large and consequently suffer from additional management and fault tolerance problems. Each search cluster also includes an index cache, which is depicted at the top, as a flat rectangle. The broker network merges the results coming from the search clusters and sends the merged results to the appropriate front-end server that will use the right document servers to generate the full results pages, including snippet and other search result page artifacts. This is an example of a more general trend to consider a whole data center as a computer [152]. Orlando *et al.* [1235] present the architecture of a parallel and distributed search engine based on two main parallelization strategies: a task parallel strategy (a query is executed independently by a set of homogeneous index servers) and a data parallel strategy (a query is processed in parallel by index servers accessing distinct partitions of the database). In practice however, very few detailed descriptions of this architecture have been published, as leading search engines treat the details (and mostly the exact figures of resources required to process the queries, such as CPUs and disks) as a business secret.

Chowdhury and Pass [381] introduce a queuing theory model of an architecture for document partitioning and use it to analyze inherent operational requirements: throughput, response time, and utilization. Their queuing model assumes a perfect balance among the execution times of index servers that process an equal number of documents per query. However, Badue *et al.* [85] found that even with a balanced distribution of the document collection among the index servers, correlation between the frequency of a term in the query log and the size of its corresponding inverted list leads to imbalances in query execution times on these very servers, because correlation affects the disk caching behavior. Furthermore, the relative sizes of the main memory at each server (with respect to disk space usage) and the number of servers participating in the parallel query processing tasks also cause an imbalance of local query execution times. This was later addressed by the same authors in [84] via a capacity planning model dedicated to this specific architecture. Nevertheless, the availability in the literature of performance models for Web search systems is rather limited.

The scale and complexity of Web search engines, as well as the volume of queries submitted every day by the users, make query logs a critical source of information to enhance the precision of results and the efficiency of different parts of the search engines. Features such as query (and terms) distribution, arrival time of each query, and clicked results are a few examples of information extracted from query logs. The important question to consider now is whether we can exploit, or transform, this information in order to partition the document collection and route queries more efficiently and effectively. In the past few years, using query logs to cache queries, partitioning the document collection and performing query routing have been the focus of some research [1304, 1474]. Next we explore some techniques that profit

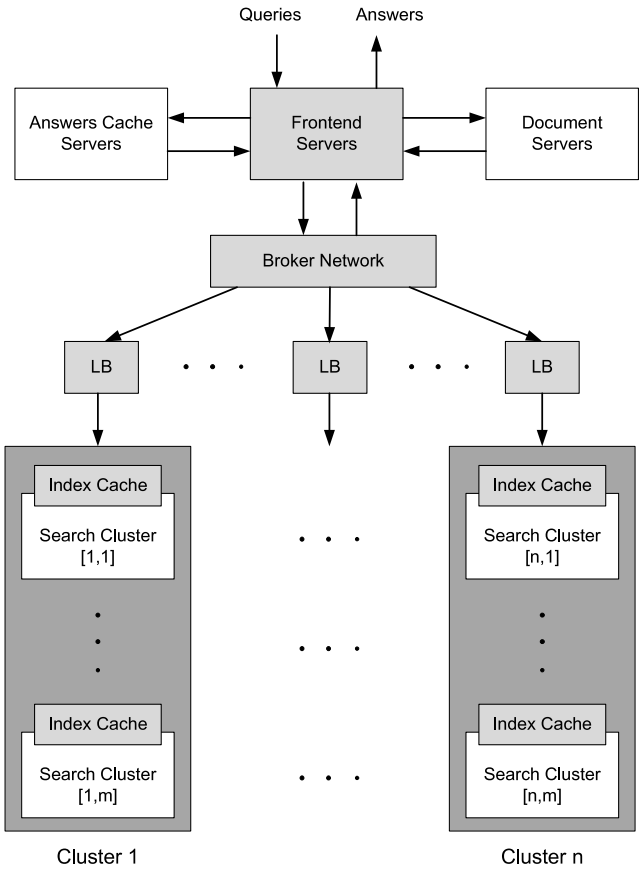

Figure 11.7: Cluster-based architecture for the search module based on [484]. Each cluster contains an index of the whole collection, *i.e.*, the index is partitioned among the m servers in the cluster. The n clusters are used to produce n replicas of the whole index.

from understanding the query load and the query distribution of the search engine.

11.4.3 Caching

As search engines need to be fast, whenever possible, most tasks should be conducted in main memory. As a consequence, caching is highly recommended and extensively used. Caching is a useful technique for Web systems that are accessed by a large number of users. It enables a shorter average response time, significantly reduces workload on back-end servers, and decreases the overall amount of bandwidth utilized. In a Web system, both clients and servers can cache items. While browsers and proxies cache Web objects on the client side, servers cache precomputed results or partial data used in the computation of new results [1284]. We focus our discussion on caching at the search engine side, as this is more efficient and is fully under control. This type of caching will be driven by the shared queries of the users, as noted initially in [1727].

The most effective caching technique in search engines is caching of search results or *caching of answers*, which is performed at the front-end level of Figure 11.7 and which allows very fast response for frequent queries. As query distribution follows a power law, a small number of queries are largely repeated and hence, a small cache can answer a large percentage of queries. For example, if we obtain a 30% hit-rate, the capacity of the search engine increases by almost 43%. On the other hand, in any time window a large fraction of the queries will be unique and hence will not be in the cache (for example, 50% in [104]). That is, there is a limit on the hit-rate of the answer cache. This can be improved by *caching inverted lists* of the index at the search cluster level. This cache is much more effective and can have hit rates over 90% [104] as overlap the terms, even of unique queries, are repeated. However, as caching of results is much faster, the fraction of the cache that we have to devote to each kind of caching is not trivial to compute and is system dependent.

Early work [1329] proposed to use a query log, built upon a set of persistent "optimal" past queries, to improve the retrieval effectiveness for similar future queries. Later, Markatos [1091] proved the existence of temporal locality in queries, compared the performance of different variants of the LRU policy using hit rate as a metric, and showed that static caching was useful for small caches. Cao [328] proposed caching policies that take into account parameters other than locality, such as the size of the object to be cached and the time needed to fetch objects from disk. This proposal organizes the index as follows: an index of precomputed results (from past user queries) and the inverted lists of the most frequent query-terms are kept in main memory, the remaining part of the index is kept in the secondary storage.

As systems are often hierarchical, there are proposals for multiple level caching architectures. Saraiva *et al.* [428] proposed a new architecture for Web search engines using a two-level dynamic caching system. The main goal of that work was to improve response time in hierarchical engines. In their architecture, both levels use an LRU eviction policy. They discovered that the second-level cache can effectively reduce disk traffic, thus increasing the overall throughput. Baeza-Yates and Saint Jean [123] proposed a three-level index organization with a frequency based inverted list static cache.

Based on Markatos' observations, Lempel and Moran [1002] proposed a new caching policy, called Probabilistic Driven Caching (PDC), which attempts to es-

timate the probability distribution of all follow-up queries (from the second result page on) submitted to a search engine. PDC is the first policy to adopt prefetching in anticipation of a user request. To this end, PDC exploits a model of user's behavior, where a user session starts with a query for the first page of results, and can proceed with one or more follow-up queries (*i.e.*, queries requesting successive page of results). When no follow-up queries are received within τ seconds, the session is considered finished.

Fagni *et al.* [544] showed that combining static and dynamic caching policies together with an adaptive prefetching policy achieves a high hit rate (SDC heuristic). In their experiments, they observed that devoting a large fraction of entries to static caching along with prefetching obtains the best hit rate. On the other hand, Zhang *et al.* [1773] studied caching of blocks of compressed inverted lists using several dynamic caching algorithms, and found that evicting from memory the least frequently used blocks of inverted lists performs very well in terms of hit rate.

Baeza-Yates *et al.* [104, 105] conducted the first comparative study of the impact of static and dynamic caching, that focused in particular on inverted list caching and memory allocation for results. They found that optimal results were achieved when dedicating around 30% to caching results and the rest to inverted lists. They also proposed a new algorithm for static caching of inverted lists that is based on a well-known heuristic for the Knapsack problem, which uses the ratio of the query frequency to the inverted list length in order to decide what to cache. The results of this algorithm as compared to LRU, LFU and previous solutions [123] are shown in Figure 11.8. They also showed that the changes on the query distribution are small and have only a little effect on the static solution that can be recomputed periodically, say on a daily basis. A similar result for dynamic caching was proposed in [1047].

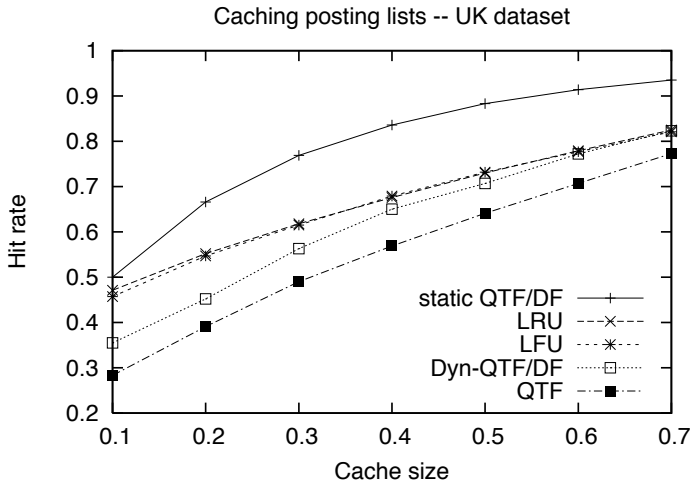

Figure 11.8: Performance of different algorithms for inverted list caching [105].

One problem of dynamic caching of results is a pollution effect produced by unique queries that never appear again. To solve this problem, Baeza-Yates *et al.* [112]

proposed an admission control mechanism that, based on simple predictions, decides whether the answer set is cached or not. In this solution, the cache memory is split in two parts. The first part is used to cache results of queries that are likely to be repeated in the future, and the second part is used to cache all other queries, so as to handle bursty queries and prediction errors. The decision to cache the query results in the first or the second part depends on features of the query, such as its past frequency or its length (in tokens or characters). Their results improve upon the SDC algorithm and shows that with very simple features we can reduce the cache pollution.

11.4.4 Multiple Indexes

Hierarchical indexes represent another type of improvement. As an example, consider a two-level or two-tier index. The first level is a small and fast index dedicated to the most frequent queries, while the second level is a larger and slower index for the rest of the queries. How to partition the collection in multiple tiers is explored in [1359, 1360, 123]. Risvik proposed to use a multi-tier system for scalability purposes [1359, 1360]. In this system, tiers act as filters to allow a query to "fall through" to the next tier, based on the number of hits in a given tier, as well as on the relevance score of the query results from that tier.

One disadvantage of the previous technique is that some queries will have slower answers, in particular if tiers are searched sequentially, one after the other. A solution is to submit the query in parallel to all tiers. However this increases the query load and hence the overall hardware infrastructure cost. A better solution is to predict which queries need to go to the next tier. For this, Baeza-Yates *et al.* [114], proposed a machine learning based predictor that decides whether the search should be conducted in parallel or not, based solely on query features. They showed that any predictor better than random yields answer time savings, but might have negative effects on infrastructure costs. Therefore, they provided an analysis of the performance-cost trade-off, showing that whenever the decrease in answer time is justified, it implies a small increase in infrastructure cost.

Liu *et al.* [1043] conducted another study of multiple indexes, where they experimentally showed, on an extract of the Chinese Web, that it is possible to reduce the corpus size by 95% to produce a "cleansed corpus" and still retain retrieval performance. More specifically, more than 90% of queries could be answered with pages from the cleansed corpus. They exploited query-independent features such as PageRank (see section 11.5.2) and the number of in-links to classify each page into a potential retrieval target page (cleansed corpus) or an ordinary page (removed).

Another hierarchical technique, proposed by Ntoulas *et al.* in [1214], consists of using a pruned index after the cache of results to provide fast answers to simple queries. They introduced a two-tier architecture, where the first tier is a small pruned index and the second tier is the full index of the search engine. A query is first answered with the small pruned index. If there is no loss in effectiveness compared to the results that would have been obtained from the full index, then the results are returned to the user, otherwise, the query is evaluated on the full index. In a first stage, this approach seemed to be beneficial, however further studies showed that when the effect of the cache was also considered, the value of the approach was less

obvious, as a pruned index has basically the same effect of inverted list caching, while not being as adaptive [1487]. In fact, these results showed how important it is to study the overall interaction of components, rather than improvements in isolation. Caching of results, for example, increases the length of the queries that actually hit the search engine by more than 30% and reduces one-term queries significantly.

Regarding how to partition a Web collection in tiers, D'Souza *et al.* [514] compared three methods of constructing partitioned corpora, using the TREC volumes one and two with data from the Associated Press [703]. They tried a random partitioning, a partitioning based on the chronological ordering of documents, and a partitioning based on the document authorship to simulate a managed environment, where the collection is partitioned based on some common characteristic. Similarly, Craswell *et al.* [437] used the wt2g collection, which is a subset of the wt10g collection. The collection is crawled from 956 document servers, and their data is more realistic than other simulations on TREC data.

Yom-Tov *et al.* [1752] tested various corpus partitioning schemes of the Web `.gov` collection within the TREC Terabyte setting. Each query is submitted to all partitions and the final result list is merged by weighing each individual result list based on a query prediction algorithm that estimates how well the partition should answer the query. The examined partition schemes are based on document clusters, domains, number of in-links and document title strings. The latter performed best overall, however random partitioning ranked second. They uses the query prediction algorithm for merging results rather than as a preprocessing step for tier selection. An additional flaw is that they do no decided in advance which corpora to search, and as a consequence every tier is always searched in parallel.

11.4.5 Distributed Architectures

There exist several variants of the crawler-indexer architecture and we describe here the most important ones. Among them, the most significant early example is Harvest [244]. Among the newest proposals, we distinguish the multi-site architecture proposed by Baeza-Yates *et al.* [106].

Harvest

Harvest uses a distributed architecture to gather and distribute data, which is more efficient than the standard Web crawler architecture (see Chapter 12). The main drawback is that Harvest requires the coordination of several Web servers. Interestingly, the Harvest distributed approach does not suffer from some of the common problems of the crawler-indexer architecture, such as:

- increased servers load caused by the reception of simultaneous requests from different crawlers,
- increased Web traffic, due to crawlers retrieving entire objects, while most content is not retained eventually, and
- lack of coordination between engines, as information is gathered independently by each crawler.

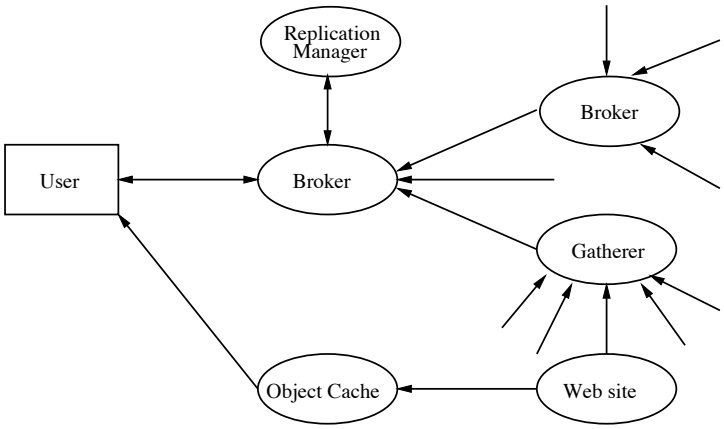

Figure 11.9: Harvest architecture.

Avoiding these issues was achieved by introducing two main components in the architecture: gatherers and brokers. A gatherer collects and extracts indexing information from one or more Web servers. Gathering times are defined by the system and are periodic (*i.e.*, there are harvesting times as the name of the system suggests). A broker provides the indexing mechanism and the query interface to the data gathered. Brokers retrieve information from one or more gatherers or other brokers, updating incrementally their indexes. Depending on the configuration of gatherers and brokers, different improvements on server load and network traffic can be achieved. For example, a gatherer can run on a Web server, generating no external traffic for that server. Also, a gatherer can send information to several brokers, avoiding work repetition. Brokers can also filter information and send it to other brokers. This design allows the sharing of work and information in a very flexible and generic manner. An example of the Harvest architecture is shown in Figure 11.9.

One of the goals of Harvest is to build topic-specific brokers, focusing the index contents and avoiding many of the vocabulary and scaling problems of generic indexes. Harvest includes a dedicated broker that allows other brokers to register information about gatherers and brokers. This is mostly useful for identifying an appropriate broker or gatherer when building a new system. The Harvest architecture also provides replicators and object caches. A replicator can be used to replicate servers, enhancing user-base scalability. For example, the registration broker can be replicated in different geographic regions to allow faster access. Replication can also be used to divide the gathering process among many Web servers. Finally, the object cache reduces network and server load, as well as response latency when accessing Web pages. More details can be found in [244].

Pointing to pages or to word positions is an indication of the granularity of the index. The index can be less dense if it points to logical blocks rather than pages. This approach allows to reduce the variance between the different document sizes, by making all blocks of roughly the same size. This not only reduces the size of the pointers (because there are less blocks than documents) but also reduces the

number of pointers due to the locality of reference between words. Indeed, all the occurrences of a non-frequent word will tend to be clustered in the same block. This idea was used in Glimpse [1078], which is at the core of Harvest [244]. Queries are resolved like in inverted indexes, obtaining a list of blocks that are then searched sequentially. Note that exact sequential search can be done over 30MB per second in RAM. Glimpse originally used only 256 blocks, which was efficient up to 200MB for searching words that were not too frequent, obtaining an index of only 2% of the text. By tuning the number of blocks and the block size, reasonable space-time trade-offs can be achieved for larger document collections (for more details see Chapter 9). These ideas cannot be used (yet) for the Web because sequential search cannot be afforded, as it implies network access. However, in a distributed architecture where the index is also distributed, logical blocks make sense.

Multi-site Architecture

As the document collection grows, the capacity of query processors has to grow as well to match the demand for high query throughput, and low latency. It is unlikely, however, that the growth in the size of single processors can be used to match the growth of very large collections such as the Web, even if a large number of servers is used. The main reason are physical and administrative constraints such as the size of a single data center and its power and cooling requirements [151]. Thus, the distributed resolution of queries using different query processors is a viable approach, as it enables a more scalable solution. However, it also imposes new challenges. One such challenge is the routing of queries to appropriate query processors, in order to utilize more efficiently the available resources and provide more precise results. Factors affecting the query routing include, for instance, geographical proximity, query topic, or language of the query. Geographical proximity aims at reducing the network latency by utilizing resources that are close to the user posing the query. A possible implementation of such a feature is DNS redirection: according to the IP address of the client, the DNS service routes the query to the appropriate Web server, usually the closest in terms of network distance [1539]. As another example, the DNS service can use the geographical location to determine where to route queries to. As there is fluctuation in submitted queries from a particular geographic region during a day [168], it is also possible to offload a server from a busy region by rerouting some queries to query servers in a less busy region.

Considering the discussion above, Baeza-Yates *et al.* [106] recently proposed a cost model for search engines as well as a simple distributed architecture that has comparable cost to a centralized search architecture. The architecture is based on several sites that are logically connected through a star topology network, such that the central site is the one with the highest load of local queries. The main idea is to answer local queries locally and forward to the other sites only queries that need external pages in their answers. To increase the percentage of local queries, the authors proposed to use caching of results and to replicate a small set of popular documents in all sites. These two techniques allow to increase the number of local results from 5% to 30% or more.

In a complementary paper, Cambazoglu *et al.* [325] showed that the resources saved by answering queries locally can be used to execute a more complex ranking

function, which can improve the results.

11.5 Search Engine Ranking

Ranking is the hardest and most important function search engines have to execute. A first challenge is to devise an adequate evaluation process that allows judging the efficacy of a ranking, in terms of its relevance to the users. Without such evaluation process it is close to impossible to fine tune the ranking function, which basically prevents achieving high quality results. There are many possible evaluation techniques and measures, as already detailed in Chapter 4. We cover this topic in the context of the Web in section 11.5.6, paying particular attention to the exploitation of user's clicks.

A second critical challenge is the identification of quality content in the Web. Evidence of quality can be indicated by several signals such as domain names (*i.e.*, .edu is a positive signal as content originating from academic institutions is more likely to be reviewed), text content and various counts (such as the number of word occurrences), links (like PageRank), and Web page access patterns as monitored by the search engine. Indeed, as mentioned before, clicks are a key element of quality. The more traffic a search engine has, the more signals it will have available. Additional useful signals are provided by the layout of the Web page, its title, metadata, font sizes, as discussed later.

The economic incentives of the current advertising based business model adopted by search engines have created a third challenge, avoiding Web spam. Spammers in the context of the Web are malicious users who try to trick search engines by artificially inflating the signals mentioned in the previous paragraph. This can be done, for instance, by repeating a term in a page a great number of times, using link farms, hiding terms from the users but keeping them visible to the search engines through weird coloring tricks, or even for the most sophisticated ones, deceiving Javascript code. More details are given later in section 11.5.7.

Finally, the fourth issue lies in defining the ranking function and computing it (which is different from evaluating its quality as mentioned above). While it is fairly difficult to compare different search engines as they evolve and operate on different Web corpora, leading search engines have to constantly measure and compare themselves, each one using its own measure, so as to remain competitive.

In the following subsections, we discuss classic signals that are typically used by search engines as an indication of relevance in their ranking functions or used as features in machine learned ranking functions. We then discuss link-based ranking functions, an area that has been "invented" for the Web as compared to classic IR. Then, we discuss three different types of ranking techniques from the simplest to the most complex. Finally, we cover quality evaluation and Web spam.

11.5.1 Ranking Signals

We distinguish among different types of signals used for ranking improvements according to their origin, namely content, structure, or usage, as follows.

Content signals are related to the text itself, to the distributions of words in the documents as has been traditionally studied in IR. The signal in this case can vary

from simple word counts to a full IR score such as BM25 (see section 3.5.1). They can also be provided by the layout, that is, the HTML source, ranging from simple format indicators (more weight given to titles/headings) to sophisticated ones such as the proximity of certain tags in the page.

Structural signals are intrinsic to the linked structure of the Web. Some of them are textual in nature, such as anchor text, which describe in very brief form the content of the target Web page. In fact, anchor text is usually used as surrogate text of the linked Web page. That implies that Web pages can be found by searching the anchor texts associated with links that point to them, even if they have not been crawled. Other signals pertain to the links themselves, such as the number of in-links to or out-links from a page. We should note that link-based signals find broad usage beyond classic search engine ranking. One example is the application of link-based signals to answer name queries in a social network, as discussed in [1638]. Since these link-based signals are typical of the Web and were not used used in traditional IR systems, the next subsection is fully devoted to them.

The next set of signals comes from *Web usage*. The main one is the implicit feedback of the user through clicks. In our case the main use of clicks are the ones in the URLs of the results set. This important source of relevance is covered in sections 4.5.5 and 5.4. Additional signals include information about the user's geographical context (IP address, language, etc.), technological context (operating system, browser, etc.), and temporal context (query history by the use of cookies). These features allow users to better localize results and customize them to a region or language, and even to partially personalize them. Notice, however, that usage follows a power law. That is, there are a few heavy users and many more light users. As a consequence, while personalization only works well for some users, personalizing the intention across many users is simpler and more effective as described in section 7.2.3.

11.5.2 Link-based Ranking

Given that there might be thousands or even millions of pages available for any given query, the problem of *ranking* those pages to generate a short list is probably one of the key problems of Web IR; one that requires some kind of relevance estimation. In this context, the number of hyperlinks that point to a page provides a measure of its popularity and quality. Further, many links in common among pages and pages referenced by a same page are often indicative of page relations with potential value for ranking purposes. Next, we present several examples of ranking techniques that exploit links, but differ on whether they are query dependent or not.

Early Algorithms

One signal of importance for a scientific paper is the number of citations that the article receives, a topic that researchers in library sciences have studied for long [880]. Based on this idea, several authors proposed to use incoming links for ranking Web pages [1086, 855, 1021]. However, it quickly became clear that just counting links was not a very reliable measure of authoritativeness (it was not in scientific citations either), because it is very easy to externally influence this count by creating new pages (which costs nearly nothing).

Yuwono and Lee [1760] proposed three ranking algorithms in addition to the classic TF-IDF scheme (see Chapter 3), namely: Boolean spread, vector spread, and most-cited. The first two are the normal ranking algorithms of the Boolean and vector space models extended to include pages pointed by a page in the answer or pages that point to a page in the answer. The third algorithm, "most-cited", is based only on the terms included in pages having a link to the pages in the answer. Their comparative study of these techniques considering 56 queries over a collection of 2,400 Web pages indicated that the vector spread model yields a better recall-precision curve, with an average precision of 75%.

Another early example is WebQuery [338], which also allows visual browsing of Web pages. WebQuery takes a set of Web pages (for example, a list of search results) and ranks them based on how connected each Web page is. Additionally, it extends the set by finding Web pages that are highly connected to the original set. A related approach is presented by Li [1021].

HITS

A better idea is due to Kleinberg [911] and used in HITS (Hypertext Induced Topic Search). This ranking scheme is query-dependent and considers the set of pages S that point to or are pointed by pages in the answer. Pages that have many links pointing to it in S are called *authorities* because they are susceptible to contain authoritative and thus, relevant content. Pages that have many outgoing links are called *hubs* and are susceptible to point to relevant similar content. A positive two-way feedback exists: better authority pages come from incoming edges from good hubs and better hub pages come from outgoing edges to good authorities. Let $H(b)$ and $A(p)$ be the hub and authority values of page p. These values are defined such that the following equations are satisfied for all pages p:

$$H(p) = \sum_{u \in S \mid p \to u} A(u) \ , \qquad A(p) = \sum_{v \in S \mid v \to p} H(v) \tag{11.5}$$

where $H(p)$ and $A(p)$ for all pages are normalized (in the original paper, the sum of the squares of each measure is set to one). These values can be determined through an iterative algorithm, and they converge to the principal eigenvector of the link matrix of S. In the case of the Web, to avoid an explosion on the size of S, a maximal number of pages pointing to the answer can be defined. This technique does not work with non-existent, repeated, or automatically generated links. One solution is to weigh each link based on the surrounding content. A second problem is that of topic diffusion, because as a consequence of link weights, the result set might include pages that are not directly related to the query (even if they have got high hub and authority values). A typical case of this phenomenon is when a particular query is expanded to a more general topic that properly contains the original answer. One solution to this problem is to associate a score with the content of each page, like in traditional IR ranking, and combine this score with the link weight. The link weight and the page score can be included in the previous formula multiplying each term of the summation [352, 199, 351]. Experiments show that the recall and precision for the first ten results increase significantly [199]. The appearance order of the links on the Web page can also be used by dividing the links into subgroups and using

the HITS algorithm on those subgroups instead of the original Web pages [351]. In Table 11.2, we show the exponent of the power law of the distribution for authority and hub values for different countries of the globe adapted from [97].

Country	PageRank	HITS	
		Hubs	Auth.
Brazil	1.83	2.9	1.83
Chile	1.85	2.7	1.85
Greece	1.83	2.6	1.83
South Korea	1.83	3.7	1.83
Spain	1.96	n/a	n/a

Table 11.2: Summary of power-law exponents for link-based measures of various countries.

PageRank

The best known link-based weight is PageRank, which is part of the ranking algorithm originally used by Google [263]. PageRank simulates a user navigating randomly on the Web, as follows. Consider that our user is currently at page a. Following, she moves to one of the pages pointed by page a by randomly selecting one of the hyperlinks present in a. Next, she repeats the process for the page she moved to, and so on. After a large number of such moves, we can compute the probability with which our user visited each page. This probability is a property of the graph, which was referred to as PageRank in the context of the Web. A real Web graph contains dead ends, that is, pages without outgoing links, as well as self-links. To avoid that the user gets trapped in these pages, an additional case is considered, in which she can also jump to any other page in the graph, with a small probability q.

Hence, our user jumps to a random page of the Web graph with probability q or follows one of the hyperlinks in the current page with probability $1 - q$. By the definition of this random walk on the Web graph, this user never goes back to a page just visited following an already traversed hyperlink backwards. This process can be modeled by a Markov chain, from which the stationary probability of being in each page can be computed. Let $L(p)$ be the number of outgoing links of page p and suppose that page a is pointed by pages p_1 to p_n. Then, the PageRank of a page a is given by the probability $PR(a)$ of finding our user in that page and is defined by

$$PR(a) = \frac{q}{T} + (1 - q) \sum_{i=1}^{n} \frac{PR(p_i)}{L(p_i)} \tag{11.6}$$

where T is the total number of pages on the Web graph and q is a parameter that must be set by the system (a typical value is 0.15). Notice that the ranking (weight) of other pages is normalized by the number of links in the page. PageRank can be computed using an iterative algorithm and corresponds to the principal eigenvector of the normalized link matrix of the Web (which is the transition matrix of the Markov chain).

There are a few technical issues when computing PageRank. The main one is related to the handling of dead ends or "sink nodes" in the Markov chain, *i.e.*, of

pages that do not have out-going links. One solution is to use $q = 1$ for these pages. A simpler solution is to remove them (which also reduces the size of the link matrix) and only compute their PageRanks at the end, using the PageRank of their parents.

In [91, 92], a family of link-based ranking algorithms that propagate page importance through links is defined. These algorithms use a damping function that decreases with distance, so that a direct link implies more endorsement than a link through a long path. The authors study three damping functions, having linear, exponential, and hyperbolic decay on the lengths of the paths. The exponential decay corresponds to PageRank, and the other functions are new. Among other results, they show how to calculate a linear approximation that induces a page ordering that is almost identical to PageRank's using a fixed small number of iterations. In addition, they shed some light on why $q = 0.15$ is a good value. In Table 11.2, we show the exponent of the power law of the distribution of PageRank in different countries.

A final historical remark is in order. A first general scheme for relevance in hypertext defined a linear differential schema for relevance interpolation of the form $y = r + Dy$, in which r is an initial relevance estimation and D is the weighted connectivity matrix [286]. By making a proper choice for both r and D, we can generate PageRank from this formulation.

11.5.3 Simple Ranking Functions

The simplest ranking scheme consists of using a global ranking function such as PageRank. In that case, the quality of a Web page is independent of the query and the query only acts as a document filter. That is, for all Web pages that satisfy a query, rank them using their PageRank order.

A more elaborated ranking scheme consists of using a linear combination of different relevance signals. For example, combining textual features, say BM25 (see Chapter 3), and link features, such as PageRank. To illustrate, consider the pages p that satisfy query Q. Then, the rank score $R(p, Q)$ of page p with regard to query Q can be computed as:

$$R(p, Q) = \alpha \; BM25(p, Q) \; + \; (1 - \alpha) PR(p) \tag{11.7}$$

Further, $R(p, Q) = 0$ if p does not satisfies Q. If we assume that all the functions are normalized and $\alpha \in [0, 1]$, then $R(p, Q) \in [0, 1]$. Notice that this linear function is convex in α. Also, while the first term depends on the query, the second term does not. If $\alpha = 1$, we have a pure textual ranking, which was the typical case in the early search engines. If $\alpha = 0$, we have a pure link-based ranking that is also independent of the query. Thus, the order of the pages is known in advance for pages that do contain q. We can tune the value of α experimentally using labeled data as ground truth or clickthrough data. In fact, α might even be query dependent. For example, for navigational queries α could be made smaller than for informational queries (see section 7.2.1).

Early work on combining text-based and link-based rankings was published by Silva *et al.* [1478]. The authors used a Bayesian network to combine the different signals and showed that the combination leads to far better results than those produced by any of the combining ranking functions in isolation. Subsequent research by Calado *et al.* [311] discussed the efficacy of a global link-based ranking versus a

local link-based ranking for computing Web results. The local link-based ranking for a page p is computed considering only the pages that link to and are linked by page p. The authors compare results produced by a combination of a text-based ranking (Vector model) with global HITS, local HITS, and global PageRank, to conclude that a global link-based ranking produces better results at the top of the ranking, while a local link-based ranking produces better results later in the ranking.

11.5.4 Learning to Rank

A rather distinct approach for computing a Web ranking is to apply machine learning techniques for learning to rank. For this, one can use their favorite machine learning algorithm, fed with training data that contains ranking information, to "learn" a ranking of the results, analogously to the supervised algorithms for text classification (see Chapter 8). The loss function to minimize in this case is the number of mistakes done by the learned algorithm, which is similar to counting the number of misclassified instances in traditional classification (see Chapter 8). The evaluation of the learned ranking must be done with another data set (which also includes ranking information) distinct from the one used for training. There exist three types of ranking information for a query Q, that can be used for training:

- pointwise: a set of relevant pages for Q.
- pairwise: a set of pairs of relevant pages indicating the ranking relation between the two pages. That is, the pair $[p_1 \succ p_2]$, implies that the page p_1 is more relevant than p_2.
- listwise: a set of ordered relevant pages: $p_1 \succ p_2 \cdots \succ p_m$.

In any case, we can consider that any page included in the ranking information is more relevant than a page without information, or we can maintain those cases undefined. Also, the ranking information does not need to be consistent (for example, in the pairwise case).

The training data may come from the so-called "editorial judgements" made by people or, better, from clickthrough data. Given that users' clicks reflect preferences that agree in most cases with relevance judgements done by human assessors (see Chapter 5, section on explicit feedback from clicks), one can consider using click-through information to generate the training data. Then, we can learn the ranking function from click-based preferences. That is, if for query Q, p_1 has more clicks than p_2, then $[p_1 \succ p_2]$.

One approach for learning to rank from clicks using the pairwise approach is to use support vector machines (SVMs), (see section 8.2.1) to learn the ranking function as proposed in [1320]. In this case, preference relations are transformed into inequalities among weighted term vectors representing the ranked documents. These inequalities are then translated into an SVM optimization problem, whose solution computes optimal weights for the document terms. This approach proposes the combination of different retrieval functions with different weights into a single ranking function.

The pointwise approach solves the problem of ranking by means of regression or classification on single documents, while the pairwise approach transforms ranking

into a problem of classification on document pairs. The advantage of these two approaches is that they can make use of existing results in regression and classification. However, ranking has intrinsic characteristics that cannot be always solved by the latter techniques.

The listwise approach tackles the ranking problem directly, by adopting listwise loss functions, or directly optimizes IR evaluation measures such as average precision. However, this case is in general more complex. Some authors have proposed to use a multi-variant function, also called relational ranking function, to perform listwise ranking, instead of using a single-document based ranking function.

A second possibility for the loss function is to maximize average precision (see section 4.3.2) with different relaxed optimization functions as proposed by Joachims [841] and Yue *et al.* [1759]. More technical details can be found in the excellent survey by Chakrabarti [350], including other variations of this problem.

For research, Microsoft has released a dataset with ranking information called LETOR [1039], which has been widely used by researchers and serves as a comparison benchmark.

11.5.5 Learning the Ranking Function

A different scheme consists of learning the ranking function rather than the ranking order, as discussed above. This is equivalent to learning the best possible function for the basic ranking case. The idea is to use a genetic algorithm where the members of the population are function instances over a given set of ranking features. At every step of the genetic algorithm, different functions are mutated or mixed, evaluating the goodness of the function by using a set of ground truth or training data. At the end of many iterations the most fitted function is selected.

This approach presents the clear advantage of offering insight about the really important features, as well as as their impact on the final ranking, simply by looking at the function. It seems that this idea was discovered in parallel and independently by Trotmann [1595] for document ranking and Lacerda *et al.* [954] for advertisement ranking. Both approaches use a similar set of functions (standard operators and typical simple functions as logarithms or exponentials), but differ in their function combination methods and in the nature of the "goodness" function that is optimized.

As this technique is quite new, further research is needed to improve the quality of the results as well as as the efficiency of the technique.

11.5.6 Quality Evaluation

To be able to evaluate quality, Web search engines typically use human judgements that indicate which results are relevant for a given query, or some approximation of a "ground truth" inferred from user's clicks, or finally a combination of both, as follows.

Precision at 5, 10, 20

One simple approach to evaluate the quality of Web search results is to adapt the standard precision-recall metrics to the Web (see section 4.3.2). For this, the following observations are important:

- on the Web, it is almost impossible to measure recall, as the number of relevant pages for most typical queries is prohibitive and ultimately unknown. Thus, standard precision-recall figures cannot be applied directly.
- most Web users inspect only the top 10 results and it is relatively uncommon that a user inspects answers beyond the top 20 results. Thus, evaluating the quality of Web results beyond position 20 in the ranking is not indicated as does not reflect common user behavior.
- since Web queries tend to be short and vague, human evaluation of results should be based on distinct relevance assessments for each query-result pair. For instance, if three separate assessments are made for each query-result pair, we can consider that the result is indeed relevant to the query if at least two of the assessments suggest so.

The compounding effect of these observations is that (a) precision of Web results should be measured only at the top positions in the ranking, say P@5, P@10, and P@20 and (b) each query-result pair should be subjected to 3-5 independent relevant assessments.

Click-through Data as an Evaluation Metric

One major advantage of using clickthrough data to evaluate the quality of answers derives from its scalability. Its disadvantage is that it works less well in smaller corpora, such as countries with little Web presence, Intranet search, or simply in the long tail of queries (see section 7.2). Note that users' clicks are not used as a binary signal but in significantly more complex ways such as considering whether the user remained a long time on the page it clicked (a good signal) or jumped from one result to the other (a signal that nothing satisfying was found). These measures and their usage are complex and kept secret by leading search engines.

For that reason, the main evaluation metric in this case could be precision at the first 10 or 20 top results, based on clickthrough data collected in large scale. For further details on clickthrough data usage for evaluation, see section 4.5.5.

An important problem when using clicks is to take into account that the click rate is biased by the ranking of the answer (better the ranking, more clicks) and the user interface (for instance, the number of clicks will have a discontinuity when going from the last result of the first page to the first result of the second page). Hence, it is important to remove the bias from clicks to uncover their real value [111, 1321, 522]. More details on this problem are discussed in section 5.4. Additionally, capturing the click in its right context is also important [1277].

Evaluating the Quality of Snippets

A related problem is to measure the quality of the snippets in the results. Search snippets are the small text excerpts associated with each result generated by a search engine. They provide a summary of the search result and indicate how it is related to the query (by presenting query terms in boldface, for instance). This provides great value to the users who can quickly inspect the snippets to decide which results are of interest.

Given that search snippets play an important role in the display of search results, evaluating their quality is important. Indeed, increased research activity has been observed in this area lately. In [861] the authors study how variations in snippet length affect search results quality. In [873] the authors study how to predict the readability of search snippets. In [35] the authors proposed to associate time information with search snippets and evaluated how it improves results. In all these studies, the preferred evaluation technique is crowdsourcing, particularly with the utilization of Amazon Mechanical Turk (AMT) platform, discussed in section 4.5.4.

11.5.7 Web Spam

The Web contains numerous profit-seeking ventures, so there is an economic incentive from Web site owners to rank high in the result lists of search engines. All deceptive actions that try to increase the ranking of a page in search engines are generally referred to as *Web spam* or *spamdexing* (a portmanteau of "spamming" and "index"). The area of research that relates to spam fighting is called Adversarial Information Retrieval, which has been the object of several publications and workshops [23].

A Web search engine must consider that "any evaluation strategy that counts replicable features of Web pages is prone to manipulation" [1238]. In practice, such manipulation is widespread, and in many cases, successful. The authors of [531] report that "among the top 20 URLs in our 100 million page PageRank calculation ... 11 were pornographic, and these high positions appear to have all been achieved using the same form of link manipulation".

In [690], *spamming* is defined as "any deliberate action that is meant to trigger an unjustifiably favorable relevance or importance for some Web page, considering the page's true value". A *spam page* is a page that is used for direct spamming or whose score is artificially inflated because of other spam pages. Another definition of spam, given in [1256], is "any attempt to deceive a search engine's relevancy algorithm" or, in the extreme, "anything that would not be done if search engines did not exist".

Multiple *spamdexing* techniques exist and new ones continue to be invented in a continuous fight between spammers and search engine companies. A spam page may contain an abnormally high number of keywords, or have other text features that are typically fought against via *content-based spam detection* techniques [1216, 512]. Link spamming includes link farms that either create a complex linking structure among Web sites of a same owner or that collude to deceive the search engine. Click spam is done by special software robots that issue specific queries and click on pages preselected to be promoted. A third and more sophisticated approach is programmatic spamming, in which Web spammers inject pieces of code in a Web page, say in Javascript.[13] This piece of code, when executed on the client side, displays information to the user that is different from the one crawled by the search engine (this is a particular form of what is called cloaking).

Some often confuse Web spam with Search Engine Optimization (SEO). However, SEO techniques can be legitimate when applied by webmasters who follow the guidelines provided by most search engines, so as to make their pages discoverable. In contrast, malicious SEO is used by Web spammers who want to deceive users and

[13] Interesting examples of sneaky Javascript are given by Matt Cutts in http://www.mattcutts.com/blog/seo-mistakes-sneaky-javascript/.

search engines alike. A great source of additional information on this topic is Matt Cutts' postings on SEO in his blog.[14] Note that identifying Web spam is one of the best examples of an application of Web mining, a related topic addressed later in section 11.10.2

11.6 Managing Web Data

In this section we briefly explore several problems related to the Web data that search engines need to store and manage.

11.6.1 Assigning Identifiers to Documents

Document identifiers are usually assigned randomly or according to the ordering with which URLs are crawled. Numerical identifiers are used to represent URLs in several data structures. In addition to inverted lists, they are also used to number nodes in Web graphs and to identify documents in search engines repositories.

It has been shown in the literature that a careful ordering of documents leads to an assignment of identifiers from which both index and Web graph storing methods can benefit [1333, 222, 209, 1482, 206, 1480, 1739]. Also an assignment based on a global ranking scheme may simplify the ranking of answers (see section 11.5.3).

Regarding the compression of inverted lists, a very effective mapping can be obtained by considering the sorted list of URLs referencing the Web documents of the collection as proposed by Silvestri in [1480]. Assigning identifiers in ascending order of lexicographically sorted URLs improves the compression rate. The hypothesis that is empirically validated by Silvestri is that documents sharing correlated and discriminant terms are very likely to be hosted by the same site and will therefore also share a large prefix of their URLs. Experiments validate the hypothesis since compression rates can be improved up to 0.4 by using the URL sorting technique. Furthermore, sorting a few million URLs takes only tens of seconds and takes only a few hundreds megabytes of main memory.

11.6.2 Metadata

As we discussed in section 11.4.1, 20 billion URLs require at least 1TB of storage to hold all the metadata on the corresponding Web pages, and this using a compressed format. Managing this amount of information efficiently implies a very fast and space efficient database, which in turn implies in the availability of a very efficient file system.

Google's BigTable [356] is perhaps the best example of a type of database incarnation at the Web scale. BigTable is currently used to store data in a distributed system, which is usually generated and modified using the Map-Reduce paradigm [485] (see also section 10.5). BigTable is not a traditional database, being described by its authors as "a sparse, distributed multi-dimensional sorted map" and designed to scale up to petabytes across "hundreds or thousands of machines". BigTable has

[14] http://www.mattcutts.com/blog/type/googleseo/

the capability of adding machines to the system and using them without any reconfiguration.

As a database, BigTable shares characteristics of both row-oriented and column-oriented databases. Each table has multiple dimensions, the values are kept in a compressed form, and it is optimized to the underlying file system, which is the Google File System or GFS [623].

An open source database inspired by BigTable is HBase [725]. HBase is also a distributed database written in Java. HBase runs on top of the Hadoop Distributed File System, HDFS [726, 240], providing BigTable-like capabilities to Hadoop [691], the open source version of map-reduce. HBase is column oriented and features compression, in-memory operations, and Bloom filters.

Other available options include Hypertable [800] and Cassandra [341]. In particular, Cassandra runs in an Amazon Dynamo-like infrastructure and hence is eventually consistent. Dynamo [486] is an Amazon proprietary key-value storage system, which is highly available and combines properties of a database with those of a distributed hash table (see section 10.8).

11.6.3 Compressing the Web Graph

Web graphs may be represented by using adjacency lists. They are basically lists that contain, for each vertex v of the graph, the list of vertexes directly reachable from v. It has been observed that almost 80% of all links are local, that is, they point to pages of the same site. Starting from this observation, it is obvious that assigning closer identifiers to URLs referring to the same site, as discussed in section 11.6.1, will result in adjacency lists that will contain very close ids. Representing these lists using a d-gapped representation will thus lead to d-gapped adjacency lists having long runs of 1's.

Exploiting this and other particular redundancies of the Web graph make it possible to reach extremely high compression rates. The purpose of Web graph compression schemes is not only to provide empirical succinct data structures but also to allow fast access, as the graph will be needed for link analysis and other applications. To exemplify, the WebGraph framework [222], which compresses typical Web graphs at about 3 bits per link, provides access to a link in few hundreds of nanoseconds, improving upon early work on this topic [197, 1333, 14].

11.6.4 Handling Duplicated Data

This problem has two flavors. One is to detect multiple URLs that represent exactly the same page and would add redundant and noisy information if they were displayed (for example, mirrors). The other is to detect multiple URLs that point to partially duplicated contents (partial deduplication can be used to improve ranking and spam detection). Identifying duplicates also reduces the size of the collection that needs to be searched and processed.

Defining what is a duplicate page is not obvious. To illustrate, two pages that contain the same text but differ on their HTML formatting (or CSS) have distinct layouts. Thus, they will not be considered as duplicates if we require that all the content be the same in duplicated pages. Indeed, most mirroring systems implement

exactly this requirement, which implies that duplicates can be detected by using a hash key computed over the whole document. The hashing function used should be easy to compute and have very low probability of collision (that is, the hashing value computed for two different documents should not be equal). Standard hashing functions normally used for this purpose are provided by the MD (Message Digest) and the SHA (Secure Hash Algorithms) hashing families. If formatting is not taken into accounting for detecting duplicates, the same approach can be applied after stripping all the formatting instructions from the HTML document.

Near duplicates are more complex to handle. An example of a near duplicate can be a mirror page which differs only by a date change or a footnote that was automatically added; neither of which would be detected by using hashing. One method to identify near duplicates is to use the cosine distance as similarity measure (see section 6.5.3). Kolcz *et al.* [927] proposed an optimization in which terms that were very popular were assumed to occur in each document and thus ignored. This is equivalent to statistically defining a set of stopwords. They report getting results within 90% percent of the best possible using just terms that appeared in less than 5% of the collection, with a performance one order of magnitude better.

Another option is to use the resemblance measure defined in section 6.5.3. In this case we can choose the function W (typically using shingles) and pick the best threshold t so as to ensure an efficient computation [267]. In both cases, cosine and resemblance, two documents are considered duplicate if the similarity (distance) between them is above (below) a threshold value t.

There are several optimizations that approximate the distance and differ in their efficiency and probability of error, like COPS [262], KOALA [742] and DSC [267]. The first idea is to use a hash value associated to each shingle based on Karp-Rabin's idea of text fingerprinting to search text [877]. That is, the hashing values can be computed incrementally in linear time. The second idea is to just consider some of the shingles, forming super shingles [742, 269]. Both optimizations reduce the number of comparisons needed in the computation. In fact, if the number of shingles per document is constant, say k super shingles, duplicate detection can be done in linear time.

Later, Chowdhury *et al.* [380] proposed I-Match to compute the hashing code of every document without considering too infrequent and too frequent terms. They show that, in the worst case, the algorithm is $O(d \log d)$ for a collection of d documents, but $O(d)$ in practice. They also show that it performs better than the super shingle approach, partly because small documents are not considered at all if they contain only common or infrequent words. This is a disadvantage in some applications, where we need to delete small duplicates such as in email spam processing [926]. On the other hand, on the Web this might not be a problem as small duplicated documents will not frequently appear high in the answer ranking.

11.7 Search Engine User Interaction

Web search engines target hundreds of millions of users, most of which have very little technical background. As a consequence, the design of the interface has been heavily influenced by a *extreme simplicity rule*, as follows.

Extreme Simplicity Rule. The design of the *user search experience*, *i.e.*, the patterns of user interaction with the search engine, must assume that the users have only minimal prior knowledge of the search task and must require as little learning on their part as possible. In fact, users are expected to read the "user manual" of a new refrigerator or DVD player more often than the help page of their favorite search engine. One immediate consequence of this state of affairs is that a user that does not "get it", while interacting with a search engine, is very likely to attempt to solve the problem by simply switching to another search engine. In this context, *extreme simplicity* has become the rule for user interaction in Web search.

In this section, we describe typical user interaction models for the most popular Web Search engines of today, their recent innovations, and the challenges they face to abide by this extreme simplicity rule. Some of the user interaction concepts we discuss here have already been introduced in Chapter 2, but we revisit them here in more depth, in the context of the Web search experience offered by major players such as Ask.com, Bing, Google and Yahoo! Search. We do not discuss here "vertical" search engines, *i.e.*, search engines restricted to a specific domains of knowledge such as Yelp[15] or Netflix,[16] or major search engines verticals, such as Google Image Search or Yahoo! Answers.

11.7.1 The Search Rectangle Paradigm

Users are now accustomed with specifying their information needs by formulating queries in a search "rectangle". This interaction mode has become so popular that many Web homepages now feature a rectangle search box, visible in prominent area of the site, even if the supporting search technology is provided by a third partner. To illustrate, Figure 11.10 displays the search rectangle of the Ask, Bing, Google, and Yahoo! search engines. The rectangle design has remained pretty much stable for some engines such as Google, whose main homepage has basically not changed in the last ten years. Others like Ask and Bing allow a more fantasy oriented design with colorful skins and beautiful photos of interesting places and objects (notice, for instance, the Golden Gate bridge background of Ask in Figure 11.10). Despite these trends, the search rectangle remains the center piece of the action in all engines. This rectangle is also commonly referred to as the "search box", as discussed in Chapter 2.

While the display of a search rectangle at the center of the page is the favored layout style, there are alternatives:

- Some Web portals embed the search rectangle in a privileged area of the homepage. Examples of this approach are provided by yahoo.com or aol.com.
- Many sites include an Advanced Search page, which provides the users with a form composed of multiple search "rectangles" and options (rarely used).
- The search toolbars provided by most search engines as a browser plug-in, or built-in in browsers like Firefox, can be seen as a leaner version of the central search rectangle. By being accessible at all times, they represent a more

[15] http://www.yelp.com
[16] http://www.netflix.com

Figure 11.10: The search rectangle as the central user interface component of four major search engines (from Ask, Bing, Google and Yahoo! Search, respectively). Ask screenshot, ©IAC Search & Media, Inc. 2010. All rights reserved. ASK.COM, ASK JEEVES, the ASK logo, the ASK JEEVES logo and other trade marks appearing on the Ask.com and Ask Jeeves websites are properti of IAC Search & Media, Inc. and/or its licensors.

convenient alternative to the homepage rectangle, yet their requirement of a download for installation prevents wider adoption. Notice that, to compensate this overhead, many search engines negotiate costly OEM deals with PC distributors/manufacturers to preinstall their toolbars.

- The "ultimate" rectangle, introduced by Google's Chrome "omnibox", merges the functionality of the address bar with that of the search box. It becomes then responsibility of the browser to decide whether the text inputted by the user aims at navigating to a given site or at conducting a search. Prior to the introduction of the omnibox, Firefox already provided functionality to recognize that certain words cannot be part of a URL and thus, should be treated as part of a query. In these cases, it would trigger Google's "I feel lucky" function to return the top search result. Interestingly enough, this Firefox feature is customizable, allowing users to trigger search engines other than Google or to obtain a full page of results.

Query Languages

As explained in section 7.1.1, users typically express their queries as a sequence of words. Indeed, after years of debate between the champions of "free-text" queries and the champions of Boolean queries, Web Search engines basically won the battle. As a consequence, the free-text format has become the *de facto* standard query language of Web search engines. Users nowadays typically enter a sequence of words that, hopefully, describe their information needs [268] or their goals [1382]. Some search engines declare that the underlying semantic of a query is an "AND" of all terms, others like Google claim that "every word matters". Most search engines reserve the right to change the semantics as needed, hence the existence of additional operators like "+" is not guaranteed in the future.

Indeed, while the most dominant query language of all search engines is a sequence of words, Web search engines implicitly agree on a basic query syntax. The query language typically consists of unary operators such as "+", "-", and "**site:**" to qualify

the immediately following word, binary operators like OR to operate on the preceding and succeeding words, or delimiters such as double quotes to indicate exact phrase match.

Most search engines do not publish an official query language, but rather provide Search tips or Options on their sites. We have compiled in Table 11.4 a short comparison between the search tips/options of leading search engines as provided on the associated pages of Ask.com [76], Bing [202, 203], Google [652] and Yahoo! Search [1736]. This table is in no way exhaustive, given that some query operators are implemented in hidden mode and publicized elsewhere. Such an example is Google's "numrange" feature [650], which was announced in 2004 and allows users to indicate numeric ranges by entering a **".."** operator between two numbers. To illustrate, the query *"DVD player $100..300"* matches any number in the range [100,300], *e.g.,* 250. In our table, we focus on "stable" operators that are well known and publicized rather than on power hacks as the "numrange" operator we just described.

In addition to these common operators, we list below some unique ones supported by a single search engine. It will be interesting to follow their usage and verify whether or not they will get adopted by other engines as time goes by.

- Ask's temporal operators:
  - *afterdate:*, *beforedate:*
    The above operators followed by a date expressed in the `yyyymmdd` format return results that occur respectively after or before the specified date.
  - *betweendate:*
    This operator works in a similar manner but accepts two dates separated by a comma and specified in the same `yyyymmdd`format.
  - *last*:
    This operator followed by a given time period, among the following 6 values: {*week, 2weeks, month, 6months, year, 2years*}, returns results that are found within that specified period of time.
- Bing:
  - *AND/&*
    This operator provided only by Bing is a Boolean AND between the preceding and succeeding terms. Interestingly enough, Bing provides this operator even if it claims that "by default all searches are AND searches".
  - ( )
    Parentheses are used to group words in conjunction with other operators like – or +. This feature increases the power of the latter operators, but is probably reserved to users with some math/logic background.
  - Bing has a long list of unique operators such as *filetype:, contains:, ip:. feed:, prefer:*, etc., some seem promising, other less so.
- Google: *
  The wildcard stands for a missing full term (rather than part of a word) and indicates to the search engine that it should be treated "as a placeholder for any unknown term".

Operator Syntax	Details	Google	Yahoo! Search	Bing	Ask
".." double quotes surrounding a string	Phrase search	yes	yes	yes	yes
+ preceded by a space, operates on the term/phrase that immediately follows	This operator ensures that the associated term is included "as is" in the results	yes	yes	yes	yes
− preceded by a space, operates on the term/phrase that immediately follows, Bing uses NOT as well	This operator ensures that the associated terms do not appear in any result	yes	yes	yes	yes
OR (as well as |) operates on preceding and succeeding terms or phrases	Equivalent to a Boolean OR	yes	yes	yes	yes
site: Followed by a site name	Returns results from the specific site only	yes	yes	yes	yes
hostname: Followed by a host name	Returns results from the specific host only	no	yes	no	yes
url: Followed by a URL	Checks that the following url exists in the engine index	no	yes	yes	no
inurl: Followed by a term	Returns results whose URL contains the specified term	no	yes	no	yes
intitle: Followed by a term	Returns results whose title contains the specific term	no	yes	yes	yes
inlink:/inanchor: Followed by a term	Returns results that contain the specific term in their link or anchor metadata	yes	no	yes	yes

Table 11.4: Common query operators.

- Yahoo! Search: *link:* followed by a url
  This operator returns documents that link to a specific url, a feature provided by the Yahoo! Site Explorer tool. It should be seen more as a shortcut to another property than as an operator *per se*. As a matter of fact, Yahoo! as a content provider is stronger than most other engines in terms of shortcut to its properties, offering direct access via a set of reserved tokens, the default ones, which can be retrieved by typing *!list* in the search rectangle. These include, for instance, *!news, !flickr, !wiki, !map*.

As can be seen in the above examples, dominant search engines provide fewer operators than the less dominant ones, probably due to usage observation. It is reasonable to guess that they are rarely used and the *de facto* standard query language on the Web is simply free-text. Nevertheless, this could change as newcomers make their apparition and try to win over the market by using new features. It comes as no surprise at the time this book is being written that Bing, as the newest search engine in the market, is the one providing the widest variety of operators, probably waiting for its users to indicate their preferences via usage.

Dynamic Query Suggestions

Dynamic query suggestions services, also referred to as "auto-complete" or "auto-suggest" in Chapter 2, enrich the search rectangle with interactive capabilities. As users enter characters in the search box, one at a time, query suggestions are offered to them in a drop-down. Examples provided by Google and Yahoo! are shown in Figure 11.11.

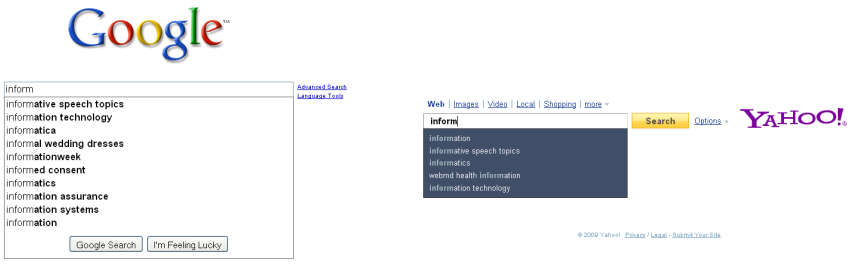

Figure 11.11: Google Suggest and Yahoo! Search Assist.

Such a service was pioneered by Google Suggest, invented by Kevin Gibbs, and deployed on Google Labs in 2004 and in its toolbar soon after, but not on the main google.com page, probably due to scalability issues. Yahoo! Search went ahead and crossed that chasm before Google, when it deployed its Search Assist feature on yahoo.com and search.yahoo.com main pages in 2007. Interestingly, Yahoo! departed from the classic prefix completion model as it offered mid-string completions as well, see the *"webmd health* inform*ation"* query offered for the prefix *"inform"* by Yahoo! Search in Figure 11.11. Google Suggest followed and did launch on Google major homepages, first on youtube.com and then on google.com by August 2008. Suggest finally reached all Google domains by early 2009. Other search engines caught up really fast and similar services are now available both on Ask and Bing in the United

States, and even on vertical search services that have enough traffic such as Amazon.com. Note that query completion services on vertical search engines such as Netflix or Nextbio, as discussed in Chapter 2, are corpus driven rather than query driven, and thus, do not scale to the Web.

Today, most users take advantage of such services without paying close attention to them. This is likely the supreme compliment that can be payed in the search user interaction realm, as per our previously mentioned extreme simplicity rule.

Dynamic query suggestions systems should be distinguished from query recommendations systems that operate only after the search query has been issued. Dynamic suggestions systems work with very little information, their main input being a prefix rather than a well formed query. They have nevertheless at their disposal all the additional signals that any search engine can use, as long as these can be leveraged without affecting response time. These additional signals include users' history and personalization features when available, geographical signals, possibly previous queries from the same user's session.

Dynamic suggestions systems are not entirely new as a similar functionality was offered by early editors like Emacs [1527], which supported command completion, or shell scripts like Korn shell [224], which would complete commands as well as file/directory names, upon request, when a user would enter a tab or space character. At a later stage, assistance services were also offered in the mobile domain to reduce the number of required keystrokes on cumbersome keypads [66, 1233]. The key differences between these early features and modern dynamic query suggestion are:

1. the source and scale of modern suggestions corpora,

2. the performance demands as modern suggestions services need to serve a huge number of users at the same time (as opposed to a single user on a single application/device), and

3. user-experience wise, the fact that modern suggestions are triggered automatically as the user types rather than upon request.

Addressing these challenges was made possible by two major changes in the Web environment: growth in search traffic and performance enhancements. Indeed, the incredible growth in Web search traffic allowed gathering a huge query corpus, which could be leveraged to serve all users. By using, as main source of data, query logs rather than the corpus at hand, a closer language model could be used so as to present more "natural" suggestions to users and thus increase precision. Additionally, the huge performance enhancements in Web search allowed centralizing a service that is, on average, five times heavier (in terms of qps, *e.g.*, queries per second) than regular search. By default, a request is sent to the server each time a new character is entered and relevant suggestions need to be returned quickly, as the user is still typing the query.

Interestingly, dynamic suggestion services had to overcome the two classic IR technical challenges in order to bring value to users: (a) efficiency, so as to return suggestions fast enough to be usable, and (b) effectiveness, so as to present the most relevant suggestions.

- Efficiency. Fast response time is typically achieved via two means. First, the candidate queries associated with a given prefix must be precomputed and stored in efficient data structures such as proposed by Ji *et al.* [834], so as to require minimal processing at query time. Second, they need to be served fast, which is typically achieved using a number of distributed data centers. In addition, the Javascript suggestion code instilled in the search page should be as lean as possible, and, as a safety precaution, not prevent the user from immediately issuing queries in case it takes more time than planned to load.
- Effectiveness. Given a string of characters entered in the rectangle, from one to a few dozens in extreme cases, the dynamic suggestion service returns a list of five candidates (in the case of Yahoo! Assist on the main search page) to ten candidates (in the case of Google Suggest or Yahoo! results page). The major criterion for relevance is popularity, as reflected not only by frequency of occurrence in the query log, but additional signals such as (possibly) click-through rate. Many technical issues need to be considered to ensure relevance, as follows.
  - Automatic spelling correction. It would be weird to propose misspelled queries as suggestions even when popular. Indeed, as the query log corpus is significantly smaller than the Web corpus, the "wisdom of crowds" [1546] principle that works so well for features like spelling, as discussed later, might work less well on the long tail of the query log. This "small corpus challenge" is common in countries with relatively small Web content, or in enterprise search, for instance. It affects not only automatic spelling correction for suggestions, but several other features that build upon the wisdom of crowds as discussed along this section.
  - Inappropriate suggestions filtering. Inappropriate queries, typically involving porn or hate, still represent a significant percentage of query logs. They need to be filtered out in order to avoid offending users and to comply with local legislation. In addition, spam is a more acute issue than in regular search, specially in long tail queries, since they are observed less frequently and are easier to foul.
  - De-duplication of queries suggestions. Slight variations of queries, expressing the same information need might all qualify, in terms of relevance, when they refer to a very popular topic, and yet add no value to the user and be annoying. Examples include plural forms and different arrangement of the words. Dynamic suggestion services work hard at identifying these quasi-duplicates and identifying the most representative ones for the suggestion box.
  - Diversity of suggestions. This issue is somehow related to the previous one, but in a more subtle way. Some senses for query completions might be so dominant for a given prefix, that rarer senses might never get a chance to be shown. This presents a danger because of the sheer nature of the suggestion service: popular queries get even more popular (the rich gets richer syndrome) and, as a result, interesting topics might see their relevance score slowly decrease until they eventually are removed from the list of suggestions for lack of clicks.

- Freshness. Popular topics are trendy and often follow the news, users cannot wait for the dynamic system to regenerate its index to make them visible. Fresh suggestions need thus to be treated differently, as also done when indexing dynamic search corpora.
- Personalization. This refers to both user and domain personalization. Users who are used to seeing their past queries history in their search box as served by the browser, might not be willing to lose them. Yet, if the user is not logged in, displaying personal queries together with community queries might be unexpected, and even give the impression that privacy has been breached. Community/domain personalization is even more critical, because interest and popularity differ so much between various locations and cultures. A user in the United States might be surprised to get as top suggestion for the prefix "real", the query "Real Madrid" rather than "real player", while users in Spain would probably be shocked if they did not. It is thus important to, at the bare minimum, personalize suggestions by country. It is also probably desirable to personalize by geographical location at a finer and finer degree. Indeed a user in Alaska issuing the query "pizza" will be probably be pretty annoyed if the first query suggestion is "pizza Palo Alto" or "pizza Manhattan Upper West Side", simply because of their sheer popularity. The technical challenge is to gather enough evidence to maintain high precision, while working with smaller and smaller corpora.

Overall, the effectiveness of dynamic suggestions can be measured by coverage and quality (a type of recall and precision). Achieving good coverage, which translates in terms of user's perception to always filling all the allocated spots in the suggestion box, becomes especially tricky when prefixes get longer. The longer the prefix, the higher the chance of hitting a long tail query without enough support evidence to qualify as a suggestion. Quality is obviously a must as well, as users will expect the suggestion service to "read their mind". Most users will actually mistake the suggestions for the "voice of the search engine", rather than for what it is in reality, namely the "voice of the community of users", as demonstrated by lawsuits [1009, 1096] and negative press motivated by individuals and companies offended by suggestions associated with their names [1008].

As a conclusion, we believe that dynamic suggestion services are here to stay and represent a promising area for further research. They occupy an extremely prominent real-estate area on the search page, Google Suggest was in fact the most visible change on Google's homepage, since its original launch more than ten years ago. They represent an obvious bottleneck because providing wrong suggestions would drive a significant[17] percentage of the users to "bad queries", which could lead to poor organic/sponsored results and thus, impact the entire revenue stream.

In addition, their successful implementation requires overcoming many classic IR challenges. As an example, the reader should consider the effectiveness issues mentioned above to verify that they remind classic retrieval issues if one replaces "prefix" with "query", and "queries" with "results". In the context of query suggestions, these

[17]Major search engines do not publish the clickthrough rate of their suggestion services but it can reasonably be expected to be in double digits.

challenges take an even more extreme form, with harsher effectiveness and efficiency constraints.

11.7.2 The Search Engine Result Page

Result Presentation

The Basic Layout

The classic presentation style of the Search Engine Result Page, often referred to as SERP, as discussed in Chapter 2, consists of a list of "organic" or "algorithmic" results, that appear on the left[18] hand side of the results page, as well as paid/sponsored results (ads) that appear on the right hand side. Additionally, the most relevant paid results might appear on top of the organic results in the North area, as illustrated in Figure 11.12. By default, most search engines show ten results in the first page, even though some engines, such as Bing, allow users to customize the number of results to show on a page.

Figure 11.12 illustrates the layout generally adopted by most engines, with common but not uniformly adopted locations being indicated by a dotted line framed box. These engines might differ on small details like the "query assistance" features, which might appear in the North, South or West region of the page, the position of the navigational tools, which might or might not be displayed on the West region, or the position of spelling correction recommendations, which might appear before of after the sponsored results in the North region.

Search engines constantly experiment with small variations of layout, and it might be the case that drastically different layouts be adopted in the future, as this is a space that calls for innovative features. To illustrate, Cuil[19] introduced a radically different layout that departs from the one dimensional ranking, but this is more the exception than the rule. In contrast, search properties other than the main engines, commonly adopt distinct, such as the image search in both Google and Yahoo! as an example, or Google Ads search results [655], all of which display results across several columns.

In this section, we focus exclusively on the organic part of search results. We will refer to them from now as "search results", note the distinction with paid/sponsored search results.

The Title/Snippet/URL Entity

Major search engines use a very similar format to display individual results composed basically of (a) a title shown in blue and underlined, (b) a short snippet consisting of two or three sentences extracted from the result page, and (c) a URL, that points to the page that contains the full text. In most cases, titles can be extracted directly from the page. When a page does not have a title, anchor texts pointing to it can be used to generate a title.

[18]Note that for languages in which the writing is done from from right-to-left, this is reversed with the organic results appearing on the right hand side and the sponsored ones on the left hand side of the page.

[19]http://www.cuil.com.

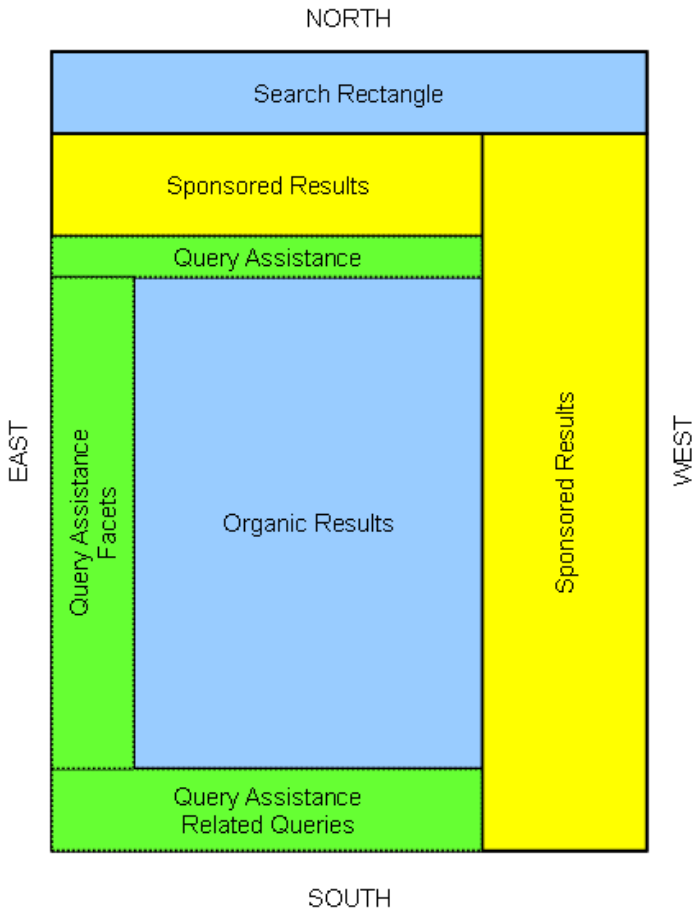

Figure 11.12: A typical SERP layout.

As discussed in Chapter 2, snippets consist of automatically generated excerpts aimed at highlighting topics in the page associated with the user's query. Hopefully, snippets help facilitate the decision of whether to click on a link or not. The key challenge behind snippets is that they need to be generated at run time, since they are query dependent. Also important, query words are usually highlighted in snippets via bold fonts, which requires a string matching operation, or some smart processing to find the right terms, when there is no obvious choice. To illustrate, consider the "arod" example given by Ben Gomes in Google blog [637], in which *"Alex and Rodriguez were bolded in the search result snippet, based on [the] analysis that you might plausibly be referring to him".* Amitay *et al.* [49] introduced an original method for generating Web summaries of a page, that rely on the anchor texts pointing to it.

When several results originate from a same site or domain, search engines group

them by indenting less relevant representatives of the same site. A superior and more recent approach is the "Sitelink" or "Quicklink" format, in which *"navigational shortcuts [..] are displayed below the Web site homepage on a search results page and let users directly jump to selected points inside the Web site"* [347]. Identifying these links automatically is not trivial as the goal is to maximize the benefits for a majority of users, while showing only relevant links in a limited real estate. Yet, the leading engines seem to do a pretty good job at it and will probably keep improving since the best links are usually inferred from learning users' behavior via clicks and toolbar data [347].

More Structured Results

In addition to results fed by the main Web corpus, search engines include additional types of results, as follows.

- "Oneboxes" results. These are very specific results, produced in response to very precise queries, that are susceptible of having one unique answer. They are displayed above regular Web results, due to their high relevance, and in a distinct format. Oneboxes are triggered by specific terms in the user's query that indicate a clear intent. They aim at either exposing the answer directly or exposing a direct link to the answer, which provides the ultimate search experience but can be achieved only in very specific cases, *i.e.*, when relevance is guaranteed and the answer is short and unambiguous.

  As an example, both Google and Yahoo! Search support a weather onebox, which is triggered by entering "weather <a location>" (see example in Figure 11.13). Examples of oneboxes supported by a number of search engines include: Sports oneboxes, package tracking (UPS or Fedex for instance), calculator, movies listings and show times, (try *"Movies Palo Alto"*), train schedules, etc. A more intriguing, and clearly more challenging example is "fact extraction", which even if not displayed in the classic onebox format, provides a similar experience. Try for example, the query "who is the governor of california" on Google and the first result will show "California – Governor: Arnold Schwarzenegger". Another type of onebox-like result can be triggered by queries that do not have one single answer, but rather one mode of querying. A query like "flights from New York to London" on Google will show an entry box that allows the user to specify their departure and return dates, and then route the new query to aggregators such as Kayak, Expedia etc. Google lists most of its current oneboxes, as well as some reserved tokens such as "define:", in its Search Features page [654]. Oneboxes are basically a quick hack, and a sort of testimony that the regular ranked list paradigm does not satisfy all users' needs.

  A related concept are Yahoo! shortcuts which are partially handcrafted oneboxes on top of the organic results for popular queries (try a city or a famous person). A more elegant solution is provided by the "universal search" vision, which was discussed by Marissa Mayer in Google blog [1102] in 2007, that we cover next.

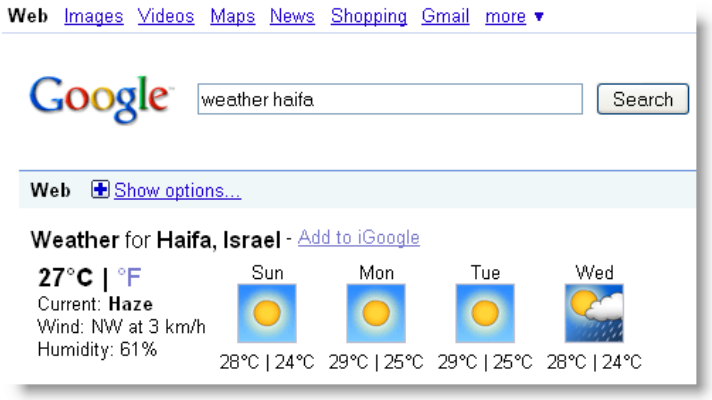

Figure 11.13: Example of weather onebox from Google, where is no need to click to get the full answer.

- Universal search results: Most Web search engines offer, in addition to core Web search, other properties, such as Images, Videos, Products, Maps, which come with their own vertical search. While users can go directly to these properties to conduct corpus-specific searches, the "universal" vision states that users should not have to specify the target corpus. The engine should guess their intent and automatically return results from the most relevant sources when appropriate. The key technical challenge here is to select these sources and to decide how many results from each sources to display. Since this is an instance of a classic federated search problem, using techniques, such as those proposed by Fagin and Wimmers for combining orthogonal scores [543] can help. As of today, search engines do not publish their approach, so it is not clear whether a principled formula is used for universal ranking or whether some heuristics are applied. Typically, a query such as "Britney Spears" will return images, videos and news together with core Web results.

Web results can also appear in a different format depending on their type. An example is provided by Ask.com, which displays answers originating from Q&A sites like Yahoo! Answers or WikiAnswers, directly without showing a snippet. More generally, a common trend is to show in a slightly different format results originating from specific sources that are either part of the main Web corpus, or serviced by other properties. This is exemplified by SiteMonkey, a Yahoo! Search open platform that allows publishers to *"share structured data with Yahoo! Search [in order] to display a standard enhanced result"*[1735]. SearchMonkey was launched in 2007 and was partly inspired on Peter Mika's earlier research on microformats [1129]. As an example, all Wikipedia results in Yahoo! are displayed in a SearchMonkey format. Google recently followed Yahoo!'s example and explored the rich result format when it launched its rich snippets in 2009 [629]. Finally, Bing is investing a great deal of efforts in structured results, offering "digest-type" results for specific domains. We list some of Bing's examples below.

- Travel results feature a fare trend indicator.

- Shopping results include some convenient shortcuts to users' reviews, experts' reviews, product details and price comparison, the best price found among results, ratings, ease of use, affordability, visual cues, and the novel "Bing cashback".
- Health results indicate authoritative sources, such as the Mayo clinic, as content provider.
- Local results comprise review-based scored cards with visual cues for topics such as "overall", "atmosphere", etc.

It will be interesting to see whether Bing will easily generalize and scale up to other verticals, as well as to other countries, while maintaining the same type of rich features.

Query Assistance on the SERP

Once users have issued their queries and looked at the search result pages, their informational, navigational as well transactional needs [268] (see more details in section 7.2.3), can be either:

- satisfied. This happens either immediately when users get the answer directly from a onebox result, such as calculator, weather, sports results, etc.) or almost immediately after they click on one or a few of the top results.
- partially satisfied. This usually happens when users have undertaken a "research task" [218], and there is no single Web page that holds all the needed information. Tools like Search Pad, which is described later in section 11.7.2, have been designed precisely to gather, annotate and organize these partial answers into one coherent unit, which can then be stored for later usage, or published/shared with others. Some needs are more susceptible than others to trigger research tasks. Examples include travel needs of a user looking for hotels, restaurants, and entertainment opportunities, homework, education needs of a student working on an assignment, or health information of a patient on an illness, its symptoms and treatment options.
- not satisfied at all. This can happen either because users did not formulate their query well or because relevant content simply does not exist. It is still almost impossible for search engines to decide when relevant content does not exist, so by default most engines will assume the first scenario and help users reformulate their queries via query assistance tools.

In this section, we explore in depth, the query assistance tools devised to help users refine or reformulate their queries, as discussed in Chapter 2.

Spelling Assistance

The most successful example of query assistance is the now famous *"Did you mean"*, offered by Google, that revolutionized spelling correction by departing from the usual dictionary-based model. Indeed, the classic approach was to use edit distances to

identify typing mistakes such as letter inversions [944] (see section 6.5.3). Instead, "Did you mean" learns its spelling corrections simply from usage and a great deal of usage. It extensively uses query logs analysis for spelling [637]. One common example that has been given by Google speakers on Campus talks is the "Britney Spears' example", in which query logs show her name misspelled in a few hundreds different manners (users are creative!) and yet the most frequent spelling by far is simply the correct one. That is, wisdom of crowds at its best (see section 11.10.2. This sheer frequency signal is less effective for long tail queries, or in domains for which logs are not large enough, and thus suffers from the "small corpus challenge" mentioned before. In such cases, other signals can be used that require less evidence. As an example, Douglas Merrill, a former Google CIO, in one of his Search 101 talks [1121], explained that by simply observing users rephrase their queries in two successive queries, the engine can learn the correct spelling of a query. Cucerzan and Brill investigated this very approach in [461] and showed how to learn query correction models from query reformulations in the query logs.

Query Recommendations

Other means of query assistance on the SERP include query recommendations. Note that SERP query recommendations differ from dynamic query suggestions provided in the search rectangle, as they can take advantage of richer types of information, from full well-formed queries (as opposed to partially specified one) to the rich set of results, with their respective snippets and their associated relevance signals. Query recommendations typically consist of queries related in some sense to the original query and are most useful when users struggle with expressing their needs and turn to related and hopefully better formulated queries that might be more successful.

There has been a number of research works in mining query logs for generating query recommendations. They can roughly be put in three categories.

1. Content-aware approaches rely on search results or target pages. An early example of this class is the work done by Raghavan and Sever [1329], who attempted to measure query similarity by determining differences in the ordering of documents returned in the results set. While this approach has the advantage of richer information as provided by the document collection, it poses challenges in terms of scalability. Similarly, Fitzpatrick and Dent [567], measured query similarity using the normalized set intersection of the top 200 results. Again, their technique suffered from scalability issues as the intersection of semantically similar queries that use different synonyms is typically very small. Sahami in [1406], used a query similarity based on the snippets of results. They treat each snippet as a query, which is submitted to the search engine aiming at finding documents that contain the terms in the original snippets. They then use these returned documents to create a context vector for the original snippet.

2. Content-ignorant approaches are well represented by Befferman and Berger [166], who infer similarity between queries from common clicked URLs. Unfortunately, the impact of such methods is somehow limited, because the number of clicks in the results pages is relatively small [89], and thus the associated query-to-query

distance matrices remain sparse. This sparsity could be diminished though by using larger query logs if allowed by legislation.

3. Query-flow approaches consider the users' sequential search behavior to better understand query intent. Fonseca *et al.* [571] and Zhang *et al.* [1777] are good example of this school. Fonseca *et al.* view query logs as a set of transactions, where each transaction represents a *session* in which a single user submits a sequence of related queries in a given time interval. The method shows good results, however two problems arise. First, it is difficult to determine sessions of successive queries that belong to the same search process, and on the other hand, the most interesting related queries, those submitted by different users, cannot be discovered.

Most modern solutions use hybrid approaches for higher precision. As an example, Baeza-Yates *et al.* [109, 110, 111] use the content of clicked Web pages to define a term-weight vector model for a query. They consider terms in the URLs clicked after a query. Each term is weighted according to the number of occurrences of the query and the number of clicks of the documents in which the term appears. One approach that seems really promising, research-wise, is to mine relations from the query flow.

For example, sessions are usually physical sessions and not logical sessions. Thus, four subsequent queries in a short time interval might be related to two drastically different tasks. With recent attempts at formalizing the query flow graph [218], we can expect to see more sophisticated mining techniques that could achieve better results.

Web search engines do not communicate the methods they favor, but one can expect that they use the "best of breed" and take advantage of multiple signals. Note that there is no unanimity on the placement of these suggestions on the SERP. This is an interesting phenomenon as position has a direct impact on usage and is probably an indicator of how much search engines trust their recommendation tools.

Google displays its query recommendations under the label "Search related to:" at the bottom of the SERP (green rectangle at the bottom in Figure 11.12) and arrange them in four columns of two candidates. Consequently, it can be expected that the clickthrough rate of this feature is relatively small. Interestingly enough, the recently launched "Search options" feature of Google "tool belt" provides access to "related searches" and to the original (yet not widely used probably due to its relatively low precision)"wonder wheel", which gives a graphical representation of related search terms. By clicking on any node of the wheel, the user obtains related topics in the interactive animated wheel, while results keep being updated on the right hand side.

Yahoo! Search also displays related results at several locations, such as right below the search rectangle, under the label "Also try" (see 2nd green rectangle from the top in Figure 11.12), on the left navigation pane and even within the search rectangle, side by side with regular dynamic query suggestions. The latter have a different scope than regular query suggestions and appear on the SERP search rectangle, only when the user continues entering a query or voluntarily expands it. Finally, Bing displays these on the West region navigation pane and labels them as "related searches".

Query Refinement via Facets

Queries can also be refined by restricting results along certain "facets", according to the faceted search paradigm. Faceted search is a navigation mechanism that "enables users to navigate a multi-dimensional information space by combining text search with a progressive narrowing of choices in each dimension" [279]. Note that we consider here faceted navigation as a query refinement mechanism since, in practice, it requires from the user to select a facet. This user-provided input augments the query with additional information so as to better specify the user's needs and narrow the results set. In Chapter 2, we review faceted navigation in detail and illustrate the use of faceted navigation in research systems such as Flamenco, University of Chicago's Aquabrowser and in vertical search services such as Yelp.com.

Figure 11.14: High level source-driven facets in Yahoo! search.

In Web search, faceted navigation has only started to appear as it is more technically challenging for reasons of scale. One approach for implementing faceted navigation in the context of the Web is to map attributes of the results, such as their type (video, audio) or their source (Wikipedia, YouTube, Yahoo! answers), into navigational facets. This approach is adopted by Yahoo! Search, as illustrated in Figure 11.14, where the West navigation pane shows a number of relevant sources for narrowing the query "Barcelona". Bing also uses a similar approach as shown in Figure 11.15. While the implementation details for this mechanism are not public, one can envision a simple implementation in which the index stores these static attributes and the engine fetches and processes them at run time.

Google offers a similar functionality through its previously mentioned tool belt feature. The user can "slice and dice" the results via various types of facets, from the type/source ones, such as "Video, Forums, Reviews" facets to time-based facets, such as "Past 24 hours", "Past week" and "Past year".

A more complex case consists of displaying the number of results associated with each facet. No Web search engine offers this feature yet. While it has been offered in the past by enterprise faceted search engines such as Endeca [534] and in multiple shopping sites, it is a difficult task to estimate these counts within decent response time at the scale of the Web.

Figure 11.15: Faceted navigation on the left side from Bing.

Faceted search research has investigated even more complex cases such as hierarchical, correlated, and even dynamic facets that require computing at run-time [1751], as well as their associated visual interfaces [739]. However since they have not been deployed by any major Web search engine yet, we will not discuss them here. It will be interesting to follow how this area progresses since major Web search engines are exploring the topic, each in its own way as demonstrated above.

Actionable Results

The last SERP topic we address here relates to a variety of promising tools that allow to do more with search results than simply interpret or navigate them. This list is not exhaustive, as this area is rich and thriving. We expect more tools to appear at a fast pace, but also expect many to disappear, as exemplified by Google stopping the development of its notebook extension [653] in early 2009.

Some features simply operate on the result itself for various purposes, examples include Yahoo! Search, or Google "Cached" link and Google "Similar" link. A more advanced feature is Google "Translate this page", which is displayed next to the results title and provides a translation of the target page into the user default language.

Another tool that now is part of of several Web search engines is a built-in search rectangle, typically displayed below sitelinks/quicklinks, which allows users to search within the site that the result belongs to. To illustrate, a search for New York Times on Google leads to a link to the newspaper homepage and to one such associated search-within-site rectangle. Issuing a query from this rectangle will augment the query with a "site:nytimes.com" qualifier (see "site:"operator in Table 11.4), which narrows the results to that site only.

More intriguing tools include Google Stars in Search and Yahoo! Search Pad. The "Stars in Search" feature was launched in March 2010 and replaced the more complex but probably less successful Searchwiki that allowed the user to provide feedback on any result via three small icons displayed next to the result URL: a

bubble for "comment", an arrow up for "promote", and an "x" for "remove". In this now discontinued feature, the user could thus annotate, promote, and get rid of any results at will. Users would see this personalization of results persist if they re-issued the same query in the future. Alternatively, users could visit their own Searchwiki notes at any time by clicking on the appropriate link at the bottom of the SERP. Searchwiki was recently replaced by a leaner version called "Stars in Search" that preserved a variation of Searchwiki "promote" capability. Namely, the three small icons were replaced by a single star that when selected turns yellow and acts as a sort of marker of favorite results. Thus, for subsequent similar searches, the user will see previously starred results appear at the top of the results list in a special section. One interesting lesson to be remembered here is that all search engines carefully monitor adoption and might modify or entirely retire features that have not gained enough traction.

Yahoo! Search Pad is an interesting feature as it belongs to the same family as Google notebook, which was mentioned earlier, yet uses a different approach. Search Pad allows users to easily keep trace of results they have consulted, and arrange and annotate them for later usage or for sharing with others. The concept is not new, it was pioneered by Bharat in [196]. What makes it novel though, and probably more usable than other related tools, is that Search Pad is triggered only when the search engine decides that the user is investigating a topic rather than looking for quick, "disposable" results. Visited pages are automatically added to the appropriate search pad, without requiring the user to specifically "mark" them like in early research work, Ask "My Stuff" [75], or the now discontinued Google Notebook [939].

11.7.3 Educating the User

We have discussed how interfaces are slowly making progresses in assisting the user with query formulation and results interpretation, with richer and richer snippets and results slicing. Yet, we should expect that users, especially younger generations, become more and more Internet savvy and take more control of the search process.

In terms of guidelines, advanced search interfaces, provide forms that allow users to better control the effects of queries. Almost the same effect can be achieved by taking advantage of advanced operators within the query itself. For more control, sophisticated users can specify as many terms as possible, as well as indicate which terms should be included in the results (via the "+" operator) and which ones should not (via the "-" operator). However, a user should not add all possible synonyms of a word, because few Web pages will use more than one or two synonyms for a given concept. If the user can restrict the search to a field (for example, the page title), limit some attributes (date, country) or use the operators mentioned in Table 11.4, the size of the results will be certainly reduced.

Even if we are able to issue good queries, the result set can still be quite large. Considering that the visual tools mentioned before are not yet available for the general public, and it is not clear whether they will ever be adopted, the user must learn from experience. There are many strategies to quickly find relevant answers. If users are looking for an institution, they can always try to guess the corresponding URL by using the `www` prefix, followed by a guessed institution acronym or brief name, and finished by a top level domain (country code or `com`, `edu`, `org`, `gov` for the US). If this

does not work, the user can search the institution name in a Web directory.

Another somewhat frequent task is for a user to search for published work on a specific topic. In order to fulfill this task, a possible strategy is to:

1. select an article related to the topic, if possible with non-common author surnames or title keywords; and

2. use a search engine to find all Web pages that have all those surnames and keywords. Many of the results are likely to be relevant, because they include references to (a) newer papers that reference the initial reference, (b) personal Web pages of the authors, and, (c) pages about the topic that point to many relevant references. This strategy can be iterated by changing the reference used initially as better references appear during the search.

The Web poses so many challenging problems that it is sometimes more effective to educate the user on how to properly profit from search engines and Web directories, rather than try to guess what the user really wants. Given that the coverage of the Web differs among search engines, one approach is use several engines or a metasearcher. The key lessons here are: (1) search engines still return too much hay together with the needle and (2) Web directories do not have enough depth to find the needle. Thus, we recommend to use the following rules of thumb, when issuing queries, namely try:

- Specific queries: look at an Encyclopedia, that is the reason that they exist, do not forget libraries.

- Broad queries: use Web directories to find good starting points.

- Vague or exploratory queries and iterative refinements: use Web search engines and improve query formulation based on relevant answers.

11.8 Browsing

In this section, we cover browsing as an additional discovery paradigm, with special attention to Web directories. Browsing is mostly useful when users have no idea of how to specify a query (which becomes rarer and rarer in the context of the global Web), or when they want to explore a specific collection and are not sure of its scope. Nowadays, browsing is no longer the discovery paradigm of choice on the Web. Despite that, it can still be useful in specific contexts such as that of an Intranet or in vertical domains, as we now discuss.

In the case of browsing, users are willing to invest some time exploring the document space, looking for interesting or even unexpected references. Both with browsing and searching, the user is pursuing discovery goals. However, in search, the user's goal is somewhat crisper. In contrast, with browsing, the user's needs are usually broader. While this distinction is not valid in all cases, we will adopt it here for the sake of simplicity. We first describe the three types of browsing namely, flat, structure driven (with special attention to Web directories), and hypertext driven. Following, we discuss attempts at combining searching and browsing in a hybrid manner.

11.8.1 Flat Browsing

In flat browsing, the user explores a document space that follows a flat organization. For instance, the documents might be represented as dots in a two-dimensional plane or as elements in a single dimension list, which might be ranked by alphabetical or by any other order. The user then glances here and there looking for information within the visited documents. Note that exploring search results is a form of flat browsing. Each single document can also be explored in a flat manner via the browser, using navigation arrows and the scroll bar.

One disadvantage is that in a given page or screen there may not be any clear indication of the context the user is in. For example, while browsing large documents, users might lose track of which part of the document they are looking at. Flat browsing is obviously not available in the global Web due to its scale and distribution, but is still the mechanism of choice when exploring smaller sets. Furthermore, it can be used in combination with search for exploring search results or attributes.

In fact, flat browsing conducted after an initial search allows identifying new keywords of interest. Such keywords can then be added to the original query in an attempt to provide better contextualization. This process is a variant form of *relevance feedback* as discussed in detail in Chapter 5.

11.8.2 Structure Guided Browsing and Web Directories

A more scalable browsing model is the structue-driven model, in which an underlying structure such as a hierarchy or a tree is used to browse the space. This model was very popular in the early days of the Web, when search engines where doing a poor job, and is still available at the global Web level through the Yahoo! directory [1737] or the Open Directory Project [1220], also known as DMOZ. Directories are hierarchies of classes that group documents covering related topics. Some directories are specific to vertical areas. For example, there are Web sites focused on business, on research bibliography (*e.g.*, CiteSeerX [387]), and so on. Web directories may also be called catalogs, yellow pages, or subject directories, depending on their domain of application.

Table 11.5 shows the first level categories used by some Web directories (the number of first level categories ranges from 12 to 26). Some subcategories are also available in the main page of Web directories, adding around 70 more topics. The largest directories, such as the previously mentioned ODP and Yahoo!, cover several millions of Web sites. In most cases, pages have to be submitted to the Web directory, where they are reviewed and classified in one or more categories of the hierarchy. ODP is to be noted as the first Web 2.0 directory, as it adopts a collaborative peer model approach, where people volunteer as editors. Note that even if the underlying structure of directories is hierarchical, they do not form real trees as cross references are frequent. Thus, in practice, directories are directed acyclic graphs.

Web directories also allow users to conduct a search on the taxonomy descriptors or on the Web pages pointed to by the taxonomy. In fact, as the number of Web pages classified is small, directories can even maintain a local copy of all pages for performance reasons. The trade-off is that it becomes the responsibility of the directory to ensure temporal validity.

Arts & Humanities	Business & Economy	Computers & Internet
Education	Entertainment & Leisure	Games
Government	Health & Fitness	Home
Investing	Kids & Family	Life & Style
Local	News	People
Philosophy & Religion	Politics	Reference
Regional	Science & Technology	Shopping & Services
Society & Culture	Sports	Travel & Tourism

Table 11.5: Examples of first level categories in Web directories.

The main advantage of directory browsing is that the information it provides is usually valuable. On the other hand, the two disadvantages are first that the classification is not always specialized enough and more problematic, that the Web coverage provided by directories is very low (less than 1% of all Web pages are covered). So at the Web scale, precision is usually achieved but not recall. Most Web directories are aware of this recall issue and also send queries to global search engines (through a strategic alliance) to complement their results.

An additional problem with directories is content growth and it becomes more acute every day, as the Web continues to expand. Efforts to generate directories automatically either by clustering or by other techniques, started more than 10 years ago. However, such efforts are computationally expensive and, due to the limitations of current natural language processing techniques (which are far from being fully effective yet for extracting key concepts), are not really successful for all cases. As a consequence, classification is still conducted mostly manually by a limited number of editors, which slows the growth of the directory. Manual classification introduces yet another limitation, namely the lack of terminology agreement not only between users and editors, but also among the editors themselves.

The same structure guided model can be applied to a single document. For example, when browsing an electronic book, a first level of content could be the chapters, the second level, all sections, and so on. The last level would be the text itself (flat). A good user interface could go down or up those levels in a focused manner, assisting the user with the task of keeping track of the context.

One common issue when browsing a tree structure is that users might lose context when deep diving in one of the paths. A common method to keep track of context is the *breadcrumbs* or *breadcrumbs trail* [215, 1244], which appear at the top of the page being visited and show the user the path taken to reach that page. Each crumb in the trail is usually clickable to allow the user to get back to a previous stage in one click. An example of breadcrumbs is given in Figure 11.16 below, where the path `Directory > Recreation > Games > Video Games` represents the 4 (clickable) steps that lead to the current "Chat and Forums" page.

A relatively successful incarnation of the structure driven model is embodied in Web site maps, which are either displayed in a dedicated page or in left navigation pane, in most Web sites. Sitemaps have been explored as early as the late 90', as demonstrated by the WWW workshop on Site Mapping [1068], but took almost a decade for some standardization to arise. Today, most of the key players agree on a sitemap protocol expressed in XML, as published by sitemap.org. However, these

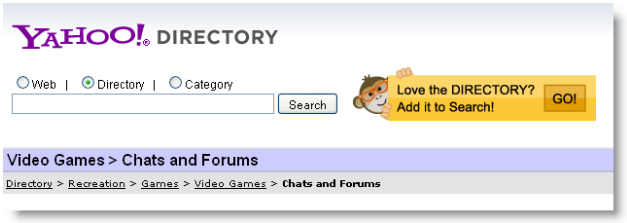

Figure 11.16: An example of breadcrumbs on the Yahoo! directory.

maps are mostly used to help webmasters communicate with search engines and inform them which pages are available for crawling.

Beyond the actual structure of the site, browsing can be guided by the history of accesses. Indeed, history maps are offered by most browsers today. Additional details on browsing large structures are provided in Chapters 2 and 3.

11.9 Beyond Browsing

11.9.1 Hypertext and the Web

Due to historical reasons, many still think of the Web as a giant distributed hypertext. This is a misconception, as the Web lacks an underlying data model, a navigational plan, and a consistently designed user interface. Millions of Web page designers devise their own interfaces independently, with their own peculiar characteristics. As a simple example, when looking for a phone number on a given Web site, we often cannot find it because it is buried in the least expected place in the site. Thus, the Web user has no underlying metaphor that facilitates the search for information of interest.

Consequently, we prefer referring to the Web as a set of (partially) interconnected Web sites. Some of these Web sites might be characterized as a local hypertext (in the sense that they have an underlying structure that enjoys some consistency), but others might be simply a collection of pages designed separately (for instance, the Web site of a university, whose departments design their own pages). Despite lacking in structure and modeling, the Web has provided us with a new dimension in communication because it is easily accessible world wide, and this at a very low cost. More importantly, the Web has no control body setting up regulations and censorship rules. As a result, for the first time in the history of mankind, individuals can publish their writings through a large medium without being subjected to the filtering of an editorial board. That is, the Web is the initial mark of the e-publishing age, as discussed in Chapter 1.

11.9.2 Combining Searching with Browsing

Combining the search and browse paradigms in a hybrid discovery model has been explored with more or less success in a variety of contexts. As mentioned before, most Web directories redirect a fraction of queries to search engines or internally allow users

to search a specific subtree of the directory. Conversely, search engines now associate small link structures to some results, such as quick links in snippets as mentioned in section 11.7.

Search and browse can be potentially combined in a drastically different manner if the Web structure, namely its hyperlinks, becomes either part of the search query (as supported in the early days of the Web by several Web query languages) or the focus of the search process in dynamic search approaches. These ideas of hyperlink driven search remain confined at the research level and are still not widely used due to several limitations, such as poor performance, poor scalability, or awkward user experience.

In the past, researchers tried a variety of approaches combining the two paradigms. As an early example, WebGlimpse [1077] attached a small search box to the bottom of every HTML page, so as to allow the search process to cover the neighborhood of that page or the whole site, while still remaining in a browsing mode. This is equivalent to following hypertext links that are constructed on the fly through a neighborhood search. The neighborhood of a Web page is defined as the set of Web pages that are reachable by a path of hypertext links within a maximum predefined distance. This distance can be set differently for local and remote pages. For example, it can be large for local pages and limited to a distance value of 3 for instance for remote sites. The neighborhood can also include all the subdirectories of the directory where the Web page is. The result is a graph of all the neighborhoods of the Web site or collection, and for each Web page, a file with all the Web pages in its neighborhood. When searching, any query in the whole index can be intersected with a neighborhood list, in an attempt to generate relevant results. This very approach of dynamic crawling around a given seed was proposed by Mapuccino [754, 1067] and its extension, Fetuccino [178]. In addition, these tools added visualization features to display small search/browse maps generated on demand. Indeed, Mapuccino allowed users to dynamically generate site maps tailored to the user's interests (expressed as a query). Navigation maps tailored to users' interests had "longer arms" in relevant directions, and color coded nodes indicating more relevant content. Fetuccino extended Mapuccino with more sophisticated XML-based visualization as well as a two-phase search process that allowed users to distinguish between a domain query and a focused query within the domain. This tool evolved into the webmaster and site analyzer tools in IBM Websphere offerings. Similarly, many site mapping tools evolved into commercial site analysis tools. Examples include the early NetCarta tool that evolved into Microsoft's SiteAnalyst, MAPA from Dynamic Diagrams, SurfSerf, Merzscope from Merzcom, CLEARweb, Astra SiteManager, WebAnalyzer from InContext and HistoryTree from SmartBrowser. One problem is that there is no well established standard for visualizing search results in spite of early proposals based on using XML, such as [34].

There are other tools that use visual metaphors for browsing, which can be broadly classified in two types: tools designed to visualize a subset of the Web and tools designed to visualize large answers. Both cases need to represent a large graph in a meaningful way. Non-commercial examples of tools to visualize Web subsets include WebMap [504], Sitemap, Ptolomeaus, and many earlier research efforts [52, 530, 1139, 1161]. We have not included more generic visualization software, where Web visualization is just a particular case, or other related visualization tools such

as Web usage analysis [619, 1274, 1513]. Metaphors to visualize large answers were already covered in Chapter 2.

In conclusion, visual tools for browsing, even if they are often pretty fancy, are not yet deployed in the whole Web probably because they have yet to demonstrate additional value to users.

11.9.3 Web Query Languages

Until now, we have focused on queries targeted at the content of each page, without considering the option of directly querying the link structure connecting Web pages. For example, we might want to search for all pages that contain at least one image and are reachable from a given site by following at most three links. To allow this type of query, different data models have been used. The most important ones are a labeled graph model that represents Web pages (nodes) and hyperlinks (edges) among Web pages and a semi-structured data model that represents the content of these pages. In the latter model, the data schema is not usually known, may change over time and might be large and descriptive [5, 294].

Although there were some models and languages for querying hypertext that were proposed before the Web appeared [167, 416, 1138], the first generation of Web query languages were aimed at combining content with structure (see also Chapter 7). These languages combined patterns that appear within the documents with graph queries describing link structure (using path regular expressions). They include W3QS [928], WebSQL [72, 1120], WebLog [964], and WQL [1020]. The second generation of results, called Web data manipulation languages, emphasized semi-structured data. However, they extend the previous languages by providing access to the structure of Web pages (the model includes also the internal structure) and by allowing the creation of new structures as a result of a query. Languages in this category include STRUQL [555], FLORID [767], and WebOQL [71]. All the languages mentioned are meant to be used by programs, not by end users. Nevertheless, there are some examples of query interfaces for these languages.

Web query languages have been extended to other Web tasks, such as extracting and integrating information from Web pages, and constructing and restructuring Web sites. More details about Web query languages can be found in the nice survey by Florescu, Levy, and Mendelzon [569]. Many of these languages inspired query languages for XML-based structured text (see section 13.6).

11.9.4 Dynamic Search

The idea behind dynamic search (sometimes called on-line focused crawling and that can be seen as the equivalent of sequential search on the Web) is to dynamically build the search corpus and discover relevant information by following links at run time. The main advantage is that it allows users to search the current "live" structure of the Web, rather than the document collection indexed by a search engine. In other words, it is equivalent to crawling on the fly. This was mostly devised for use in small and dynamic subsets of the Web, rather than on the entire Web for obvious scalability reasons. The first heuristic devised was the *fish search* [248], which exploited the intuition that relevant documents often have neighbors that are relevant. It was subse-

quently improved by the *shark search* [754], which does a better relevance assessment of neighboring pages, and was adopted by the Mapuccino/Fetuccino tools previously mentioned for their visual browsing features. Related work includes software agents that search specific information on the Web [1202, 970]. This implies dealing with heterogeneous sources of information that have to be combined. Important issues in this case are how to determine relevant sources (see also Chapters 10 and 17).

11.10 Related Problems

11.10.1 Computational Advertising

A related Web search problem is computational advertising, *"a new scientific discipline, at the intersection of information retrieval, machine learning, optimization, and microeconomics. Its central challenge is to find the best ad to present to a user engaged in a given context, such as querying a search engine ("sponsored search"), reading a Web page ("content match"), watching a movie, and IM-ing"* [270]. This discipline has been offered as a course at Stanford University, for the first time, in September 2009.

Computational advertising has two main flavors. The best known is matching advertisements (ads) to a given query and displaying them on the right side of the results page. This is called *sponsored search* and is at the heart of systems like Google Adwords. The second flavor is finding the right ads to be included on a given page that is being requested by a user. This case is called *contextual match* and is at the heart of systems like Google Adsense or Yahoo! Context Match.

The advertisement itself can be of two types: image or text-based (or a combination of both). The initial model of display advertisement was mainly images and the search based model is usually text. In many cases, placing the ad is usually coordinated by an intermediary, called the *ad-network*. The business model is usually a payment per impression, in the case of display advertisement, or a payment per click, in the case of sponsored search or contextual match. A third model is cost per action in which the advertiser only pays if a specific target action is achieved (*e.g.*, sell a product).

Computational advertising can be seen as a search problem, where the search input, either a query or the content of a page, has to be matched against a database of ads. Every advertisement in the database has at least a title, a target URL, and a description, called creative, which is in most case textual. One important difference with classic Web search is that the search input, in the case of contextual match, can be much larger than the ads themselves. Because of the small size of creatives, pure textual similarity with the query (be it short or long like in contextual search) will not bring enough relevant ads to users. To address this problem, creatives are systematically augmented by large lists of keywords, which are either generated by the advertisers or offered by the ads system in the more sophisticated scenarios.

The matched ads must be then ranked so as to only present the "best" ones to the users. However, the measure of "goodness" here is not only based on relevance, but also on commercial considerations. In fact, in both cases, advertisers pay according to the number of clicks issued by users on specific creatives, up to a certain plateau fixed by their budget. When the budget is spent, no more ads are shown. This payment

is modeled by an auction mechanism where advertisers bid (and thus compete) on the previously mentioned keywords associated with each creative. In the original scheme invented by the Goto search engine (later renamed Overture and bought by Yahoo!), the order was solely based on the bidding price. However, if no clicks are made for lack of relevance or value to users, the model fails altogether. The actual approach adopted by most search engines thus uses a combination of auction bids (where advertisers bid on specific keywords) and predicted relevance models based on expected *clickthrough rate* (CTR) of the ad, estimated using past history.

Several research efforts have been conducted to better understand the relationship between the characteristics of the ads, and the number of clicks that users make on them, observing as expected that more relevant ads increase the number of clicks. Computational advertising is too wide and complex (and often too confidential) an area to be fully covered here, but we cover next some of the recent published research results in this area.

Jones *et al.* [850] address this problem by trying to match large number of queries (the set of queries submitted by users) to a much smaller corpus of advertiser listings. They propose generating query substitutions of the original query, in order to broaden the set of possible ads, and then ranking the proposed queries. This is achieved by modifying parts of the original query, based on precomputed query and phrase similarity. The data is obtained from the users-sessions of a search engine query logs, since most of the users reformulate their original query by adding or removing words. They compare several machine learning techniques, by combining a set of features, and generating a model for the combination that describes best the query substitution. They observed that better suggestions where obtained by making small edit distances and few changes in the words contained.

Other approaches have been used for tackling the problem of contextual advertisement. Ribeiro-Neto *et al.* [1350] describe an *impedance coupling* technique for context-targeted advertising that constitutes, likely, the first extensive published work on the problem. They focused on algorithms that directly match ads (*i.e.*, advertisements) related keywords to the text of the Web pages. Their experiments, which compared 10 distinct vector-based ranking functions, showed that better ads can be generated for a given page by running more sophisticated ranking functions. Following that work, Lacerda *et al.* [954] explored the use of genetic algorithms to learn a ranking function for contextual ads targeting (likely, this was the first application of learning techniques to the problem). They showed that ranking functions as effective as the best impedance coupling functions can be generated in fully automatic fashion. Immediately subsequent research focused on extracting relevant keywords or phrases from the Web page, which are then used to match the keywords that describe the ads. This is important for reasons of efficiency, given that approaches based on matching all words on the page to the ads related keywords, such as the impedance coupling algorithm just discussed, are expensive to compute in practice.

A technique that extracts a set of features from the phrases and keywords contained in a document and determines which are the relevant ones for ads targeting is presented in [1749]. In total, they used 40 features, and some of these features are whether the keyword is capitalized; contained in the title, in the Meta section, or in the URL; is a noun; etc. Also, they found useful to use the queries contained in the query logs of a search engine, since these are the keywords that are used by the people.

So if a document contains some of these phrases, they can be used as descriptors of the document. They extracted the top 7.5 million English queries from MSN Search. After collecting these features from the documents, they made a manual classification of the relevant keywords of a set of training documents, and then used a supervised learning in order to classify unseen documents. To compare the performance of their approach, they used KEA [891] as a baseline and obtained better results, perhaps due to the fact that they use much more features. This approach permits to extract more relevant key phrases from the content, hence obtaining more relevant ads, increasing the overall revenue.

Another approach, is presented by Broder *et al.* [277]. Instead of using descriptive keywords in order to match the phrase bids, they present a system that extracts semantic and syntactic features for describing the contents of the text, and then matching it with ads. They used a taxonomy built by Yahoo! USA, consisting on 6,000 concepts that describe queries, where each node contains a set of approximately 100 queries. In order to use this taxonomy for classifying pages and ads, they tried several classification schemes. Best results were obtained by concatenating all the queries classified into each node, and generating a meta document. Afterwards, they used these meta documents as centroids for a nearest-neighbor classifier based on Rocchio classifier, and using the cosine distance between the document to be classified and the centroids they assign to each page a set of classes (topics from the taxonomy). To obtain the final set of relevant ads for a particular Web page, they combine the semantic information (*taxonomy score* or $TaxScore$) obtained from the classification with the syntactic characteristics of them (*keyword score* or $KeywordScore$), using a convex combination of both scores:

$$Score(p_i, a_i) = \alpha \times TaxScore(Tax(p_i), Tax(a_i)) + (1 - \alpha) \times KeywordScore(p_i, a_i)$$

where p_i and a_i correspond to the Web page and the ad, respectively, and $Tax(x)$ corresponds to the set of classes obtained from the taxonomy for the element x (a Web page or an ad). The $TaxScore$ should reflect the semantic distance between the ad and Web page by giving high score when both elements share the same nodes, or share a common ancestor. The $KeywordScore$ is obtained by representing the Web page and the ad in an n-dimensional space (each dimension corresponds to a term) and calculating the cosine similarity between the vectors. This semantic-syntactic approach was compared to a pure syntactic approach, and analyzed which values of α gave better results. From their observations, it is possible to infer that the semantic information is useful in the matching process since purely syntactic matches depend on the quality of the pages.

11.10.2 Web Mining

One fundamental difference between data mining and IR is that data mining must discover information in the absence of a clear query or information need. Unlike some, we take the stand that Web mining goes beyond IR, precisely for this reason. Web mining is typically conducted in three sequential stages, namely data recollection, information extraction, and analysis, aimed at mining three basic types of data, namely content, usage, and structure, as follows:

1. Content data consists of text and multimedia.

2. Structure data includes the link structure of the Web (and possibly XML structure at a finer level).

3. Usage data includes Web logs, clickthrough data, and usage access patterns.

In addition, an orthogonal temporal dimension must be considered that reflects the dynamics of the Web growth and evolution. Thus content, usage and structure are augmented by temporal data. The first and third types are covered in [419], while the second is the main topic of [349]. Another generic data mining book is [1037].

Content mining can be subdivided in text mining and multimedia mining. Text mining is a classic area that goes beyond the scope of this book [552, 1706], being opinion mining one of the trendy problems [1240]. Multimedia mining is newer and has been recently combined with other contexts such as geography and diversity [1778].

Link mining is intrinsic to the Web, so we give a few more examples below. The ParaSite system [1512] used hyperlink information to find pages that have moved, related pages, and personal Web pages. HITS has also been used to find communities and similar pages [625, 911]. Other examples of exploiting hyperlink structure can be found in [352, 1086, 1273]. Further improvements in this area include Web document clustering [269, 361, 1679] (already mentioned), connectivity services (for example, asking which Web pages point to a given page [197]), automatic link generation [671] and information extraction [226, 261]. Some of the results mentioned for Web spam represent a particular case of link mining (see section 11.5.7).

Web usage mining is one of the best examples of what is today called "wisdom of crowds"[1546]. Web usage mining can be used for adaptive Web design (for example, user-driven Web design), Web site reorganization, Web site personalization, and several performance improvements. One important class of Web usage mining related to search is *query mining*, which we cover in more detail next, as is intrinsically related to search engines.

Query Mining

The simplest kind of query mining is directly related to search usage and is called *search analytics*. This is the study of the searches related to a given Web site. They include external searches coming from search engines and internal searches done in the search box provided by the Web site. The first case allows to distinguish pages that are only found by searching from the ones found by navigating the site [122]. A careful analysis of the searches leads to the identification of new and better words to improve anchor text, as well as the Web site organization. The property of a word being better than another to satisfy an information need has been coined *information scent* by Pirolli [1268]. Internal searches provide information about needs that are not well satisfied in the site, including non-clicked results or null answers. These provide insight and new keywords related to a given need that the site owner might not have been aware of. Even more important, new keywords signal the needs for new content, services or products missing at given sites [122] and might bring great insights for future site developments. In fact, queries can describe documents better than their content [1282]. Other applications of queries to improve Web sites are covered in [1383].

Another main class of applications, is to use query mining to improve search engines. We already covered many examples of query mining in section 7.2, in particular

regarding intention, topic and ambiguity prediction. Another example is query recommendation as covered in section 11.7.2 and the query-based caching and indexing techniques that we presented in section 11.4.2. Baeza-Yates *et al.* [88, 89] and recently Silvestri [1481] cover most of the applications of mining query logs to improve search engines.

Yet another application of query logs is for extracting semantic relations. By analyzing the behavior of the users when submitting different queries (*i.e.*, the click-through data), it is possible to infer the semantics of the queries, and find semantic similarities. Baeza-Yates *et al.* [125] analyze a click graph and obtain interesting semantic relations. First, for each query on the log, the set of URLs clicked as a result for the query is obtained. This set of URLs is referred to as *URL cover* (UC_q) or *click-graph*. Each query is represented as nodes in a n-dimensional space where each dimension is a unique URL, and each component of a query is assigned a weight representing the click-frequency of the URL. Afterwards, nodes (queries) in the graph are connected if they share a common URL, and the edges are weighted using the cosine similarity between the queries. Finally, each node is assigned a *weighted degree*, as the sum of the weights of all its edges divided by the degree of the node. Using this type of graph, they define three types of relations between queries:

- identical cover ($UC_{q1} = UC_{q2}$): undirected edge representing the fact that both queries cover the same URLs, and then we define them as equivalent queries.
- strict complete cover ($UC_{q1} \subset UC_{q2}$): directed edge from q_1 to q_2, representing the semantic fact that q_1 is more specific than q_2.
- partial cover ($UC_{q1} \cap UC_{q2} \neq \emptyset$ and does not fulfill any of the previous conditions): this is the most typical case and represents a partial similarity among the queries.

Using this representation of the queries and their cover graph, it is possible to extract semantic information. There exists URLs that contain multi-topical contents, and affect negatively the identification of interesting semantic relations between queries. They observe that edges with low weight are likely to represent poor quality semantic relations, implying that the URLs related to those edges are possible candidates of being multi-topical pages. This observation is used to remove these URLs from the graph in order to reduce the noise they caused. As a result of their analysis, it is possible to observe relations between queries that could not have been identified via a purely linguistic approach, as this technique is language independent. For example, some typos that recurrently occur lead to the construction of a *Web-slang* that can only be identified from the analysis of usage. An interesting additional research direction consists of building a hierarchical folksonomy from the queries [584].

11.10.3 Metasearch

Metasearchers are Web servers that send a given query to several search engines, Web directories and other databases, collect the answers and combine them in a single ranked list. They can be seen as a type of federated search where the federated sources are independent search engines (see section 10.7). Metasearch was popular in the early days of the Web when search engines had very little overlap. Old examples are

Metacrawler [1448], SavvySearch [511] and Vivisimo [1643]. Most metasearch engines have disappeared or evolved (like Vivisimo that turned to enterprise search), except for a few remaining players like Clusty,[20] which is powered by Vivisimo, Dogpile[21] and Mamma,[22] which calls itself the "mother of all search engines".

Metasearchers differ from each other in how ranking is performed (if performed) in the combined list of results, and how well they translate the user's query into the specific query language of each search engine or Web directory (the query language common to all of them could be small).

The advantages of metasearchers is that the results can be sorted by different attributes such as host, keyword and date, which can be more informative that the output of a single search engine. Therefore, browsing the results should be simpler. On the other hand, the result does not necessarily cover the most relevant pages, as the number of results per search engine is limited and poor results from one source might be promoted at the detriment of better results generated by a second search engine., a classic federated search limitation when working in one pass. Nevertheless, the intuition is that by diversifying sources, more relevant results are susceptible to be retrieved.

One of first steps in this direction was taken by the NEC Research Institute metasearch engine, Inquirus [987, 986]. Inquirus downloaded and analyzed each Web page obtained from different sources, and then displayed each page (highlighting all query terms) in a progressive manner, as soon as they became available.

The use of metasearchers was justified by early coverage studies that showed that a small percentage of Web pages were in all search engines [198]. In fact, this first study showed that less than 1% of the Web pages indexed by AltaVista, HotBot, Excite and Infoseek are in all of those search engines. Recent studies show a larger overlap [136, 1518], which might partially explain the slow decrease of metasearchers popularity. Metasearch in the context of enterprise search is covered in section 15.3.8. Metasearchers for specific topics can be considered as dynamic search software agents and are covered in section 11.9.4.

One of the main criticisms to metasearchers is that they piggy-back on top of search engines, like a parasite, without investing in expensive computer infrastructure. For this reason, large search engines may limit the number of queries per day they will accept from metasearchers.

11.11 Trends and Research Issues

The future of the Web might surprise us, considering that its massive use started less than five years ago. There are many distinct trends and each one opens up new and particular research problems. What follows is a quick review of the upcoming sources of data that should become more and more available, as well as a compilation of the major trends and challenges to be addressed for better Web retrieval.

[20] http://www.clusty.com
[21] http://www.dogpile.com
[22] http://www.mamma.com

11.11.1 Beyond Static Text Data

We consider here the more challenging types of data that keep appearing, namely hidden or dynamic pages, multimedia data, and semantic data.

Dynamic Data

The static Web has become small compared to content generated on demand, in particular when querying e-business or information services sites. Current crawling software can follow dynamic links, but that has to be done with care, as there might be no limits on the number of pages dynamically generated.

Accessing pages behind query forms is even more difficult, as the crawler does not have *a priori* knowledge of the database. Pages of this type compose what is called the deep Web. On the other hand, even if the database is known, asking all possible queries might be too time consuming (exponential on the size of the database) and even if we stick to simple queries, some of them might be never posed by real persons. Web services might be a partial solution to this problem if they allow learning from the database, particularly learning how people query it. For example, obtaining the most frequent one thousand queries could be enough.

Multimedia Data

Multimedia data includes images, animations, audio in several forms, and video. None of them have universally agreed standard formats. Dominant ones are JPG, GIF and PNG for images, MP3 for music, Real Video or Quicktime for video, etc. The ideal solution is to search any kind of data, including text, using the same model and with a single query language. This ambitious goal is probably not attainable.

For a particular data type we can develop a similarity model, and depending on the type, the query language will change. Examples include query by example for images or query by humming (or by recording in the Shazam model[23]) for audio. All this area belongs more to image and signal processing, rather than to classic IR.

Searching for non-textual objects will keep gaining importance in the near future. There are already many research results which are mentioned in Chapter 14.

Semantic Data

The two main problems with semantic information are standards for metadata that describe the semantics and the quality or degree of trust for a given information source. The first is being carried out by the W3 Consortium, while the second require certification schemes that have not been developed yet.

Other problems are common issues, such as scaling, rate of change, lack of referential integrity (links are physical, not logical), distributed authority, heterogeneous content and quality, and multiple sources. One of the main efforts today is the Open Linking Data [1231] effort, which tries to add and improve links among semantic resources already available on the Web.

[23]Shazam is a cool mobile phone application available on iphones and Android that recognizes songs over 3G connection, once fed with a few second-long recording conducted on the mobile phone itself.

Semantic search engines represent a more recent development. These engines search semantic Web data. The most interesting representative of this class of engines is Sindice [1483], that allows to search over dozens of millions of RDF files that may contain billions of triples. A more pragmatic approach to semantic search is discussed in [101]. A completely different approach has been taken by the Wolfram Alpha[24] search engine, where a knowledge database of facts is used to infer the answer to the query. For this reason is called an answer engine.

11.11.2 Current Challenges

Web retrieval is a fast moving research and development area and we list below a few existing challenges that require more efforts.

- Distributed architectures. New distributed schemes to traverse and search the Web must be devised to cope with its growth. This will have an impact on current crawling and indexing techniques, as well as on caching techniques for the Web. It will be interesting to find out which will be the bottleneck in the future, server capacity or network bandwidth?

- Modeling. More IR models tailored for the Web are still needed. In addition, search is still mostly dominated by the pull paradigm, where users proactively initiate search, yet the push paradigm is still to be further explored for better user experience. In both cases, we need better search paradigms and better information filtering.

- Querying. Further work on combining structure and content in the queries is needed as well as new visual metaphors to pose those queries and visualize answers [118]. Future query languages may include concept-based search and natural language processing, as well as searching by example (this implies document clustering and categorization on the Web. Another critical issue is identifying the need behind the query: informational, navigational, or transactional, as well as more refined intents. It is estimated that less than 50% of the queries are of the first kind, which was the classic case. One alternative is adding to the query language the context of the information needed, such as genre or time. Extensively studying query logs is thus required in order to better understand users' behavior.

- Ranking. Better ranking schemes are needed, exploiting both content, and structure (internal to a page and hyperlinks). In particular, it would be interesting to combine and compare query-dependent and independent techniques. One problem related to advertisement is that search engines may rank higher some pages due to reasons that are not based on the actual relevance of a page (this is called the search engine persuasion problem in [1086]). This are also covers better handling of Web spam and identifying, in general, content of good quality. The Web is full of low quality (syntactically and semantically) content, including noisy, unreliable and contradictory data, without even mentioning trust issues regarding suspicious (malicious or not) sites. Another challenge is

[24] http://www.wolframalpha.com

adapting ranking to specific people or groups of people, that is, personalization around individual or intents. Relevance is based in personal judgements, so ranking based in user profiles or other user-based context information can help.

- Indexing. In spite of the long record of research and innovation in this field, many additional questions need to be addressed. Examples include: Which is the best logical view for the text? What should be indexed? How to exploit better text compression schemes to achieve fast searching and get lower network traffic? How to compress efficiently word lists, URL tables, and update them without significant run-time penalty? How to maintain the index fresh? Many implementation details must be addressed and improved.

- Elimination of duplicate data. Better mechanisms to detect and eliminate repeated Web pages (or pages that are syntactically very similar) are needed. Initial approaches are based on resemblance measures using document fingerprints [267, 269], as we saw in section 11.6.4. This is related to the important problem in databases of finding similar objects. A variation of this problem deals with the content generated from other Web pages found by using search engines, hence possibly biased by their ranking [120]. We also cover this topic in Chapter 12.

- User interaction. The front-end of Web search engines, both in the rectangle and SERP paradigms are basically the front-line of the search engines war, and will be decisive in retaining or stealing users from the competition. Indeed, relevance is becoming more and more difficult to evaluate by end users, and their perception is strongly influenced by their user experience on the page itself. The danger as usual is to clutter the page and fall into the "more is less" trap, yet this area is likely to continue evolving at a very fast pace. Additional areas to explore include better extraction of the main content of a page or the formulation of content-based queries [1593]. Another challenge is to better exploit users' feedback, either from explicit user evaluation or implicitly from Web logs.

- Browsing. This is an area that deserves to be revisited, exploiting links, popularity of Web pages, content similarity, collaboration, 3D, and virtual reality, with special attention to hybrid searching/browsing approaches.

- Adapting to small corpora. This is a sensitive area especially for enterprises that do not want to open their Intranet to the world, as well as emerging countries with still little Web presence. Due to the limited size of the corpus, not enough content/usage and link data is available to make the wisdom of crowds approach work. This leads to the paradox that it is easier to find information on the huge Internet than in small companies domains.

- Content delivery as search. Search can be thought as a very special case of the more general problem of delivering the content that the user wants at a given time. In the case of search, this action is user driven. On the other hand, can be also context driven (*e.g.*, for example the information supply paradigm of Broder [272]). Hence, how we can design the content of a requested page to

match the current user-based in the whole context? (that includes the user itself, history, location, etc.)

- Query log privacy. The use of query logs is very important to improve search engines. However, after the AOL incident in 2006[25] it is clear that it is possibly to identify some users from queries in the long tail of the distribution [141]. Even if we cannot identify a user, information like age, gender or income also compromises privacy. For that reason some users prefer not to be tracked by search engines and for example they delete cookies after every session [780]. Due to this trend many researchers have tried to find ways to share query logs with researchers maintaining full anonymity. However, for very infrequent queries, that does not seem to be possible. For more details see the excellent survey by Cooper [420] and also further references at the end of the chapter.

- Social networks. Our understanding on how social networks evolve is still little. Also, how social networks can be leveraged for other purposes, such as finding experts or communities, route important messages, etc., needs more research. Another important aspect is the relation of social networks and the Semantic Web [1128].

11.12 Bibliographical Discussion

There are hundreds of books about the Web. Many of them include some information about searching the Web and tips for users. An early one was edited by Abrams and includes a chapter on searching the Web [8]. Other early sources are the special numbers of Scientific American on Internet (March 1997) and IEEE Internet Computing on Search Technologies (July/August 1998).

For more information on modeling the Web we recommend the book by Baldi *et al.* [130].

Recent books that cover Web retrieval include the "Web Dragons" of Witten *et al.* [1708], the multidisciplinary collection edited by Spink and Zimmer [1522] and the search engines book of Croft *et al.* [449]. More information on faceted search can be found in [1607] and on collaborative Web search in [1156].

Early surveys about Web retrieval include [64, 268, 348, 745, 747, 1586]. A good survey in link analysis is [746]. The topic of information scent is expanded in [369, 370, 1269]. More in power laws and scale-free networks can be found in [907, 1142, 1199, 1341]. Regarding the impact of the long tail on the Web, see the book by Anderson [50].

The most recent comparison on caching answers is due to Gan *et al.* [617].

A lot of research has been done in computing PageRank efficiently as well as its properties, perhaps more than what is really needed. The main results on this area are [19, 200, 220, 221, 636, 718, 870, 869, 992, 1111]. For more details we refer the reader to the book of Langville and Meyer [976] and Berkhin's survey [189].

More information on link analysis can be found in the book by Thelwall [1575] and the survey by Henzinger [746]. Other algorithms that exploit links for ranking include [238, 239, 311, 576, 943, 1200].

[25] See http://en.wikipedia.org/wiki/AOL_search_data_scandal.

A nice example of how to optimize a simple ranking function is due to Singhal *et al.* [1484].

In recent years, several learning to rank techniques have been proposed. In the pointwise approach we have the discriminative model for IR [1170] and MCRank [1019]. In the pairwise case we can mention RankBoost [591], Ranking SVM [748], RankNet [296], IR-SVM [330], FRank [1603], MHR [1308] and QBRank [1782]. In the case of the listwise technique we have LamdaRank [295, 507], ListNet [331], RankCosine [1307], SVM-MAP [1759], AdaRank [1733], and SoftRank [1566, 684]. Fen *et al.* studied the loss functions of the later case [1726]. A recent survey is due to Lui [1064].

Further work on learning to rank from clickthrough data can be found in [846, 1322, 1323]. Learning has also be used for other problems, like clustering Web search results [1768], predicting clicks [18, 1280] or for on-line ranking [1632]. In general, "learning to rank" has become an important research problem in IR, even if solutions to the problem are still partial. Despite these challenges, this area is promising research wise due to its ability for incorporating implicit and explicit users' signals so as to improve the quality of the results.

Many researchers have looked at using resemblance given some threshold to find duplicate documents, in particular de work of Garcia-Molina and collaborators in the context of digital libraries [618, 1465, 1466, 1467]. Other people have looked at overlaps among documents [565, 1154, 1155]. Improvements of I-Match are discussed in [924, 925].

For further research in computational advertising see [275, 1319, 1350]. More results can be found in the Monetization track of the WWW conference.

Recently, a stream of papers about query log privacy have been published, in particular to explore anonymization techniques [11, 848, 849, 929, 946, 1728, 1781]. Bar-Yossef and Gurovich have shown how to estimate the frequency of queries and impressions of Web pages using query suggestion tools [137, 138]. The problem of query log privacy has also been extended to business privacy in the context of Web sites [1283]. Privacy is also important in the context of social networks [1781].

In addition, the best source for references to the Web is the Web itself. To start, there are many Web sites devoted to inform and rate search engines and Web directories. Among them we can distinguish Search Engine Watch [1543] and Search Engine Showdown [1209]. A good directory to Web characteristics is [503]. Other sources that provide pointers and references related to searching the Web, are the World Wide Web Consortium, or W3C, (www.w3c.org), the World Wide Web journal (w3j.com) and the WWW conference series (http://www.iw3c2.org/). These and other pointers are available on the Web page of this book (see Chapter 1).