6

Advanced Crawling Techniques

In this chapter we refine the basic design of Web crawlers, building on the material presented in Section 2.5.

6.1 Selective Crawling

The search engine coverage experiments reported in Section 2.5.2 suggest that no crawler can realistically harvest the entire Web. An important fact is that the time required to complete a large crawl can be significant, independent of whatever technology is available at the site where the search engines operate. Moreover, fetching and indexing a larger set of documents has significant implications on the scalability of the overall system and, consequently, on the cost of the required hardware and maintenance services. Thus, to optimize available resources, a crawler should ideally be capable of recognizing the relevance or the importance of sites or pages, and limiting fetching to the most important subset of pages that can be downloaded in a given amount of time.

Relevance can be estimated with respect to several different criteria. To formalize the intuitive idea of selective crawling, we need to introduce, for each URL u, a scoring function $s_\theta^{(\xi)}(u)$, with respect to some underlying relevance criterion ξ and parameters θ. In the simplest case we can assume a Boolean relevance function, i.e. $s(u) = 1$ if the document pointed to by u is relevant and $s(u) = 0$ if the document is not interesting. More generally, we can think of $s(d)$ as a real-valued function, such as the conditional probability that the document belongs to a certain category given its contents. Note that in all of these cases the scoring function depends only on the URL and the criterion, but does not depend on the state of the crawler. Moreover, we will assume that the scoring function does not depend on time.

One general approach for building selective crawlers consists of changing the policy of insertions and extractions in the queue of discovered URLs. For simplicity, let us consider again the algorithm SIMPLE-CRAWLER on page 47. Suppose now that URLs in Q are sorted according to the value returned by $s(u)$. In this way we obtain a *best-first* search strategy (see, for example, Russell and Norvig 1995) such that URLs

Modeling the Internet and the Web P. Baldi, P. Frasconi and P. Smyth
 ISBN: 0-470-84906-1

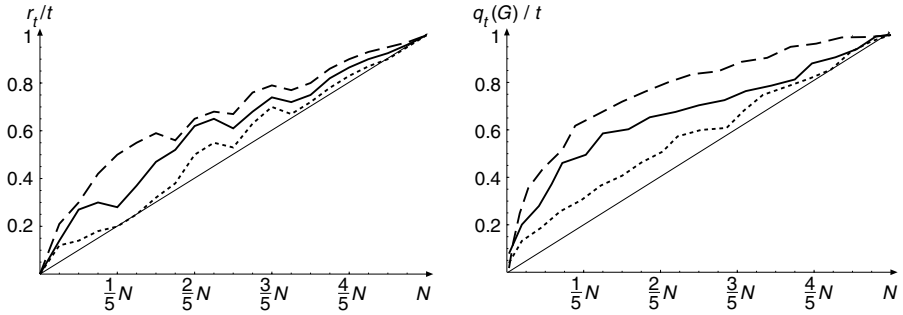

Figure 6.1 Efficiency curves obtained by Cho *et al.* (1998) for a selective crawler. Both were obtained on a set of 784 592 Stanford University URLs. Dotted lines correspond to a BFS crawler, solid lines to a crawler guided by the estimated number of backlinks, and dashed lines to a crawler guided by estimated PageRank. The diagonal lines are the reference performance of a random crawler. The target measure of relevance is the actual number of backlinks. In the right-hand plot, the importance target G is 100 backlinks.

having higher score are fetched first. If $s(u)$ provides a good model of the relevance of the document pointed to by the URL, then we expect that the search process will be guided toward the most relevant regions of the Web. From this perspective the problem is to define interesting scoring functions and to devise good algorithms to compute them.

In the following we consider in detail some specific examples of scoring functions.

Depth. As we have seen in Chapter 3, the distribution of the number of pages on different Web servers generally follows a power law. Thus, many sites have few pages and, at the same time, there is a small but significant fraction of sites containing a huge number of pages. A very simple strategy for maximizing coverage breadth consists of limiting the total number of documents downloaded from a single site, for example, by setting a threshold, or by keeping track of the depth in the site directory tree, or by limiting the acceptable length of the path from the site homepage to the document,

$$s_\delta^{(\text{depth})}(u) = \begin{cases} 1, & \text{if } |\text{root}(u) \rightsquigarrow u| < \delta, \\ 0, & \text{otherwise,} \end{cases} \tag{6.1}$$

where $\text{root}(u)$ is the root of the site containing u. The rationale behind this approach is that by maximizing breadth it will be easy for the end-user to eventually find the desired information. For example a page that is not indexed by the search engine may be easily reachable from other pages in the same site.

Popularity. It is often the case that we can define criteria according to which certain pages are more important than others. For example, search engine queries are answered by proposing to the user a sorted list of documents. Typically, users tend to inspect only the few first documents in the list (in Chapter 8 we will discuss

empirical data that support this statement). Thus, if a document rarely appears near the top of any lists, it may be worthless to download it and its importance should be small. A simple way of assigning importance to documents that are more popular is to introduce a relevance function based on the number of backlinks

$$s_\tau^{(\text{backlinks})}(u) = \begin{cases} 1, & \text{if indegree}(u) > \tau, \\ 0, & \text{otherwise,} \end{cases} \tag{6.2}$$

where τ is an assigned threshold.

Clearly $s_\tau^{(\text{backlinks})}(u)$ can be computed exactly only if we have already crawled the entire Web. In practice we can use an approximated value from the partial subgraph obtained in a previous crawl, or we can incrementally update the score as soon as new parents of u are encountered during the crawl.

PageRank. As discussed in Section 5.4, PageRank is a measure of popularity that recursively assigns each link a weight that is proportional to the popularity of the source document. It may be seen as a refinement of the indegree measure defined above in Equation (6.2). Also PageRank needs to be estimated using partial knowledge of the graph that is refined during crawling.

The efficiency of a selective crawler can be measured by comparing it to an ideal crawler that fetches documents in order of relevance (whatever the relevance function). Suppose we are given a website with N pages and for $n = 1, \ldots, N$ let s_n denote the score of the page that is ranked in position n by the relevance function. Suppose at some point in time the crawler has fetched t pages and let r_t denote the number of fetched pages whose score is higher than s_t. In the ideal crawler we would have $r_t = t$ and the efficiency curve r_t/t would be constant and equal to one. In a random crawler, only a fraction t/N of the fetched pages will have a score higher than s_t, i.e. $r_t/2 = t/N$. Selective crawlers should obtain intermediate efficiency curves. Alternatively, one may assign a fixed 'importance target', G, and consider as relevant all the pages whose score is higher than G. For example, we might be interested in quickly fetching pages having at least 20 backlinks. The efficiency in this case may be measured as $q_t(G)/t$, where q_t is the number of fetched pages having a score higher than G. Cho *et al.* (1998) report several experimental results comparing the above strategies on a controlled data set of Stanford University pages. Figure 6.1 shows some of the results when the target measure of relevance is the number of backlinks. Interestingly, the estimated PageRank outperforms the estimated number of backlinks as a guiding criterion.

In a more recent study by Najork and Wiener (2001) it was found that traversing the Web in breadth-first order is a good strategy and allows us to download 'hot' pages first. The study was conducted on a large set of 328 million pages from 7 million distinct servers.

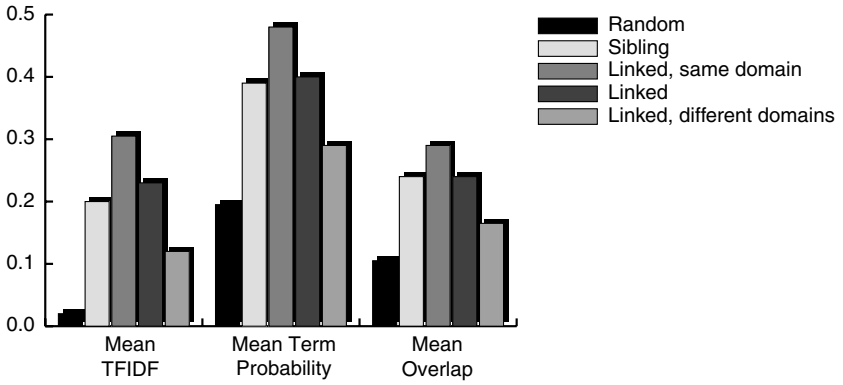

Figure 6.2 Results of the study carried out by Davison in 2000. (a) Three indicators that measure text similarity (see Section 4.3) are compared in five different linkage contexts. In this data set, about 56% of the linked documents belong to the same domain. It can be seen that the similarity between linked documents is significantly higher than the similarity between random documents. (b) Similarities measured between anchor text and text of documents in five different contexts.

6.2 Focused Crawling

In many cases a search engine does not need to be completely general purpose and cover every possible site on the Web, but instead can be focused on particular topics that are of interest to a specific community of users. In these cases, relevance can be defined with respect to the expected utility of the pages for users of the search engine. For example, a crawler that provides information to a vertical portal specialized in, say, music, could be quite safely programmed to ignore sites with content in different topics such as health or sports. A *focused crawler* is a refined selective crawler that searches for information related to certain topics rather than being driven by generic quality measures. The fraction of sites that specialize in a particular topic may be small enough to enable a focused crawler to download them almost exhaustively and in a relatively short time.

6.2.1 Focused crawling by relevance prediction

The URL ordering technique suggested for selective crawlers can be also extended to focused crawlers. In this case, it is necessary to determine whether a document is relevant or not to the topic of interest in order to define a scoring function for driving the search process. One way to do this is to use the text categorization techniques presented in Section 4.6. More precisely, if we denote by c the topic of interest, a score can be computed as the conditional probability that the document is relevant, given the text in the document:

$$s_\theta^{(\text{topic})}(u) = P(c \mid d(u), \theta). \tag{6.3}$$

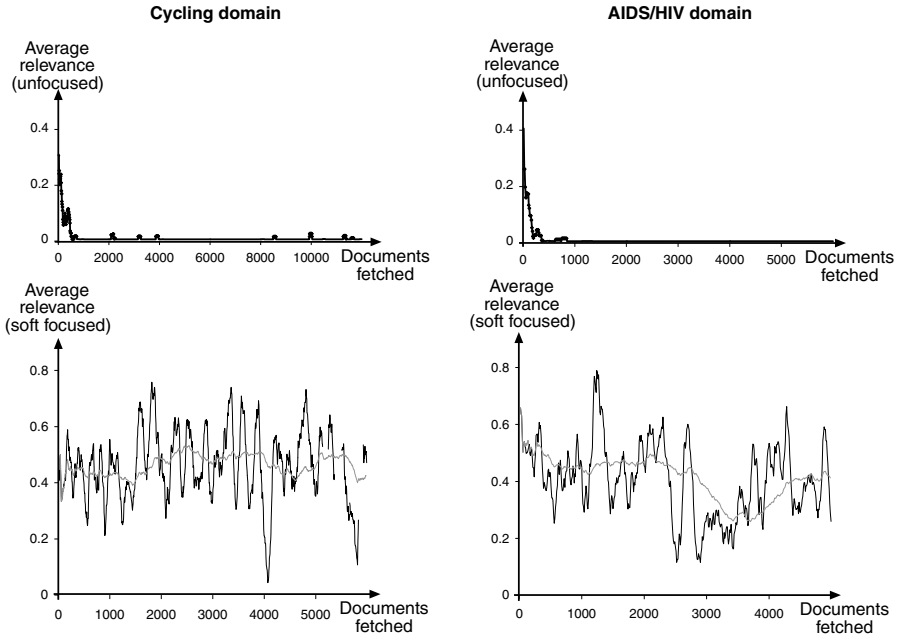

Figure 6.3 Efficiency of a BFS and a focused crawler compared in the study by Chakrabarti *et al.* (1999b). The plots show the average relevance of fetched documents versus the number of fetched documents. Results for two topics are shown (cycling on the left, AIDS/HIV on the right).

Here θ represents the adjustable parameters of the classifier. An obvious consideration is that the score cannot be computed exactly (we have not yet downloaded the document pointed to by u). There are a number of different strategies for approximating the topic score.

Parent based. In this case we compute the score for a fetched document and extend the computed value to all the URLs in that document. More precisely, if v is a parent of u, we approximate the score of u as

$$s_\theta^{(\text{topic})}(u) \simeq P(c \mid d(v), \theta). \tag{6.4}$$

The rationale is a general principle of 'topic locality'. If a page deals with, say, music, it may be reasonable to believe that most of the outlinks of that page will deal with music as well. In a systematic study based on the analysis of about 200 000 documents, Davison (2000b) found that topic locality is measurably present in the Web, under different text similarity measures (see Figure 6.2a).

Chakrabarti *et al.* (1999b) use a hierarchical classifier and suggest two implementations of the parent-based scoring approach. In *hard focusing*, they check if at least one node in the category path of $d(v)$ is associated with a topic of interest; if not

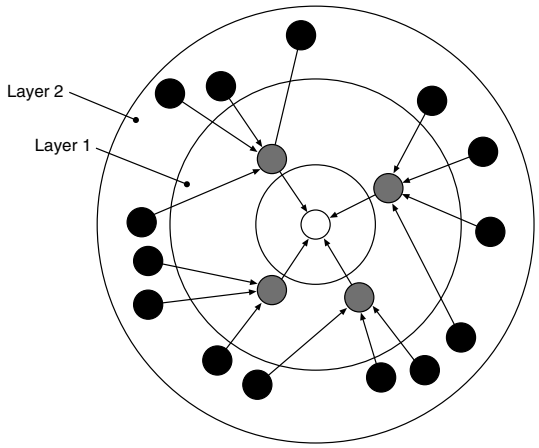

Figure 6.4 Example of a two-layered context graph. The central white node is a target document. Adapted from Diligenti *et al.* (2000).

the outlinks of v are simply discarded (not even inserted in the crawling queue Q). In *soft focusing*, relevance is computed according to Equation (6.4), but if the same URL is discovered from multiple parents, a revision strategy is needed in order to update $s_\theta^{(\text{topic})}(u)$ when new evidence is collected. Chakrabarti *et al.* (1999b) found no significant difference between the two approaches.

Anchor based. Instead of the entire parent document, $d(v)$, we can just use the text $d(v, u)$ in the anchor(s) where the link to u is referred to, as the anchor text is often very informative about the contents of the document pointed to by the corresponding URL. This 'semantic linkage' was also quantified by Davison (2000b), who showed that the anchor text is most similar to the page it references (see Figure 6.2b).

To better illustrate the behavior of a focused crawler on real data, consider the efficiency diagrams in Figure 6.3 that summarize some results obtained by Chakrabarti *et al.* (1999b). An unfocused crawler starting from a seed set of pages that are relevant to a given topic will soon begin to explore irrelevant regions of the Web. As shown in the top diagrams, the average relevance of the fetched pages dramatically decreases as crawling goes on. Using a focused crawler (in this case, relevance is predicted by a Naive Bayes classifier trained on examples of relevant documents) allows us to maintain an almost steady level of average relevance.

There are several alternatives to a focused crawler based on a single best-first queue, as detailed in the following.

6.2.2 Context graphs

Diligenti *et al.* (2000) suggested a strategy that takes advantage of knowledge of the Internet topology to train a machine-learning system to predict 'how far' some

relevant information can be expected to be found starting from a given page. Intuitively, suppose the crawler is programmed to gather homepages of academic courses in artificial intelligence. The backlinks of these pages are likely to lead to professors' home pages or to the teaching sections of the department site. Going one step further, backlinks of backlinks are likely to lead into higher level sections of department sites (such as those containing lists of the faculty). More precisely, the *context graph* of a node u (see Figure 6.4) is the layered graph formed inductively as follows. Layer 0 contains node u. Layer i contains all the parents of all the nodes in layer $i-1$. No edges jump across layers. Starting from a given set of relevant pages, Diligenti *et al.* (2000) used context graphs to construct a data set of documents whose distance from the relevant target was known (backlinks were obtained by querying general purpose search engines). After training, the machine-learning system predicts the layer a new document belongs to, which indicates how many links need to be followed before relevant information will be reached, or it returns 'other' to indicate that the document and its near descendants are all irrelevant. Denoting by n the depth of the considered context graph, the crawler uses n best-first queues, one for each layer, plus one extra queue for documents of class 'other'. This latter queue is initialized with the seeds. In the main loop, the crawler extracts URLs from the first nonempty queue and in this manner favors those that are more likely to rapidly lead to relevant information.

6.2.3 Reinforcement learning

Reinforcement learning (see, for example, Sutton and Barto 1998) is a framework for deciding in an optimal way what actions an agent operating in a discrete environment should take in order to maximize its expected future reward. Compared to supervised learning, reinforcement learning is characterized by the absence of external supervision. The agent learns from the received rewards (or punishments). More formally, the discrete-time environment is characterized by a finite set of states S. At time t the environment perceived by the agent is in state $s_t \in S$ and, correspondingly, the agent has access to a finite set of actions $A(s_t)$. After performing action $a_t \in A(s_t)$ the state transitions to s_{t+1} and the agent receives an immediate reward r_{t+1}. A policy π describes the joint probability distribution on states and actions, i.e. $\pi(s, a)$ is the probability of taking action a when in state s. An optimal policy should maximize the expected rewards received by the agent over time. In order to specify what an optimal policy is, we first define the value of state s under a policy π as

$$V^{\pi}(s) = E_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s\right]. \tag{6.5}$$

Similarly, we can define the action-value function as

$$Q^{\pi}(s, a) = E_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s, a_t = a\right]. \tag{6.6}$$

An optimal policy π^* maximizes the value function over all the states: $V^*(s) \geqslant V^{\pi}(s)$ for all $s \in S$. According to the Bellman optimality principle, underlying the foundations of dynamic programming (Bellman 1957), a sequence of optimal decisions has the property that, regardless of the action taken at the initial time, the subsequent sequence of decisions must be optimal with respect of the outcome of the first action. This translates into

$$V^*(s) = \max_{a \in A(s)} E[r_{t+1} + \gamma V^*(s_{t+1}) \mid s_t = s, a_t = a], \tag{6.7}$$

which allows us to determine the optimal policy once V^* is known for all s. An optimal policy π^* also maximizes $Q^{\pi}(s, a)$ for all $s \in S, a \in A(s)$:

$$Q^*(s, a) = E[r_{t+1} + \gamma \max_b Q^*(s_{t+1}, b) \mid s_t = s, a_t = a]. \tag{6.8}$$

The advantage of the state-action representation is that once Q^* is known for all s and a, all the agent needs to do in order to maximize the expected future reward is to choose the action that maximizes $Q^*(s, a)$.

In the context of focused Web search, immediate rewards are obtained (downloading relevant documents) and the policy learned by reinforcement can be used to guide the agent (the crawler) toward high long-term cumulative rewards. As we have seen in Section 2.5.3, the internal state of a crawler is basically described by the sets of fetched and discovered URLs. Actions correspond to fetching a particular URL that belongs to the queue of discovered URLs. Even if we simplify the representation by removing from the Web all the off-topic documents, it is clear that the sets of states and actions are overwhelming for a standard implementation of reinforcement learning.

LASER (Boyan *et al.* 1996) was one of the first proposals to combine reinforcement learning ideas with Web search engines. The aim in LASER is to answer queries rather than to crawl the Web. The system begins by computing a relevance score $r_0(u) = \text{TFIDF}(d(u), q)$ and then propagates it in the Web graph using the recurrence

$$r_{t+1}(u) = r_o(u) + \gamma \sum_{v \in \text{pa}[u]} \frac{r_t(v)}{|\text{pa}[u]|^{\ni}}, \tag{6.9}$$

which is iterated until convergence for each document in the collection, where γ and $\ni$ are free parameters. After convergence, documents at distance k from u provide a contribution proportional to γ^k times their relevance to the relevance of u.

McCallum *et al.* (2000c) used reinforcement to search computer science papers in academic websites. In order to simplify the problem of learning $Q^*(s, a)$ for an enormous number of states and actions they propose the following two assumptions:

- the state is independent of the relevant documents that have been fetched so far;
- actions can only be distinguished by means of the words in the neighborhood of the hyperlink that correspond to each action (e.g. the anchor text).

In this way, learning Q reduces to learning a mapping from text (e.g. bag of words representing anchors) to a real number (the expected future reward).

6.2.4 Related intelligent Web agents

The Fish algorithm by De Bra and Post (1994) uses a population of agents that autonomously spider the Web. Like fishes in an information sea, agents accumulate energy as they collect relevant documents for a given query while they consume energy for using network resources. Since agents need energy to survive, those which end up exploring irrelevant portions of the Web perish, implementing a selection mechanism. The Shark algorithm by Hersovici *et al.* (1998) improves the Fish search by introducing a real-valued relevance score that also depends on anchor text and a discounted fraction of the score that was given to ancestor pages. One of the main limitations of these approaches is the lack of adaptation.

WebWatcher (Armstrong *et al.* 1995) is an interactive recommendation system that assists a user during browsing. It introduced the use of machine learning for predicting the best hyperlink to follow according to a given user goal. A similar approach based on heuristic searching was proposed in Lieberman (1995). Arachnid (Menczer 1997) is based on a distributed population of adaptive agents that search information related to user-provided keywords. Similar in spirit to the Fish algorithm, Arachnid agents receive energy by relevance feedback provided by the user. An important feature of this approach is that hyperlinks are selected by a neural network associated with the agent, whose weights are adjusted by reinforcement learning. An extension of this system is described in Menczer and Belew (2000).

A more recent framework that generalizes focused crawling is *intelligent crawling* (Aggarwal *et al.* 2001). In this case, the crawler does not need a collection of topical examples for training. Users describe their information needs by means of *predicates*, i.e. general specifications that generalize keyword-based queries to also include document-to-document similarity queries, or characterizations of a topic as obtained from a text classifier. The intelligent crawler proposed in Aggarwal *et al.* (2001) is capable of auto-focusing using documents that satisfy the query predicate.

CiteSeer (`http://citeseer.nj.nec.com/cs`) is a search engine focused on computer science literature (Bollacker *et al.* 1998; Lawrence *et al.* 1999). It fetches PostScript or PDF papers from paper repositories (for example, a researcher's Web pages) and autonomously builds a citation index. As a major difference with respect to the system described by McCallum *et al.* (2000c), documents are located by querying traditional Web search engines and by collecting submitted URLs of pages containing links to research articles. The system performs several information extraction operation from the documents and, in particular, it extracts citations and references in the body of a paper in order to build a scientific literature web. Having represented the collection of papers as a web, CiteSeer can also apply link analysis algorithm and compute for example authority and hubness weights of each paper. Link analysis in this case can also be used to determine which documents are related to any given one.

Interestingly, since CiteSeer creates a Web page for each online article, it effectively maps the online subset of the computer science literature web to a subset of the World Wide Web. A recent study has shown that papers that are available online tend to receive a significantly higher number of citations (Lawrence 2001).

The DEADLINER system described in Kruger *et al.* (2000) is a search engine specialized in conference and workshop announcements. One of the input components of the system is a context-graph focused crawler (see Section 6.2.2) that gathers potentially related Web documents. An SVM text classifier is subsequently used to refine the set of documents retrieved by the focused crawler.

6.3 Distributed Crawling

A single crawling process, even if multithreading is used, will typically be insufficient for large-scale engines that need to fetch large amounts of data rapidly. When using a single centralized crawler, all the fetched data must pass through a single physical link. This is often problematic, regardless of the bandwidth available at the crawling center, because it will be unlikely to have comparable connection speeds from different geographical regions. Distributing the crawling activity via multiple processes can be seen as a form of 'divide and conquer' that can help build a scalable system. Splitting the load decreases hardware requirements and at the same time increases the overall download speed and reliability if separate processes access the Internet through different physical links. The advantage is particularly evident if separate crawlers run in separate data centers that are located in different countries or continents. However, while the physical links reflect geographical neighborhoods, we know that the edges of the Web graph are instead associated with 'communities' that can and often do cross geographical borders. Hence, running separate and independent crawling processes can result in a significant overlap among the collections of fetched documents. The performance of a parallelization approach can be measured in terms of

- communication overhead – the fraction of bandwidth spent to coordinate the activity of the separate processes, with respect to the bandwidth usefully spent for document fetching;
- overlap – the fraction of duplicate documents downloaded by all the processes;
- coverage – the fraction of documents reachable from the seeds that are actually downloaded; and
- quality – e.g. some of the scoring functions defined in Section 6.1 depend on the link structure and this information can be partially lost using separate crawling processes.

The literature on this important topic is not abundant. In a recent study, Cho and Garcia-Molina (2002) have defined a framework based on several dimensions that characterize the interaction among a set of crawlers.

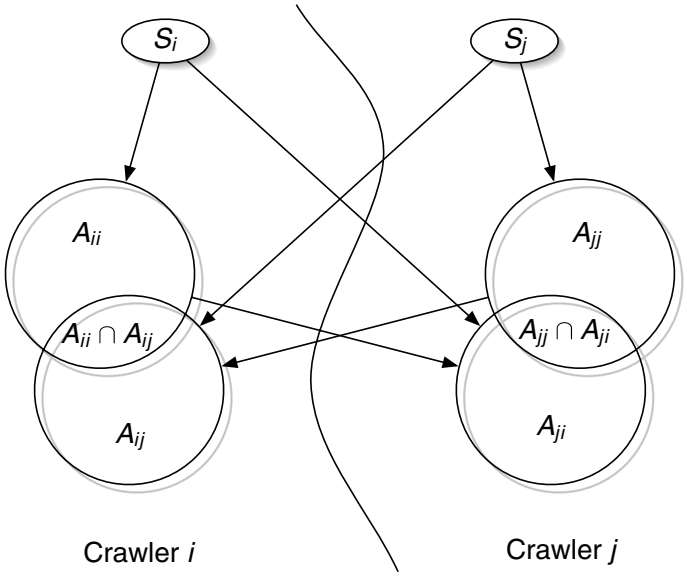

Figure 6.5 Two crawlers statically coordinated.

Coordination refers to the way different processes agree about the subset of pages each of them should be responsible for. If two crawling processes i and j are completely *independent* (not coordinated), then the degree of overlap can only be controlled by having different seeds S_i and S_j. If we assume the validity of topological models such as those presented in Section 3, then we can expect that the overlap will eventually become significant unless a partition of the seed set is properly chosen. On the other hand, making a good choice is a challenge, since the partition that minimizes overlap may be difficult to compute, for example, because current models of the Web are not accurate enough. In addition it may be suboptimal with respect to other desiderata that motivated distributed crawling in the first place, such as distributing the load and scaling-up.

A pool of crawlers can be coordinated by partitioning the Web into a number of subgraphs and letting each crawler be mainly responsible for fetching documents from its own subgraph. If the partition is decided before crawling begins and not changed thereafter, we refer to this as *static coordination*. This option has the great advantage of simplicity and implies little communication overhead. Alternatively, if the partition is modified during the crawling process, the coordination is said to be *dynamic*. In the static approach the crawling processes, once started, can be seen as agents that operate in a relatively autonomous way. In contrast, in the dynamic case each process is subject to a reassignment policy that must be controlled by an external supervisor.

Confinement specifies, assuming statically coordinated crawlers, how strictly each crawler should operate within its own partition. Consider two processes, i and

j, and let A_{ij} denote the set of documents belonging to partition i that can be reached from the seeds S_j (see Figure 6.5). The question is what should happen when crawler i pops from its queue 'foreign' URLs pointing to nodes in a different partition. Cho and Garcia-Molina (2002) suggest three possible modes: *firewall*, *crossover*, and *exchange*. In firewall mode, each process remains strictly within its partition and never follows interpartition links. In crossover mode, a process can follow interpartition links when its queue does not contain any more URLs in its own partition. In exchange mode, a process never follows interpartition links, but it can periodically open a communication channel to dispatch the foreign URLs it has encountered to the processes that operate in those partitions. To see how these modes affect performance measures, consider Figure 6.5 again. The firewall mode has, by definition, zero overlap but can be expected to have poor coverage, since documents in $A_{ij} \setminus A_{ii}$ are never fetched (for all i and j). Cross-over mode may achieve good coverage but can have potentially high overlap. For example, documents in $A_{ii} \cap A_{ij}$ can be fetched by both process i and j. The exchange mode has no overlap and can achieve perfect coverage. However, while the first two modes do not require extra bandwidth, in exchange mode there will be a communication overhead.

Partitioning defines the strategy employed to split URLs into non-overlapping subsets that are then assigned to each process. A straightforward approach is to compute a hash function of the IP address in the URL, i.e. if $n \in \{0, \ldots, 2^{32} - 1\}$ is the integer corresponding to the IP address and m the number of processes, documents such that $n \mod m = i$ are assigned to process i. In practice, a more sophisticated solution would take into account the geographical dislocation of networks, which can be inferred from the IP address by querying Whois databases such as the Réseaux IP Européens (RIPE) or the American Registry for Internet Numbers (ARIN).

6.4 Web Dynamics

In the final section of this chapter we address the question of how information on the Web changes over time. Knowledge of this 'rate of change' is crucial, as it allows us to estimate how often a search engine should visit each portion of the Web in order to maintain a fresh index.

In Chapter 3 we discussed general 'aging' properties of Web graphs. A precise notion of recency in this context has been proposed by Brewington and Cybenko (2000). The index entry for a certain document, indexed at time t_0, is said to be β-current at time t if the document has not changed in the time interval between t_0 and $t - \beta$. Basically β is a 'grace period': if we pretend that the user query was made β time units ago rather than now, then the information in the search engine would be up to date. A search engine for a given collection of documents is said to be (α, β)-current if the probability that a document is β-current is at least α. According to this definition, we can ask interesting questions like 'how many documents per day should a search

engine refresh in order to guarantee it will remain (0.9,1 week)-current?' Answering this question requires that we develop a probabilistic model of Web dynamics. The model will be complicated because of two concomitant factors: pages change over time, and the Web itself grows and evolves in time (see Chapter 3).

6.4.1 Lifetime and aging of documents

To begin with consider a single document. The model we are going to develop is based on the same ideas underpinning reliability theory in industrial engineering (see, for example, Barlow and Proshan 1975). Let T be a continuous random variable representing the *lifetime* of a component in a piece of machinery or equipment. Assuming the component was initially installed or replaced at time zero, lifetime is the time when the component breaks down (dies). Let $F(t)$ be the cumulative distribution function (cdf) of lifetime. The reliability (or survivorship function) is defined as

$$S(t) \doteq 1 - F(t) = P(T > t), \tag{6.10}$$

i.e. the probability that the component will be functioning at time t. A variable closely related to lifetime is the *age* of a component, i.e. the time elapsed since the last replacement (see Figure 6.6 to better understand the relationship between lifetime and age). Its cdf, defined as $G(t) = P(\text{age} < t)$, is obtained by integrating the survivorship function and normalizing:

$$G(t) = \frac{\int_0^t S(\tau)\,\mathrm{d}\tau}{\int_0^\infty S(\tau)\,\mathrm{d}\tau}. \tag{6.11}$$

Note that $G(t)$ is the expected fraction of components that are still operating at time t. The age probability density function (pdf) $g(t)$ is thus proportional to the survivorship function. Returning from the reliability metaphor to Web documents, $S(t)$ is the probability that a document that was last changed at time zero will remain unmodified at time t, while $G(t)$ is the expected fraction of documents that are older than t. The probability that the document will be modified before an additional time h has passed is expressed by the conditional probability $P(t < T \leqslant t + s \mid T > t)$. The change rate $\lambda(t)$ (also known as the hazard rate in reliability theory, or mortality force in demography) is then obtained by dividing by h and taking the limit for small h,

$$\begin{aligned}\lambda(t) &\doteq \lim_{h\to 0} \frac{1}{s} P(t < T \leqslant t + h \mid T > t) \\ &= \lim_{h\to 0} \frac{1}{S(t)} \frac{1}{h} \int_t^{t+h} f(\tau)\,\mathrm{d}\tau = \frac{f(t)}{S(t)},\end{aligned} \tag{6.12}$$

where $f(t)$ denotes the lifetime pdf. Combining (6.10) and (6.12) we have the ordinary differential equation (see, for example, Apostol 1969)

$$F'(t) = \lambda(t)(1 - F(t)), \tag{6.13}$$

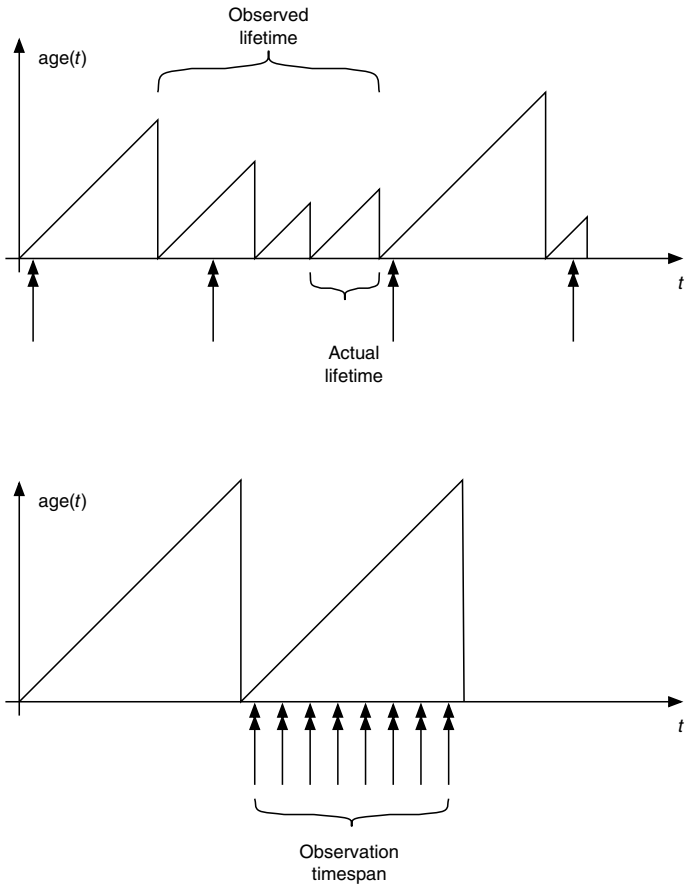

Figure 6.6 Sampling lifetimes can be problematic as changes can be missed for two reasons. Top, two consecutive changes are missed and the observed lifetime is overestimated. Bottom, the observation time-span must be large enough to catch changes that occur in a long range. Sampling instants are marked by double arrowheads.

with $F(0) = 0$. We will assume that changes happen randomly and independently. According to a Poisson process, the probability of a change event at any given time is independent of when the last change happened (see Appendix A). For a constant change rate $\lambda(t) = \lambda$, the solution of Equation (6.13) is

$$F(t) = 1 - \mathrm{e}^{-\lambda t}, \qquad f(t) = \lambda \mathrm{e}^{-\lambda t}.$$

Brewington and Cybenko (2000) observed that the model could be particularly valuable for analyzing Web documents. In practice, however, the estimation of $f(t)$ is problematic for any method based on sampling. If a document is observed at two different instants t_1 and t_2, we can check for differences in the document but we cannot know how many times the document was changed in the interval $[t_1, t_2]$, a phe-

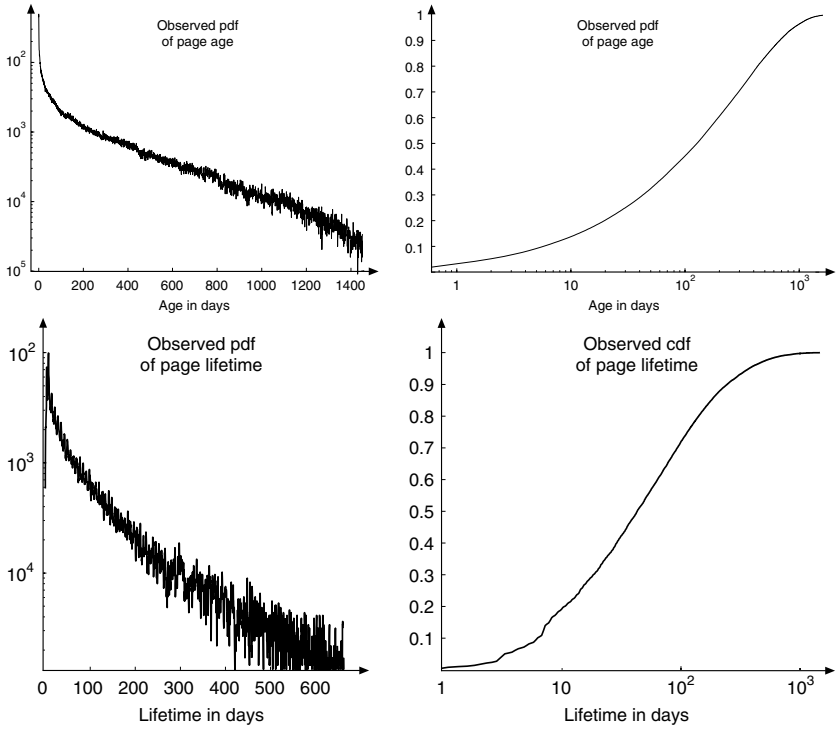

Figure 6.7 Empirical distributions of page age (top) and page lifetime (bottom) on a set of 7 million Web pages. Adapted from Brewington and Cybenko (2000).

nomenon known as *aliasing* (see Figure 6.6). On the other hand, the age of a document may be readily available, if the Web server correctly returns the `Last-Modified` timestamp in the HTTP header (see Section 2.3.4 for a sample script that obtains this information from a Web server). Sampling document ages is not subject to aliasing and lifetime can be obtained indirectly via Equation (6.11). In particular, if the change rate is constant, it is easy to see that the denominator in Equation (6.11) is a/λ and thus from Equation (6.12) we obtain

$$g(t) = f(t) = \lambda e^{-\lambda t}. \tag{6.14}$$

In other words, assuming a constant change rate, it is possible to estimate lifetime from observed age.

This simple model, however, does not capture the essential property that the Web is growing with time. Brewington and Cybenko (2000) collected a large data set of roughly 7 million Web pages, observed between 1999 and 2000 in a time period of seven months, while operating a service called 'The Informant'.[1] The resulting distributions of age and lifetime are reported in Figure 6.7.

[1] Originally `http://informant.dartmouth.edu`, now `http://www.tracerlock.com`.

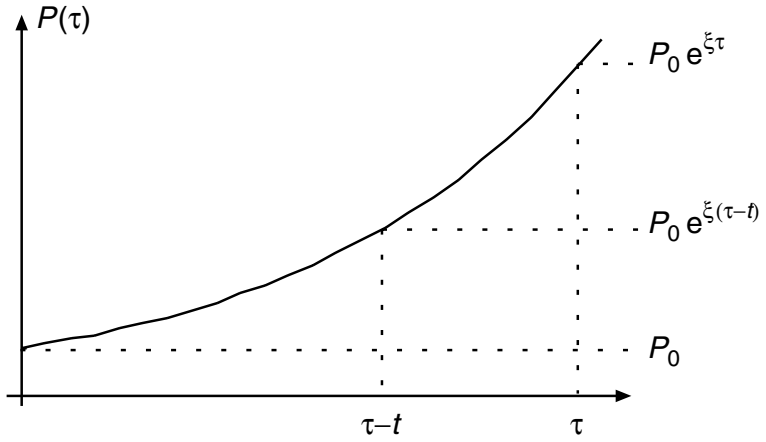

Figure 6.8 Assuming an exponentially growing Web whose pages are never changed, it follows that the age distribution is also exponential at any time τ.

What is immediately evident is that most of the collected Web documents are 'young', but what is interesting is why this is the case. Suppose that growth is modeled by an exponential distribution, namely that the size at time τ is $P(\tau) = P_0 e^{\xi\tau}$, where P_0 is the size of the initial population and ξ the growth rate. If documents were created and never edited, their age would be simply the time since their creation. The probability that a random document at time τ has an age less than t is

$$G_g(t, \tau) = \frac{\text{new docs in } (\tau - t, \tau)}{\text{all docs}} = \frac{e^{\xi\tau} - e^{\xi(\tau - t)}}{e^{\xi\tau} - 1} \tag{6.15}$$

(see Figure 6.8) and thus the resulting age density at time τ is

$$g_g(t, \tau) = \frac{\xi e^{\xi(\tau - t)}}{e^{\xi\tau} - 1}[H(t) - H(t - \tau)], \tag{6.16}$$

where $H(t)$ is the Heaviside step function, i.e. $H(t) = 0$ for $t < 0$ and $H(t) = 1$ otherwise.

In other words, this trivial growth model yields an exponential age distribution, like the model in Section 6.4.1 that assumed a static Web with documents refreshed at a constant rate. Clearly, the effects of both Web growth and document refreshing should be taken into account in order to obtain a realistic model. Brewington and Cybenko (2000) experimented with a hybrid model (not reported here) that combines an exponential growth model and exponential change rate. Fitting this model with ages obtained from timestamps, they estimated $\xi = 0.001\,76$ (in units of days^{-1}). This estimate corresponds to the Web size doubling in about 394 days. In contrast, if we estimated ξ from the lower bounds of 320 million pages in December 1997 (Lawrence and Giles 1998b) and 800 million pages in February 1999 (Lawrence and Giles 1999), roughly 426 days later, we would obtain $\xi = 0.022$, or a doubling time

of 315 days. Despite the differences in these two estimates, they are of the same order of magnitude and give us some idea of the growth rate of the Web.

Another important problem when using empirically measured ages is that servers often do not return meaningful timestamps. This is particularly true in the case of highly dynamic Web pages. For example, it is not always possible to assess from the timestamp whether a document was just edited or was generated on-the-fly by the server. These considerations suggest that the estimation of change rates should be carried out using lifetimes rather than ages. The main difficulty is how to deal with the problem of potentially poorly chosen timespans, as exemplified in Figure 6.6. Brewington and Cybenko (2000) suggest a model that explicitly takes into account the probability of observing a change, given the change rate and the timespan. Their model is based on the following assumptions.

- Document changes are events controlled by an underlying Poisson process, where the probability of observing a change at any given time does not depend on previous changes. Given the timespan τ and the change rate λ, the probability that we observe one change (given that it actually was made) is therefore

$$P(c \mid \lambda, \tau) = 1 - \mathrm{e}^{-\lambda\tau}. \tag{6.17}$$

  It should be observed that, in their study, Brewington and Cybenko (2000) found that pages are changed with different probabilities at different hours of the day or during different days of the week (most changes being concentrated during office hours). Nonetheless, the validity of a memoryless Poisson model is assumed.

- Mean lifetimes are Weibull distributed (see Appendix A), i.e. denoting the mean lifetime by $t = 1/\lambda$, the pdf of t is

$$w(t) = \frac{\sigma}{\delta}\left(\frac{t}{\delta}\right)^{\sigma-1} \mathrm{e}^{(t/\delta)^{\sigma}}, \tag{6.18}$$

  where δ is a scale parameter and σ is a shape parameter.

- Change rates and timespans are independent and thus

$$P(c \mid \lambda) = \int_0^{\infty} P(c, \tau \mid \lambda)\,\mathrm{d}\tau = \int_0^{\infty} P(\tau)P(c \mid \tau, \lambda)\,\mathrm{d}\tau.$$

The resulting lifetime distribution is

$$f(t) = \int_0^{\infty} \lambda \mathrm{e}^{-\lambda t}\hat{w}(1/\lambda)\,\mathrm{d}(1/\lambda), \tag{6.19}$$

where $\hat{w}(1/\lambda)$ is an estimate of the mean lifetime. Brewington and Cybenko (2000) used the data shown in Figure 6.7 to estimate the Weibull distribution parameters, obtaining $\sigma = 1.4$ and $\delta = 152.2$. The estimated mean lifetime distribution is plotted in Figure 6.9.

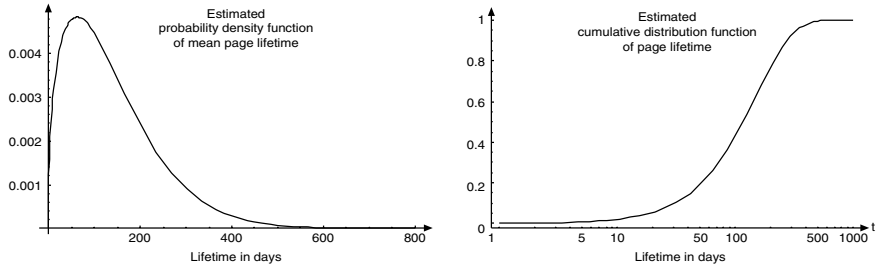

Figure 6.9 Estimated density and distribution of mean lifetime resulting from the study of Brewington and Cybenko (2000).

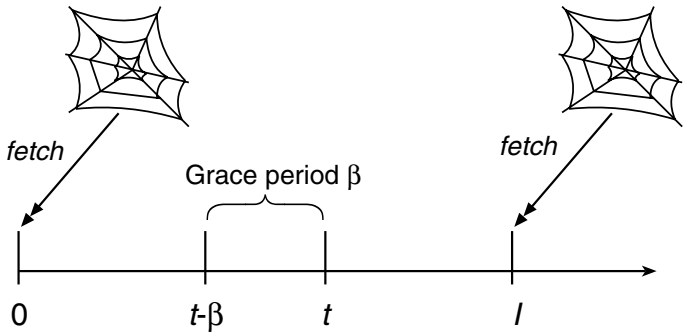

Figure 6.10 A document is β-current at time t if no changes have occurred before the grace period that extends backward in time until $t - \beta$.

These results allow us to estimate how often a crawler should refresh the index of a search engine to guarantee that it will remain (α, β)-current. Let us consider first a single document and, for simplicity, let $t = 0$ be the time when the crawler last fetched the document. Also, let I be the time interval between two consecutive visits (see Figure 6.10). The probability that for a particular time t the document is unmodified in $[0, t - \beta]$ is $e^{-\lambda(t-\beta)}$ for $t \in [\beta, I)$ and 1 for $t \in (0, \beta)$. Thus, the probability that a specific document is β-current at time t is

$$\int_0^{\beta} \frac{1}{I}\,dt + \int_{\beta}^{I} \frac{1}{I} e^{-\lambda(t-\beta)}\,dt = \frac{\beta}{I} + \frac{1 - e^{-\lambda(I-\beta)}}{\lambda I}, \tag{6.20}$$

but since each document has a change rate λ, whose reciprocal is Weibull distributed, the probability that the collection of documents is β-current is

$$\alpha = \int_0^{\infty} w(t)\left[\frac{\beta}{I} + \frac{1 - e^{-(I-\beta/t)}}{t/I}\right] dt. \tag{6.21}$$

This allows us to determine the minimum refresh interval I to guarantee (α, β)-currency once the parameters of the Weibull distribution for the mean change rate are known. Assuming a Web size of 800 million pages, Brewington and Cybenko (2000)

determined that a reindexing period of about 18 days was required to guarantee that 95% of the repository was current up to one week ago.

6.4.2 Other measures of recency

Freshness and *index age* are two alternative and somewhat simpler measures of recency (Cho and Garcia-Molina 2000b). The freshness $\phi(t)$ at time t of a given document is a binary function that indicates whether the document is up-to-date in the index at time t. The expected freshness is therefore the probability that the document did not change in $(0, t]$, i.e. $E[\phi(t)] = \mathrm{e}^{-\lambda t}$ (see Equation (6.17)). Note that freshness essentially corresponds to the concept of β-currency for $\beta = 0$. Hence, if d is refreshed regularly each I time units, the average freshness is

$$\bar{\phi} = \frac{1 - \mathrm{e}^{-\lambda I}}{\lambda I}$$

as follows from Equation (6.20) with $\beta = 0$.

The *index age* of a document is the age of the document if the local copy is outdated, or zero if the local copy is fresh. Thus if the document was modified at time $s \in (0, t]$, its index age is $t - s$. From Equation (6.14) it follows that the expected age at time t is

$$E[a(t)] = \int_0^t (t - s)\lambda \mathrm{e}^{-\lambda s}\, \mathrm{d}s = t - \frac{1 - \mathrm{e}^{-\lambda t}}{\lambda},$$

whose average in $(0, I]$ is

$$\overline{a} = \frac{1 - \mathrm{e}^{-\lambda I}}{\lambda^2 I} - \frac{1}{\lambda} + \frac{I}{2}.$$

When a collection of documents is considered, the above quantities can be averaged over the collection. Because of the linearity of integral, we can average both time punctual values and time averages.

6.4.3 Recency and synchronization policies

The bandwidth requirements that can be inferred from the analysis reported in Section 6.4.1 could be somewhat pessimistic. Clearly, not all the sites change their pages at the same rate. Cho and Garcia-Molina (2000b) conducted another study in 1999, monitoring changes of about 720 000 popular[2] Web pages. Results are reported in Figure 6.11. In terms of the overall 'popular' Web, these diagrams are in good qualitative accordance with the findings of Brewington and Cybenko (2000), indicating that a vast fraction of this Web is dynamic. However, it is interesting to note that a very simple Web partitioning, based on four top-level domains, yields dramatically different results (Cho and Garcia-Molina 2000b). In particular, 'dot com' websites

[2] According to PageRank (see Section 5.4), so this forms a biased representative of the Web population.

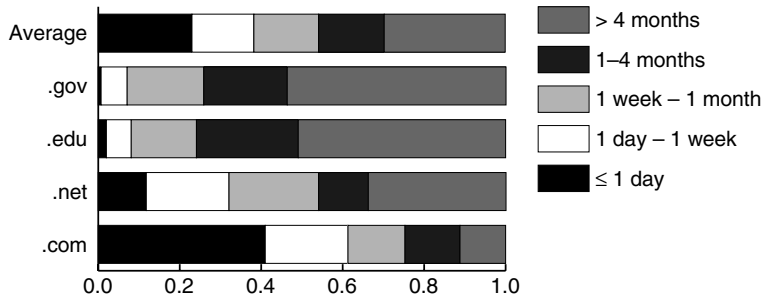

Figure 6.11 Average change interval found in a study conducted by Cho and Garcia-Molina (2000a). 270 popular sites were monitored for changes from 17 February to 24 June 1999. A sample of 3000 pages was collected from each site by visiting in breadth first order from the homepage.

are much more dynamic compared to educational or governmental sites. While this is not surprising, it suggests that a resource allocation policy that does not take into account site (or even document) *specific* dynamics may waste bandwidth re-fetching old information that, however, is recent in the index. From a different viewpoint, this nonuniformity suggests that, for a given bandwidth, the recency of the index can be improved if the refresh rate is differentiated for each document.

To understand how to design an optimal synchronization policy we will make several simplifying assumptions. Suppose there are N documents of interest and suppose we can estimate the change rate $\lambda_i, i = i, \ldots, N$, of each document. Suppose also that it will be practical to program a crawler that regularly fetches each document i with a refresh interval I_i. Suppose also that the time required to fetch each document is constant. The fact that we have limited bandwidth should be reflected in a constraint involving I_i. If B is the available bandwidth, expressed as the number of documents that can be fetched in a time unit, this constraint is

$$\sum_{i=1}^{N} \frac{1}{I_i} \leqslant N. \tag{6.22}$$

The problem of *optimal resource allocation* consists of selecting the refresh intervals I_i so that a recency measure of the resulting index (e.g. freshness or index age) will be maximized (Cho and Garcia-Molina 2000b). For example, we may want to maximize freshness

$$(I_1^*, \ldots, I_N^*) = \arg \max_{I_i, \ldots, I_N} \sum_{i=1}^{N} \bar{\phi}(\lambda_i, \phi_i) \tag{6.23}$$

subject to (6.22). We might be tempted to arrange a policy that assigns to each document a refresh interval I_i that is proportional to the change rate λ_i. However, this intuitive approach is suboptimal, and can be proven to be even worse than assigning the same interval to each document. The optimal intervals can be easily derived

as the solution of a constrained optimization problem such as the one described by Equation (6.23).

If very large collections of documents need to be monitored, bandwidth limitations may even prevent us from frequently monitoring document changes. Cho and Ntoulas (2002) have recently proposed a sampling approach where a small fraction of pages from a given site is checked for changes and the results are used to estimate the change rate of the entire site.

WebFountain (Edwards *et al.* 2001) is a fully distributed and *incremental* crawler with no central control or centralized queue of URLs. 'Incremental' in this case means that the repository entry of a given document is updated as soon as it is fetched from the Web and the crawling process is never regarded as complete. The goal is to keep the repository as fresh and as complete as possible. The model in this case is not based on assumptions on the distribution of document change rates. Changes are simply detected when a document is re-fetched and documents are grouped into a set of buckets, each containing documents having similar rates of change. The trade-off between re-fetching (to improve freshness) and exploring (to improve coverage) is controlled in this case by maintaining separate queues for 'old' and 'new' URLs. The optimal ratio of between the number of old and new URLs is determined as the solution of a constrained optimization problem (see Edwards *et al.* (2001) for details).

Wolf *et al.* (2002) also formulate crawling as an optimization problem and suggest a pragmatic metric, the *embarrassment level*, that focuses on the expected number of times a search engine client is returned a stale URL.

Exercises

Exercise 6.1. Write a simplified crawling program that organizes the list of URLs to be fetched as a priority queue. Order the priority queue according to the expected indegree of the page pointed to by each URL and compare your results to a best-first search algorithm, using the actual indegree as a target measure of relevance.

Exercise 6.2. Extend the crawler developed in Exercise 6.1 to search for documents related to a specific topic of interest. Collect a set of documents of interest to be used as training examples and use one of text categorization tools studied in Chapter 4 to guide the crawler. Compare results obtained using the parent-based and the anchor-based strategies.

Exercise 6.3. Write a program that recursively scans your hard disk and estimates the lifetime of your files.

Exercise 6.4. Suppose the page mean lifetime is Weibull distributed with parameters δ and σ. What are the average mean lifetime, the most likely mean lifetime, and the median mean lifetime? What would be reasonable values for these quantities starting from an estimated distribution like the one shown in Figure 6.9?

Exercise 6.5. Explain what we mean when we say that a collection of stored documents is (α, β)-current.

Exercise 6.6. Suppose you want to build a (0.8,1-week)-current search engine for a collection of 2 billion documents whose average size is 10 Kb. Suppose the mean change is Weibull distributed with $\sigma = 1$ and $\delta = 100$ days. What is the required bandwidth (suppose the time necessary to build the index is negligible)?