15 Web Search

Apart from an occasional example or exercise, the preceding chapters deal with information retrieval in a generic context. We assume the IR system contains a collection of documents, with each document represented by a sequence of tokens. Markup may indicate titles, authors, and other structural elements. We assume nothing further about the IR system's operating environment or the documents it contains.

In this chapter we consider IR in the specific context of Web search, the context in which it may be most familiar to you. Assuming this specific context provides us with the benefit of document features that cannot be assumed in the generic context. One of the most important of these features is the structure supplied by hyperlinks. These links from one page to another, often labeled by an image or anchor text, provide us with valuable information concerning both the individual pages and the relationship between them.

Along with these benefits come various problems, primarily associated with the relative "quality", "authority" or "popularity" of Web pages and sites, which can range from the carefully edited pages of a major international news agency to the personal pages of a high school student. In addition, many Web pages are actually *spam* — malicious pages deliberately posing as something that they are not in order to attract unwarranted attention of a commercial or other nature. Although the owners of most Web sites wish to enjoy a high ranking from the major commercial search engines, and may take whatever steps are available to maximize their ranking, the creators of spam pages are in an adversarial relationship with the search engine's operators. The creators of these pages may actively attempt to subvert the features used for ranking, by presenting a false impression of content and quality.

Other problems derive from the scale of the Web — billions of pages scattered among millions of hosts. In order to index these pages, they must be gathered from across the Web by a *crawler* and stored locally by the search engine for processing. Because many pages may change daily or hourly, this snapshot of the Web must be refreshed on a regular basis. While gathering data the crawler may detect duplicates and near-duplicates of pages, which must be dealt with appropriately. For example, the standard documentation for the Java programming language may be found on many Web sites, but in response to a query such as ⟨"java", "vector", "class"⟩ it might be best for a search engine to return only the official version on the `java.sun.com` site.

Another consideration is the volume and variety of queries commercial Web search engines receive, which direct reflect the volume and variety of information on the Web itself. Queries are often short — one or two terms — and the search engine may know little or nothing about the user entering a query or the context of the user's search. A user entering the query ⟨"UPS"⟩ may be interested in tracking a package sent by the courier service, purchasing a universal power supply, or attending night classes at the University of Puget Sound. Although such query ambiguity is a consideration in all IR applications, it reaches an extreme level in Web IR.

15.1 The Structure of the Web

Figure 15.1 provides an example of the most important features related to the structure of the Web. It shows Web pages on three sites: W, M, and H.[1] Site W provides a general encyclopedia including pages on Shakespeare (`w0.html`) and two of his plays, *Hamlet* (`w1.html`) and *Macbeth* (`w2.html`). Site H provides historical information including information on Shakespeare's wife (`h0.html`) and son (`h1.html`). Site M provides movie and TV information including a page on the 1971 movie version of *Macbeth* directed by Roman Polanski (`m0.html`).

The figure illustrates the link structure existing between these pages and sites. For example, the HTML *anchor* on page `w0.html`

```
<a href="http://H/h0.html">Anne Hathaway</a>
```

establishes a link between that page and `h0.html` on site H. The anchor text associated with this link ("Anne Hathaway") provides an indication of the content on the target page.

15.1.1 The Web Graph

The relationship between sites and pages indicated by these hyperlinks gives rise to what is called a *Web graph*. When it is viewed as a purely mathematical object, each page forms a node in this graph and each hyperlink forms a directed edge from one node to another. Figure 15.2 shows the Web graph corresponding to Figure 15.1. For convenience we have simplified the labels on the pages, with `http://W/w0.html` becoming w_0, for example.

From a practical standpoint it is important to remember that when we refer to a Web graph we are referring to more than just this mathematical abstraction — that pages are grouped into sites, and that anchor text and images may be associated with each link. Links may reference a labeled position within a page as well as the page as a whole. The highly dynamic and fluid nature of the Web must also be remembered. Pages and links are continuously added and removed at sites around the world; it is never possible to capture more than a rough approximation of the entire Web graph.

At times our interest may be restricted to a subset of the Web, perhaps to all the pages from a single site or organization. Under these circumstances, capturing an accurate snapshot of the Web graph for that site or organization is easier than for the entire Web. Nonetheless, most Web sites of any size grow and change on a continuous basis, and must be regularly recrawled for the snapshot to remain accurate.

[1] For simplicity we use single-letter site names and simplified page names rather than actual site and page names — for instance, `http://W/w0.html` instead of a full URL such as `http://en.wikipedia.org/wiki/William_Shakespeare`.

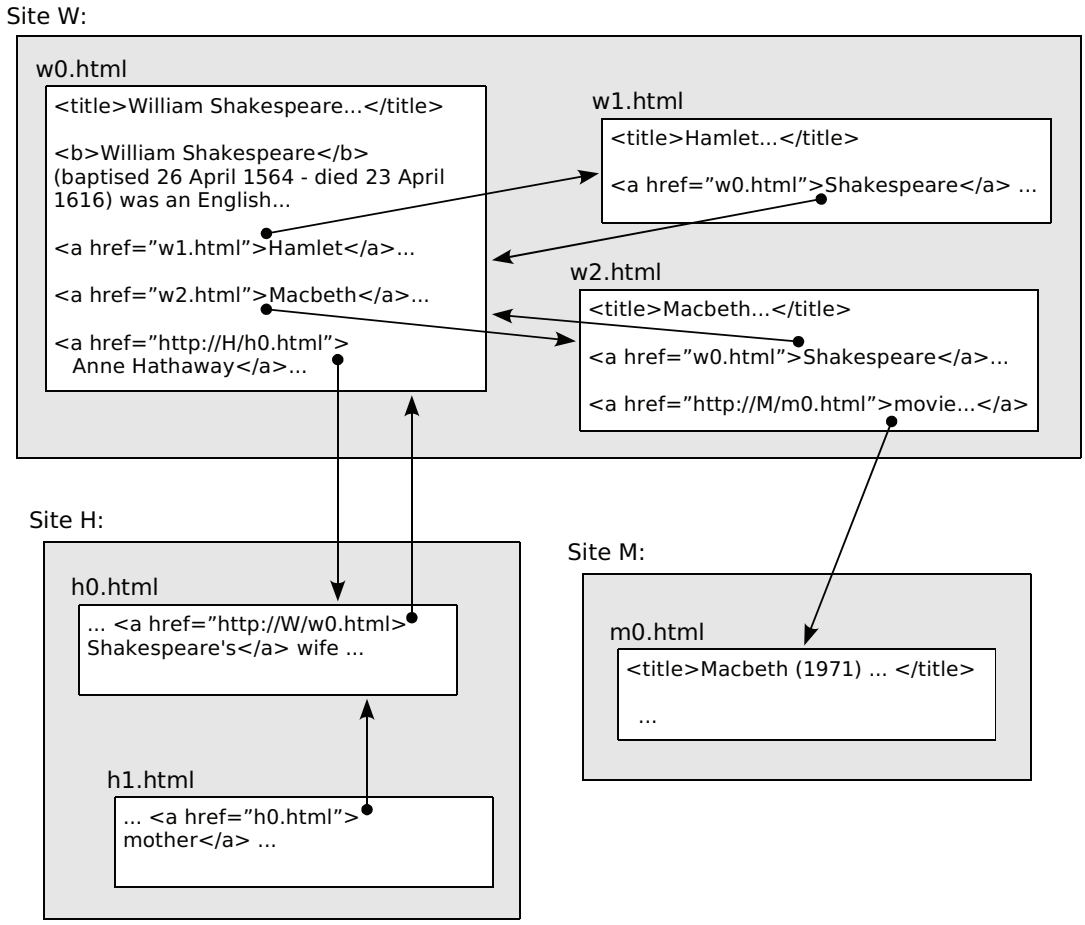


Figure 15.1 Structure on the Web. Pages can link to each other (`<a href=...>`) and can associate each link with anchor text (e.g., "mother").

We now introduce a formal notation for Web graphs that we use in later sections. Let Φ be the set of all pages in a Web graph, let $N = |\Phi|$ be the number of pages, and E the number of links (or edges) in the graph. Given a page $\alpha \in \Phi$, we define *out(α)* as the number of *out-links* from α to other pages (the *out-degree*). Similarly, we define *in(α)* as the number of *in-links* from other pages to α (the *in-degree*). In the Web graph of Figure 15.2, *out(w_0)* $= 3$ and *in(h_0)* $= 2$. If *in(α)* $= 0$, then α is called a *source*; if *out(α)* $= 0$, it is called a *sink*. We define Γ as the set of sinks. In the Web graph of Figure 15.2, page m_0 is a sink and page h_1 is a source.

The individual Web pages themselves may be highly complex and structured objects. They often include menus, images, and advertising. Scripts may be used to generate content when the page is first loaded and to update the content as the user interacts with it. A page may

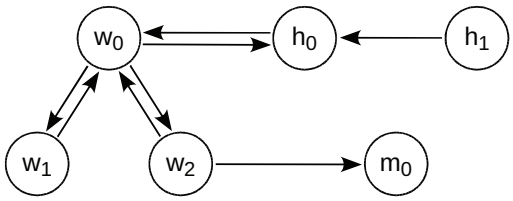


Figure 15.2 A Web graph for the Web pages shown in Figure 15.1. Each link corresponds to a directed edge in the graph.

serve as a frame to hold other pages or may simply redirect the browser to another page. These redirections may be implemented through scripts or through special HTTP responses, and may be repeated multiple times, thus adding further complexity. Along with HTML pages the Web includes pages in PDF, Microsoft Word, and many other formats.

15.1.2 Static and Dynamic Pages

It is not uncommon to hear Web pages described as "static" or "dynamic". The HTML for a static Web page is assumed to be generated in advance of any request, placed on disk, and transferred to the browser or Web crawler on demand. The home page of an organization is a typical example of a static page. A dynamic Web page is assumed to be generated at the time the request is made, with the contents of the page partially determined by the details of the request. A search engine result page (SERP) is a typical example of a dynamic Web page for which the user's query itself helps to determine the content.

There is often the implication in this dichotomy that static Web pages are more important for crawling and indexing than dynamic Web pages, and it is certainly the case that many types of dynamic Web pages are not suitable for crawling and indexing. For example, on-line calendar systems, which maintain appointments and schedules for people and events, will often serve up dynamically generated pages for dates far into the past and the future. You can reach the page for any date if you follow the links far enough. Although the flexibility to book appointments 25 years from now is appropriate for an online calendar; indexing an endless series of empty months is not appropriate for a general Web search engine.

Nonetheless, although many dynamic Web pages should not be crawled and indexed, many should. For example, catalog pages on retail sites are often generated dynamically by accessing current products and prices in a relational database. The result is then formatted into HTML and wrapped with menus and other fixed information for display in a browser. To meet the needs of a consumer searching for a product, a search engine must crawl and index these pages.

A dynamic page can sometimes be identified by features of its URL. For example, servers accessed through the Common Gateway Interface (CGI) may contain the path element `cgi-bin` in their URLs. Pages dynamically generated by Microsoft's Active Server Pages technology include the extension `.asp` or `.aspx` in their URLs. Unfortunately, these URL features are not

always present. In principle any Web page can be static or dynamic, and there is no sure way to tell. For crawling and indexing, it is the content that is important, not the static or dynamic nature of the page.

15.1.3 The Hidden Web

Many pages are part of the so-called "hidden" or "invisible" or "deep" Web. This hidden Web includes pages that have no links referencing them, those that are protected by passwords, and those that are available only by querying a digital library or database. Although these pages can contain valuable content, they are difficult or impossible for a Web crawler to locate.

Pages in *intranets* represent a special case of the hidden Web. Pages in a given intranet are accessible only within a corporation or similar entity. An enterprise search engine that indexes an intranet may incorporate many of the same retrieval techniques that are found in general Web search engines, but may be tuned to exploit specific features of that intranet.

15.1.4 The Size of the Web

Even if we exclude the hidden Web, it is difficult to compute a meaningful estimate for the size of the Web. Adding or removing a single host can change the number of accessible pages by an arbitrary amount, depending on the contents of that host. Some hosts contain millions of pages with valuable content; others contain millions of pages with little or no useful content (see Exercise 15.14).

However, many Web pages would rarely, if ever, appear near the top of search results. Excluding those pages from a search engine would have little impact. Thus, we may informally define what is called the *indexable Web* as being those pages that should be considered for inclusion in a general-purpose Web search engine (Gulli and Signorini, 2005). This indexable Web would comprise all those pages that could have a substantial impact on search results.

If we may assume that any page included in the index of a major search engine forms part of the indexable Web, a lower bound for the size of the indexable Web may be determined from the combined coverage of the major search engines. More specifically, if the sets $A_1, A_2, \ldots$ represent the sets of pages indexed by each of these search engines, a lower bound on the size of the indexable Web is the size of the union of these sets $|\cup A_i|$.

Unfortunately, it is difficult to explicitly compute this union. Major search engines do not publish lists of the pages they index, or even provide a count of the number of pages, although it is usually possible to check if a given URL is included in the index. Even if we know the number of pages each engine contains, the size of the union will be smaller than the sum of the sizes because there is considerable overlap between engines.

Bharat and Broder (1998) and Lawrence and Giles (1998) describe a technique for estimating the combined coverage of major search engines. First, a test set of URLs is generated. This step may be achieved by issuing a series of random queries to the engines and selecting a random URL from the results returned by each. We assume that this test set represents a uniform

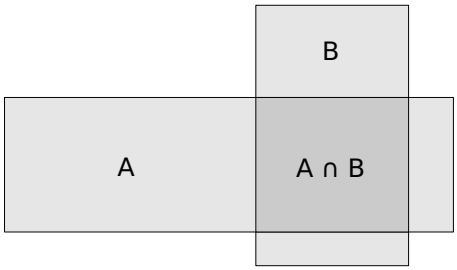


Figure 15.3 The collection overlap between search engines A and B can be used to estimate the size of the indexable Web.

sample of the pages indexed by the engines. Second, each URL from the test set is checked against each engine and the engines that contain it are recorded.

Given two engines, A and B, the relationship between their collections is illustrated by Figure 15.3. Sampling with our test set of URLs allows us to estimate $\Pr[A \cap B \,|\, A]$, the probability that a URL is contained in the intersection if it is contained in A:

$$\Pr[A \cap B \,|\, A] \;=\; \frac{\#\text{ of test URLs contained in both } A \text{ and } B}{\#\text{ of test URLs contained in } A}\,. \tag{15.1}$$

We may estimate $\Pr[A \cap B \,|\, B]$ in a similar fashion. If we know the size of A, we may then estimate the size of the intersection as

$$|A \cap B| \;=\; |A| \cdot \Pr[A \cap B \,|\, A]\,. \tag{15.2}$$

and the size of B as

$$|B| \;=\; \frac{|A \cap B|}{\Pr[A \cap B \,|\, B]}\,. \tag{15.3}$$

Thus, the size of the union may be estimated as

$$|A \cup B| \;=\; |A| + |B| - |A \cap B|\,. \tag{15.4}$$

If sizes for both A and B are available, the size of the intersection may be estimated from both sets, and the average of these estimates may be used to estimate the size of the union

$$|A \cup B| \;=\; |A| + |B| - \frac{1}{2}\left(|A| \cdot \Pr[A \cap B \,|\, A] + |B| \cdot \Pr[A \cap B \,|\, B]\right)\,. \tag{15.5}$$

This technique may be extended to multiple engines. Using a variant of this method, Bharat and Broder (1998) estimated the size of the indexable Web in mid-1997 at 160 million pages. In a study conducted at roughly the same time, Lawrence and Giles (1998) estimated the size at 320 million pages. Approximately a year later the size had grown to 800 million pages (Lawrence and Giles, 1999). By mid-2005 it was 11.5 billion (Gulli and Signorini, 2005).

15.2 Queries and Users

Web queries are short. Several studies of query logs from major search engines are summarized by Spink and Jansen (2004). Although the exact numbers differ from study to study, these studies consistently reveal that many queries are just one or two terms long, with a mean query length between two and three terms. The topics of these queries range across the full breadth of human interests, with sex, health, commerce, and entertainment representing some of the major themes.

Perhaps it is not surprising that Web queries are short. Until quite recently, the query processing strategies of Web search engines actually discouraged longer queries. As we described in Section 2.3, these processing strategies filter the collection against a Boolean conjunction of the query terms prior to ranking. For a page to be included in the result set, all query terms must be associated with it in some way, either by appearing on the page itself or by appearing in the anchor text of links referencing it. As a result, increasing the length of the query by adding related terms could cause relevant pages to be excluded if they are missing one or more of the added terms. This strict filtering has been relaxed in recent years. For example, synonyms may be accepted in place of exact matches — the term "howto" might be accepted in place of the query term "FAQ". Nonetheless, for efficiency reasons filtering still plays a role in query processing, and Web search engines may still perform poorly on longer queries.

The distribution of Web queries follows Zipf's law (see Figure 15.4). In a representative log of 10 million queries, the single most frequent query may represent more than 1% of the total, but nearly half of the queries will occur only once. The tuning and evaluation of a search engine must consider this "long tail" of Zipf's law. In the aggregate, infrequent queries are at least as important as the frequent ones.

15.2.1 User Intent

Several researchers have examined collections of queries issued to Web search engines in an attempt to characterize the user intent underlying these queries. Broder (2002) surveyed users of the Altavista search engine and examined its query logs to develop a taxonomy of Web search. He classified Web queries into three categories reflecting users' apparent intent, as follows:

- The intent behind a *navigational* query is to locate a specific page or site on the Web. For example, a user intending to locate the home page of the CNN news network might enter the query ⟨"CNN"⟩. A navigational query usually has a single correct result. However, this correct result may vary from user to user. A Spanish-speaking user from the United States might want the CNN en Español home page (`www.cnn.com/espanol`); a user from Tunisia may want the Arabic edition (`arabic.cnn.com`).

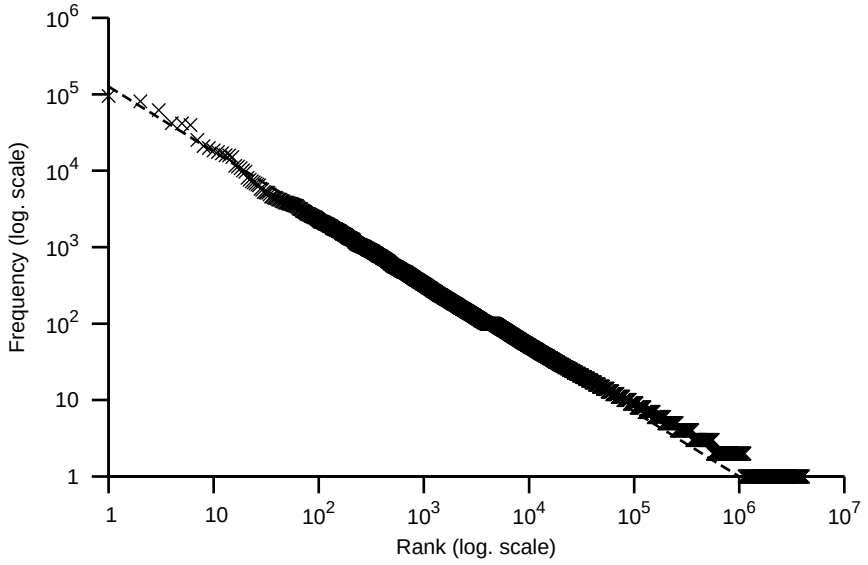


Figure 15.4 Frequency of queries by rank order, based on 10 million queries taken from the logs of a commercial search engine. The dashed line corresponds to Zipf's law with $\alpha = 0.85$.

- A user issuing an *informational* query is interested in learning something about a particular topic and has a lesser regard for the source of the information, provided it is reliable. The topics forming the test collections introduced in Chapter 1 and used for evaluation throughout the first four parts of the book reflect informational intent. A user entering the query ⟨"president", "obama"⟩ might find the information she is seeking on the CNN Web site, on Wikipedia, or elsewhere. Perhaps she will browse and read a combination of these before finding all the information she requires. The need behind an informational query such as this one may vary from user to user, and may be broad or narrow in scope. A user may be seeking a detailed description of government policy, a short biography, or just a birth date.
- A user issuing a *transactional* query intends to interact with a Web site once she finds it. This interaction may involve activities such as playing games, purchasing items, booking travel, or downloading images, music, and videos. This category may also include queries seeking services such as maps and weather, provided the user is not searching for a specific site providing that service.

Rose and Levinson (2004) extended the work of Broder by expanding his three categories into a hierarchy of user goals. They retained Broder's categories at the top level of their hierarchy but renamed the transactional category as the "resource" category. Under both the informational and the transactional categories they identified a number of subcategories. For example, the goal

of a user issuing a *directed informational* query is the answer to a particular question ("When was President Obama born?"), whereas the goal of a user issuing an *undirected informational* query is simply to learn about the topic ("Tell me about President Obama."). In turn, directed informational queries may be classified as being *open* or *closed*, depending on whether the question is open-ended or has a specific answer.

The distinction between navigational and informational queries is relatively straightforward: Is the user seeking a specific site or not? The distinction between transactional queries and the other two categories is not always as clear. We can assume the query ⟨"mapquest"⟩ is navigational, seeking the site `www.mapquest.com`, but it is likely that the user will then interact with the site to obtain directions and maps. A user entering the query ⟨"travel", "washington"⟩ may be seeking both tourist information about Washington, D.C., and planning to book a hotel, making the intent both informational and transactional. In cases such as these, it may be reasonable to view such queries as falling under a combination of categories, as navigational/transactional and informational/transactional, respectively.

According to Broder (2002), navigational queries comprise 20–25% of all Web queries. Of the remainder, transactional queries comprise at least 22%, and the rest are informational queries. According to Rose and Levinson (2004), 12–15% of queries are navigational and 24–27% are transactional. A more recent study by Jansen et al. (2007) slightly contradicts these numbers. Their results indicate that more than 80% of queries are informational, with the remaining queries split roughly equally between the navigational and transactional categories. Nonetheless, these studies show that all three categories represent a substantial fraction of Web queries and suggest that Web search engines must be explicitly aware of the differences in user intent.

Referring to "navigational queries" and "informational queries" is common jargon. This usage is understandable when the goal underlying a query is the same, or similar, regardless of the user issuing it. But for some queries the category may differ from user to user. As a result we stress that these query categories fundamentally describe the goals and intentions of the users issuing the queries, and are not inherent properties of the queries themselves. For example, a user issuing our example query ⟨"UPS"⟩ may have

- Informational intent, wanting to know how universal power supplies work
- Transactional intent, wanting to purchase an inexpensive UPS for a personal computer
- Transactional/navigational intent, wanting to track a package or
- Navigational/informational intent, wanting information on programs offered by the University of Puget Sound.

Although this query is atypical, falling into so many possible categories, the assignment of any given query to a category (by anyone other than the user) may be little more than an educated guess.

15.2.2 Clickthrough Curves

The distinction between navigational and informational queries is visible in user behavior. Lee et al. (2005) examined *clickthroughs* as one feature that may be used to infer user intent. A clickthrough is the act of clicking on a search result on a search engine result page. Clickthroughs are often logged by commercial search engines as a method for measuring performance (see Section 15.5.2).

Although almost all clicks occur on the top ten results (Joachims et al., 2005; Agichtein et al., 2006b), the pattern of clicks varies from query to query. By examining clickthroughs from a large number of users issuing the same query, Lee et al. (2005) identified clickthrough distributions that are typical of informational and navigational queries. For navigational queries the clickthroughs are skewed toward a single result; for informational queries the clickthrough distribution is flatter.

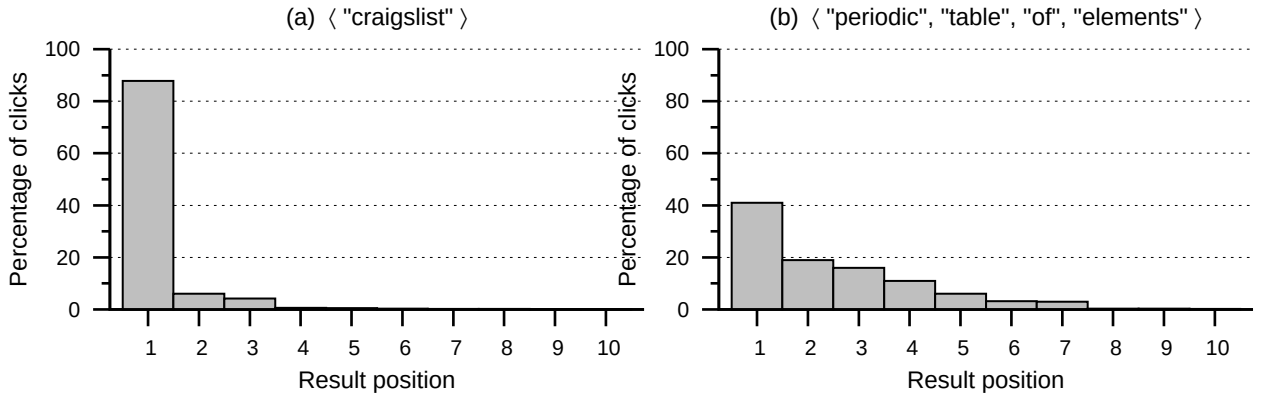


Figure 15.5 Clickthrough curve for a typical navigational query (⟨"craigslist"⟩) and a typical informational query (⟨"periodic", "table", "of", "elements"⟩).

Clarke et al. (2007) analyzed logs from a commercial search engine and provided further analysis and examples. The plots in Figure 15.5 are derived from examples in that paper. Both plots show the percentage of clickthroughs for results ranked 1 to 10. Each clickthrough represents the first result clicked by a different user entering the query. Figure 15.5(a) shows a clickthrough distribution for a stereotypical navigational query, exhibiting a spike at `www.craigslist.org`, a classified advertising site and the presumed target of the query. Figure 15.5(b) shows a clickthrough distribution for a typical informational query. For both queries the number of clickthroughs decreases with rank; the informational query receives proportionally more clicks at lower ranks.

15.3 Static Ranking

Web retrieval works in two phases. The first phase takes place during the indexing process, when each page is assigned a *static rank* (Richardson et al., 2006). Informally, this static rank may be viewed as reflecting the quality, authority, or popularity of the page. Ideally, static rank may correspond to a page's prior probability of relevance — the higher the probability, the higher the static rank.

Static rank is independent of any query. At query time the second phase of Web retrieval takes place. During this phase the static rank is combined with query-dependent features, such as term proximity and frequency, to produce a *dynamic rank*.

Assigning a static rank during indexing allows a Web search engine to consider features that would be impractical to consider at query time. The most important of these features are those derived from *link analysis* techniques. These techniques extract information encoded in the structure of the Web graph. Other features may also contribute to static rank, including features derived from the content of the pages and from user behavior (Section 15.3.5).

Our presentation of static ranking begins with the fundamentals of PageRank, easily the most famous of the link analysis techniques, in Section 15.3.1. Although this section presents the version of the algorithm generally known by the name PageRank, its practical value in Web search may be relatively limited because it depends on naïve assumptions regarding the structure of the Web. Sections 15.3.2 and 15.3.3 present and analyze an extended version of the algorithm, which accommodates a more sophisticated view of Web structure. Section 15.3.4 provides an overview of other link analysis techniques and Section 15.3.5 briefly discusses other features applicable to static ranking. Dynamic ranking is covered in Section 15.4.

15.3.1 Basic PageRank

PageRank was invented in the mid-1990s through the efforts of Larry Page and Sergey Brin, two Stanford computer science graduate students. The algorithm became a key element of their Backrub search engine, which quickly matured into Google.

The classic intuition behind the PageRank algorithm imagines a person surfing the Web at random. At any point a Web page is visible in her browser. As a next step she can either

1. Follow a link from the current page by clicking on it or
2. Select a page uniformly at random and jump to it, typing its URL into the address bar.

At any step the probability she will follow a link is fixed as δ. Thus, the probability of a jump is $1 - \delta$. Reasonable values for δ might range from 0.75 to 0.90, with 0.85 being the value quoted most often in the research literature. For simplicity we use $\delta = 3/4$ for our experiments and examples, unless otherwise stated.

Sinks in the Web graph force a jump: When our surfer reaches a sink, she always takes the second option. Assuming that our surfer could surf quickly and tirelessly for a long period of time, the value of PageRank $r(\alpha)$ for a page α would indicate the relative frequency at which the surfer visits that page.

The probability δ is sometimes referred to as the *restart probability* or *damping factor*, because it reduces the probability that the surfer will follow a link. The use of a damping factor to allow random jumps is known to improve the stability of PageRank, in a statistical sense, in comparison to an equivalent algorithm without jumps (Section 15.3.3). Informally, small changes to the Web graph will not generate large changes in PageRank.

The value of $r(\alpha)$ may be expressed in terms of the PageRank values of the pages that link to it and the PageRank values of the sinks in the graph, as follows:

$$r(\alpha) \;=\; \delta \cdot \left(\sum_{\beta \to \alpha} \frac{r(\beta)}{out(\beta)} + \sum_{\gamma \in \Gamma} \frac{r(\gamma)}{N} \right) + (1 - \delta) \cdot \sum_{\alpha \in \Phi} \frac{r(\alpha)}{N} \tag{15.6}$$

For simplicity, and because the choice is arbitrary, we assume

$$\sum_{\alpha \in \Phi} r(\alpha) \;=\; N, \tag{15.7}$$

reducing the equation slightly to

$$r(\alpha) \;=\; \delta \cdot \left(\sum_{\beta \to \alpha} \frac{r(\beta)}{out(\beta)} + \sum_{\gamma \in \Gamma} \frac{r(\gamma)}{N} \right) + (1 - \delta). \tag{15.8}$$

Because $\sum_{\alpha \in \Phi} r(\alpha)/N \;=\; 1$, the probability that the random surfer will be at page α at any given point is thus $r(\alpha)/N$.

Let us consider the composition of Equation 15.8 in detail. First, the value $(1 - \delta)$ reflects the contribution from random jumps to α. The other contributions to the PageRank value of α — from links and sinks — are more complex. In the case of links, the page α may be reached by following a link from a page β. Because β may have links to multiple pages, its PageRank value is distributed according to its out-degree $out(\beta)$. Finally, jumps from sinks contribute in a small way to the PageRank value of α. Because a jump from sink γ may target any of the N pages in the graph — even itself — its contribution is $r(\gamma)/N$.

The application of Equation 15.8 to the Web graph of 15.2 gives the following set of equations:

$$\begin{aligned} r(w_0) &= \delta \cdot \left(r(w_1) + \frac{r(w_2)}{2} + r(h_0) + \frac{r(m_0)}{6} \right) + (1 - \delta) \\ r(w_1) &= \delta \cdot \left(\frac{r(w_0)}{3} + \frac{r(m_0)}{6} \right) + (1 - \delta) \end{aligned}$$

$$
\begin{aligned}
r(w_2) &= \delta \cdot \left(\frac{r(w_0)}{3} + \frac{r(m_0)}{6} \right) + (1-\delta) \\
r(h_0) &= \delta \cdot \left(\frac{r(w_0)}{3} + r(h_1) + \frac{r(m_0)}{6} \right) + (1-\delta) \\
r(h_1) &= \delta \cdot \left(\frac{r(m_0)}{6} \right) + (1-\delta) \\
r(m_0) &= \delta \cdot \left(\frac{r(w_2)}{2} + \frac{r(m_0)}{6} \right) + (1-\delta)
\end{aligned}
$$

Setting $\delta = 3/4$ and simplifying gives

$$
\begin{aligned}
r(w_0) &= \frac{3\,r(w_1)}{4} + \frac{3\,r(w_2)}{8} + \frac{3\,r(h_0)}{4} + \frac{r(m_0)}{8} + \frac{1}{4} \\
r(w_1) &= \frac{r(w_0)}{4} + \frac{r(m_0)}{8} + \frac{1}{4} \\
r(w_2) &= \frac{r(w_0)}{4} + \frac{r(m_0)}{8} + \frac{1}{4} \\
r(h_0) &= \frac{r(w_0)}{4} + \frac{3\,r(h_1)}{4} + \frac{r(m_0)}{8} + \frac{1}{4} \\
r(h_1) &= \frac{r(m_0)}{8} + \frac{1}{4} \\
r(m_0) &= \frac{3\,r(w_2)}{8} + \frac{r(m_0)}{8} + \frac{1}{4}
\end{aligned}
$$

To compute PageRank, we solve for the six variables in the resulting system of linear equations $r(w_0)$, $r(w_1)$, $r(w_2)$, $r(h_0)$, $r(h_1)$, and $r(m_0)$. Many algorithms exist for the numerical solution of linear systems such as this one. The method particularly suited to PageRank is a form of *fixed-point iteration*.

Fixed point iteration is a general technique for solving systems of equations, linear and otherwise. To apply it, each variable must be expressed as a function of the other variables and itself. The PageRank equations are already expressed in this form, with the PageRank value for each page appearing alone on the left-hand side, and functions of these PageRank values appearing on the right-hand side.

The algorithm begins by making an initial guess for the value of each variable. These values are substituted into the right-hand equations, generating new approximations for the variables. We repeat this process, substituting the current values to generate new approximations. If the values converge, staying the same from iteration to iteration, we have solved the system of equations.

The method produces a series of approximations to $r(\alpha)$ for each page α: $r^{(0)}(\alpha)$, $r^{(1)}(\alpha)$, $r^{(2)}(\alpha), \ldots$ If we choose as our initial guess $r^{(0)}(\alpha) = 1$ for all pages, we have

$$\sum_{\alpha \in \Phi} r^{(0)}(\alpha) \;=\; N, \tag{15.9}$$

as required by Equation 15.7. From Equation 15.8 we then compute new approximations from existing approximations with the equation:

$$r^{(n+1)}(\alpha) \;=\; \delta \cdot \left(\sum_{\beta \to \alpha} \frac{r^{(n)}(\beta)}{\mathit{out}(\beta)} + \sum_{\gamma \in \Gamma} \frac{r^{(n)}(\gamma)}{N} \right) + (1 - \delta). \tag{15.10}$$

To compute PageRank for the Web graph of Figure 15.2, we iterate the following equations:

$$\begin{aligned}
r^{(n+1)}(w_0) &= \frac{3\,r^{(n)}(w_1)}{4} + \frac{3\,r^{(n)}(w_2)}{8} + \frac{3\,r^{(n)}(h_0)}{4} + \frac{r^{(n)}(m_0)}{8} + \frac{1}{4} \\
r^{(n+1)}(w_1) &= \frac{r^{(n)}(w_0)}{4} + \frac{r^{(n)}(m_0)}{8} + \frac{1}{4} \\
r^{(n+1)}(w_2) &= \frac{r^{(n)}(w_0)}{4} + \frac{r^{(n)}(m_0)}{8} + \frac{1}{4} \\
r^{(n+1)}(h_0) &= \frac{r^{(n)}(w_0)}{4} + \frac{3\,r^{(n)}(h_1)}{4} + \frac{r^{(n)}(m_0)}{8} + \frac{1}{4} \\
r^{(n+1)}(h_1) &= \frac{r^{(n)}(m_0)}{8} + \frac{1}{4} \\
r^{(n+1)}(m_0) &= \frac{3\,r^{(n)}(w_2)}{8} + \frac{r^{(n)}(m_0)}{8} + \frac{1}{4}
\end{aligned}$$

Table 15.1 shows the successive approximations to PageRank for this system of equations. After 18 iterations the values have converged to within three decimal places.

Figure 15.6 presents a detailed algorithm for computing PageRank. The algorithm assumes that pages in the Web graph are numbered consecutively from 1 to N. The array *link*, of length E, stores the links in the Web graph, with *link*[i].*from* indicating the source of the link and *link*[i].*to* indicating the destination of the link. The array R stores the current approximation to PageRank; the array R' accumulates the new approximation.

The loop over lines 1–2 sets R to the initial approximation. At the end of each iteration of the main loop (lines 3–14) the array R contains the next approximation. As written, these lines form an infinite loop. In practice this loop would be terminated when the approximation converges or after a fixed number of iterations. We omit the exact termination condition because it depends on the accuracy desired and the precision of the computations. Over most Web graphs, acceptable values can be achieved after a few hundred iterations. Each iteration of the main loop takes $O(N + E)$ time; the time complexity of the overall algorithm depends on the number of iterations of the main loop.

Table 15.1 Iterative computation of PageRank for the Web graph of Figure 15.1.

n	$r^{(n)}(w_0)$	$r^{(n)}(w_1)$	$r^{(n)}(w_2)$	$r^{(n)}(h_0)$	$r^{(n)}(h_1)$	$r^{(n)}(m_0)$
0	1.000	1.000	1.000	1.000	1.000	1.000
1	2.250	0.625	0.625	1.375	0.375	0.750
2	2.078	0.906	0.906	1.188	0.344	0.578
3	2.232	0.842	0.842	1.100	0.322	0.662
4	2.104	0.891	0.891	1.133	0.333	0.648
5	2.183	0.857	0.857	1.107	0.331	0.665
6	2.128	0.879	0.879	1.127	0.333	0.655
7	2.166	0.864	0.864	1.114	0.332	0.661
8	2.140	0.874	0.874	1.123	0.333	0.657
9	2.158	0.867	0.867	1.116	0.332	0.660
10	2.145	0.872	0.872	1.121	0.332	0.658
11	2.154	0.868	0.868	1.118	0.332	0.659
12	2.148	0.871	0.871	1.120	0.332	0.658
13	2.152	0.869	0.869	1.119	0.332	0.659
14	2.149	0.870	0.870	1.120	0.332	0.658
15	2.151	0.870	0.870	1.119	0.332	0.659
16	2.150	0.870	0.870	1.119	0.332	0.658
17	2.151	0.870	0.870	1.119	0.332	0.659
18	2.150	0.870	0.870	1.119	0.332	0.658
19	2.150	0.870	0.870	1.119	0.332	0.659
20	2.150	0.870	0.870	1.119	0.332	0.658
...	...	...	...	...	...	...

Lines 4–5 initialize R' by storing the contribution from jumps. Lines 6–9 consider each link in turn, computing the contribution of the source page to the destination's PageRank value. Because $\sum_{\alpha\in\Phi} r(\alpha) = N$, the set of sinks need not be explicitly considered. Instead, after initializing R' and applying links we sum the elements of R'. The difference between the resulting sum and N represents $\sum_{\gamma\in\Gamma} r(\gamma)$, the contribution from sinks in the graph.

As an alternative to lines 6–9 we might consider iterating over all nodes. For each node we would then sum the PageRank contributions for the nodes linking to it, directly implementing Equation 15.8. Although this alternative is reasonable for small Web graphs, the current approach of iterating over links requires simpler data structures and saves memory, an important consideration for larger Web graphs.

Iterating over links also allows us to store the links in a file on disk, further reducing memory requirements. Instead of iterating over an array, the loop of lines 6–9 would sequentially read this file of links from end to end during each iteration of the main loop. If links are stored in a file, the algorithm requires only the $O(N)$ memory to store the arrays R and R'.

```
1    for i ← 1 to N do
2        R[i] ← 1
3    loop
4        for i ← 1 to N do
5            R′[i] ← 1 − δ
6        for k ← 1 to E do
7            i ← link[k].from
8            j ← link[k].to
9            R′[j] ← R′[j] + δ·R[i] / out(i)
10       s ← N
11       for i ← 1 to N do
12           s ← s − R′[i]
13       for i ← 1 to N do
14           R[i] ← R′[i] + s/N
```

Figure 15.6 Basic PageRank algorithm. The array *link* contains the links in the Web graph. Lines 3-14 form an infinite loop. At the end of each interation of this loop, the array R contains a new estimate of PageRank for each page. In practice the loop would be terminated when the change in PageRank from iteration to iteration drops below a threshold or after a fixed number of iterations.

To demonstrate PageRank over a larger Web graph, we applied the algorithm to the English Wikipedia (see Exercise 1.9). The top 12 pages are listed in Table 15.2. Unsurprisingly, given that the collection includes only the English Wikipedia, major English-speaking countries figure prominently on the list. The remaining countries have large economies and close ties to the United States and to other English-speaking countries. Dates are often linked in Wikipedia, which explains the high ranks given to recent years. The high rank given to World War II, the pivotal event of the twentieth century, is also unsurprising. Overall, these results reflect the culture and times of Wikipedia's authors, an appropriate outcome.

15.3.2 Extended PageRank

The limitations of basic PageRank become evident if we revisit and reconsider its original motivation, the random surfer. No real user accesses the Web in the fashion of this random surfer. Even if we view the random surfer as an artificial construct — as a homunculus exhibiting the average behavior of Web users as a group — its selection of links and jumps uniformly at random remains unrealistic.

More realistically, our random surfer should have preferences for both links and jumps. For example, she might prefer navigational links over links to ads, prefer links at the top of the page to those at the bottom, and prefer links in readable fonts to those in tiny or invisible fonts. When the random surfer is jumping randomly, top-level pages may be preferred over deeply nested pages, longstanding pages may be preferred over newly minted pages, and pages with more text may be preferred over pages with less.

Table 15.2 The top 12 pages from a basic PageRank of Wikipedia.

Page: α	**PageRank: $r(\alpha)$**	**Probability: $r(\alpha)/N$**
United States	10509.50	0.004438
United Kingdom	3983.74	0.001682
2006	3781.65	0.001597
England	3421.03	0.001445
France	3340.53	0.001411
2007	3301.65	0.001394
2005	3290.57	0.001389
Germany	3218.33	0.001359
Canada	3090.20	0.001305
2004	2742.86	0.001158
Australia	2441.65	0.001031
World War II	2417.38	0.001021

Fortunately, we can easily extend PageRank to accommodate these preferences. To accommodate jump preferences, we define a *teleport vector* or *jump vector* J, of length N, where the element $J[i]$ indicates the probability that the random surfer will target page i when jumping. Because J is a vector of probabilities, we require $\sum_{i=1}^{N} J[i] = 1$. It is acceptable for some elements of J to be 0, but this may result in some pages having a zero PageRank value. If all elements of J are positive, $J[i] > 0$ for all $1 \leq i \leq N$, then all pages will have non-zero PageRank (Section 15.3.3).

To accommodate link preferences, we define an $N \times N$ *follow matrix* F, where element $F[i, j]$ indicates the probability that our random surfer will following a link to page j from page i. Unless page i is a sink, the elements in its row of F must sum to 1, $\sum_{i=1}^{N} F[i, j] = 1$. If page i is a sink, all elements of the row are 0.

F is *sparse*. At most E of the N^2 elements are non-zero, with one element corresponding to each link. We exploit this property of F in our implementation of the extended PageRank algorithm. The algorithm is given in Figure 15.7. It is similar to the basic PageRank algorithm of Figure 15.6, with lines 5, 9, and 14 generalized to use the jump vector and follow matrix. This generalization has no impact on the algorithm's time complexity, with each iteration of the main loop requiring $O(N + E)$ time.

Because F is sparse, the elements of F may be stored with their corresponding links. More specifically, we may add a *follow* field to each element of the *link* array, defined so that

$$link[k].follow = F[link[k].from, link[k].to].$$

```
1    for i ← 1 to N do
2        R[i] ← J[i] · N
3    loop
4        for i ← 1 to N do
5            R′[i] ← (1 − δ) · J[i] · N
6        for k ← 1 to E do
7            i ← link[k].from
8            j ← link[k].to
9            R′[j] ← R′[j] + δ · R[i] · F[i, j]
10       s ← N
11       for i ← 1 to N do
12           s ← s − R′[i]
13       for i ← 1 to N do
14           R[i] ← R′[i] + s · J[i]
```

Figure 15.7 Extended PageRank algorithm. Each element $J[i]$ of the jump vector J indicates the probabilty of reaching page i when jumping randomly. Each element $F[i, j]$ of the follow matrix F indicates the probabilty of reaching page i from page j when following a link.

This extended link array could then be stored in a file on disk, with the loop of lines 6–9 changed to iterate over this file. With the links stored on disk, the algorithm requires only $O(N)$ RAM to store the arrays R and R' and the jump vector J.

To compute basic PageRank using this extended algorithm, we construct a jump vector with all elements equal to $1/N$. For the follow matrix we set its elements equal to $1/out(\alpha)$ for all pages to which a page α links, and to 0 otherwise. If we number the nodes in the graph of Figure 15.2 from 1 to 6 in the order w_0, w_1, w_2, h_0, h_1, and m_0, we obtain the corresponding follow matrix and jump vector for basic PageRank:

$$F = \begin{pmatrix} 0 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{2} & 0 & 0 & 0 & 0 & \frac{1}{2} \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}, \qquad J = \begin{pmatrix} \frac{1}{6} \\ \frac{1}{6} \\ \frac{1}{6} \\ \frac{1}{6} \\ \frac{1}{6} \\ \frac{1}{6} \end{pmatrix}. \tag{15.11}$$

Now, suppose we know from external information that site W contains only high-quality and carefully edited information, and the relative quality of the information on the other sites remains unknown. We might adjust the follow matrix and jump vector to reflect this knowledge by assuming the random surfer is twice as likely to link or jump to site W than to any other

site, as follows:

$$F = \begin{pmatrix} 0 & \frac{2}{5} & \frac{2}{5} & \frac{1}{5} & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ \frac{2}{3} & 0 & 0 & 0 & 0 & \frac{1}{3} \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}, \qquad J = \begin{pmatrix} \frac{2}{9} \\ \frac{2}{9} \\ \frac{2}{9} \\ \frac{1}{9} \\ \frac{1}{9} \\ \frac{1}{9} \end{pmatrix}. \tag{15.12}$$

For general Web retrieval there are numerous adjustments we might make to the follow matrix and jump vector to reflect external knowledge. In general, however, we should not make adjustments to reflect properties of the Web graph itself. For example, if many pages link to a given page, we might view that page as having high "popularity" and be tempted to increase its probability in the jump vector. Avoid this temptation. Adjustments of this type are not required or recommended. Focus on external sources when adjusting the follow matrix and jump vector. It is the job of PageRank to account for the properties of the Web graph itself.

Many external sources can be enlisted when setting the follow matrix and jump vector. Although space does not allow us to describe and analyze all possibilities, we list a number of the possible sources below. This list is not exhaustive (see Exercise 15.6) nor is it the case that all (or any) of these suggestions are useful in practice.

- **Page content and structure.** During crawling and indexing, a search engine might analyze the structure and organization of a page as it would appear in a browser. This analysis might then be used to assign probabilities to links based on their layout and appearance. For example, users may be more likely to follow links near the top of the page or in menus. Various aspects of page content might also be interpreted as indicators of quality, thus influencing the jump vector by making the page a likelier target for a jump. Large blocks of readable text might be interpreted positively, while a low ratio of text to HTML tags might be interpreted negatively. Text in tiny and unreadable fonts may indicate an attempt to trick the search engine by misrepresenting what a user would actually see when the page is displayed.
- **Site content and structure.** Users may be more likely to jump to a site's top-level page than to a deeply nested page. Moreover, the longer the URL, the less likely a user will be to remember it or be willing to type it directly into a browser's address bar. The number of pages in a site may also be a factor when setting the jump vector: A site should not receive a high jump probability just because it contains a large number of pages. Depending on the site, users may be more likely to follow a link within a site, as they navigate about, than to follow a link off-site. On the other hand, a link between an educational site and a commercial site may represent a strong recommendation because these links are not usually motivated by commercial considerations. Older, more established, sites may be preferred over newer sites.

- **Explicit judgments.** Human editors might be recruited to manually identify (and classify) high-quality sites, which then receive higher jump probabilities. These editors could be professionals working for the search service itself or volunteers working for a Web directory such as the Open Directory Project (ODP—see Open Directory Project[2]). A major search engine might employ in-house editors to maintain their own Web directories as part of their overall service.
- **Implicit feedback.** A clickthrough on a Web search result might be interpreted as an implicit judgment regarding the quality of that page (see Section 15.5.2). Many of the major search services offer *toolbars*. These browser extensions provide additional features complementing the basic search service. With a user's explicit permission, a toolbar transmits information regarding the user's browsing behavior back to the search service. Pages and sites that the user visits might be recorded and then used to adjust probabilities in the follow matrix and jump vector. Many search services also provide free e-mail accounts. Links sent through these e-mail services might be accessible to these search services and may represent recommendations regarding site and page quality. Of course, any such use of implicit feedback must respect the privacy of users.

Adjustments in the jump vector may also be used to compute special variants of PageRank. To compute a *personalized PageRank* we assign high jump probabilities to pages of personal interest to an individual user (Page et al., 1999). These pages may be extracted from the user's browser bookmarks or her home page, or determined by monitoring her browsing behavior over a period of time. A *topic-oriented PageRank*, or *focused PageRank*, assigns high jump probabilities to pages known to be related to a specified topic, such as sports or business (Haveliwala, 2002). These topic-oriented pages may be taken from a Web directory such as the ODP.

For example, we can select a page from our Wikipedia corpus and generate a topic-specific PageRank focused just on that page. To generate the jump vector we assign a jump probability of 50% to the selected page and assign the remaining 50% uniformly to the other pages. We assign non-zero values to all elements of the jump vector in order to ensure that every page receives a non-zero PageRank value, but the outcome does not change substantially if we assign 100% probability to the selected page.

Table 15.3 shows the top 12 pages for a pair of topics: "William Shakespeare" and "Information Retrieval". The topic-specific PageRank for the topic "Information Retrieval" assigns high ranks to many pages related to that topic: The names Karen Spärck Jones and Gerard Salton should be familiar from Chapters 2 and 8; Rutgers University, City University London, and the University of Glasgow are home to prominent information retrieval research groups; the presence of Google and SIGIR is hardly unexpected. Surprisingly, however, the topic-specific PageRank for the topic "William Shakespeare" appears to be little different from the general

[2] `www.dmoz.org`

Table 15.3 The top 12 pages from a focused PageRank over Wikipedia for two topics.

William Shakespeare		**Information Retrieval**	
Article	PageRank	Article	PageRank
William Shakespeare	303078.44	Information retrieval	305677.03
United States	7200.15	United States	8831.25
England	5357.85	Association for Computing Machinery	6238.30
London	3637.60	Google	5510.16
United Kingdom	3320.49	GNU General Public License	4811.08
2007	3185.71	World Wide Web	4696.78
France	2965.52	SIGIR	4456.67
English language	2714.88	Rutgers University	4389.07
2006	2702.72	Karen Spärck Jones	4282.03
Germany	2490.50	City University, London	4274.76
2005	2377.21	University of Glasgow	4222.44
Canada	2058.84	Gerard Salton	4171.45

PageRank shown in Figure 15.2. Apart from a page on the "English language" and the Shakespeare page itself, all the pages appear in the top twelve of the general PageRank. Despite our topic-specific focus, the page for United States appears second in both lists.

A topic-specific PageRank and a general PageRank both represent probability distributions over the same set of pages, and we may obtain clearer results by comparing these distributions. In Section 9.4 (page 296) we defined the Kullback-Leibler divergence, or KL divergence, between two discrete probability distributions f and g as

$$\sum_x f(x) \cdot \log \frac{f(x)}{g(x)}\,. \tag{15.13}$$

Here we define f to be the topic-specific PageRank and g to be the general PageRank, normalized to represent probabilities. For each page α we rerank the page according to its *contribution* to the KL divergence (pointwise KL divergence) between f and g:

$$f(\alpha) \cdot \log \frac{f(\alpha)}{g(\alpha)}\,. \tag{15.14}$$

Recall that KL divergence indicates the average number of extra bits per symbol needed to compress a message if we assume its symbols are distributed according to g instead of the correct distribution f. Equation 15.14 indicates the extra bits due to α. Table 15.4 shows the impact of adjusting a focused PageRank using Equation 15.14.

All the top pages are now on-topic. Andrew Cecil Bradley, a Professor at Oxford University, was famous for his books on Shakespeare's plays and poetry. Ben Jonson and George Wilkins

Table 15.4 Reranking a focused PageRank according to each page's contribution to the KL divergence between the focused PageRank and the general PageRank.

William Shakespeare		**Information Retrieval**	
Article	KL divergence	Article	KL divergence
William Shakespeare	1.505587	Information retrieval	2.390223
First Folio	0.007246	SIGIR	0.027820
Andrew Cecil Bradley	0.007237	Karen Spärck Jones	0.026368
King's Men (playing company)	0.005955	C. J. van Rijsbergen	0.024170
Twelfth Night, or What You Will	0.005939	Gerard Salton	0.024026
Lord Chamberlain's Men	0.005224	Text Retrieval Conference	0.023260
Ben Jonson	0.005095	Cross-language information retrieval	0.022819
Stratford-upon-Avon	0.004927	Relevance (information retrieval)	0.022121
Richard Burbage	0.004794	Assoc. for Computing Machinery	0.022051
George Wilkins	0.004746	Sphinx (search engine)	0.021963
Henry Condell	0.004712	Question answering	0.021773
Shakespeare's reputation	0.004710	Divergence from randomness model	0.021620

were playwrights and contemporaries of Shakespeare. Richard Burbage and Henry Condell were actors and members of the King's Men playing company, for which Shakespeare wrote and acted. On the information retrieval side, Spärck Jones and Salton are joined by C. J. (Keith) van Rijsbergen, another great pioneer of the field. Sphinx is an open-source search engine targeted at relational database systems. The remaining topics should be familiar to readers of this book.

15.3.3 Properties of PageRank

In our presentation of PageRank we have been ignoring several important considerations: Will the fixed-point iteration procedure of Figure 15.7 always converge, or will the algorithm fail to work for some Web graphs? If it does converge, how quickly will it converge? Will it always convergence to the same PageRank vector, regardless of the initial estimate?

Fortunately the PageRank algorithm possesses properties that guarantee good behavior. To simplify our discussion of these properties, we combine the follow matrix and the jump vector into a single *transition matrix*. First we extend the follow matrix to handle sinks. Let F' be the $N \times N$ matrix

$$F'[i,j] \;=\; \begin{cases} J[j] & \text{if } i \text{ is a sink,} \\ F[i,j] & \text{otherwise.} \end{cases} \tag{15.15}$$

Next, let J' be the $N \times N$ matrix with each row equal to the jump vector. Finally, we define the transition matrix as

$$M \;=\; \delta \cdot F' + (1-\delta) \cdot J' \,. \tag{15.16}$$

For pages i and j, $M[i,j]$ represents the probability that a random surfer on page i will transition to page j without considering if this transition is made through a jump or by following a link. For example, the transition matrix corresponding to F and J in Equation 15.12 is

$$M = \begin{pmatrix} \frac{1}{18} & \frac{16}{45} & \frac{16}{45} & \frac{37}{180} & \frac{1}{18} & \frac{1}{18} \\ \frac{29}{36} & \frac{1}{18} & \frac{1}{18} & \frac{1}{18} & \frac{1}{18} & \frac{1}{18} \\ \frac{5}{9} & \frac{1}{18} & \frac{1}{18} & \frac{1}{18} & \frac{1}{18} & \frac{11}{36} \\ \frac{7}{9} & \frac{1}{36} & \frac{1}{36} & \frac{1}{36} & \frac{1}{36} & \frac{1}{36} \\ \frac{1}{36} & \frac{1}{36} & \frac{1}{36} & \frac{7}{9} & \frac{1}{36} & \frac{1}{36} \\ \frac{7}{36} & \frac{7}{36} & \frac{7}{36} & \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{pmatrix}. \tag{15.17}$$

You may recognize M as a stochastic matrix representing the transition matrix of a Markov chain, as introduced in Section 1.3.4. Each page corresponds to a state; the surfer's initial starting location corresponds to a starting state. Moreover, the PageRank vector R has the property that

$$M^T R = R. \tag{15.18}$$

You may recognize R as being an eigenvector[3] of M^T, with a corresponding eigenvalue of 1. Given an $n \times n$ matrix A, recall that $\vec{x}$ is an *eigenvector* of A and λ is the corresponding *eigenvalue* of A, if $A\vec{x} = \lambda\vec{x}$.

A considerable amount is known about Markov chains and their eigenvectors, and we can turn to this knowledge to determine the properties of PageRank. For simplicity we temporarily assume that all elements of the jump vector are positive, but these properties also hold when elements are 0. We consider jump vectors with 0 elements later in the section.

Underlying PageRank is the idea that as our random surfer surfs for longer periods of time, the relative time spent on each page converges to a value that is independent of the surfer's starting location. This informal notion corresponds to an important property of Markov chains known as *ergodicity*. A Markov chain is *ergodic* if two properties hold, both of which trivially hold for M when J contains only positive values. The first property, *irreducibility*, captures the idea that the random surfer can get from any state to any other state after some finite number of steps. More specifically for any states i and j, if the surfer is currently in state i, there is a positive probability that she can reach state j after taking no more than k steps, where $k \leq N$. If the jump vector contains only positive elements, then $M[i,j] > 0$ for all states i and j, and there is a positive probability that the surfer will reach j from i after $k = 1$ step.

The second property, that the transition matrix be *aperiodic*, eliminates those Markov chains in which the probability of being in a given state cycles between multiple values. A state i is *periodic* with period k if it is possible to return to it only after a number of steps that is a multiple of k. For example, if the surfer can return to a state only after an even number of

[3] If linear algebra is a distant memory, you may safely skip to the start of Section 15.3.4.

steps, then the state is periodic with period 2. If all states in a matrix have period 1, then it is aperiodic. Because $M[i,j] > 0$ for all i, the surfer can reach any state from any other state. Thus, the period of all states is 1 and the property holds.

Let the eigenvectors of M^T be $\vec{x}_1$, $\vec{x}_2$,..., $\vec{x}_N$, with associated eigenvalues λ_1,..., λ_N. Following convention we order these eigenvectors so that $|\lambda_1| \geq |\lambda_2| \geq ... \geq |\lambda_N|$. Because all elements of M^T are positive and each row of M sums to 1, a theorem known as the *Perron-Frobenius theorem* tells us that $\lambda_1 = 1$, that all other eigenvectors satisfy $|\lambda_i| < 1$, and that all elements of $\vec{x}_1$ are positive. Thus, *the principal eigenvector of M^T is the PageRank vector*. Following convention, eigenvectors are scaled to unit length, with $\|\vec{x}_i\| = 1$ for all i, so that $\vec{x}_1 = R/N$.

We now re-express our algorithm for computing PageRank in matrix notation. Let $\vec{x}^{(0)}$ be our initial estimate of PageRank (normalized to have a length of 1). Each iteration of the main loop in Figure 15.7 multiplies M^T by our current estimate to produce a new estimate. The first iteration produces the estimate $\vec{x}^{(1)} = M^T\vec{x}^{(0)}$, the second iteration produces the estimate $\vec{x}^{(2)} = M^T\vec{x}^{(1)}$, and so on. After n iterations we have the estimate

$$\vec{x}^{(n)} = \left(M^T\right)^n \vec{x}^{(0)}. \tag{15.19}$$

Thus, the algorithm computes PageRank if

$$\lim_{n\to\infty} \vec{x}^{(n)} = \vec{x}_1. \tag{15.20}$$

The ergodicity of M guarantees this convergence. In the terminology of Markov chains, $\vec{x}_1$ is called the *stationary distribution* of M. This algorithm for computing the principal eigenvector of a matrix is called the *power method* (Golub and Van Loan, 1996).

To determine the rate of convergence, suppose the initial estimate $\vec{x}^{(0)}$ is expressed as a linear combination of the (unknown) eigenvectors.

$$\vec{x}^{(0)} = \vec{x}_1 + a_2\,\vec{x}_2 + a_3\,\vec{x}_3 + \cdots + a_N\,\vec{x}_N \tag{15.21}$$

After the first iteration we have

$$\begin{aligned}
\vec{x}^{(1)} &= M^T\vec{x}^{(0)} \\
&= M^T\left(\vec{x}_1 + a_2\,\vec{x}_2 + \cdots + a_N\,\vec{x}_N\right) \\
&= M^T\vec{x}_1 + a_2\,M^T\vec{x}_2 + \cdots + a_N\,M^T\vec{x}_N \\
&= \vec{x}_1 + a_2\,\lambda_2\,\vec{x}_2 + \cdots + a_N\,\lambda_N\,\vec{x}_N.
\end{aligned}$$

After n iterations, and recalling that λ_2 is the second-largest eigenvalue after λ_1, we have

$$\begin{aligned}\vec{x}^{(n)} &= \vec{x}_1 + a_2\,\lambda_2^n\,\vec{x}_2 + \cdots + a_N\,\lambda_N^n\,\vec{x}_N \\ &\leq \vec{x}_1 + \lambda_2^n\,(a_2\,\vec{x}_2 + \cdots + a_N\,\vec{x}_N) \\ &= \vec{x}_1 + O(\lambda_2^n).\end{aligned}$$

Thus, the rate of convergence depends on the value of the second eigenvalue of M^T. The smaller the value, the faster the convergence.

For PageRank, Haveliwala and Kamvar (2003) proved the remarkable result that the value of the second eigenvector is δ, the damping factor.[4] Provided that δ is not too close to 1, convergence should be acceptably fast. More important, the rate of convergence does not depend on characteristics of the Web graph. If convergence is acceptably fast for one Web graph, it should remain acceptably fast as the Web graph evolves over time.

Haveliwala and Kamvar note that the stability of PageRank is also associated with δ. With smaller values of δ, PageRank becomes less sensitive to small changes in the Web graph. Of course, if the value of δ is close to 0, PageRank may fail to capture any useful information regarding the Web graph because a random jump will happen at nearly every step. The traditional value of $\delta = 0.85$ appears to be a reasonable compromise, giving good stability and convergence properties while allowing PageRank to do its job.

Thus, it is the random jumps of the imaginary surfer that determines the convergence and stability of PageRank. The jump vector guarantees the ergodicity of M, and therefore the convergence of the PageRank algorithm. The damping factor determines stability and the rate of convergence. The structure of the Web graph itself does not play a major role.

The above discussion assumes that all elements of the jump vector are positive. If this assumption does not hold, and the jump vector contains a zero, M may not be irreducible. An element with value zero in the jump vector indicates a page to which the surfer will never jump at random. Suppose there is a page α that cannot be reached by following links from any of the pages with a non-zero jump probability. Once the surfer makes a random jump after leaving α, she can never return to it.

As the length of the surf goes to infinity, the probability that the surfer eventually makes a jump goes to one. After making a jump the probability of visiting α is zero. Thus, we can set the PageRank value for α to zero, without explicitly computing it.

Let Φ' be the set of pages reachable after a jump (Figure 15.8). The pages outside Φ' have a PageRank value of zero, and do not need to be involved in the PageRank computation for the pages in Φ'. Let M' be the transition matrix corresponding to Φ' and let J' be its jump vector. J' may still contain elements with value zero, but the pages associated with these elements will be reachable from pages with non-zero values. Because a jump is possible at any step, and any

[4] More accurately, $|\lambda_2| \leq \delta$, but $\lambda_2 = \delta$ for any realistic Web graph.

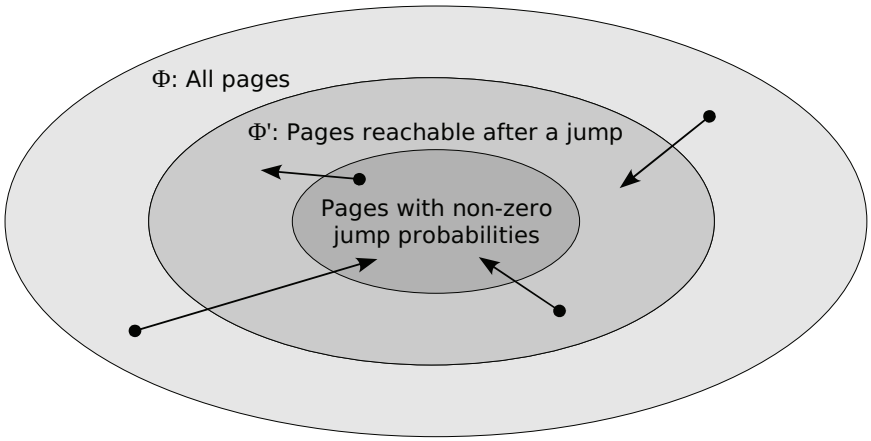


Figure 15.8 Pages reachable after a jump.

page in Φ' may be reached after a jump, M' is irreducible. The possibility of a jump at any step also helps to guarantee that M' is aperiodic (Exercise 15.5). Because M' is irreducible and aperiodic, it is ergodic and PageRank will converge. The other stability and convergence properties can also be shown to hold for M'.

15.3.4 Other Link Analysis Methods: HITS and SALSA

In addition to PageRank a number of other link analysis techniques have been proposed in the context of the Web. Two of these methods are Kleinberg's HITS algorithm (Kleinberg, 1998, 1999) and the related SALSA algorithm due to Lempel and Moran (2000). HITS was developed independently of PageRank during roughly the same time frame. SALSA was developed shortly after, in an attempt to combine features from both algorithms.

The intuition underlying HITS derives from a consideration of the roles a page may play on the Web with respect to a given topic. One role is that of an *authority*, a page that contains substantial and reliable information on the topic. A second role is that of a *hub*, a page that aggregates links to pages related to the topic. A good hub points to many authorities; a good authority is referenced by many hubs. We emphasize that these are idealized roles; a page may be both a hub and an authority to some extent.

In contrast to PageRank, which was envisioned as operating over the entire Web, Kleinberg envisioned HITS as operating over smaller Web graphs. These graphs may be generated by executing a query on a search engine to retrieve the contents of the top few hundred pages, along with pages in their immediate neighborhood. Based on the retrieved pages, HITS computes hubs and authorities with respect to the query. At the time HITS was invented, Web search engines were not known to incorporate link analysis techniques (Marchiori, 1997), and HITS provided a way of obtaining the benefit of these techniques without explicit support from the engine.

For a given page α, HITS computes two values: an authority value $a(\alpha)$ and a hub value $h(\alpha)$. The authority value of a page is derived from the hub values of the pages that link to it:

$$a(\alpha) \;=\; w_a \cdot \sum_{\beta \to \alpha} h(\beta). \tag{15.22}$$

The hub value of a page is derived from the authority values of the pages it references.

$$h(\alpha) \;=\; w_h \cdot \sum_{\alpha \to \beta} h(\beta). \tag{15.23}$$

The significance of the weights w_a and w_h is discussed shortly. Links from a page to itself (*self-loops*) and links within sites are ignored because these are assumed to represent navigational relationships rather than hub-authority relationships.

We may express Equations 15.22 and 15.23 in matrix/vector notation by defining $\vec{a}$ to be the authority values and $\vec{h}$ to be the hub values for the N pages in the Web graph.[5]. Let W be the *adjacency matrix* of the Web graph, where $W[i,j] = 1$ if there is a link from the ith page to the jth page, and $W[i,j] = 0$ otherwise. For example, the adjacency matrix for the Web graph in Figure 15.2 (retaining links within sites) is

$$W \;=\; \begin{pmatrix} 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}. \tag{15.24}$$

In matrix/vector notation, Equations 15.22 and 15.23 become

$$\vec{a} \;=\; w_a \cdot W^T \vec{h} \quad \text{and} \quad \vec{h} \;=\; w_h \cdot W \vec{a}\,. \tag{15.25}$$

Substituting gives

$$\vec{a} \;=\; w_a \cdot w_h \cdot W^T W \vec{a} \quad \text{and} \quad \vec{h} \;=\; w_a \cdot w_h \cdot W W^T \vec{h}\,. \tag{15.26}$$

If we define $A = W^T W$, $H = W W^T$, and $\lambda = 1/(w_a \cdot w_h)$, we have

$$\lambda \vec{a} \;=\; A \vec{a} \quad \text{and} \quad \lambda \vec{h} \;=\; H \vec{h}\,. \tag{15.27}$$

Thus, the authority vector $\vec{a}$ is an eigenvector of A and the hub vector $\vec{h}$ is an eigenvector of H.

[5] If you wish to avoid the details, you may safely skip to Section 15.3.5.

The matrices A and H have interesting interpretations taken from the field of bibliometrics (Lempel and Moran, 2000; Langville and Meyer, 2005): A is the *co-citation matrix* for the Web graph, where $A[i,j]$ is the number of pages that link to both i and j; H is the *co-reference* or *coupling matrix* for the Web graph, where $H[i,j]$ is the number of pages referenced by both i and j. Both A and H are symmetric, a property of importance for the computation of $\vec{a}$ and $\vec{h}$.

As we did for PageRank, we may apply fixed-point iteration — in the form of the power method — to compute $\vec{a}$ and $\vec{h}$. Let $\vec{a}^{(0)}$ and $\vec{h}^{(0)}$ be initial estimates for $\vec{a}$ and $\vec{h}$. The nth estimate, $\vec{a}^{(n)}$ and $\vec{h}^{(n)}$, may be computed from the previous estimate by the equations

$$\vec{a}^{(n)} = W^T\vec{h}^{(n-1)}/\|W^T\vec{h}^{(n-1)}\| \quad \text{and} \quad \vec{h}^{(n)} = W\vec{a}^{(n-1)}/\|W\vec{a}^{(n-1)}\|\,. \tag{15.28}$$

Normalization allows us to avoid explicit computation of the eigenvalue by ensuring that each estimate has unit length. Successive application of these equations will compute $\vec{a}$ and $\vec{h}$ if

$$\lim_{n\to\infty}\vec{a}^{(n)} = \vec{a} \quad \text{and} \quad \lim_{n\to\infty}\vec{h}^{(n)} = \vec{h}\,. \tag{15.29}$$

For PageRank the ergodicity of M guarantees convergence. For HITS the symmetric property of A and H provides this guarantee, as long as the initial estimates have a component in the direction of the principal eigenvector (Kleinberg, 1999; Golub and Van Loan, 1996). Any unit vector with all positive elements will suffice to satisfy this requirement, such as

$$\vec{a}^{(0)} = \vec{h}^{(0)} = \langle\, 1/\sqrt{N},\, 1/\sqrt{N}, \ldots \rangle\,.$$

Unfortunately, convergence to a unique solution is not guaranteed (Langville and Meyer, 2005). Depending on the initial estimate, HITS may convergence to different solutions (see Exercise 15.9).

SALSA, the *stochastic approach for link-structure analysis*, introduces PageRank's random surfer into HITS (Lempel and Moran, 2000). The creation of SALSA was motivated by the observation that a small set of highly interconnected sites (or "tightly connected communities") can have a disproportionate influence on hub and authority values, by generating inappropriately high values for members of these communities.

SALSA imagines a random surfer following both links and *backlinks* (i.e., reversed links), alternating between them. On odd-numbered steps the surfer chooses an outgoing link from the current page uniformly at random and follows it. On even-numbered steps the surfer chooses an incoming link uniformly at random, from a page linking to the current page, and follows it backwards to its source. As the number of steps increases, the relative time spent on a page just before taking an odd-numbered step represents the page's hub score; the relative time spent on a page just before taking an even-numbered step represents the page's authority score.

Although we do not provide the details, SALSA may be formulated as an eigenvector problem. Lempel and Moran (2000) discuss properties that simplify the computation of SALSA. Rafiei and Mendelzon (2000) propose the introduction of random jumps into a SALSA-like algorithm, with the possibility for a random jump occurring after each odd-even pair of steps.

15.3.5 Other Static Ranking Methods

Static ranking provides a query-independent score for each Web page. Although link analysis is an essential component in the computation of static rank, other features may contribute.

As we mentioned with respect to link analysis in Section 15.3.2, some of the most important features are provided through *implicit user feedback*. By "implicit" we mean that the user provides this feedback while engaged in other activities, perhaps without being aware that she is providing feedback. For example, a click on a Web search result may indicate a preference for that site or page. If higher-ranked results were skipped, the click may also indicate a negative assessment of these skipped pages (Joachims et al., 2005). The toolbars provided by commercial search services track the sites visited by a user and return this information to the search service (with the user's permission). The popularity of a site may be measured through the number of users visiting it, and through the frequency and length of their visits (Richardson et al., 2006).

The content of the pages themselves may also contribute to their static rank. Ivory and Hearst (2002) describe how quantitative measures of page content and structure predict the quality scores assigned by expert assessors. These measures consider the quantity and complexity of text, the placement and formatting of graphical elements, and the choice of fonts and colors.

Finally, the content and structure of a URL may be considered. Short and simple URLs might be favored over long and complex ones, particularly for navigational queries (Upstill et al., 2003). Sites in the `com` domain might be more appropriate for commercial queries than sites in the `edu` domain; the opposite is true for academic queries.

Richardson et al. (2006) apply machine learning techniques, similar to those described in Section 11.7, to the computation of static rank. By combining popularity, page content, and URL features with basic PageRank, they demonstrate substantial improvements over basic PageRank alone. In Section 15.3.2 we discussed how these same features might be applied to adjust the follow and jump vectors in extended PageRank. The best methods for exploiting these features in the computation of static rank remains an area open for exploration.

15.4 Dynamic Ranking

At query time the search engine combines each page's static rank with query-dependent features — such as term frequency and proximity — to generate a dynamic ranking for that query. Although the dynamic ranking algorithms used in commercial search engines are grounded in the theory of Part III, the details of these algorithms vary greatly from one Web search engine to another. Moreover, these algorithms evolve continuously, reflecting the experience gained from the tremendous numbers of queries and users these search engines serve.

Although the scope of this book does not allow us to provide the details of the dynamic ranking algorithms used in specific Web search engines, two aspects of dynamic ranking deserve some attention. The first of these aspects — the availability of anchor text — represents a ranking feature of particular importance in Web search. Anchor text often provides a description of

Table 15.5 Anchor text for in-links to the Wikipedia page `en.wikipedia.org/wiki/William_Shakespeare`, ordered by number of appearances.

#	Anchor Text	#	Anchor Text
3123	Shakespeare	2	Will
3008	William Shakespeare	2	Shakesperean
343	Shakespeare's	2	Shakespere
210	Shakespearean	2	Shakespearean studies
58	William Shakespeare's	2	Shakepeare
52	Shakespearian	2	Bill Shakespeare...
10	W. Shakespeare	1	the Bard's
7	Shakespeare, William	1	lost play of Shakespeare
3	William Shakepeare	1	Shakespearesque
3	Shakesphere	1	Shakespearean theatre
3	Shakespeare's Theatre	1	Shakespearean plays
3	Bard		...
2	the Bard		...

the page referenced by its link, which may be exploited to improve retrieval effectiveness. The importance of the second aspect — novelty — derives from the scale and structure of the Web. Although many pages on a given site may be relevant to a query, it may be better to return one or two results from several sites rather than many pages from a single site, thereby providing more diverse information.

15.4.1 Anchor Text

Anchor text often provides a label or description for the target of its link. Table 15.5 lists the anchor text linking to the page `en.wikipedia.org/wiki/William_Shakespeare` from other pages within Wikipedia, ordered by the number of times each string appears. Pages outside of Wikipedia may also link to this page, but that anchor text can be discovered only through a crawl of the general Web. The table shows a mixture of text, including misspellings and nicknames ("the Bard"). In all, 71 different strings are used in 6889 anchors linking to the page. However, more than 45% of these anchors use the text "Shakespeare", and nearly as many use the text "William Shakespeare".

On the general Web the number of links to a page may vary from one to many millions. When a page is linked many times, the anchor text often repeats, and when it repeats, it usually represents an accurate description of the page (but see Exercise 15.12). Misspellings, nicknames, and similar anchor text can also prove valuable because these terms may not appear on the page, but may be entered by a user as part of a query.

For anchor text to be usable as a ranking feature, it must be associated with the target of the link. Before building a Web index, anchor text must be extracted from each page to create a set of tuples of the form

$$\langle \textit{target URL}, \textit{anchor text} \rangle .$$

These tuples are sorted by URL. The anchor text for each URL is then merged with the page contents for that URL to form a composite document for indexing purposes.

For retrieval, anchor text may be treated as a document field, much as we did for titles and other fields in Section 8.7. When computing term weights, anchor text may be weighted in the same way as other text, or term frequencies may be adjusted to dampen the influence of repeated terms (Hawking et al., 2004). The static rank of the page on which the anchor appears may also play a role in term weights (Robertson et al., 2004). Anchor text appearing on pages with high static rank may be given greater weight than text appearing on pages with low static rank.

15.4.2 Novelty

Given the variety and volume of information on the Web, it is important for a search engine to present a diverse set of results to the user. Given our example query ⟨"UPS"⟩, a search engine might best return a mix of results related to the parcel service, to power supplies, and to the university. Simple ways to reduce redundancy include the identification of duplicate pages in the index and post-retrieval filtering to eliminate excessive results from a single Web site.

Although the Web contains many pages that are duplicates (or near-duplicates) of others, the result of a search should usually contain only a single copy. These duplicated pages occur on the Web for several reasons. Many sites return a default page when a URL path is invalid, and all invalid URLs from these sites appear to be duplicates of this default page. Certain types of content are frequently mirrored across multiple sites. For example, a newswire article might appear on the Web sites of multiple newspapers. These duplicates may be detected and flagged by the Web crawler during the crawl (Section 15.6.3), and static rank may be used to choose the best copy at query time.

In general, duplicated pages within a given site should be eliminated from the index. However, it may be reasonable to retain duplicates when they appear on different sites. Most commercial Web search engines allow searches to be restricted to a specified site or domain. For example, including the term `site:wikipedia.org` in a query would restrict a search to Wikipedia. Retaining pages duplicated on other sites allows these searches to be handled correctly.

Another method for improving diversity in search results is to apply post-retrieval filtering. After dynamic ranking, most commercial search engines filter the results to reduce redundancy.

For example, the documentation for the (now-deprecated) Google SOAP API[6] describes one simple post-retrieval filtering algorithm. After retrieval, results may be filtered in two ways:

1. If several results have identical titles and snippets, then only one is retained.
2. Only the two best results from a given Web site are retained.

The API calls this second filter "host crowding". Interestingly, the first filter may eliminate relevant results that are not in fact redundant in order to reduce the *appearance* of redundancy in the search results.

15.5 Evaluating Web Search

In principle it is possible to apply the traditional IR evaluation framework (e.g., P@10 and MAP) to Web search. However, the volume of material available on the Web introduces some problems. For example, an informational query could have hundreds or thousands of relevant documents. It is not uncommon that all of the top ten results returned for such a query are relevant. In that situation the binary relevance assessments ("relevant"/"not relevant") underlying the traditional methodology might not be a sufficient basis for a meaningful evaluation; graded relevance assessments may be more appropriate.

When graded relevance values are available, evaluation measures such as nDCG (Section 12.5.1) may be applied to take advantage of them (Richardson et al., 2006). For example, Najork (2007) uses nDCG in a Web search evaluation based on more than 28,000 queries taken from the Windows Live search engine. Nearly half a million manual judgments were obtained for these queries. These judgments were made on a six-point scale: definitive, excellent, good, fair, bad, and detrimental.

The nature of the Web introduces several novel aspects of evaluation. As we discussed in Section 15.2, many Web queries are navigational. For these queries only a single specific page may be relevant. In Section 15.5.1 we present an evaluation methodology addressing this query type. In addition, the volume of queries and users handled by commercial Web search services provides an opportunity to infer relevance judgments from user behavior, which may be used to augment or replace manual judgments. In Section 15.5.2 we provide an overview of techniques for interpreting clickthroughs and similar user actions for evaluation purposes.

15.5.1 Named Page Finding

A *named page finding* task is a Web evaluation task that imagines a user searching for a specific page, perhaps because she has seen it in the past or otherwise learned about it. The user enters a query describing the content of the page and expects to receive it as the first (or only) search

[6] `code.google.com/apis/soapsearch/reference.html` (accessed Dec 23, 2009)

Table 15.6 Named page finding results for selected retrieval methods from Part III. The best run from TREC 2006 (Metzler et al., 2006) is included for comparison.

Method	MRR	% Top 10	% Not Found
BM25 (Ch. 8)	0.348	50.8	16.0
BM25F (Ch. 8)	0.421	58.6	16.6
LMD (Ch. 9)	0.298	48.1	15.5
DFR (Ch. 9)	0.306	45.9	19.9
Metzler et al. (2006)	0.512	69.6	13.8

result. For example, the query ⟨"Apollo", "11", "mission"⟩ might be used to describe NASA's history page on the first moon landing. Although other pages might be related to this topic, only that page would be considered correct. Named page finding tasks were included in TREC as part of the Web Track from 2002 to 2004 and in the Terabyte Track in 2005 and 2006 (Hawking and Craswell, 2001; Craswell and Hawking, 2004; Clarke et al., 2005; Büttcher et al., 2006).

The premise behind named page finding represents an important subset of navigational queries (see page 515). Nonetheless, many navigational queries (e.g., ⟨"UPS"⟩) are seeking specific sites rather than specific content. To address this issue the scope of named page finding may be expanded to include "home page finding" queries (Craswell and Hawking, 2004).

For the 2006 Terabyte Track, 181 topics were created by track participants. For each topic the creator specified the answer page within the GOV2 collection. Each submitted run consisted of up to 1000 results. During the evaluation of the runs, near-duplicates of answer pages were detected using an implementation of Bernstein and Zobel's (2005) DECO algorithm, a variant of the near-duplicate detection algorithm from Section 15.6.3. All near-duplicates of a correct answer page were also considered correct. Three measures were used in the evaluation:

- **MRR**: The *mean reciprocal rank* of the first correct answer.
- **% Top 10**: The proportion of queries for which a correct answer was found in the top ten search results.
- **% Not Found**: The proportion of queries for which no correct answer was found in the top 1000 search results.

For a given topic, reciprocal rank is the inverse of the rank at which the answer first appears. If the answer is the top result, its reciprocal rank is 1; if the answer is the fifth result, its reciprocal rank is 1/5. If the answer does not appear in the result list, it may be assigned a reciprocal rank of $1/\infty = 0$. Mean reciprocal rank is the average of reciprocal rank across all topics.

Full results for the track are provided by Büttcher et al. (2006). Table 15.6 shows the results of applying the retrieval methods from Part III to this task. For this task, BM25F improves upon BM25. For comparison we include the best result from the track itself (Metzler et al., 2006). This run incorporated a number of Web-specific techniques, such as link analysis (static ranking) and anchor text. The gap between the standard retrieval methods from Part III and Metzler's run illustrates the importance of these techniques for navigational search tasks.

15.5.2 Implicit User Feedback

Implicit feedback is provided by users as a side effect of their interaction with a search engine. Clickthroughs provide one important and readily available source of this implicit feedback. Whenever a user enters a query and clicks on the link to a result, a record of that clickthrough may be sent by the browser to the search engine, where it is logged for later analysis.

For a given query the clickthroughs from many users may be combined into a *clickthrough curve* showing the pattern of clicks for that query. The plots in Figure 15.5 provide examples of stereotypical clickthrough curves for navigational and informational queries. In both plots the number of clickthroughs decreases with rank. If we interpret a clickthrough as a positive preference on the part of the user, the shapes of these curves are just as we would expect: Higher-ranked results are more likely to be relevant and to receive more clicks. Even when successive results have the same relevance, we expect the higher ranked result to receive more clicks. If the user scans the results in order, higher-ranked results will be seen before lower-ranked ones. There is also a *trust bias* (Joachims et al., 2005) in user behavior: Users expect search engines to return the best results first, and therefore tend to click on highly ranked results even though they might not contain the information sought.

Figure 15.9 presents a third clickthrough curve, taken from the same paper as the previous two (Clarke et al., 2007). This figure plots clickthroughs for the informational/transactional query ⟨"kids", "online", "games"⟩. The plot includes a number of *clickthrough inversions*, in which a particular result received fewer clicks than the result ranked immediately below it (e.g., the result at rank 2 versus rank 3, or rank 7 versus rank 8).

Clickthrough inversions may indicate a suboptimal ranking, in which a less relevant document is ranked above a more relevant one. However, they may also arise for reasons unrelated to relevance. For example, the title and snippet of the higher-ranked result may not accurately describe the underlying page (Clarke et al., 2007; Dupret et al., 2007). If a user does not understand how or why a result is relevant after reading its title and snippet, she may be inclined to ignore it and to move on to other results. Nonetheless, when the titles and snippets provide accurate descriptions of results, a clickthrough inversion may be interpreted as a pairwise preference for the lower-ranked result (Joachims and Radlinski, 2007).

Query *abandonment* — issuing a query but not clicking on any result — suggests user dissatisfaction of a more substantial nature (Joachims and Radlinski, 2007). When many users abandon a given query, this behavior may indicate that none of the top results are relevant. In some cases an abandonment is followed by a query reformulation in which the user adds, removes, or corrects terms. A search service may capture these query reformulations by tracking search activity through browser cookies and similar mechanisms.[7]

By examining query reformulations from large numbers of users, it is possible to correct spelling errors, to suggest expansion terms, and to identify acronyms (Cucerzan and Brill, 2004; Jones et al., 2006). For example, a user entering the query ⟨"brittany", "spears"⟩ may correct

[7] `www.w3.org/Protocols/rfc2109/rfc2109`

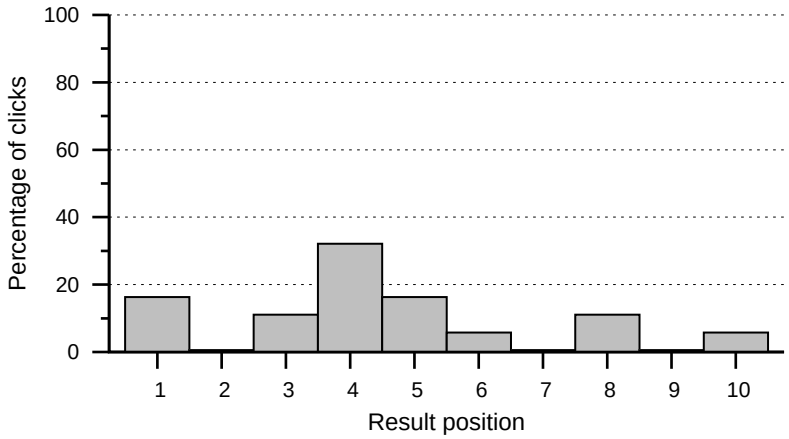


Figure 15.9 Clickthrough curve for the query ⟨"kids", "online", "games"⟩. Inversions (e.g., rank 2 versus rank 3) indicate a suboptimal ranking of search results.

it to ⟨"britney", "spears"⟩.[8] A click after a reformulation may indicate a page that is relevant to the original query. For example, Joachims and Radlinski (2007) report that users frequently reformulate ⟨"oed"⟩ to ⟨"oxford", "english", "dictionary"⟩ and then click on the first result.

Capturing other forms of implicit feedback requires the cooperation of the user, who must install a browser toolbar or similar application, and give explicit permission to capture this feedback (Kellar et al., 2007). Page *dwell time* provides one example of feedback that may be captured in this way. If a user clicks on a result and then immediately backtracks to the result page, this action may indicate that the result is not relevant.

Taken together, these and other sources of implicit feedback can lead to substantial improvements in the quality of Web search results, both by facilitating evaluation (Agichtein et al., 2006b) and by providing additional ranking features (Agichtein et al., 2006a).

15.6 Web Crawlers

The functioning of a Web crawler is similar to that of a user surfing the Web, but on a much larger scale. Just as a user follows links from page to page, a crawler successively downloads pages, extracts links from them, and then repeats with the new links. Many of the challenges associated with the development of a Web crawler relate to the scale and speed at which this process must take place.

[8] `labs.google.com/britney.html` (accessed Dec 23, 2009)

Suppose our goal is to download an 8-billion-page snapshot of the Web over the course of a week (a small snapshot by the standards of a commercial Web search engine). If we assume an average page is 64 KB in size, we must download data at a steady rate of

$$\begin{aligned} 64 \text{ KB/page} \cdot 8 \text{ giga-pages/week} &= 512 \text{ TB/week} \\ &= 888 \text{ MB/second} \end{aligned}$$

To achieve this tremendous download rate, the crawler must download pages from multiple sites concurrently because it may take an individual site several seconds to respond to a single download request. As it downloads pages, the crawler must track its progress, avoiding pages it has already downloaded and retrying pages when a download attempt fails.

The crawler must take care that its activities do not interfere with the normal operation of the sites it visits. Simultaneous downloads should be distributed across the Web, and visits to a single site must be carefully spaced to avoid overload. The crawler must also respect the wishes of the sites it visits by following established conventions that may exclude the crawler from certain pages and links.

Many Web pages are updated regularly. After a period of time the crawler must return to capture new versions of these pages, or the search engine's index will become stale. To avoid this problem we might recrawl the entire Web each week, but this schedule wastes resources and does not match the practical requirements of Web search. Some pages are updated regularly with "popular" or "important" information and should be recrawled more frequently, perhaps daily or hourly. Other pages rarely change or contain information of minimal value, and it may be sufficient to revisit these pages less frequently, perhaps biweekly or monthly.

To manage the order and frequency of crawling and recrawling, the crawler must maintain a priority queue of seen URLs. The ordering of URLs in the priority queue should reflect a combination of factors including their relative importance and update frequency.

15.6.1 Components of a Crawler

In this section we examine the individual steps and components of a crawler by tracing the journey of a single URL (`en.wikipedia.org/wiki/William_Shakespeare`) as it travels through the crawling process. Tracing a single URL allows us to simplify our presentation, but we emphasize that crawling at the rates necessary to support a Web search engine requires these steps to take place concurrently for large numbers of URLs.

Figure 15.10 provides an overview of these steps and components. The crawling process begins when a URL is removed from the front of the priority queue and ends when it is returned to the queue along with other URLs extracted from the visited page. Although specific implementation details vary from crawler to crawler, all crawlers will include these steps in one form or another. For example, a single thread may execute all steps for a single URL, with thousands of such threads executing concurrently. Alternatively, URLs may be grouped into large batches, with

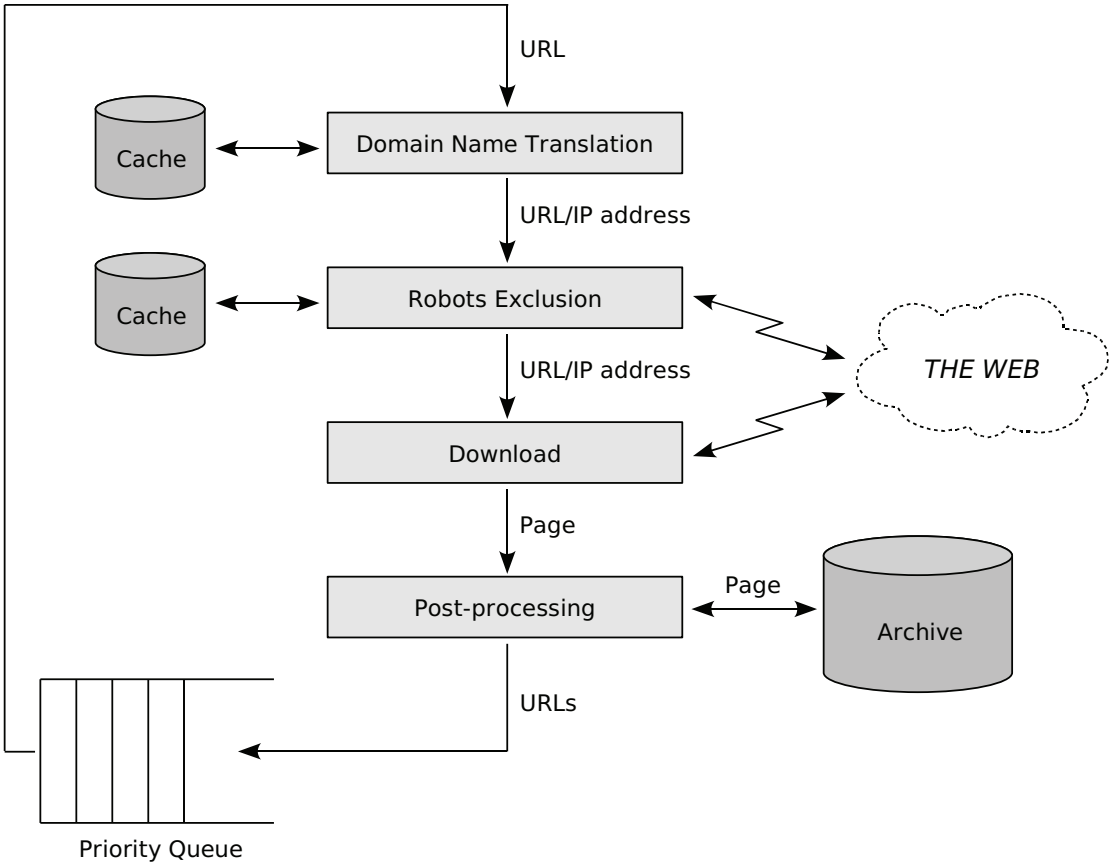


Figure 15.10 Components of a Web crawler.

each step executed for all URLs in the batch before the crawler moves on to execute the next step for the same batch.

Like the search engine it supports (Section 14.1), the activities of a large-scale Web crawler must be distributed across multiple machines in order to sustain the necessary download rates. This distribution may be achieved by assigning subsets of URLs to each machine, essentially giving each machine responsibility for a portion of the Web. The construction of these subsets may be based on host name, IP address, or other factors (Chung and Clarke, 2002). For example, we might assign each machine responsibility for certain hosts. The priority queue might be centralized at a single machine that would distribute URLs to other machines for crawling, or each machine might implement its own priority queue for its assigned subset.

Domain name translation

The processing of a URL begins by translating its host name into a 32-bit IP address. For the host `en.wikipedia.org` the corresponding IP address is `208.80.152.2` (at time and place of writing). This address will be downloaded to contact the machine hosting the page.

As it is for browsers and other Internet applications, this translation is effected through the Domain Name System (DNS), a standard Internet service that is implemented by a distributed hierarchy of servers and caches spread across the Internet. Although the speed of DNS is acceptable for most applications, its speed may be insufficient to satisfy the demands of a Web crawler, which may require thousands of translations per second. To satisfy this translation rate, the crawler may need to maintain its own translation cache. This cache would maintain mappings between host names and IP addresses, expiring and refreshing these entries as they grow stale. Although the inclusion of a custom DNS cache represents a minor part of a crawler, the need for this cache illustrates the problems encountered when crawling at high speeds.

Robots exclusion protocol

With the IP address available the crawler must check that access to the page is permitted by the Web site. This permission is determined through an unofficial (but well-established) convention known as the *robots exclusion protocol* (more informally, the "`robots.txt` protocol"). For a given site, access permissions are obtained by appending the path `"/robots.txt"` to the host name and downloading the corresponding page.

In our example the URL is `http://en.wikipedia.org/robots.txt`. Figure 15.11 shows a small portion of the corresponding page. The page is structured as a series of comments and instructions. A "`User-agent`" instruction begins a list that applies to the crawler named on the line, with a "`*`" indicating that the instructions apply to all crawlers. For example, a "`Disallow`" instruction indicates a path prefix to which access is disallowed. Downloading any page having a path starting with a disallowed prefix is not permitted. The example in the figure denies all access to the `wget` program, and requests other crawlers to avoid the random article link and the search link, both of which generate content dynamically. Full details on the robots exclusion protocol may be found on the Web Robots site.[9]

A Web crawler will generally cache the `robots.txt` page for each host it encounters in order to avoid repeated downloads whenever a URL from that host is processed. This cached information should be expired on a regular basis, perhaps after a few hours or days.

When access by the crawler is not permitted, the URL may be returned to the priority queue and labeled to indicate that download was disallowed. The crawler may retry the URL at a future time, checking first that access has been permitted, or the URL may be permanently flagged as disallowed, never to be retried but maintained as a placeholder to prevent further attempts.

[9] `www.robotstxt.org`

```
#
# robots.txt for http://www.wikipedia.org/ and friends
#
...

#
# Sorry, wget in its recursive mode is a frequent problem.
#
User-agent: wget
Disallow: /
...

#
# Friendly, low-speed bots are welcome viewing article pages, but not
# dynamically-generated pages please.
#
User-agent: *
Disallow: /wiki/Special:Random
Disallow: /wiki/Special:Search
...
```

Figure 15.11 Extracts from a `robots.txt` file.

Download

After confirming that download is permitted, the crawler accesses the Web site via the HTTP protocol and downloads the page. The page may be formatted as HTML (e.g., Figure 8.1 on page 278) or in other formats, such as PDF. Associated pages may be identified and downloaded at the same time, or they may be left for a future crawling cycle. For example, if the page defines a frame set, all pages in the set might be downloaded together. Images may also be downloaded, perhaps in support of an image search service.

During download the crawler may be required to resolve redirections, which indicate that the actual content of the page is found elsewhere. These redirections can add considerable complexity to the download process, in part because redirections may be implemented in multiple ways. At the HTTP protocol level various responses generated by the Web server may indicate that a page has moved permanently or temporarily to a new location. Redirections may also be specified by tags appearing in HTML pages. Finally, JavaScript loaded with HTML pages may generate redirections during execution.

For example, Wikipedia uses an HTTP "301 Moved Permanently" response to redirect the URL `en.wikipedia.org/wiki/william_shakespeare` to `en.wikipedia.org/wiki/William_Shakespeare`. Redirections to correct spelling and capitalization are common in Wikipedia.

Other sites may use HTTP redirections when sites are redesigned or restructured, so that older URLs continue to work.

The use of JavaScript to generate redirections causes the most trouble for Web crawlers because correct determination of the redirection's target potentially requires the crawler to execute the JavaScript. Moreover, this JavaScript may redirect to different pages depending on factors such as the user's browser type. Covering all possibilities theoretically requires the crawler to repeatedly execute the JavaScript under different configurations until all possible redirection targets are determined.

Given resource limitations, it may not be feasible for a crawler to execute JavaScript for millions of downloaded pages. Instead the crawler may attempt partial evaluation or other heuristics, which may be sufficient to determine redirection targets in many cases. On the other hand, if the crawler ignores JavaScript redirections, it may be left with little useful content, perhaps no more than a message suggesting that JavaScript be enabled.

Post-processing

After download, the crawler stores the page in an archive for indexing by the search engine. Older copies of the page may be retained in this archive, thus allowing the crawler to estimate the frequency and types of changes the page undergoes. If the download fails, perhaps because the site is temporarily inaccessible, these older copies remain available for indexing. After a failed download the URL may be returned to the priority queue and retried at a later time. If several download attempts fail over a period of days, the older copies may be allowed to expire, thus removing the page from the index.

In addition to being stored in the archive, the page is analyzed, or *scraped*, to extract any URLs it contains. Much like a browser, the crawler parses the HTML to locate anchor tags and other elements containing links. Anchor text and other information required for indexing may be extracted at the same time, then stored in the archive for the convenience of the search engine. Pages that are duplicates (or near-duplicates) of the downloaded page may also be identified during this analysis (see Section 15.6.3).

During post-processing, the crawler must respect conventions requesting that it not index a page or follow certain links. If the tag

```
<meta name="robots" content="noindex">
```

appears in the header of page, it indicates that a search engine should not include the page in its index. The "`rel=nofollow`" attribute appearing in an anchor tag indicates that the crawler should essentially ignore the link. For example, the following external link appeared on the Wikipedia Shakespeare page at the time of writing:

```
<a href="http://www.opensourceshakespeare.org" rel="nofollow">
  Open Source Shakespeare
</a>
```

The crawler should crawl this page only if it finds a link elsewhere, and the link should not influence ranking in any way. Sites such as blogs and wikis, which allow their users to create external links, may automatically add "`rel=nofollow`" to all such links. This policy is intended to discourage the creation of inappropriate links solely for the purpose of increasing the PageRank value of the target page. With the addition of "`rel=nofollow`" users generating this link spam will garner no benefit from it.

As it did during download, JavaScript poses problems during post-processing. When it is executed, JavaScript is capable of completely rewriting a page with new content and links. If execution is not feasible, the crawler may apply heuristics to extract whatever URLs and other information it can, but without any guarantee of success.

Priority queue

URLs extracted during post-processing are inserted into the priority queue. If a URL already appears in the queue, information gathered during post-processing may alter its position.

Implementing the priority queue is a challenging problem. If we assume an average URL is 64 bytes in length, a priority queue for 8 billion pages requires half a terabyte just to store the URLs (ignoring compression). These storage requirements, coupled with the need to support millions of updates per second, may limit the sophistication of strategies used to manage the priority queue. In Section 15.6.2, where these strategies are discussed, we ignore implementation difficulties. However, when deploying an operational crawler, these difficulties cannot be ignored.

15.6.2 Crawl Order

Suppose we are crawling the Web for the first time, with the goal of capturing and maintaining a multibillion-page snapshot. We have no detailed knowledge of the sites and pages contained on the Web, which remain to be discovered. To begin, we might start with a small *seed set* of well-known URLs including major portals, retail sites, and news services. The pages linked from the ODP[10] or other Web directory could form one possible seed set. If we then proceed in a breadth-first manner, we should encounter many high-quality pages early in the crawl (Najork and Wiener, 2001).

As we conduct the crawl, our knowledge of the Web increases. At some point, especially if the crawler is feeding an operational search engine, it becomes important to revisit pages as well as to crawl new URLs. Otherwise, the index of the search service will become stale. From this point forward, crawling proceeds *incrementally*, with the crawler continuously expanding and updating its snapshot of the Web. As it visits and revisits pages, new URLs are discovered and deleted pages are dropped.

[10] `www.dmoz.org`

The activities of the crawler are then determined by its *refresh policy* (Olston and Pandey, 2008; Pandey and Olston, 2008; Cho and Garcia-Molina, 2000, 2003; Wolf et al., 2002; Edwards et al., 2001). Two factors feed into this policy: (1) the frequency and nature of changes to pages and (2) the impact that these changes have on search results, compared to the impact of crawling new URLs. The simplest refresh policy is to revisit all pages at a fixed rate, once every X weeks, while continuing to crawl new URLs in breadth-first order. However, although this policy is appropriate for pages that change infrequently, high-impact pages that undergo frequent alterations (e.g., `www.cnn.com`) should be revisited more often. Moreover, priority should be placed on crawling those new URLs that have the highest potential to affect search results. This potential may, for instance, be estimated by processing existing query logs and analyzing user behavior (e.g., clickthrough data). It is the role of the refresh policy to determine revisit rates for existing pages and the crawl order for new URLs.

The Web changes continuously. Ntoulas et al. (2004) tracked the changes to 154 Web sites over the course of a year and estimated that new pages were created at a rate of 8% per week and new links at a rate of 25% per week. The deletion rate was also high. After one year only 20% of the initial pages remained available. However, once created, most pages changed very little until they were deleted. Even after a year less than half of the pages changed by more than 5% before deletion, as determined through a measure based on TF-IDF. When pages do change, Cho and Garcia-Molina (2003) demonstrate that the frequency of these changes may be modeled by the Poisson distribution (see page 268). Thus, if we know the history of a page, we can predict how often future changes will take place.

Even when a page does change frequently, the determination of the revisit rate depends on the nature of the changes. Olston and Pandey (2008) recognize that different parts of a Web page have different update characteristics, and these characteristics must be considered when establishing a refresh policy for that page. Some parts of a Web page may remain static, while other parts exhibit *churning behavior*. For example, the ads on a page may change each time it is accessed, or a page may update daily with a "quote of the day" while its other content remains the same. Blogs and forums may exhibit *scrolling behavior*, with new items pushing down older items that eventually fall away. Some pages, such as the home pages of news services, may have the bulk of their content revised several times an hour.

Both revisit rates and the ordering of unvisited URLs must be determined in light of the impact they have on search results. Pandey and Olston (2008) define impact in terms of the number of times a page appears, or would appear, in the top results for the queries received by the search engine. Their work suggests that when choosing a new URL to visit, priority should be given to URLs that are likely to improve the search results for one or more queries. Similarly, we may consider impact when determining revisit rates. It might be appropriate to revisit a news service every hour or more, because the top news services change constantly and have high impact. A user searching on the topic of a breaking story wants to know the latest news. On the other hand, a daily visit to a page featuring a "quote of the day" may be unnecessary if the rest of the page remains unchanged and the searches returning the page as a top result depend only on this static content.

Impact is related to static rank. Pages with high static rank may often be good candidates for frequent visits. However, impact is not equivalent to static rank (Pandey and Olston, 2008). For example, a page may have high static rank because it is linked by other pages on the same site with higher static rank. However, if it always appears well below other pages from the same site in search results, changes to that page will have little or no impact. On the other hand, a page may have a low static rank because its topic is relatively obscure, of interest only to a small subset of users. But for those users the absence of the page would have high impact on their search results.

15.6.3 Duplicates and Near-Duplicates

Roughly 30–40% of Web pages are exact duplicates of other pages and about 2% are near-duplicates (Henzinger, 2006). The volume of this data represents a major problem for search engines because retaining unnecessary duplicates increases storage and processing costs. More important, duplicates and near-duplicates can impact novelty and lead to less-than-desirable search results. Although host crowding or similar post-retrieval filtering methods might ameliorate the problem, the performance costs must still be paid and near-duplicates may still slip past the filter.

Detecting exact duplicates of pages is relatively straightforward. The entire page, including tags and scripts, may be passed to a hash function. The resulting integer hash value may then be compared with the hash value for other pages. One possible hash function for detecting duplicates is provided by the MD5 algorithm (Rivest, 1992). Given a string, MD5 computes a 128-bit "message digest" corresponding to that string. MD5 is often used to validate the integrity of files after a data transfer, providing a simple way of checking that the entire file was transferred correctly. With a length of 128 bits inadvertent collisions are unlikely (however, see Exercise 15.11).

Detection of exact duplicates through these hash values is sufficient to handle many common sources of duplication, even though pages will match only if they are byte-for-byte the same. Pages mirrored across multiple sites, such as the Java documentation, may be identified in this way. Many sites will return the same "Not Found" page for every invalid URL. Some URLs that contain a user or session id may represent the same content regardless of its value. Ideally, exact duplicates should be detected during the crawling processing. Once a mirrored page is detected, other mirrored pages linked from it can be avoided.

From the perspective of a user, two pages may be considered duplicates because they contain essentially the same material even if they are not byte-for-byte copies of one another. To detect these near-duplicates, pages may be canonicalized into a stream of tokens, reducing their content to a standard form much as we do for indexing. This canonicalization usually includes the removal of tags and scripts. In addition we may remove punctuation, capitalization, and extraneous white space. For example, the HTML document of Figure 8.1 (page 278) might be canonicalized to:

> william shakespeare wikipedia the free encyclopedia william shakespeare william shakespeare baptised 26 April 1564 died 23 April 1616 was an english poet and playwright he is widely regarded as the...

After canonicalization a hash function may be applied to detect duplicates of the remaining content.

However, near-duplicates often contain minor changes and additions, which would not be detected by a hash value over the entire page. Menus, titles, and other boilerplate may differ. Material copied from one site to another may have edits applied. Detecting these near-duplicates requires that we compare documents by comparing the substrings they contain. For example, a newswire article appearing on multiple sites will have a large substring (the news story) in common, but other parts of the pages will differ.

The degree to which a page is a copy of another may be measured by comparing the substrings they have in common. Consider a newswire story appearing on two sites; the page on the first site is 24 KB in size and the page on the second site is 32 KB, after canonicalization. If the story itself has a size of 16 KB, then we might compute the similarity of the two pages in terms of this duplicate content as

$$\frac{16 \text{ KB}}{24 \text{ KB} + 32 \text{ KB} - 16 \text{ KB}} = 50\%.$$

Thus, the news story represents 50% of the combined content of the pages.

Broder et al. (1997) describe a method to permit the computation of content similarity and the detection of near-duplicates on the scale of the Web. Their method extracts substrings known as *shingles* from each canonicalized page and compares pages by measuring the overlap between these shingles. We provide a simplified presentation of this method; full details can be found in Broder et al. (1997) and in related papers (Henzinger, 2006; Bernstein and Zobel, 2005; Charikar, 2002).

The shingles of length w from a page (the w-shingles) consist of all substrings of length w tokens appearing in the document. For example, consider the following three lines from *Hamlet*:

#1: To be, or not to be: that is the question
#2: To sleep: perchance to dream: ay, there's the rub
#3: To be or not to be, ay there's the point

The first two lines are taken from the standard edition of Shakespeare's works used throughout the book. The last is taken from what may be a "pirated" copy of the play, reconstructed from memory by a minor actor.[11]

[11] `internetshakespeare.uvic.ca/Library/SLT/literature/texts+1.html` (accessed Dec 23, 2009)

After canonicalization we have the following 2-shingles for each line, where the shingles have been sorted alphabetically and duplicates have been removed:

#1: be or, be that, is the, not to, or not, that is, the question, to be
#2: ay there, dream ay, perchance to, s the, sleep perchance, the rub, there s, to dream, to sleep
#3: ay there, be ay, be or, not to, or not, s the, the point, there s, to be

For this example we use $w = 2$. For Web pages $w = 11$ might be an appropriate choice (Broder et al., 1997).

Given two sets of shingles A and B, we define the *resemblance* between them according to the number of shingles they have in common

$$\frac{|A \cap B|}{|A \cup B|}. \tag{15.30}$$

Resemblance ranges between 0 and 1 and indicates the degree to which A and B contain duplicate content. To simplify the computation of resemblance, we calculate a hash value for each shingle; 64-bit hash values are usually sufficient for this purpose (Henzinger, 2006). Although the range of a 64-bit value is small enough that the Web is likely to contain different shingles that hash to the same value, pages are unlikely to contain multiple matching shingles unless they contain duplicate content. For the purpose of our example we assign 8-bit hash values arbitrarily, giving the following sets of shingle values:

#1: 43, 14, 109, 204, 26, 108, 154, 172
#2: 132, 251, 223, 16, 201, 118, 93, 197, 217
#3: 132, 110, 43, 204, 26, 16, 207, 93, 172.

A Web page may consist of a large number of shingles. One way of reducing this number is to eliminate all but those that are equal to 0 $\mod m$, where $m = 25$ might be appropriate for Web data (Broder et al., 1997). Another approach is to keep the smallest s shingles. Resemblance is then estimated by comparing the remaining shingles. For our example we are following the first approach with $m = 2$, leaving us with the even-valued shingles:

#1: 14, 204, 26, 108, 154, 172
#2: 132, 16, 118
#3: 132, 110, 204, 26, 16, 172.

For this example it is easy to make pairwise comparisons between the sets of shingles. However, for billions of Web documents pairwise comparisons are not feasible. Instead, we construct tuples consisting of pairs of the form

$\langle$ *shingle value*, *document id* $\rangle$

and sort the pairs by shingle value (essentially constructing an inverted index). For our example this gives the following tuples:

$\langle 14, 1\rangle$, $\langle 16, 2\rangle$, $\langle 16, 3\rangle$, $\langle 26, 1\rangle$, $\langle 26, 3\rangle$ $\langle 108, 1\rangle$, $\langle 110, 3\rangle$, $\langle 118, 2\rangle$, $\langle 132, 2\rangle$, $\langle 132, 3\rangle$, $\langle 154, 1\rangle$, $\langle 172, 1\rangle$, $\langle 172, 3\rangle$, $\langle 204, 1\rangle$, $\langle 204, 3\rangle$.

Our next step is to combine tuples with the same shingle value to create tuples of the form

$\langle$ *id1*, *id2* $\rangle$,

where each pair indicates that the two documents share a shingle. In constructing these pairs we place the lower-valued document identifier first. The actual shingle values may now be ignored because there will be one pair for each shingle the documents have in common. For our example we have the following pairs:

$\langle 2, 3\rangle$, $\langle 1, 3\rangle$, $\langle 2, 3\rangle$, $\langle 1, 3\rangle$, $\langle 1, 3\rangle$.

Sorting and counting gives triples of the form

$\langle$ *id1*, *id2*, *count* $\rangle$.

For our example these triples are

$\langle 1, 3, 3\rangle$, $\langle 2, 3, 2\rangle$.

From these triples we may estimate the resemblance between #1 and #3 as

$$\frac{3}{6+6-3} = \frac{1}{3},$$

and the resemblance between #2 and #3 as

$$\frac{2}{3+6-2} = \frac{2}{7}.$$

The resemblance between #1 and #2 is 0.

15.7 Summary

Although this chapter is long and covers a wide range of topics, three factors occur repeatedly: scale, structure, and users.

- The Web contains an enormous volume of material on all topics and in many languages, and is constantly growing and changing. The scale of the Web implies that for many queries there are large numbers of relevant pages. With so many relevant pages, factors such as novelty and quality become important considerations for ranking. The same query will mean different things to different users. Maintaining an accurate and appropriate index for millions of sites requires substantial planning and resources.
- The structure represented by HTML tags and links provides important features for ranking. Link analysis methods have been extensively explored as methods for determining the relative quality of pages. The weighting of terms appearing in titles and anchor text may be adjusted to reflect their status. Without links, Web crawling would be nearly impossible.
- User experience drives Web search. Search engines must be aware of the range of interpretations and intent underlying user queries. Informational queries require different responses than navigational queries, and search engines must be evaluated with respect to their performance on both query types. Clickthroughs and other implicit user feedback captured in search engine logs may be analyzed to evaluate and improve performance.

15.8 Further Reading

A Web search engine is a mysterious and complex artifact that undergoes constant tuning and enhancement by large teams of dedicated engineers and developers. In this chapter we have covered only a small fraction of the technology underlying Web search.

The commercial importance of Web search and Web advertising is now significant enough that a large community of search engine marketing and search engine optimization (SEO) companies have grown around it. These SEOs advise Web site owners on how to obtain higher rankings from the major engines by means both fair and (occasionally) foul. Members of this community often blog about their work, and the technical tidbits appearing in these blogs can make for fascinating reading. A good place to start is the Search Engine Roundtable.[12] The personal blog of Matt Cutts, the head of Google's Webspam team, is another good entry point.[13]

[12] `www.seroundtable.com`

[13] `www.mattcutts.com`

The first Web search engines appeared in the early 1990s, soon after the genesis of the Web itself. By the mid-1990s commercial search services, such as Excite, Lycos, Altavista, and Yahoo!, were handling millions of queries per day. A history of these early search engines may be found at Search Engine Watch, a site that has tracked the commercial aspects of search engine technology since 1996.[14] Along with the content-based features discussed in Part III, these early engines incorporated simple link-based features — for example, using a page's in-degree as an indicator of its popularity (Marchiori, 1997).

15.8.1 Link Analysis

By the late 1990s the importance of static ranking and link analysis was widely recognized. In 1997 Marchiori outlined a link analysis technique that foreshadowed the intuition underlying PageRank. In the same year Carrière and Kazman (1997) presented a tool for exploring the linkage relationships between Web sites.

On 15 April 1998, at the 7th World Wide Web Conference in Brisbane, Australia, Brin and Page presented their now-classic paper describing the basic PageRank algorithm and the architecture of their nascent Google search engine (Brin and Page, 1998). In papers published soon afterwards, they and their colleagues built upon this work, describing foundational algorithms for personalized PageRank and Web data mining (Brin et al., 1998; Page et al., 1999). While Brin and Page were creating PageRank (and the Google search engine), Kleinberg independently developed the HITS algorithm and presented it at the 9th Annual Symposium on Discrete Algorithms in January 1998 (Kleinberg, 1998, 1999). A paper co-authored by Kleinberg, extending HITS to accommodate anchor text, was presented at the 7th World Wide Web Conference on the same day as Brin and Page's paper (Chakrabarti et al., 1998). Together the work of Brin, Page, and Kleinberg engendered a flood of research into link analysis techniques.

Bharat and Henzinger (1998) combined HITS with content analysis. They also recognized the problems that tightly connected communities cause for HITS and suggested the solution that Lempel and Moran (2000) later developed into SALSA. Rafiei and Mendelzon (2000) extended PageRank along the lines of SALSA to determine the topics for which a page is known (its "reputation"). Davidson (2000) recognized that some links should be assigned lower weights than others and trained a classifier to recognize links arising from commercial relationships rather than solely from merit. Cohn and Chang (2000) applied *principal component analysis* (PCA) to extract multiple eigenvectors from the HITS matrix. A number of other authors explore efficient methods for computing personalized and topic-oriented PageRank (Haveliwala, 2002; Jeh and Widom, 2003; Chakrabarti, 2007).

Ng et al. (2001a,b) compare the stability of HITS and PageRank, demonstrating the role of PageRank's jump vector and suggesting the addition of a jump vector to HITS. Borodin et al. (2001) provide a theoretical analysis of HITS and SALSA and suggest further improvements.

[14] `searchenginewatch.com/showPage.html?page=3071951` (accessed Dec 23, 2009)

Richardson and Domingos (2002) describe a query-dependent version of PageRank in which the random surfer is more likely to follow links to pages related to the query. Kamvar et al. (2003) present a method for accelerating the computation of PageRank.

More recently, Langville and Meyer (2005, 2006) provide a detailed and readable survey of the mathematics underlying PageRank and HITS. Bianchini et al. (2005) explore further properties of PageRank. Craswell et al. (2005) discuss the extension of BM25F to incorporate static ranking. Baeza-Yates et al. (2006) generalize the role of the jump vector and replace it with other damping functions to create a family of algorithms related to PageRank. Najork et al. (2007) compare the retrieval effectiveness of HITS and basic PageRank, both alone and in combination with BM25F, illustrating the limitations of basic PageRank. In a related study Najork (2007) compares the retrieval effectiveness of HITS and SALSA, finding that SALSA outperforms HITS as a static ranking feature.

Gyöngyi et al. (2004) present a link analysis technique called *TrustRank* for identifying Web spam. A related paper by Gyöngyi and Garcia-Molina (2005) provides a general overview and discussion of the Web spam problem. In addition to link analysis techniques, content-oriented techniques such as the e-mail spam filtering methods described in Chapter 10 may be applied to identify Web spam. The AIRWeb[15] workshops provide a forum for the testing and evaluating methods for detecting Web spam as part of a broader theme of adversarial IR on the Web.

Golub and Van Loan (1996) is the bible of numerical methods for matrix computations. They devote two long chapters to the solution of eigenvalue problems, including a thorough discussion of the power method.

15.8.2 Anchor Text

Anchor text was used as a ranking feature in the earliest Web search engines (Brin and Page, 1998). Craswell et al. (2001) demonstrate its importance in comparison to content features. Robertson et al. (2004) describe the incorporation of anchor text into the probabilistic retrieval model. Hawking et al. (2004) present a method for attenuating weights for anchor text, adjusting term frequency values when anchor text is repeated many times.

15.8.3 Implicit Feedback

Joachims and Radlinski (2007) provide an overview of methods for interpreting implicit feedback; Kelly and Teevan (2003) provide a bibliography of early work. Dupret et al. (2007), Liu et al. (2007), and Carterette and Jones (2007) all discuss the use of clickthroughs for Web search evaluation. Agichtein et al. (2006b) learn relevance judgments by combining multiple browsing and clickthrough features. Qiu and Cho (2006) present a method for determining a user's interests from clickthroughs, thus allowing search results to be personalized to reflect these interests.

[15] `airweb.cse.lehigh.edu`

15.8.4 Web Crawlers

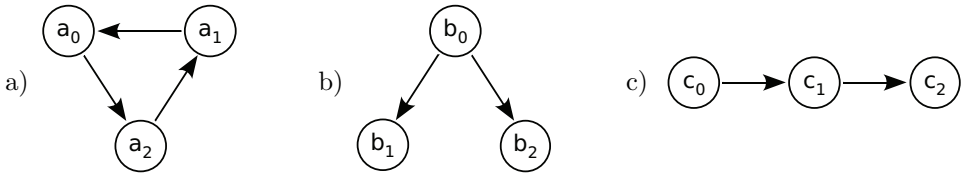
A basic architecture for a Web crawler is described by Heydon and Najork (1999). Olston and Najork (2010) provide a recent and thorough survey of the area.

The problem of optimizing the refresh policy of an incremental crawler has been studied by several groups (Edwards et al., 2001; Wolf et al., 2002; Cho and Garcia-Molina, 2003; Olston and Pandey, 2008). Dasgupta et al. (2007) examine the trade-offs between visiting new URLs and revisiting previously crawled pages. Pandey and Olston (2008) consider the impact of new pages on search results. Chakrabarti et al. (1999) describe focused crawlers that are dedicated to crawling pages related to a specific topic. Little has been published regarding the efficient implementation of priority queues for Web crawlers, but Yi et al. (2003) provide a possible starting point for investigation.

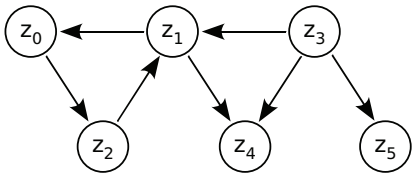
Broder et al. (1997) introduced shingles as a method for detecting near-duplicates. Henzinger (2006) experimentally compared this method with a competing method by Charikar (2002) and introduced a combined algorithm that outperforms both. Bernstein and Zobel (2005) used a version of shingling to evaluate the impact of near-duplicates on retrieval effectiveness.

15.9 Exercises

Exercise 15.1 Compute basic PageRank for the following Web graphs. Assume $\delta = 3/4$.



Exercise 15.2 Compute basic PageRank for the following Web graph. Assume $\delta = 0.85$.



Exercise 15.3 Given Equations 15.9 and 15.10, show that for all $n \geq 0$

$$\sum_{\alpha \in \Phi} r^{(n)}(\alpha) \;=\; N.$$

Exercise 15.4 Compute extended PageRank for the follow matrix and jump vector in Equation 15.12 (page 527). Assume $\delta = 0.85$.

Exercise 15.5 Show that the transition matrix M' (page 533) is aperiodic for any Web graph.

Exercise 15.6 Suggest additional sources of information, beyond those listed on page 527, that might be enlisted when setting the follow matrix and jump vector

Exercise 15.7 Compute the co-citation matrix $A = W^T W$ and the bibliographic coupling matrix $H = WW^T$ for the adjacency matrix W in Equation 15.24 (page 535).

Exercise 15.8 Compute the authority vector $\vec{a}$ and hub vector $\vec{h}$ for the adjacency matrix W in Equation 15.24 (page 535).

Exercise 15.9 Langville and Meyer (2005) suggest the following example to illustrate the convergence properties of HITS. Compute the authority vector $\vec{a}$ for the following adjacency matrix, starting with the initial estimate $\vec{a}^{(0)} = \langle\, 1/2, 1/2, 1/2, 1/2 \,\rangle^T$.

$$W = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \end{pmatrix} \tag{15.31}$$

Recompute the authority vector $\vec{a}$ starting with the initial estimate $\vec{a}^{(0)} = \langle\, 1/\sqrt{3}, 1/3, 1/3, 2/3 \,\rangle^T$.

Exercise 15.10 View the `robots.txt` file on well-known Web sites. Are any pages or crawlers disallowed? Why?

Exercise 15.11 The MD5 algorithm is known to be insecure. Given an MD5 value for a string, it is possible to construct a different string that gives the same value. How could this vulnerability be exploited by a malicious site to cause problems for a Web search service? Search this topic and suggest possible solutions.

Exercise 15.12 Search the term "Google bombing". Suggest ways in which a search engine might cope with this problem.

Exercise 15.13 (project exercise) Using the technique described by Bharat and Broder (1998), as updated by Gulli and Signorini (2005), estimate the size of the indexable Web.

Exercise 15.14 (project exercise) Build a "crawler trap", which generates dynamic context to make a Web site appear much larger than it really is (billions of pages). The trap should generate random pages, containing random text (Exercise 1.13) with random links to other random pages in the trap. URLs for the trap should have seeds for a random number generator embedded within them, so that visiting a page will consistently give the same contents.

Warning: Deploying the trap on a live Web site may negatively impact search results for that site. Be cautious. Ask for permission if necessary. If you do deploy the trap, we suggest that you add a `robots.txt` entry to steer well-behaved crawlers away from it.

15.10 Bibliography

Agichtein, E., Brill, E., and Dumais, S. (2006a). Improving Web search ranking by incorporating user behavior information. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 19–26. Seattle, Washington.

Agichtein, E., Brill, E., Dumais, S., and Ragno, R. (2006b). Learning user interaction models for predicting Web search result preferences. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 3–10. Seattle, Washington.

Baeza-Yates, R., Boldi, P., and Castillo, C. (2006). Generalizing PageRank: Damping functions for link-based ranking algorithms. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 308–315. Seattle, Washington.

Bernstein, Y., and Zobel, J. (2005). Redundant documents and search effectiveness. In *Proceedings of the 14th ACM International Conference on Information and Knowledge Management*, pages 736–743. Bremen, Germany.

Bharat, K., and Broder, A. (1998). A technique for measuring the relative size and overlap of public Web search engines. In *Proceedings of the 7th International World Wide Web Conference*, pages 379–388. Brisbane, Australia.

Bharat, K., and Henzinger, M. R. (1998). Improved algorithms for topic distillation in a hyperlinked environment. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 104–111. Melbourne, Australia.

Bianchini, M., Gori, M., and Scarselli, F. (2005). Inside PageRank. *ACM Transactions on Internet Technology*, 5(1):92–128.

Borodin, A., Roberts, G. O., Rosenthal, J. S., and Tsaparas, P. (2001). Finding authorities and hubs from link structures on the World Wide Web. In *Proceedings of the 10th International World Wide Web Conference*, pages 415–429. Hong Kong, China.

Brin, S., Motwani, R., Page, L., and Winograd, T. (1998). What can you do with a Web in your pocket? *Data Engineering Bulletin*, 21(2):37–47.

Brin, S., and Page, L. (1998). The anatomy of a large-scale hypertextual Web search engine. In *Proceedings of the 7th International World Wide Web Conference*, pages 107–117. Brisbane, Australia.

Broder, A. (2002). A taxonomy of Web search. *ACM SIGIR Forum*, 36(2):3–10.

Broder, A. Z., Glassman, S. C., Manasse, M. S., and Zweig, G. (1997). Syntactic clustering of the Web. In *Proceedings of the 6th International World Wide Web Conference*, pages 1157–1166. Santa Clara, California.

Büttcher, S., Clarke, C. L. A., and Soboroff, I. (2006). The TREC 2006 Terabyte Track. In *Proceedings of the 15th Text REtrieval Conference*. Gaithersburg, Maryland.

Carrière, J., and Kazman, R. (1997). WebQuery: Searching and visualizing the Web through connectivity. In *Proceedings of the 6th International World Wide Web Conference*, pages 1257–1267.

Carterette, B., and Jones, R. (2007). Evaluating search engines by modeling the relationship between relevance and clicks. In *Proceedings of the 21st Annual Conference on Neural Information Processing Systems*. Vancouver, Canada.

Chakrabarti, S. (2007). Dynamic personalized PageRank in entity-relation graphs. In *Proceedings of the 16th International World Wide Web Conference*. Banff, Canada.

Chakrabarti, S., Dom, B., Raghavan, P., Rajagopalan, S., Gibson, D., and Kleinberg, J. (1998). Automatic resource list compilation by analyzing hyperlink structure and associated text. In *Proceedings of the 7th International World Wide Web Conference*. Brisbane, Australia.

Chakrabarti, S., van den Burg, M., and Dom, B. (1999). Focused crawling: A new approach to topic-specific Web resource discovery. In *Proceedings of the 8th International World Wide Web Conference*, pages 545–562. Toronto, Canada.

Charikar, M. S. (2002). Similarity estimation techniques from rounding algorithms. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing*, pages 380–388. Montreal, Canada.

Cho, J., and Garcia-Molina, H. (2000). The evolution of the Web and implications for an incremental crawler. In *Proceedings of the 26th International Conference on Very Large Data Bases*, pages 200–209.

Cho, J., and Garcia-Molina, H. (2003). Effective page refresh policies for Web crawlers. *ACM Transactions on Database Systems*, 28(4):390–426.

Chung, C., and Clarke, C. L. A. (2002). Topic-oriented collaborative crawling. In *Proceedings of the 11th International Conference on Information and Knowledge Management*, pages 34–42. McLean, Virginia.

Clarke, C. L. A., Agichtein, E., Dumais, S., and White, R. W. (2007). The influence of caption features on clickthrough patterns in Web search. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 135–142. Amsterdam, The Netherlands.

Clarke, C. L. A., Scholer, F., and Soboroff, I. (2005). The TREC 2005 Terabyte Track. In *Proceedings of the 14th Text REtrieval Conference*. Gaithersburg, Maryland.

Cohn, D., and Chang, H. (2000). Learning to probabilistically identify authoritative documents. In *Proceedings of the 17th International Conference on Machine Learning*, pages 167–174.

Craswell, N., and Hawking, D. (2004). Overview of the TREC 2004 Web Track. In *Proceedings of the 13th Text REtrieval Conference*. Gaithersburg, Maryland.

Craswell, N., Hawking, D., and Robertson, S. (2001). Effective site finding using link anchor information. In *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 250–257. New Orleans, Louisiana.

Craswell, N., Robertson, S., Zaragoza, H., and Taylor, M. (2005). Relevance weighting for query independent evidence. In *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 416–423. Salvador, Brazil.

Cucerzan, S., and Brill, E. (2004). Spelling correction as an iterative process that exploits the collective knowledge of Web users. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 293–300.

Dasgupta, A., Ghosh, A., Kumar, R., Olston, C., Pandey, S., and Tomkins, A. (2007). The discoverability of the Web. In *Proceedings of the 16th International World Wide Web Conference*. Banff, Canada.

Davidson, B. D. (2000). Recognizing nepotistic links on the Web. In *Proceedings of the AAAI-2000 Workshop on Artificial Intelligence for Web Search*, pages 23–28.

Dupret, G., Murdock, V., and Piwowarski, B. (2007). Web search engine evaluation using clickthrough data and a user model. In *Proceedings of the 16th International World Wide Web Conference Workshop on Query Log Analysis: Social and Technological Challenges*. Banff, Canada.

Edwards, J., McCurley, K., and Tomlin, J. (2001). An adaptive model for optimizing performance of an incremental Web crawler. In *Proceedings of the 10th International World Wide Web Conference*, pages 106–113. Hong Kong, China.

Golub, G. H., and Van Loan, C. F. (1996). *Matrix Computations* (3rd ed.). Baltimore, Maryland: Johns Hopkins University Press.

Gulli, A., and Signorini, A. (2005). The indexable Web is more than 11.5 billion pages. In *Proceedings of the 14th International World Wide Web Conference*. Chiba, Japan.

Gyöngyi, Z., and Garcia-Molina, H. (2005). Spam: It's not just for inboxes anymore. *Computer*, 38(10):28–34.

Gyöngyi, Z., Garcia-Molina, H., and Pedersen, J. (2004). Combating Web spam with TrustRank. In *Proceedings of the 30th International Conference on Very Large Databases*, pages 576–584.

Haveliwala, T., and Kamvar, S. (2003). *The Second Eigenvalue of the Google Matrix*. Technical Report 2003-20. Stanford University.

Haveliwala, T. H. (2002). Topic-sensitive PageRank. In *Proceedings of the 11th International World Wide Web Conference*. Honolulu, Hawaii.

Hawking, D., and Craswell, N. (2001). Overview of the TREC-2001 Web Track. In *Proceedings of the 10th Text REtrieval Conference*. Gaithersburg, Maryland.

Hawking, D., Upstill, T., and Craswell, N. (2004). Toward better weighting of anchors. In *Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 512–513. Sheffield, England.

Henzinger, M. (2006). Finding near-duplicate Web pages: A large-scale evaluation of algorithms. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and development in Information Retrieval*, pages 284–291. Seattle, Washington.

Heydon, A., and Najork, M. (1999). Mercator: A scalable, extensible web crawler. *World Wide Web*, 2(4):219–229.

Ivory, M. Y., and Hearst, M. A. (2002). Statistical profiles of highly-rated Web sites. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 367–374. Minneapolis, Minnesota.

Jansen, B. J., Booth, D., and Spink, A. (2007). Determining the user intent of Web search engine queries. In *Proceedings of the 16th International World Wide Web Conference*, pages 1149–1150. Banff, Canada.

Jeh, G., and Widom, J. (2003). Scaling personalized Web search. In *Proceedings of the 12th International World Wide Web Conference*, pages 271–279. Budapest, Hungary.

Joachims, T., Granka, L., Pan, B., Hembrooke, H., and Gay, G. (2005). Accurately interpreting clickthrough data as implicit feedback. In *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 154–161. Salvador, Brazil.

Joachims, T., and Radlinski, F. (2007). Search engines that learn from implicit feedback. *IEEE Computer*, 40(8):34–40.

Jones, R., Rey, B., Madani, O., and Greiner, W. (2006). Generating query substitutions. In *Proceedings of the 15th International World Wide Web Conference*, pages 387–396. Edinburgh, Scotland.

Kamvar, S. D., Haveliwala, T. H., Manning, C. D., and Golub, G. H. (2003). Extrapolation methods for accelerating PageRank computations. In *Proceedings of the 12th International World Wide Web Conference*, pages 261–270. Budapest, Hungary.

Kellar, M., Watters, C., and Shepherd, M. (2007). A field study characterizing web-based information-seeking tasks. *Journal of the American Society for Information Science and Technology*, 58(7):999–1018.

Kelly, D., and Teevan, J. (2003). Implicit feedback for inferring user preference: A bibliography. *ACM SIGIR Forum*, 37(2):18–28.

Kleinberg, J. M. (1998). Authoritative sources in a hyperlinked environment. In *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 668–677. San Francisco, California.

Kleinberg, J. M. (1999). Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5):604–632.

Langville, A. N., and Meyer, C. D. (2005). A survey of eigenvector methods of Web information retrieval. *SIAM Review*, 47(1):135–161.

Langville, A. N., and Meyer, C. D. (2006). *Google's PageRank and Beyond: The Science of Search Engine Rankings*. Princeton, New Jersey: Princeton University Press.

Lawrence, S., and Giles, C. L. (1998). Searching the World Wide Web. *Science*, 280:98–100.

Lawrence, S., and Giles, C. L. (1999). Accessibility of information on the Web. *Nature*, 400:107–109.

Lee, U., Liu, Z., and Cho, J. (2005). Automatic identification of user goals in Web search. In *Proceedings of the 14th International World Wide Web Conference*, pages 391–400. Chiba, Japan.

Lempel, R., and Moran, S. (2000). The stochastic approach for link-structure analysis (SALSA) and the TKC effect. *Computer Networks*, 33(1-6):387–401.

Liu, Y., Fu, Y., Zhang, M., Ma, S., and Ru, L. (2007). Automatic search engine performance evaluation with click-through data analysis. In *Proceedings of the 16th International World Wide Web Conference Workshop on Query Log Analysis: Social and Technological Challenges*, pages 1133–1134. Banff, Canada.

Marchiori, M. (1997). The quest for correct information on the Web: Hyper search engines. In *Proceedings of the 6th International World Wide Web Conference*. Santa Clara, California.

Metzler, D., Strohman, T., and Croft, W. (2006). Indri TREC notebook 2006: Lessons learned from three Terabyte Tracks. In *Proceedings of the 15th Text REtrieval Conference*. Gaithersburg, Maryland.

Najork, M., and Wiener, J. L. (2001). Breadth-first search crawling yields high-quality pages. In *Proceedings of the 10th International World Wide Web Conference*. Hong Kong, China.

Najork, M. A. (2007). Comparing the effectiveness of HITS and SALSA. In *Proceedings of the 16th ACM Conference on Information and Knowledge Management*, pages 157–164. Lisbon, Portugal.

Najork, M. A., Zaragoza, H., and Taylor, M. J. (2007). HITS on the Web: How does it compare? In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 471–478. Amsterdam, The Netherlands.

Ng, A. Y., Zheng, A. X., and Jordan, M. I. (2001a). Link analysis, eigenvectors and stability. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence*, pages 903–910. Seattle, Washington.

Ng, A. Y., Zheng, A. X., and Jordan, M. I. (2001b). Stable algorithms for link analysis. In *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 258–266. New Orleans, Louisiana.

Ntoulas, A., Cho, J., and Olston, C. (2004). What's new on the Web?: The evolution of the web from a search engine perspective. In *Proceedings of the 13th International World Wide Web Conference*, pages 1–12.

Olston, C., and Najork, M. (2010). Web crawling. *Foundations and Trends in Information Retrieval*.

Olston, C., and Pandey, S. (2008). Recrawl scheduling based on information longevity. In *Proceedings of the 17th International World Wide Web Conference*, pages 437–446. Beijing, China.

Page, L., Brin, S., Motwani, R., and Winograd, T. (1999). *The PageRank Citation Ranking: Bringing Order to the Web*. Technical Report 1999-66. Stanford InfoLab.

Pandey, S., and Olston, C. (2008). Crawl ordering by search impact. In *Proceedings of the 1st ACM International Conference on Web Search and Data Mining*. Palo Alto, California.

Qiu, F., and Cho, J. (2006). Automatic identification of user interest for personalized search. In *Proceedings of the 15th International World Wide Web Conference*, pages 727–736. Edinburgh, Scotland.

Rafiei, D., and Mendelzon, A. O. (2000). What is this page known for? Computing Web page reputations. In *Proceedings of the 9th International World Wide Web Conference*, pages 823–835. Amsterdam, The Netherlands.

Richardson, M., and Domingos, P. (2002). The intelligent surfer: Probabilistic combination of link and content information in PageRank. In *Advances in Neural Information Processing Systems 14*, pages 1441–1448.

Richardson, M., Prakash, A., and Brill, E. (2006). Beyond PageRank: Machine learning for static ranking. In *Proceedings of the 15th International World Wide Web Conference*, pages 707–715. Edinburgh, Scotland.

Rivest, R. (1992). *The MD5 Message-Digest Algorithm.* Technical Report 1321. Internet RFC.

Robertson, S., Zaragoza, H., and Taylor, M. (2004). Simple BM25 extension to multiple weighted fields. In *Proceedings of the 13th ACM International Conference on Information and Knowledge Management*, pages 42–49. Washington, D.C.

Rose, D. E., and Levinson, D. (2004). Understanding user goals in web search. In *Proceedings of 13th International World Wide Web Conference*, pages 13–19. New York.

Spink, A., and Jansen, B. J. (2004). A study of Web search trends. *Webology*, 1(2).

Upstill, T., Craswell, N., and Hawking, D. (2003). Query-independent evidence in home page finding. *ACM Transactions on Information Systems*, 21(3):286–313.

Wolf, J. L., Squillante, M. S., Yu, P. S., Sethuraman, J., and Ozsen, L. (2002). Optimal crawling strategies for Web search engines. In *Proceedings of the 11th International World Wide Web Conference*, pages 136–147. Honolulu, Hawaii.

Yi, K., Yu, H., Yang, J., Xia, G., and Chen, Y. (2003). Efficient maintenance of materialized top-k views. In *Proceedings of the 19th International Conference on Data Engineering*, pages 189–200.