19 *Web search basics*

Thus far in this book, we have considered search engines for content whose authorship is of relatively high quality; furthermore, the users of such engines tended to be relatively skilled users. In this and the following two chapters, we consider web search engines. Sections 19.1–19.3 provide some background and history to help the reader appreciate the forces that conspire to make the web chaotic, fast-changing and (from the standpoint of information retrieval) very different from the "traditional" collections studied thus far in this book. Sections 19.5 and 19.6 deal with estimating the number of documents indexed by web search engines, and the elimination of duplicate documents in web indexes, respectively. These two sections serve as background material for the following two chapters.

19.1 Background and history

The web is unprecedented in many ways: unprecedented in scale, unprecedented in the almost-complete lack of coordination in its creation, and unprecedented in the diversity of backgrounds and motives of its participants. Each of these contributes to making web search different – and generally far harder – than searching "traditional" documents.

The invention of hypertext, envisioned by Vannevar Bush in the 1940's and realized in working systems in the 1970's, significantly precedes the formation of the World Wide Web (which we will simply refer to as the web), in the 1990's. Web usage has shown tremendous growth to the point where it now claims a good fraction of humanity as participants, by relying on a simple, open client-server design: (1) the server communicates with the client via a protocol (the *http* or hypertext transfer protocol) that is lightweight and simple, asynchronously carrying a variety of payloads (text, images and – over time – richer media such as audio and video files) encoded in a simple markup language called *html* (for hypertext markup language); (2) the client – generally a *browser*, an application within a graphical user environment –

HTTP

HTML

can ignore what it does not understand. Each of these seemingly innocuous features has contributed enormously to the growth of the web, so it is worthwhile to examine them further.

URL

The basic operation is as follows: a client (such as a browser) sends an *http request* to a *web server*. The browser specifies a *URL* (for *Universal Resource Locator*) such as `http://www.stanford.edu/home/atoz/contact.html`. In this example URL, the string `http` refers to the protocol to be used for transmitting the data. The string `www.stanford.edu` is known as the *domain* (sometimes the *top-level domain*) and specifies the root of a hierarchy of web pages (typically mirroring a filesystem hierarchy underlying the web server). In this example, `/home/atoz/contact.html` is a path in this hierarchy with a file `contact.html` that contains the information to be returned by the web server at `www.stanford.edu` in response to this request. The html-encoded file `contact.html` holds the hyperlinks and the content (in this instance, contact information for Stanford University), as well as formatting rules for rendering this content in a browser. Such an http request thus allows us to fetch the content of a page, something that will prove to be useful to us for crawling and indexing documents (Chapter 20).

The designers of the first browsers made it easy to view the html markup tags on the content of a URL. This simple convenience allowed new users to create their own html content without extensive training and experience; rather, they learned from example content that they liked. As they did so, a second feature of browsers supported the rapid proliferation of web content creation and usage: browsers ignored what they did not understand. This did not, as one might fear, lead to the creation of numerous incompatible dialects of html. What it did promote was amateur content creators who could freely experiment with and learn from their newly created web pages without fear that a simple syntax error would "bring the system down". Publishing on the web became a mass activity that was not limited to a few trained programmers, but rather open to tens and eventually hundreds of millions of individuals. For most users and for most information needs, the web quickly became the best way to supply and consume information on everything from rare ailments to subway schedules.

The mass publishing of information on the web is essentially useless unless this wealth of information can be discovered and consumed by other users. Early attempts at making web information "discoverable" fell into two broad categories: (1) full-text index search engines such as Altavista, Excite and Infoseek and (2) taxonomies populated with web pages in categories, such as Yahoo! The former presented the user with a keyword search interface supported by inverted indexes and ranking mechanisms building on those introduced in earlier chapters. The latter allowed the user to browse through a hierarchical tree of category labels. While this is at first blush a convenient and intuitive metaphor for finding web pages, it has a number

of drawbacks: first, classifying web pages into taxonomy tree nodes is for the most part a manual editorial process, which is difficult to scale with the size of the web. Arguably, we only need to have "high-quality" web pages in the taxonomy, with only the best web pages for each category. However, just discovering these and classifying them accurately and consistently into the taxonomy entails significant human effort. Further, in order for a user to effectively discover web pages classified into the nodes of the taxonomy tree, the user's idea of what sub-tree(s) to seek for a particular topic should match that of the editors performing the classification. This quickly becomes challenging as the size of the taxonomy grows; the Yahoo! taxonomy tree surpassed 1000 distinct nodes fairly early on. Given these challenges, the popularity of taxonomies declined over time, even though variants (such as About.com and the Open Directory Project) sprang up with subject-matter experts collecting and annotating web pages for each category.

The first generation of web search engines transported classical search techniques such as those in the preceding chapters to the web domain, focusing on the challenge of scale. The earliest web search engines had to contend with indexes containing millions of documents, which was a few orders of magnitude larger than any prior information retrieval system in the public domain. Indexing, query serving and ranking at this scale required the harnessing together of tens of machines to create highly available systems, again at scales not witnessed hitherto in a consumer-facing search application. The first generation of web search engines was largely successfully at solving these challenges while continually indexing a significant fraction of the web, all the while serving queries with sub-second response times. However, the quality and relevance of web search results left much to be desired owing to the idiosyncracies of content creation on the web. This necessitated the invention of new ranking and spam-fighting techniques in order to ensure the quality of the search results.

19.2 Web characteristics

The essential feature that led to the explosive growth of the web – decentralized content publishing with essentially no central control of authorship – turned out to be the biggest challenge for web search engines in their quest to index and retrieve this content. Web page authors created content in dozens of (natural) languages and thousands of dialects, thus demanding many different forms of stemming and other linguistic operations. Because publishing was now open to tens of millions, web pages exhibited heterogeneity at a daunting scale, in many crucial aspects. First, content-creation was no longer the privy of editorially-trained writers; while this represented a tremendous democratization of content creation, it also resulted in a tremendous varia-

tion in grammar and style (and in many cases, no recognizable grammar or style). Indeed, web publishing in a sense unleashed the best and worst of desktop publishing on a planetary scale, so that pages quickly became riddled with wild variations in colors, fonts and structure. Some web pages, including the professionally created home pages of some large corporations, consisted entirely of images (which, when clicked, led to richer textual content) – and therefore, no indexable text.

What about the substance of the text in web pages? The democratization of content creation on the web meant a new level of granularity in *opinion* on virtually any subject. This meant that the web contained truth, lies, contradictions and suppositions on a grand scale. This gives rise to the question: which web pages does one trust? In a simplistic approach, one might argue that some publishers are trustworthy and others not – begging the question of how a search engine is to assign such a measure of trust to each website or web page. In Chapter 21 we will examine approaches to understanding this question. More subtly, there may be no universal, user-independent notion of trust; a web page whose contents are trustworthy to one user may not be so to another. In traditional (non-web) publishing this is not an issue: users self-select sources they find trustworthy. Thus one reader may find the reporting of *The New York Times* to be reliable, while another may prefer *The Wall Street Journal*. But when a search engine is the only viable means for a user to become aware of (let alone select) most content, this challenge becomes significant.

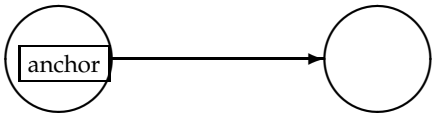
While the question "how big is the web?" has no easy answer (see Section 19.5), the question "how many web pages are in a search engine's index" is more precise, although, even this question has issues. By the end of 1995, Altavista reported that it had crawled and indexed approximately 30 million *static web pages*. Static web pages are those whose content does not vary from one request for that page to the next. For this purpose, a professor who manually updates his home page every week is considered to have a static web page, but an airport's flight status page is considered to be dynamic. Dynamic pages are typically mechanically generated and one sign of such a page is that the URL has the character "?" in it. Since the number of static web pages was believed to be doubling every few months in 1995, engines such as Altavista had to constantly add hardware and bandwidth for crawling and indexing web pages.

STATIC WEB PAGES

19.2.1 The web graph

We can view the static web consisting of static html pages together with the hyperlinks between them as a directed graph in which each web page is a node and each hyperlink a directed edge.

Figure 19.1 shows two nodes A and B from the web graph, each corre-



► **Figure 19.1** Two nodes of the web graph joined by a link.

sponding to a web page, with a hyperlink from A to B. We refer to the set of all such nodes and directed edges as the web graph. Figure 19.1 also shows that (as is the case with most links on web pages) there is some text surrounding the origin of the hyperlink on page A. This text is generally encapsulated in the `href` tag that encodes the hyperlink in the html code of page A, and is referred to as *anchor text*. As one might suspect, this directed graph is not *strongly connected*: there are pairs of pages such that one cannot proceed from one page of the pair to the other by following hyperlinks. We refer to the hyperlinks into a page as *in-links* and those out of a page as *out-links*. The number of in-links to a page (also known as its *in-degree*) has averaged from roughly 8 to 15, in a range of studies. We similarly define the out-degree of a web page to be the number of links out of it. There is ample evidence that these links are not randomly distributed; for one thing, the distribution of the number of links into a web page does not follow the Poisson distribution one would expect if every web page were to pick the destinations of its links uniformly at random. Rather, this distribution is widely reported to be a *power law*, in which the total number of web pages with in-degree i is proportional to $1/i^{\alpha}$; the value of α typically reported by studies is 2.1.[1]

ANCHOR TEXT

IN-LINKS

OUT-LINKS

POWER LAW

Exercise 19.1

If the number of pages with in-degree i is proportional to $1/i^{2.1}$, write down the probability that a randomly chosen web page has in-degree 1.

Exercise 19.2

If the number of pages with in-degree i is proportional to $1/i^{2.1}$, what is the average in-degree of a web page?

Exercise 19.3

If the number of pages with in-degree i is proportional to $1/i^{2.1}$, then as the largest in-degree goes to infinity, does the fraction of pages with in-degree i grow, stay the

1. Cf. Zipf's law of the distribution of words in text in Chapter 5 (page 79), which is a power law with $\alpha = 1$.

same, or diminish? How would your answer change for values of the exponent other than 2.1?

Exercise 19.4

The average in-degree of all nodes in a snapshot of the web graph is 9. What can we say about the average out-degree of all nodes in this snapshot?

19.2.2 Spam

Early in the history of web search, it became clear that web search engines were an important means for connecting advertisers to prospective buyers. A user searching for maui golf real estate is not merely seeking news or entertainment on the subject of housing on golf courses on the island of Maui, but instead likely to be seeking to purchase such a property. Sellers of such property and their agents, therefore, have a strong incentive to create web pages that rank highly on such a query. In a search engine whose scoring was based on term frequencies, a web page with numerous repetitions of maui golf real estate would rank highly. This led to the first generation of *spam*, which (in the context of web search) is the manipulation of web page content for the purpose of appearing high up in search results for selected keywords. To avoid irritating users with these repetitions, sophisticated *spammers* resorted to such tricks as rendering these repeated terms in the same color as the background. Despite these words being consequently invisible to the human user, a search engine indexer would parse the invisible words out of the html representation of the web page and index these words as being present in the page.

SPAM

At its root, spam stems from the heterogeneity of motives in content creation on the web. In particular, many web content creators have commercial motives and therefore stand to gain from manipulating search engine results. You might argue that this is no different from a company that uses large fonts to list its phone numbers in the yellow pages; but this generally costs the company more and is thus a fairer mechanism. A more apt analogy, perhaps, is the use of company names beginning with a long string of A's to be listed early in a yellow pages category. In fact, the yellow pages' model of companies paying for larger/darker fonts has been replicated in web search: in many engines, it is possible to pay to have one's web page included in the engine's search index – a model known as *paid inclusion*. Different engines have different policies on whether to allow paid inclusion, and whether such a payment has any effect on ranking in search results.

PAID INCLUSION

Search engines soon became sophisticated enough in their spam detection to screen out an unusually large number of repetitions of particular keywords. Spammers responded with a richer set of spam techniques, the best known of which we now describe. The first of these techniques is *cloaking*: the spammer's web server returns different pages depending on whether

the http request comes from a web search engine's crawler, or from a human user's browser. The former causes the web page to be indexed by the search engine under misleading keywords. When the user searches for these keywords and elects to view the page, he receives a web page that has altogether different content than that indexed by the engine. Such deception of search indexers is unknown in the traditional world of information retrieval; it stems from the fact that the web is partly collaborative but also partly competitive.

A *doorway page* contains text and meta-data carefully chosen to rank highly on selected search keywords. When a browser requests the doorway page, it is redirected to a page containing content of a more commercial nature. More complex spamming techniques involve manipulation of the meta-data related to a page including (for reasons we will see in Chapter 21) the links into a web page. Given that spamming is inherently an economically motivated activity, there has sprung around it an industry of *Search Engine Optimizers*, or SEO's to provide consultancy services for clients who seek to have their web pages rank highly on selected keywords. Web search engines frown on this business of attempting to decipher and adapt to their proprietary ranking techniques and indeed announce policies on forms of SEO behavior they do not tolerate (and have been known to shut down search requests from certain SEO's for violation of these). Inevitably, the parrying between such SEO's (who gradually infer features of each web engine's ranking methods) and the web search engines (who adapt in response) is an unending struggle; indeed, the research sub-area of *adversarial information retrieval* has sprung up around this battle. One potent technique for addressing spammers that fabricate the textual content of their web pages is the exploitation of the link structure of the web – a technique known as *link analysis*. The first web search engine to apply link analysis (to be detailed in Chapter 21) was Google, although all web search engines currently make use of it (and correspondingly, spammers now invest considerable effort in subverting it).

SEARCH ENGINE OPTIMIZERS

ADVERSARIAL INFORMATION RETRIEVAL

19.3 Advertising as the economic model

Early in the history of the web, companies used graphical banner advertisements on web pages at popular websites (news and entertainment sites such as MSN, America Online, Yahoo! and CNN). The primary purpose of these advertisements was *branding*: to convey to the viewer a positive feeling about the brand of the company placing the advertisement. Typically these advertisements were priced on a *cost per mil* (*CPM*) basis: the cost to the company of having its banner advertisement displayed 1000 times. Some websites struck contracts with their advertisers in which an advertisement was priced not by the number of times it is displayed (also known as *impressions*), but

CPM

rather by the number of times it was *clicked on* by the user. This pricing model is known as the *cost per click* (*CPC*) model. In such cases, clicking on the advertisement leads the user to a web page set up by the advertiser, where the user is induced to make a purchase. Here the goal of the advertisement is not so much brand promotion as to induce a transaction. This distinction between brand and transaction-oriented advertising was already widely recognized in the context of conventional media such as broadcast and print. The interactivity of the web allowed the CPC billing model – clicks could be metered and monitored by the website and billed to the advertiser.

CPC

However, the user at the news/entertainment website is typically not there with an intent to make a purchase, as much as to consume news and entertainment. How could a company better target its audience? Companies do in fact achieve some measure of focus by carefully selecting the websites on which they advertised. For instance, a company selling golf clubs might wish to advertise on a web page containing sports news and even better on a web pages that discuss golf news, but perhaps not on a web page announcing recent breakthroughs on biochemistry.

Demographic focusing of web advertising was already in use in the late 1990's, when web search engines were growing in usage (and therefore constantly in need of capital expenditures for hardware and bandwidth). The challenge for web search companies was to create a revenue stream that outweighed these expenditures. The pioneer in this direction was a company named Goto, which changed its name to Overture prior to eventual acquisition by Yahoo! Goto was not, in the traditional sense, a search engine; rather, for every query term q it accepted *bids* from companies who wanted their web page shown on the query q. In response to the query q, Goto would return the pages of all advertisers who bid for q, ordered by their bids. Furthermore, when the user clicked on one of the returned results, the corresponding advertiser would make a payment to Goto (in the initial implementation, this payment equalled the advertiser's bid for q).

Several aspects of Goto's model are worth highlighting. First, a user typing the query q into Goto's search interface was actively expressing an interest and intent related to the query q. For instance, a user typing golf clubs is more likely to be imminently purchasing a set than one who is simply browsing news on golf. Second, Goto only got compensated when a user actually expressed interest in an advertisement – as evinced by the user clicking the advertisement. Taken together, these created a powerful mechanism by which to connect advertisers to consumers, quickly raising the annual revenues of Goto/Overture into hundreds of millions of dollars. This style of search engine came to be known variously as *sponsored search* or *paid placement*.

SPONSORED SEARCH
PAID PLACEMENT

Given these two kinds of search engines – the "pure" engines such as Google and Altavista, versus the sponsored search engines – the logical next step was to combine them into a single user experience. Current search en-

gines follow precisely this model: they provide pure search results (generally known as *algorithmic search* results) as the primary response to a user's search, together with sponsored search results displayed separately and distinctively to the right of the algorithmic results. Retrieving sponsored search results and ranking them in response to a query has now become considerably more sophisticated than the simple Goto scheme; the process entails a blending of ideas from information retrieval and microeconomics, and is beyond the scope of this book. From the standpoint of advertisers, understanding how search engines do this ranking and how to allocate marketing campaign budgets to different sponsored search engines has become a profession known as *search engine marketing* (SEM).

ALGORITHMIC SEARCH

SEARCH ENGINE MARKETING

The inherently economic motives underlying sponsored search give rise to attempts by some participants to subvert the system to their advantage. This can take many forms, one of which is known as *click spam*. There is currently no universally accepted definition of click spam. It refers (as the name suggests) to clicks on sponsored search results that are not from bona fide search users. For instance, a devious advertiser may attempt to exhaust the advertising budget of a competitor by clicking repeatedly (through the use of a robotic click generator) on that competitor's sponsored search advertisements. Search engines face the challenge of discerning which of the clicks they observe are part of a pattern of click spam, to avoid charging their advertiser clients for such clicks.

CLICK SPAM

Exercise 19.5

The Goto method ranked advertisements matching a query by *bid*: the highest-bidding advertiser got the top position, the second-highest the next, and so on. What can go wrong with this when the highest-bidding advertiser places an advertisement that is irrelevant to the query? Why might an advertiser with an irrelevant advertisement bid high in this manner?

Exercise 19.6

Suppose that, in addition to bids, we had for each advertiser their *click-through rate*: the ratio of the historical number of times users click on their advertisement to the number of times the advertisement was shown. Suggest a modification of the Goto scheme that exploits this data to avoid the problem in Exercise 19.5 above.

19.4 The search user experience

It is crucial that we understand the users of web search as well. This is again a significant change from traditional information retrieval, where users were typically professionals with at least some training in the art of phrasing queries over a well-authored corpus whose style and structure they understood well. In contrast, web search users tend to not know (or care) about the heterogeneity of web content, the syntax of query languages and the art of phrasing queries; indeed, a mainstream tool (as web search has come to become) should not place such onerous demands on billions of people. A

range of studies has concluded that the average number of keywords in a web search is somewhere between 2 and 3. Syntax operators (Boolean connectives, wildcards, etc.) are seldom used, again a result of the composition of the audience – "normal" people, not information scientists.

It is clear that the more user traffic a web search engine can attract, the more revenue it stands to earn from sponsored search. How do search engines differentiate themselves and grow their traffic? Here Google identified two principles that helped it grow at the expense of its competitors: (1) a focus on relevance, specifically precision rather than recall in the first few results; (2) a user experience that is lightweight, meaning that both the search query page and the search results page are uncluttered and almost entirely textual, with very few graphical elements. The effect of the first was simply to save users time in locating the information they sought; more on this below. The effect of the second is to provide a user experience that is extremely responsive, or at any rate not bottlenecked by the time to load the search query or results page.

19.4.1 User query needs

There appear to be three broad categories into which common web search queries can be grouped: (i) informational, (ii) navigational and (iii) transactional. We now explain these categories; it should be clear that some queries will fall in more than one of these categories, while others will fall outside them.

INFORMATIONAL QUERIES

Informational queries seek general information on a broad topic, such as leukemia or Provence. There is typically not a single web page that contains all the information sought; indeed, users with informational queries typically try to assimilate information from multiple web pages.

NAVIGATIONAL QUERIES

Navigational queries seek the website or home page of a single entity that the user has in mind, say Lufthansa airlines. In such cases, the user's expectation is that the very first search result should be the home page of Lufthansa.

TRANSACTIONAL QUERY

A *transactional query* is one that is a prelude to the user performing a transaction on the web – such as purchasing a product, downloading a file or making a reservation. In such cases, the engine should return results listing services that provide form interfaces for such transactions.

Discerning which of these categories a query falls into can be challenging. The category not only governs the algorithmic search results, but the suitability of the query for sponsored search results (since the query may reveal an intent to purchase). For navigational queries, some have argued that the engine should return only a single result or even the target web page directly. Nevertheless, web search engines have historically engaged in a battle of bragging rights over which one indexes more web pages. Does the user really care? Perhaps not, but the media does highlight measurements

(often statistically indefensible) of the sizes of various search engines. Users are influenced by these reports and thus, search engines do have to pay attention to how their index sizes compare to competitors'. For informational (and to a lesser extent, transactional) queries, the user does care about the comprehensiveness of the engine.

19.5 Index size and estimation

To a first approximation, comprehensiveness grows with index size, although it does matter which specific pages an engine indexes – some pages are more informative than others. It is also difficult to reason about the fraction of the web indexed by an engine, because there is an infinite number of dynamic web pages; for instance, `http://www.yahoo.com/any_string` returns a valid html page rather than an error, politely informing the user that there is no such page at Yahoo! Such a "soft 404 error" is only one example of many ways in which web servers can generate an infinite number of valid web pages. Indeed, some of these are malicious spider traps devised to cause a search engine's crawler (the component that systematically fetches web pages for the engine's index, described in Chapter 20) to stay within a spammer's website and index many pages from that site.

We could ask the following better-defined question: given two search engines, what are the relative sizes of their indexes? Even this question turns out to be imprecise, because:

1. In response to queries a search engine can return web pages whose contents it has not (fully or even partially) indexed. For one thing, engines generally index only the first few thousand words in a web page. In some cases, an engine is aware of a page p that is *linked to* by pages it has indexed, but has not indexed p itself. As we will see in Chapter 21, it is still possible to meaningfully return p in search results.

2. Search engines generally organize their indexes in various tiers and partitions, not all of which are examined on every search. For instance, a web page deep inside a website may be indexed but not retrieved on general web searches; it is however retrieved as a result on a search specific to that website.

Thus, search engine indexes include multiple classes of indexed pages, so that there is no single measure of index size. These issues notwithstanding, a number of techniques have been devised for crude estimates of the ratio of the index sizes of two search engines, E_1 and E_2. The basic hypothesis underlying these techniques is that each engine indexes a fraction of the web chosen independently and uniformly at random. This involves some questionable assumptions: first, that there is a finite size for the web from which

each engine chooses a subset, and second, that each engine chooses an independent, uniformly chosen subset. As will be clear from the discussion of crawling in Chapter 20, this is far from true. However, if we begin with these assumptions, then we can invoke a classical estimation technique known as the *capture-recapture method*.

CAPTURE-RECAPTURE METHOD

Suppose that we could pick a random page from the index of E_1 and test whether it is in E_2's index and symmetrically, test whether a random page from E_2 is in E_1. These experiments give us fractions x and y such that our estimate is that a fraction x of the pages in E_1 are in E_2, while a fraction y of the pages in E_2 are in E_1. Then, letting $|E_i|$ denote the size of the index of engine E_i, we have

$$x|E_1| \approx y|E_2|,$$

from which we have the form we will use

$$\frac{|E_1|}{|E_2|} \approx \frac{y}{x}. \tag{19.1}$$

If our assumption about E_1 and E_2 being independent and uniform random subsets of the web were true, and our sampling process unbiased, then Equation (19.1) should give us an unbiased estimator for $|E_1|/|E_2|$. We distinguish between two scenarios here. Either the measurement is performed by someone with access to the index of one of the engines (say an employee of E_1), or the measurement is performed by an independent party with no access to the innards of either engine. In the former case, we can simply pick a random document from one index. The latter case is more challenging; we begin with the sampling process by which we pick a random page from one engine *from outside the engine*, then describe the checking process by which we verify whether the random page is present in the other engine.

To implement the sampling phase, we might generate a random page from the entire (idealized, finite) web and test it for presence in each engine. Unfortunately, picking a web page uniformly at random is a difficult problem. We briefly outline several attempts to achieve such a sample, pointing out the biases inherent to each; following this we describe in some detail one technique that much research has built on.

1. *Random searches:* Begin with a search log of web searches; send a random search from this log to E_1 and a random page from the results. Since such logs are not widely available outside a search engine, one implementation is to trap all search queries going out of a work group (say scientists in a research center) that agrees to have all its searches logged. This approach has a number of issues, including the bias from the types of searches made public by the work group. Further, a random document from the results of such a random search to E_1 is not the same as a random document from E_1.

2. *Random IP addresses:* A second approach is to generate random IP addresses and send a request to a web server residing at the random address, collecting all pages at that server. The biases here include the fact that many hosts might share one IP (due to a practice known as virtual hosting) or not accept http requests from the host where the experiment is conducted. Further, this technique is more likely to hit one of the many sites with few pages, skewing the document probabilities; we may be able to correct for this effect if we understand the distribution of pages on a website.

3. *Random walks:* If the web graph were a strongly connected directed graph, we could run a random walk starting at an arbitrary web page. This walk would converge to a steady state distribution (see Chapter 21, Section 21.2.1 for more background material on this), from which we could in principle pick a web page with a fixed probability. This method, too has a number of biases. First, the web is not strongly connected so that, even with various corrective rules, it is difficult to argue that we can reach a steady state distribution starting from any page. Second, the time it takes for the random walk to settle into this steady state is unknown and could exceed the length of the experiment.

Clearly each of these approaches is far from perfect. We now describe a fourth sampling approach, *random queries*. This approach is noteworthy for two reasons: it has been successfully built upon for a series of increasingly refined estimates, and conversely it has turned out to be the approach most likely to be misinterpreted and carelessly implemented, leading to misleading measurements. The idea is to pick a page (almost) uniformly at random from an engine's index by posing a random query to it. It should be clear that picking a set of random terms from (say) Webster's dictionary is not a good way of implementing this idea. For one thing, not all dictionary terms occur equally often, so this approach will not result in documents being chosen uniformly at random from the engine. For another, there are a great many terms in the web corpus that do not occur in a standard dictionary such as Webster's. To address the problem of lexicon terms not in a standard dictionary, we begin by amassing a sample web lexicon. This could be done by crawling a limited portion of the web, or by crawling a manually-assembled representative subset of the web such as Yahoo! (as was done in the earliest experiments with this method). Consider a conjunctive query with two or more randomly chosen words from this lexicon.

The probability of the event that a page is in the results set of such a random conjunctive query induces a distribution over all pages in the union of the two engines. Then, we estimate $|E_1|/|E_2|$ by taking the ratio of the corresponding induced distributions. We can improve the estimate by repeating the experiment a large number of times.

Operationally, we proceed as follows: we use a random conjunctive query on E_1 and pick from the top 100 returned results a page p at random. We then test p for presence in E_2 by choosing 6-8 low-frequency terms in p and using them in a conjunctive query for E_2. Both the sampling process and the testing process have a number of issues.

1. Our sample is biased towards longer documents.
2. Picking from the top 100 results of E_1 induces a bias from the ranking algorithm of E_1. Picking from all the results of E_1 makes the experiment slower. This is particularly so because most web search engines put up defenses against excessive robotic querying.
3. During the checking phase, a number of additional biases are introduced: for instance, E_2 may not handle 8-word conjunctive queries properly.
4. Either E_1 or E_2 may refuse to respond to the test queries, treating them as robotic spam rather than as bona fide queries.
5. There could be operational problems like connection time-outs.

A sequence of research has built on this basic paradigm to eliminate some of these issues; there is no perfect solution yet, but the level of sophistication in statistics for understanding the biases is increasing.

Exercise 19.7

Two web search engines A and B each generate a large number of pages uniformly at random from their indexes. 30% of A's pages are present in B's index, while 50% of B's pages are present in A's index. What is the number of pages in A's index relative to B's?

19.6 Near-duplicates and shingling

One aspect we have ignored in the discussion of index size in Section 19.6 is *duplication*: the web contains multiple copies of the same content. By some estimates, as many as 40% of the pages on the web are duplicates of other pages. Many of these are legitimate copies; for instance, certain information repositories are mirrored simply to provide redundancy and access reliability. Search engines try to avoid indexing multiple copies of the same content, to keep down storage and processing overheads.

The simplest approach to detecting duplicates is to compute, for each web page, a *fingerprint* that is a succinct (say 64-bit) digest of the sequence of characters on that page. Then, whenever the fingerprints of two web pages are equal, we test whether the pages themselves are equal and if so declare one of them to be a duplicate copy of the other. This simplistic approach fails

to capture a crucial and widespread phenomenon on the web: *near duplication*. In many cases, the contents of one web page are identical to those of another except for a few characters – say, a notation showing the date and time at which the page was last modified. Even in such cases, we want to be able to declare the two pages to be close enough that we only index one copy. Short of exhaustively comparing all pairs of web pages, an infeasible task at the scale of billions of pages, how can we detect and filter out such near duplicates?

Exercise 19.8

Web search engines A and B each crawl a random subset of the same size of the web. Some of the pages crawled are duplicates – exact textual copies of each other at different URLs. Assume that duplicates are distributed uniformly amongst the pages crawled by A and B. Further, assume that a duplicate is a page that has exactly two copies – no pages have more than two copies. A indexes pages without duplicate elimination whereas B indexes only one copy of each duplicate page. The two random subsets have the same size before duplicate elimination. If, 45% of A's indexed URLs are present in B's index, while 50% of B's indexed URLs are present in A's index, what fraction of the web consists of pages that do not have a duplicate?

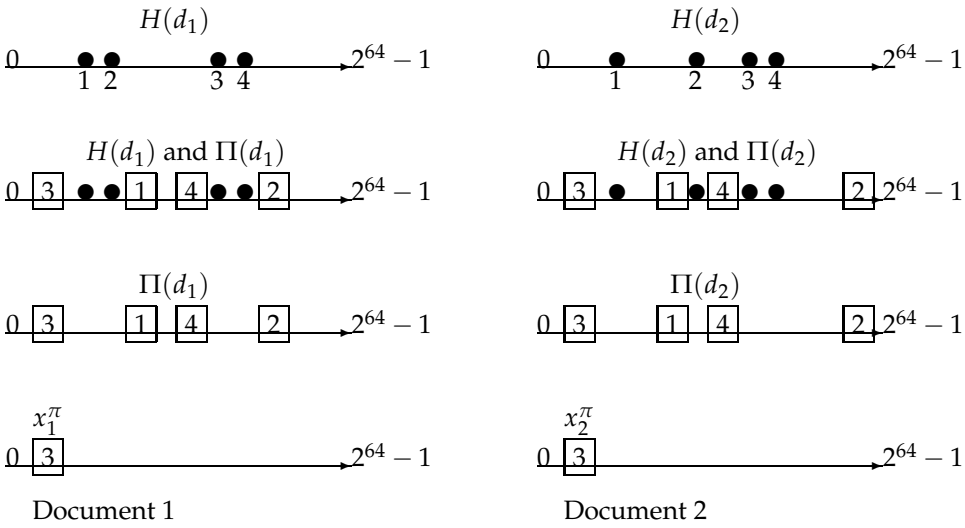
19.6.1 Shingling

SHINGLING

We now describe a solution to the problem of detecting near-duplicate web pages. The answer lies in a technique known as *shingling*. Given a positive integer k and a sequence of terms in a document d, define the k-shingles of d to be the set of all consecutive sequences of k terms in d. As an example, consider the following text: a rose is a rose is a rose. The 4-shingles for this text ($k = 4$ is a typical value used in the detection of near-duplicate web pages) are a rose is a, rose is a rose and is a rose is. The first two of these shingles each occur twice in the text. Intuitively, two documents are near duplicates if the sets of shingles generated from them are nearly the same. We now make this intuition precise, then develop a method for efficiently computing and comparing the sets of shingles for all web pages.

Let $S(d_j)$ denote the set of shingles of document d_j. Recall the Jaccard coefficient (Chapter 3, page 47), which measures the degree of overlap between the sets $S(d_1)$ and $S(d_2)$ as $|S(d_1) \cap S(d_2)|/|S(d_1) \cup S(d_2)|$; denote this by $J(S(d_1), S(d_2))$. Our test for near duplication between d_1 and d_2 is to compute this Jaccard coefficient; if it exceeds a preset threshold (say, 0.9), we declare them near duplicates and eliminate one from indexing. However, this does not appear to have simplified matters: we still have to compute Jaccard coefficients pairwise.

To avoid this, we use a form of hashing. First, we map every shingle into a hash value over a large space, say 64 bits. For $j = 1, 2$, let $H(d_j)$ be the corresponding set of 64-bit hash values derived from $S(d_j)$. We now invoke the following trick to detect document pairs whose sets $H()$ have large Jaccard overlaps. Let π be a random permutation from the 64-bit integers to the



► **Figure 19.2** Illustration of shingle sketches. We see two documents going through four stages of shingle sketch computation. In the first step (top row), we apply a 64-bit hash to each shingle from each document to obtain $H(d_1)$ and $H(d_2)$ (circles). Next, we apply a random permutation Π to permute $H(d_1)$ and $H(d_2)$, obtaining $\Pi(d_1)$ and $\Pi(d_2)$ (squares). The third row shows only $\Pi(d_1)$ and $\Pi(d_2)$, while the bottom row shows the minimum values x_1^π and x_2^π for each document.

64-bit integers. Denote by $\Pi(d_j)$ the set of permuted hash values in $H(d_j)$; thus for each $h \in H(d_j)$, there is a corresponding value $\pi(h) \in \Pi(d_j)$.

Let x_j^π be the smallest integer in $\Pi(d_j)$. Then

Theorem 19.1.

$$J(S(d_1), S(d_2)) = Pr[x_1^\pi = x_2^\pi].$$

Proof. We give the proof in a slightly more general setting: consider a family of sets whose elements are drawn from a common universe. View the sets as columns of a matrix A, with one row for each element in the universe. The element $a_{ij} = 1$ if element i is present in the set S_j that the jth column represents.

Let Π be a random permutation of the rows of A; denote by $\Pi(S_j)$ the column that results from applying Π to the jth column. Finally, let x_j^π be the index of the first row in which the column $\Pi(S_j)$ has a 1. We then prove that for any two columns j_1, j_2,

$$\Pr[x_{j_1}^\pi = x_{j_2}^\pi] = J(S_{j_1}, S_{j_2}).$$

S_{j_1}	S_{j_2}
0	1
1	0
1	1
0	0
1	1
0	1

► **Figure 19.3** Two sets S_{j_1} and S_{j_2}; their Jaccard coefficient is 2/5.

If we can prove this, the theorem follows.

Consider two columns j_1, j_2 as shown in Figure 19.3, which shows that the ordered pairs of entries of S_{j_1} and S_{j_2} partition the rows into four types: those with 0's in both of these columns, those with a 0 in S_{j_1} and a 1 in S_{j_2}, those with a 1in S_{j_1} and a 0in S_{j_2}, and finally those with 1's in both of these columns. Indeed, the first four rows of Figure 19.3 exemplify all of these four types of rows. Denote by C_{00} the number of rows of the first of these types, C_{01} the second, C_{10} the third and C_{11} the fourth. Then,

$$J(S_{j_1}, S_{j_2}) = \frac{C_{11}}{C_{01} + C_{10} + C_{11}}. \tag{19.2}$$

To complete the proof by showing that the right-hand side of (19.2) equals $\Pr[x_{j_1}^{\pi} = x_{j_2}^{\pi}]$, consider scanning columns j_1, j_2 in increasing row index until the first non-zero entry is found in either column. Because Π is a random permutation, the probability that this smallest row has a 1 in both columns is exactly the right-hand side of (19.2). □

Thus, our test for the Jaccard coefficient of the shingle sets is probabilistic: we compare the computed values x_i^{π} from different documents. If a pair coincides, we have candidate near duplicates. Repeat the test independently for 200 random permutations π (a choice suggested in the literature). Call the set of these 200 values of x_i^{π} the *sketch* $\psi(d_i)$ of d_i. We can then estimate the Jaccard coefficient for any pair of documents d_i, d_j to be $|\psi_i \cap \psi_j|/200$; if this exceeds a preset threshold, we declare that d_i and d_j are similar.

How can we quickly compute $|\psi_i \cap \psi_j|/200$ for all pairs i, j? Indeed, how do we represent all pairs of documents that are similar, without incurring a blowup that is quadratic in the number of documents? First, we use fingerprints to remove all but one copy of *identical* documents. We may also remove common html tags and integers from the shingle computation, to eliminate shingles that occur very commonly in documents without telling us anything about duplication. Next we use a *union-find* algorithm to create

clusters that contain documents that are similar. To do this, we must accomplish a crucial step: going from the set of sketches to the set of pairs i, j such that d_i and d_j are similar.

To this end, we compute the number of shingles common for any pair of documents whose sketches have any members in common. We begin with the list of sorted $< x_i^\pi, d_i >$ pairs and for each x_i^π, generate all pairs i, j for which x_i^π is present in both their sketches. We then accumulate these into counts for each pair i, j with non-zero sketch overlap; by applying the preset threshold, we know which pairs i, j have heavily overlapping sketches. For instance, if the preset threshold were 80%, we would need the merged count to be at least 160. As we identify such pairs, we run the union-find to group documents into near-duplicate "syntactic clusters". This is essentially a variant of the single-link clustering algorithm introduced in Section 17.2 (page 290).

One final trick cuts down the space needed in the computation of $|\psi_i \cap \psi_j|/200$ for pairs i, j, which in principle could still demand space quadratic in the number of documents. To remove from consideration those pairs i, j whose sketches have few shingles in common, we preprocess the sketch for each document as follows: sort the x_i^π in the sketch, then shingle this sorted sequence to generate a set of *super-shingles* for each document. If two documents have a super-shingle in common, we proceed to compute the precise value of $|\psi_i \cap \psi_j|/200$. This again is a heuristic but can be highly effective in cutting down the number of i, j pairs for which we accumulate the sketch overlap counts.

Exercise 19.9

Instead of using the process depicted in Figure 19.2, consider instead the following process for estimating the Jaccard coefficient of the overlap between two sets S_1 and S_2. We pick a random subset of the elements of the universe from which S_1 and S_2 are drawn; this corresponds to picking a random subset of the rows of the matrix A in the proof. We exhaustively compute the Jaccard coefficient of these random subsets. Why is this estimate an unbiased estimator of the Jaccard coefficient for S_1 and S_2?

Exercise 19.10

Explain why this estimator would be very difficult to use in practice.

19.7 References and further reading

Bush (1945b) foreshadowed the web when he described an information management system that he called *memex*. Berners-Lee et al. (1992) describes one of the earliest incarnations of the web. Kumar et al. (2000) and Broder et al. (2000) provide comprehensive studies of the web as a graph. The use of anchor text was first described in McBryan (1994a). The taxonomy of web queries in Section 19.4 is due to Broder (2002).

The estimation of web search index sizes has a long history of development covered by Bharat and Broder (1998), Lawrence and Giles (1998), Rusmevichientong et al. (2001), Lawrence and Giles (1999), Henzinger et al. (2000a), Henzinger et al. (2000b), Bar-Yossef and Gurevich (2006). Shingling was introduced by Broder et al. (1997) and used for detecting websites (rather than simply pages) that are identical by Bharat et al. (2000).