Effective Web Crawling

by

Carlos Castillo

Submitted to the University of Chile in fulfillment
of the thesis requirement to obtain the degree of
Ph.D. in Computer Science

Advisor	**Dr. Ricardo Baeza-Yates** University of Chile
Committee	Dr. Mauricio Marin University of Magallanes, Chile
	Dr. Alistair Moffat University of Melbourne, Australia
	Dr. Gonzalo Navarro University of Chile
	Dr. Nivio Ziviani Federal University of Minas Gerais, Brazil

This work has been partially funded by the Millennium Nucleus “Center for Web Research”
of the Millennium Program, Ministry of Planning and Cooperation – Government of Chile

Dept. of Computer Science - University of Chile

November 2004

Abstract

The key factors for the success of the World Wide Web are its large size and the lack of a centralized control over its contents. Both issues are also the most important source of problems for locating information. The Web is a context in which traditional Information Retrieval methods are challenged, and given the volume of the Web and its speed of change, the coverage of modern search engines is relatively small. Moreover, the distribution of quality is very skewed, and interesting pages are scarce in comparison with the rest of the content.

Web crawling is the process used by search engines to collect pages from the Web. This thesis studies Web crawling at several different levels, ranging from the long-term goal of crawling important pages first, to the short-term goal of using the network connectivity efficiently, including implementation issues that are essential for crawling in practice.

We start by designing a new model and architecture for a Web crawler that tightly integrates the crawler with the rest of the search engine, providing access to the metadata and links of the documents that can be used to guide the crawling process effectively. We implement this design in the WIRE project as an efficient Web crawler that provides an experimental framework for this research. In fact, we have used our crawler to characterize the Chilean Web, using the results as feedback to improve the crawler design.

We argue that the number of pages on the Web can be considered infinite, and given that a Web crawler cannot download all the pages, it is important to capture the most important ones as early as possible during the crawling process. We propose, study, and implement algorithms for achieving this goal, showing that we can crawl 50% of a large Web collection and capture 80% of its total Pagerank value in both simulated and real Web environments.

We also model and study user browsing behavior in Web sites, concluding that it is not necessary to go deeper than five levels from the home page to capture most of the pages actually visited by people, and support this conclusion with log analysis of several Web sites. We also propose several mechanisms for server cooperation to reduce network traffic and improve the representation of a Web page in a search engine with the help of Web site managers.

Publications related to this thesis

The crawling model and architecture described in Chapter 3 was presented in the second Hybrid Intelligent Systems conference [BYC02] (HIS 2002, proceedings published by IOS Press), and introduced before in preliminary form in the eleventh World Wide Web conference [CBY02].

The analysis and comparison of scheduling algorithms, in terms of long-term and short-term scheduling in Chapter 4 was presented in the second Latin American Web conference [CMRBY04] (LA-WEB 2004, published by IEEE CS Press).

The model and analysis of browsing behavior on the "Infinite Web" on Chapter 5 was presented in the third Workshop on Algorithms and Models for the Web-Graph [BYC04] (WAW 2004, published by Springer LNCS).

Most of the proposals about Web server cooperation shown in Chapter 6 were introduced in preliminary form in the first Latin American Web conference [Cas03] (LA-WEB 2003, published by IEEE CS Press).

Portions of the studies on Web structure and dynamics shown in Chapter 8 appear as a chapter in the book "Web Dynamics" [BYCSJ04] (published by Springer), and were presented in the 8th and 9th String Processing and Information Retrieval conferences [BYC01, BYSJC02] (SPIRE 2001, published by IEEE CS Press and SPIRE 2002, published by Springer LNCS).

An application of the WIRE crawler to characterize images, not described in this thesis, was presented in the first Latin American Web conference [JdSV$^+$03] (LA-WEB 2003, published by IEEE CS Press) and the third Conference on Image and Video Retrieval [BYdSV$^+$04] (CIVR 2004, published by Springer LNCS).

The WIRE crawler developed during this thesis is available under the GNU public license, and can be freely downloaded at `http://www.cwr.cl/projects/WIRE/`. The user manual, including step-by-step instructions on how to use the crawler, is available at the same address.

Acknowledgements

This thesis would not have been possible without **O.P.M.** During the thesis I received mostly the financial support of grant P01-029F of the Millennium Scientific Initiative, Mideplan, Chile. I also received financial support from the Faculty of Engineering and the Computer Science Department of the University of Chile, among other sources.

What you are is a consequence of whom you interact with, but just saying "thanks everyone for everything" would be wasting this opportunity. I have been very lucky of interacting with really great people, even if some times I am prepared to understand just a small fraction of what they have to teach me. I am sincerely grateful for the support given by my advisor **Ricardo Baeza-Yates** during this thesis. The comments received from the committee members **Gonzalo Navarro**, Alistair Moffat, Nivio Ziviani and Mauricio Marin during the review process were also very helpful and detailed. For writing the thesis, I also received data, comments and advice from Efthimis Efthimiadis, Marina Buzzi, Patrizia Andrónico, Massimo Santini, Andrea Rodríguez and Luc Devroye. I also thank Susana Docmac and everybody at **Newtenberg**.

This thesis is just a step on a very long road. I want to thank the professors I met during graduate studies: **Vicente López**, Claudio Gutierrez and José Pino; also, I was lucky to have really inspiring professors during the undergraduate studies: Martin Matamala, Marcos Kiwi, Patricio Poblete, Patricio Felmer and **José Flores**. There were some teachers in high and grade school that trusted in me and helped me get the most out of what I was given. During high school: Domingo Almendras, Belfor Aguayo, and in grade school: Manuel Guiñez, Ivonne Saintard and specially **Carmen Tapia**.

I would said at the end that I owe everything to my parents, but that would imply that they also owe everything to their parents and so on, creating an infinite recursion that is outside the context of this work. Therefore, I thank **Myriam** and **Juan Carlos** for being with me even from before the beginning, and sometimes giving everything they have and more. I am also thankful for the support of all the members of my family, specially Mercedes Pincheira.

Finally, my beloved wife **Fabiola** was exactly 10,000 days old on the day I gave my dissertation, and I need no calculation to say that she has given me the best part of those days – thank you.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This thesis is about Web crawling, the process used by Web search engines to download pages from the Web. This opening chapter starts with the main motivations for studying this process in Section 1.1. Section 1.2 introduces the WIRE project, the system that we use as a context for working on this topic. Section 1.3 explains the scope and organization of our work.

1.1 Motivation

1.1.1 From organic to mineral memory

As technology advances, concerns arise about how the new inventions may impair human capabilities. Plato, in his dialogue *Phaedrus*, tells the story of Theuth (Hermes) presenting his inventions to Pharaoh Thamus, who dislikes the idea of writing:

> "This, said Theuth, will make the Egyptians wiser and will give them better memories; it is a specific for both the memory and for the wit. Thamus replied: Oh most ingenious Theuth (...) this discovery of yours will create forgetfulness in the learners souls, because they will not use their memories; they will trust to the external written characters and not remember of themselves, (...) they will be hearers of many things and will have learned nothing; they will appear to be omniscient and will generally know nothing; they will be tiresome company, having the show of wisdom without the reality." [PlaBC]

Thamus considers that writing is a bad invention because it replaces human memory. There are others who consider writing as just an extension of the human memory, such as Umberto Eco, who in November 2003 gave a lecture about the future of books at the newly opened Library of Alexandria in Egypt:

> "We have three types of memory. The first one is organic, which is the memory made of flesh and

> blood and the one administrated by our brain. The second is mineral, and in this sense mankind has known two kinds of mineral memory: millennia ago, this was the memory represented by clay tablets and obelisks, pretty well known in this country, on which people carved their texts. However, this second type is also the electronic memory of today's computers, based upon silicon. We have also known another kind of memory, the vegetal one, the one represented by the first papyruses, again well known in this country, and then on books, made of paper." [Eco03]

The World Wide Web, a vast mineral memory, has become in a few years the largest cultural endeavour of all times, equivalent in importance to the first Library of Alexandria. How was the ancient library created? This is one version of the story:

> "By decree of Ptolemy III of Egypt, all visitors to the city were required to surrender all books and scrolls in their possession; these writings were then swiftly copied by official scribes. The originals were put into the Library, and the copies were delivered to the previous owners. While encroaching on the rights of the traveler or merchant, it also helped to create a reservoir of books in the relatively new city." [wik04]

The main difference between the Library of Alexandria and the Web is not that one was vegetal, made of scrolls and ink, and the other one is mineral, made of cables and digital signals. The main difference is that while in the Library books were copied by hand, most of the information on the Web has been reviewed only once, by its author, at the time of writing.

Also, modern mineral memory allows fast reproduction of the work, with no human effort. The cost of disseminating content is lower due to new technologies, and has been decreasing substantially from oral tradition to writing, and then from printing and the press to electronic communications. This has generated much more information than we can handle.

1.1.2 The problem of abundance

The signal-to-noise ratio of the products of human culture is remarkably high: mass media, including the press, radio and cable networks, provide strong evidence of this phenomenon every day, as well as more small-scale actions such as browsing a book store or having a conversation. The average modern working day consists of dealing with 46 phone calls, 15 internal memos, 19 items of external post and 22 e-mails [Pat00].

We live in an era of information explosion, with information being measured in exabytes (10^{18} bytes):

> "Print, film, magnetic, and optical storage media produced about 5 exabytes of new information in 2002. (...) We estimate that new stored information grew about 30% a year between 1999

and 2002. (...) Information flows through electronic channels – telephone, radio, TV, and the Internet – contained almost 18 exabytes of new information in 2002, three and a half times more than is recorded in storage media. (...) The World Wide Web contains about 170 terabytes of information on its surface." [LV03]

On the dawn of the World Wide Web, finding information was done mainly by scanning through lists of links collected and sorted by humans according to some criteria. Automated Web search engines were not needed when Web pages were counted only by thousands, and most directories of the Web included a prominent button to "add a new Web page". Web site administrators were encouraged to submit their sites. Today, URLs of new pages are no longer a scarce resource, as there are thousands of millions of Web pages.

The main problem search engines have to deal with is the size and rate of change of the Web, with no search engine indexing more than one third of the publicly available Web [LG98]. As the number of pages grows, it will be increasingly important to focus on the most "valuable" pages, as no search engine will be able of indexing the complete Web. Moreover, in this thesis we state that the number of Web pages is essentially infinite, which makes this area even more relevant.

1.1.3 Information retrieval and Web search

Information Retrieval (IR) is the area of computer science concerned with retrieving information about a subject from a collection of data objects. This is not the same as Data Retrieval, which in the context of documents consists mainly in determining which documents of a collection contain the keywords of a user query. Information Retrieval deals with satisfying a user need:

"... the IR system must somehow 'interpret' the contents of the information items (documents) in a collection and rank them according to a degree of relevance to the user query. This 'interpretation' of a document content involves extracting syntactic and semantic information from the document text ..." [BYRN99]

Although there was an important body of Information Retrieval techniques published before the invention of the World Wide Web, there are unique characteristics of the Web that made them unsuitable or insufficient. A survey by Arasu *et al.* [ACGM+01] on searching the Web notes that:

"IR algorithms were developed for relatively small and coherent collections such as newspaper articles or book catalogs in a (physical) library. The Web, on the other hand, is massive, much less coherent, changes more rapidly, and is spread over geographically distributed computers ..." [ACGM+01]

This idea is also present in a survey about Web search by Brooks [Bro03], which states that a distinction could be made between the "closed Web", which comprises high-quality controlled collections on which a

search engine can fully trust, and the "open Web", which includes the vast majority of Web pages and on which traditional IR techniques concepts and methods are challenged.

One of the main challenges the open Web poses to search engines is "search engine spamming", i.e.: malicious attempts to get an undeserved high ranking in the results. This has created a whole branch of Information Retrieval called "adversarial IR", which is related to retrieving information from collections in which a subset of the collection has been manipulated to influence the algorithms. For instance, the vector space model for documents [Sal71], and the TF-IDF similarity measure [SB88] are useful for identifying which documents in a collection are relevant in terms of a set of keywords provided by the user. However, this scheme can be easily defeated in the "open Web" by just adding frequently-asked query terms to Web pages.

A solution to this problem is to use the hypertext structure of the Web, using links between pages as citations are used in academic literature to find the most important papers in an area. Link analysis, which is often not possible in traditional information repositories but is quite natural on the Web, can be used to exploit links and extract useful information from them, but this has to be done carefully, as in the case of Pagerank:

> "Unlike academic papers which are scrupulously reviewed, web pages proliferate free of quality control or publishing costs. With a simple program, huge numbers of pages can be created easily, artificially inflating citation counts. Because the Web environment contains profit seeking ventures, attention getting strategies evolve in response to search engine algorithms. For this reason, any evaluation strategy which counts replicable features of web pages is prone to manipulation" [PBMW98].

The low cost of publishing in the "open Web" is a key part of its success, but implies that searching information on the Web will always be inherently more difficult that searching information in traditional, closed repositories.

1.1.4 Web search and Web crawling

The typical design of search engines is a "cascade", in which a Web crawler creates a collection which is indexed and searched. Most of the designs of search engines consider the Web crawler as just a first stage in Web search, with little feedback from the ranking algorithms to the crawling process. This is a cascade model, in which operations are executed in strict order: first crawling, then indexing, and then searching.

Our approach is to provide the crawler with access to all the information about the collection to guide the crawling process effectively. This can be taken one step further, as there are tools available for dealing with all the possible interactions between the modules of a search engine, as shown in Figure 1.1.

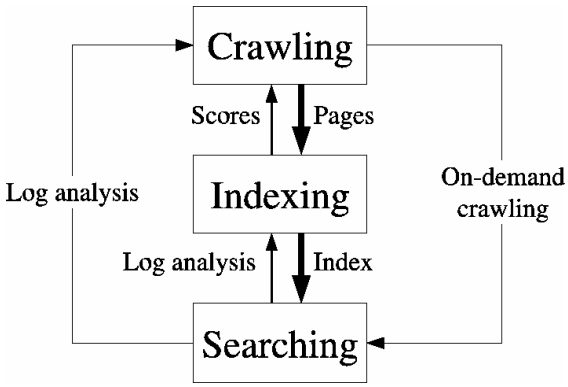


Figure 1.1: Cyclic architecture for search engines, showing how different components can use the information generated by the other components. The typical cascade model is depicted with thick arrows.

The indexing module can help the Web crawler by providing information about the ranking of pages, so the crawler can be more selective and try to collect important pages first. The searching process, through log file analysis or other techniques, is a source of optimizations for the index, and can also help the crawler by determining the "active set" of pages which are actually seen by users. Finally, the Web crawler could provide on-demand crawling services for search engines. All of these interactions are possible if we conceive the search engine as a whole from the very beginning.

1.2 The WIRE project

At the Center for Web Research (`http://www.cwr.cl/`) we are developing a software suite for research in Web Information Retrieval, which we have called WIRE (Web Information Retrieval Environment). Our aim is to study the problem of Web search by creating an efficient search engine. Search engines play a key role on the Web. Web search currently generates more than 13% of the traffic to Web sites [Sta03]. Furthermore, 40% of the users arriving to a Web site for the first time are following a link from a list of search results [Nie03].

The WIRE software suite generated several sub-projects, including some of the modules depicted in Figure 1.2.

During this thesis, the following parts of the WIRE project were developed:

- An efficient general-purpose Web crawler.
- A format for storing a Web collection.
- A tool for extracting statistics from the collection and generating reports.

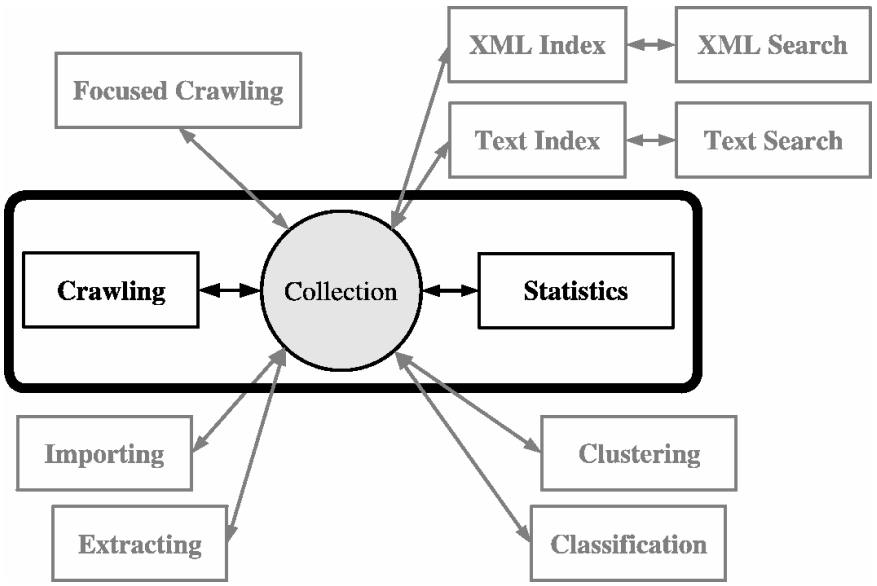


Figure 1.2: Some of the possible sub-projects of WIRE. The thick box encloses the sub-projects in which we worked during this thesis.

Our objective was to design a crawler that can be used for a collection in the order of millions or tens of millions of documents ($10^6 - 10^7$). This is bigger than most Web sites, but smaller than the complete Web, so we worked mostly with national domains (ccTLDs: country-codes top level domains such as `.cl` or `.gr`).

The main characteristics of the WIRE crawler are:

Good scalability It is designed to work with large volumes of documents, and tested with several million documents. The current implementation would require further work to scale to billions of documents (e.g.: process some data structures on disk instead of in memory).

Highly configurable All of the parameters for crawling and indexing can be configured, including several scheduling policies.

High performance It is entirely written in C/C++ for high performance. The downloader modules of the WIRE crawler ("harvesters") can be executed in several machines.

Open-source The programs and the code are freely available under the GPL license.

1.3 Scope and organization of this thesis

This thesis focuses on Web crawling, and we study Web crawling at many different levels. Our starting point is a crawling model, and in this framework we develop algorithms for a Web crawler. We aim at designing an efficient Web crawling architecture, and developing a scheduling policy to download pages from the Web that is able to download the most "valuable" pages early during a crawling process.

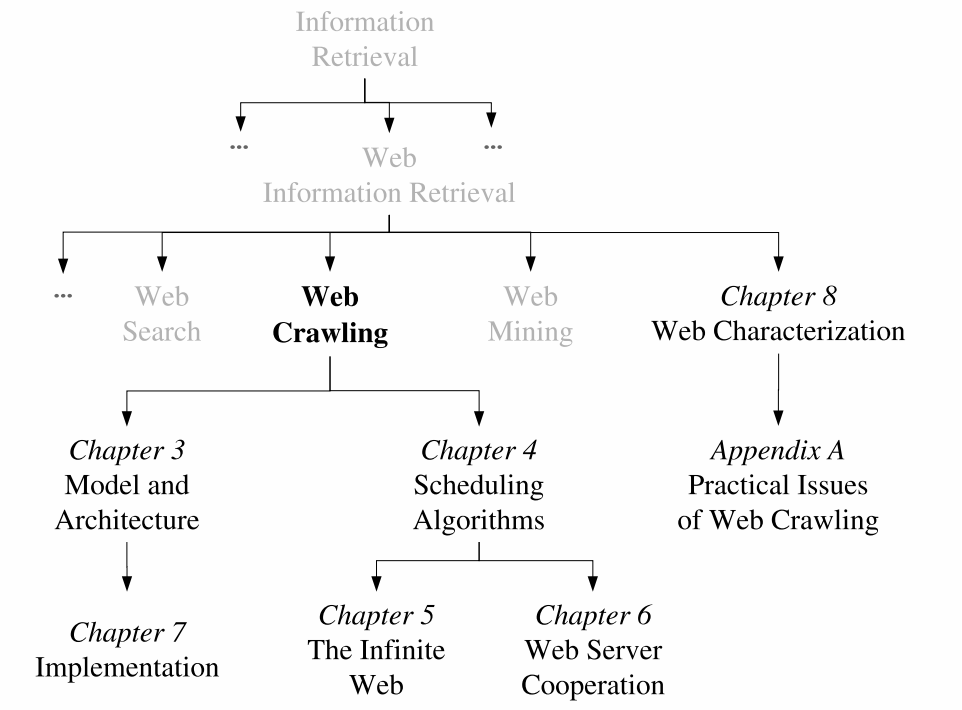


Figure 1.3: Main topics covered in this thesis. Web crawling is important in the context of Web information retrieval, because it is required for both Web search and Web characterization.

The topics covered in this thesis are shown in Figure 1.3. The topics are entangled, i.e., there are several relationships that make the development non-linear. The crawler implementation is required for Web characterization, but a good crawler design needs to consider the characteristics of the collection. Also, the crawler architecture is required for implementing the scheduling algorithms, but the result of the scheduling experiments drives the design of the crawler's architecture.

We try to linearize this research process to present it in terms of chapters, but this is not the way the actual research was carried out: it was much more cyclic and iterative than the way it is presented here.

The following is an outline of the contents of this thesis. The first chapters explore theoretical aspects of Web crawling:

- Chapter 2 reviews selected publications related to the topics covered in this thesis, including Web search, link analysis and Web crawling. The next chapters are organized into two parts: one theoretical and one practical.

- Chapter 3 introduces a new model for Web crawling, and a novel design for a Web crawler that integrates it with the other parts of a search engine. Several issues of the typical crawling architectures are discussed and the architecture of the crawler is presented as a solution to some of those problems.

- Chapter 4 compares different policies for scheduling visits to Web pages in a Web crawler. These

algorithms are tested on a simulated Web graph, and compared in terms of how soon they are able to find pages with the larger values of Pagerank. We show how in practice we can reach 80% of the total Pagerank value downloading just 50% of the Web pages.

- Chapter 5 studies an important problem of Web crawling, namely, the fact that the number of Web pages in a Web site can be potentially infinite. We use observations from actual Web sites to model user browsing behavior and to predict how "deep" we must explore Web sites to download a large fraction of the pages that are actually visited.

- Chapter 6 proposes several schemes for Web server cooperation. The administrator of a Web site has incentives to improve the representation of the Web site in search engines, and this chapter describes how to accomplish this goal by helping the Web crawler.

The last chapters empirically explore the problems of Web crawling:

- Chapter 7 details implementation issues related to the design and to algorithms presented in the previous chapters, including the data structures and key algorithms used.

- Chapter 8 presents the results of a characterization study of the Chilean Web, providing insights that are valuable for Web crawler design.

Finally, Chapter 9 summarizes our contributions and provides guidelines for future work in this area. We have also included in Appendix A a list of practical issues of Web crawling that were detected only after carrying several large crawls. We propose solutions for each problem to help other crawler designers.

Finally, the bibliography includes over 150 references to publications in this area. The next chapter is a survey about the most important ones in the context of this thesis.

Chapter 2

Related Work

In this chapter we review selected publications related to the topics covered in this thesis.

We start in section 2.1 with a summary of several studies about Web characterization that include results relevant for Web crawling. We continue in section 2.2 with an outline on how search engines index pages from the Web. Section 2.3 provides an overview of publications on link analysis in general, in section 2.4 we review specific issues of Web crawling and their solutions, and in section 2.5 we cover the architecture of existing Web crawlers.

2.1 Web characterization

2.1.1 Methods for sampling

One of the main difficulties involved in any attempt of Web characterization is how to obtain a good sample. As there are very few important pages lost in a vast amount of unimportant pages (according to any metric: Pagerank, reference count, page size, etc.), just taking a URL at random is not enough. For many applications, pages with little or no meaningful content should be excluded, so it is important to estimate the importance of each page [HHMN00], even if we have only partial information.

We distinguish two main methods for sampling Web pages:

Vertical sampling involves gathering pages restricted by domain names. As the domain name system induces a hierarchical structure, vertical sampling can be done at different levels of the structure. When vertical sampling is done at top-level it can select entire countries such as `.cl`, `.it`, `.au`, which are expected to be cohesive in terms of language, topics, history, or it can select general top-level domains such as `.edu` or `.com`, which are less coherent, except for the `.gov` domain. When vertical sampling is done at second level, it will choose a set of pages produced by members of the same organization (e.g. `stanford.edu`).

Countries that have been the subject of Web characterization studies include Brazil [VdMG+00], Chile [BYP03], Portugal [GS03], Spain [BY03], Hungary [BCF+03] and Austria [RAW+02].

Horizontal sampling involves a criteria of selection that is not based on domain names. In this case, there are two approaches for gathering data: using a log of the transactions in the proxy of a large organization or ISP, or using a Web crawler. There are advantages and disadvantages for each method: when monitoring a proxy it is easy to find popular pages, but the revisit period is impossible to control, as it depends on users; using a crawler the popularity of pages has to be estimated but the revisit period can be fine-tuned.

In horizontal sampling, a "random walk" can be used to obtain a set in which pages are roughly visited with probability proportional to their Pagerank values, and then obtain a sample taken from this set with probability inversely proportional to Pagerank, so the sample is expected to be unbiased [HHMN99, HHMN00].

2.1.2 Web dynamics

There are two areas of Web dynamics: studying the Web growth and studying the document updates [RM02]; we will focus on the study of document updates, i.e.: the change of the Web in terms of creations, updates and deletions. For a model of the growth of the number of pages per Web site, see the study by Huberman and Adamic [HA99].

When studying document updates, the data is obtained by repeated access to a large set of pages during a period of time.

For each page p and each visit, the following information is available:

- The access time-stamp of the page: visit_p.
- The last-modified time-stamp (given by most Web servers; about 80%-90% of the requests in practice): modified_p.
- The text of the page, which can be compared to an older copy to detect changes, especially if modified_p is not provided.

The following information can be estimated if the re-visiting period is short:

- The time at which the page first appeared: created_p.
- The time at which the page was no longer reachable: deleted_p. Koehler [Koe04] noted that pages that are unreachable may become reachable in the future, and many pages exhibit this behavior, so he prefers the term "comatose page" instead of "dead page".

In all cases, the results are only an estimation of the actual values because they are obtained by **polling** for events (changes), not by the resource **notifying** events, so it is possible that between two accesses a Web page changes more than once.

Estimating freshness and age

The probability that a copy of p is up-to-date at time t, $u_p(t)$ decreases with time if the page is not re-visited.

Brewington and Cybenko [BCS+00] considered that if changes to a given page occur at independent intervals, i.e., page change is a memory-less process, then this can be modeled as a Poisson process. However, it is worth noticing that most Web page changes exhibit certain periodicity –because most of the updates occur during business hours in the relevant time zone for the studied sample– so the estimators that do not account for this periodicity are more valid on the scales of weeks or months than on smaller scales.

When page changes are modeled as a Poisson process, if t units of time have passed since the last visit, then:

$$u_p(t) \propto e^{-\lambda_p t} \tag{2.1}$$

The parameter λ_p characterizes the rate of change of the page p and can be estimated based on previous observations, especially if the Web server provides the last modification date of the page whenever it is visited. This estimation for λ_p was obtained by Cho and Garcia-Molina [CGM03b]:

$$\lambda_p \approx \frac{(X_p - 1) - \frac{X_p}{N_p \log(1 - X_p/N_p)}}{S_p T} \tag{2.2}$$

- N_p number of visits to p.
- S_p time since the first visit to p.
- X_p number of times the server has informed that the page has changed.
- T_p total time with no modification, according to the server, summed over all the visits.

If the server does not give the last-modified time, we can still check for modifications by comparing the downloaded copies at two different times, so X_p now will be the number of times a modification is detected. The estimation for the parameter in this case is:

$$\lambda_p \approx \frac{-N_p \log(1 - X_p/N_p)}{S_p} \tag{2.3}$$

The above equation requires $X_p < N_p$, so if the page changes every time it is visited, we cannot estimate its change frequency.

Characterization of Web page changes

There are different time-related metrics for a Web page, the most used are:

- Age: $\text{visit}_p - \text{modified}_p$.
- Lifespan: $\text{deleted}_\text{p} - \text{created}_p$.
- Number of changes during the lifespan: changes_p.
- Average change interval: $\text{lifespan}_p / \text{changes}_p$.

Once an estimation of the above values has been obtained for Web pages in the sample, useful metrics for the entire sample are calculated, for instance:

- Distribution of change intervals.
- Average lifespan of pages.
- Median lifespan of pages, i.e.: time it takes for 50% of the pages to change. This is also called the "half-life" of the Web –a term borrowed from physics.

Selected results about Web page changes are summarized in Table 2.1.

The methods for the study of these parameters vary widely. Some researchers focus on the lifespan of pages, as they are concerned with the "availability" of Web content. This is an important subject from the point of view of researchers, as it is being common to cite on-line publications as sources, and they are expected to be persistent over time –but they are not.

Other publications focus on the rate of change of pages, which is more directly related to Web crawling, as knowing the rate of change can help to produce a good re-visiting order.

2.1.3 Link structure

About computer networks, Barabási [Bar01] noted: "While entirely of human design, the emerging network appears to have more in common with a cell or an ecological system than with a Swiss watch."

The graph representing the connections between Web pages has a scale-free topology and a macroscopic structure that are different from the properties of a random graph. A Web crawler designer must be aware of these special characteristics.

Table 2.1: Summary of selected results about Web page changes, ordered by increasing sample size. In general, methods for Web characterization studies vary widely and there are few comparable results.

Reference	Sample	Observations
[Koe04]	360 random pages, long-term study	Half-life $\approx$ 2 years 33% of pages lasted for 6 years
[MB03]	500 scholarly publications	Half-life $\approx$ 4.5 years
[GS96]	2,500 pages, university Website	Average lifespan $\approx$ 50 days Median age $\approx$ 150 days
[Spi03]	4,200 scholarly publications	Half-life $\approx$ 4 years
[Cho00]	720,000 pages, popular sites	Average lifespan $\approx$ 60 – 240 days 40% of pages in `.com` change every day 50% of pages in `.edu` and `.gov` remain the same for 4 months
[DFKM97]	950,000 pages	Average age between 10 days and 10 months Highly-linked pages change more frequently
[NCO04]	4 million pages, popular sites	8% of new pages every week 62% of the new pages have novel content 25% of new links every week 80% of page changes are minor
[FMNW03]	150 million pages,	65% of pages don't change in a 10-week period 30% of pages have only minor changes Large variations of availability across domains
[BCS+00]	800 million pages	Average lifespan $\approx$ 140 days

Scale-free networks

Scale-free networks, as opposed to random networks, are characterized by an uneven distribution of links. These networks have been the subject of a series of studies by Barabási [Bar02], and are characterized as networks in which the distribution of the number of links $\Gamma(p)$ to a page p follows a power law:

$$Pr(\Gamma(p) = k) \propto k^{-\theta} \tag{2.4}$$

A scale-free network is characterized by a few highly-linked nodes that act as "hubs" connecting several nodes to the network. The difference between a random network and a scale-free network is depicted in

Figure 2.1.

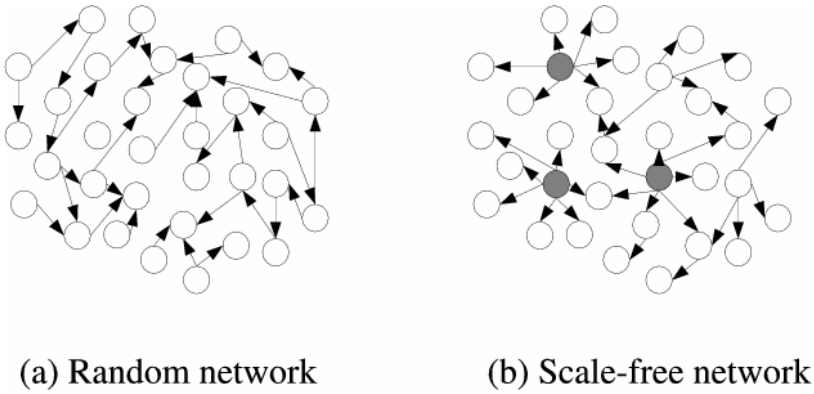


Figure 2.1: Examples of a random network and a scale-free network. Each graph has 32 nodes and 32 links. Note that both were chosen to be connected and to look nice on the plane, so they are not entirely random.

Scale-free networks arise in a wide variety of contexts, and there is a substantial amount of literature about them, so we will cite in the following just a few selected publications.

Some examples of scale-free network arising outside the realm of computer networks include:

- Acquaintances, friends and social popularity in human interactions. The Economist commented "in other words, some people have all the luck, while others have none." [Eco02].
- Sexual partners in humans, which is highly relevant for the control of sexually-transmitted diseases.
- Power grid designs, as most of them are designed in such a way that if a few key nodes fail, the entire system goes down.
- Collaboration of movie actors in films.
- Citations in scientific publications.
- Protein interactions in cellular metabolism.

Examples of scale-free networks related to the Internet are:

- Geographic, physical connectivity of Internet nodes.
- Number of links on Web pages.
- User participation in interest groups and communities.
- E-mail exchanges.

These scale-free networks do not arise by chance alone. Erdõs and Rényi [ER60] studied a model of growth for graphs in which, at each step, two nodes are chosen uniformly at random and a link is inserted between them. The properties of these random graphs are not consistent with the properties observed in scale-free networks, and therefore a model for this growth process is needed.

The connectivity distribution over the entire Web is very close to a power law, because there are a few Web sites with huge numbers of links, which benefit from a good placement in search engines and an established presence on the Web. This has been called the "winners take all" phenomenon.

Barabási and Albert [BA99] propose a "rich get richer" generative model in which each new Web page creates link to existent Web pages with a probability distribution with is not uniform, but proportional to the current in-degree of Web pages. According to this process, a page with many in-links will attract more in-links that a regular page. This generates a power-law but the resulting graph differs from the actual Web graph in other properties such as the presence of small tightly connected communities.

A different generative model is the "copy" model studied by Kumar *et al.* [KRR+00], in which new nodes choose an existent node at random and copy a fraction of the links of the existent node. This also generates a power law.

However, if we look at communities of interests in a specific topic, discarding the major hubs of the Web, the distribution of links is no longer a power law but resembles more a Gaussian distribution, as observed by Pennock *et al.* [PFL+02] in the communities of the home pages of universities, public companies, newspapers and scientists. Based on these observations, the authors propose a generative model that mixes preferential attachment with a baseline probability of gaining a link.

Macroscopic structure

The most complete study of the Web structure [BKM+00] focuses on the connectivity of a subset of 200 million Web pages from the Altavista search engine. This subset is a connected graph, if we ignore the direction of the links.

The study starts by identifying in the Web graph a single large strongly connected component (i.e.: all of the pages in this component can reach one another along directed links). They call the larger strongly connected component "MAIN". Starting in MAIN, if we follow links forward we find OUT, and if we follow links backwards we find IN. All of the Web pages with are part of the graph but do not fit neither MAIN, IN, nor OUT are part of a fourth component called TENTACLES.

A page can describe several documents and one document can be stored in several pages, so we decided to study the structure of how Web sites were connected, as Web sites are closer to real logical units. Not surprisingly, we found in [BYC01] that the structure in the `.cl` (Chile) domain at the Web site level was similar to the global Web – another example of the autosimilarity of the Web – and hence we use the same

notation of [BKM+00]. The components are defined as follows:

(a) MAIN, sites that are in the strong connected component of the connectivity graph of sites (that is, we can navigate from any site to any other site in the same component);

(b) IN, sites that can reach MAIN but cannot be reached from MAIN;

(c) OUT, sites that can be reached from MAIN, but there is no path to go back to MAIN; and

(d) other sites that can be reached from IN or can only reach OUT (TENTACLES), sites in paths between IN and OUT (TUNNEL), and unconnected sites (ISLANDS).

Figure 2.2 shows all these components.

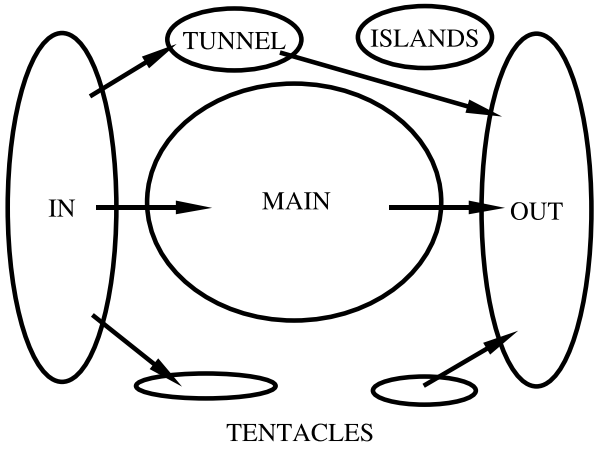


Figure 2.2: Macroscopic structure of the Web. The MAIN component is the biggest strongly connected component in the graph. The IN and OUT components can reach and be reached from the MAIN components, and there are other minor structures. There is a significant portion of Web sites which are disconnected from the Web graph in the ISLAND portion.

2.1.4 User sessions on the Web

User sessions on the Web are usually characterized through models of random surfers, such as the ones studied by Diligenti *et al.* [DGM04]. As we have seen, these models have been used for page ranking with the Pagerank algorithm [PBMW98], or to sample the Web [HHMN00].

The most used source for data about the browsing activities of users are the access log files of Web servers, and there are several log file analysis software available: [Tur04, web04b, Bou04, Bar04]. A common goal for researchers in this area is to try to infer rules in user browsing patterns, such as “40% users that visit page *A* also visit page *B*” to assist in Web site re-design. Log file analysis has a number of restrictions arising from the implementation of HTTP, especially caching and proxies, as noted by Haigh and Megarity

[HM98]. *Caching* implies that re-visiting a page is not always recorded, and re-visiting pages is a common action, and can account for more than 50% of the activity of users, when measuring it directly in the browser [TG97]. *Proxies* implies that several users can be accessing a Web site from the same IP address.

To process log file data, careful data preparation must be done [CMS99, BS00, TT04]. An important aspect of this data preparation is to separate automated sessions from user sessions. Robot session characterization was studied by Tan and Kumar [TK02].

The visits to a Web site have been modeled as a sequence of decisions by Huberman *et al.* [HPPL98]. After each click, the user finds a page with a certain value that is the value of the last page plus a random variable with a normal distribution. Under this model, the maximum depth on a Web site follows an inverse Gaussian distribution that gives a better fit than a geometric distribution but uses two parameters instead of one. The probability of a session of length t is approximately $t^{-3/2}$.

Lukose and Huberman later extended this model [LH98] by adding a third parameter that represents the discount value for future pages. This model can be used to design an algorithm for automatic browsing, which is also the topic of a recent work by Liu *et al.* [LZY04].

In [AH00], it is shown that the surfing paths for different categories have different length, for instance, user seeking for adult content tend to see more pages than users seeking for other types of information. This is a motivation to study several different Web sites as user sessions can be different among them.

Levene *et al.* [LBL01] proposed to use an absorbing state to represent the user leaving the Web site, and analyzed the lengths of user sessions when the probability of following a link is either constant (as in Model B presented later), or decreases with session length. In the two Web sites studied, the distribution of the length of user sessions is better modeled by an exponential decrease of the probability of following a link as the user enters the Web site.

2.2 Indexing and querying Web pages

The Web search process has two main parts: off-line and on-line.

The off-line part is executed periodically by the search engine, and consists in downloading a sub-set of the Web to build a collection of pages, which is then transformed into a searchable index.

The on-line part is executed every time a user query is executed, and uses the index to select some candidate documents that are sorted according to an estimation on how relevant they are for the user's need. This process is depicted in Figure 2.3.

Web pages come in many different formats such as plain text, HTML pages, PDF documents, and other proprietary formats. The first stage for indexing Web pages is to extract a standard logical view from the documents. The most used logical view for documents in search engines is the "bag of words" model, in

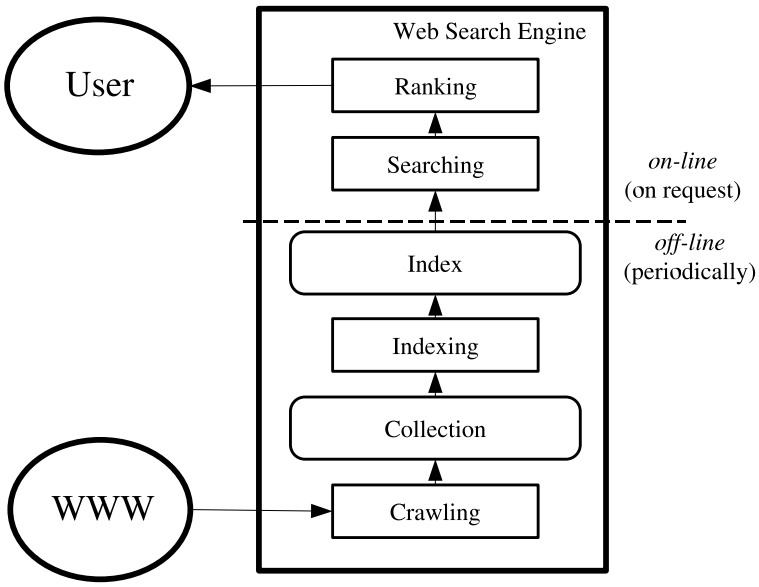


Figure 2.3: A Web search engine periodically downloads and indexes a sub-set of Web pages (off-line operation). This index is used for searching and ranking in response to user queries (on-line operation). The search engine is an interface between users and the World Wide Web.

which each document is seen only as an unordered set of words. In modern Web search engines, this view is extended with extra information concerning word frequencies and text formatting attributes, as well as meta-information about Web pages including embedded descriptions and explicit keywords in the HTML markup.

There are several text normalization operations [BY04] that are executed for extracting keywords, the most used ones are: tokenization, stopword removal and stemming.

Tokenization involves dividing the stream of text into words. While in some languages like English this is very straightforward and involves just splitting the text using spaces and punctuation, in other languages like Chinese finding words can be very difficult.

Stopwords are words that carry little semantic information, usually functional words that appear in a large fraction of the documents and therefore have little discriminating power for asserting relevance. In information retrieval stopwords are usually discarded also for efficiency reasons, as storing stopwords in an index takes considerable space because of their high frequency.

Stemming extracts the morphological root of every word. In global search engines, the first problem with stemming is that it is language dependent, and while an English rule-based stemming works well, in some cases like Spanish, a dictionary-based stemmer has to be used, while in other languages as German and Arabic stemming is quite difficult.

Other, more complex operations such as synonym translation, detecting multi-word expressions, phrase identification, named entity recognition, word sense disambiguation, etc. are used in some application do-

mains. However, some of these operations can be computationally expensive and if they have large error rates, then they can be useless and even harm retrieval precision.

2.2.1 Inverted index

An inverted index is composed of two parts: a vocabulary and a list of occurrences. The vocabulary is a sorted list of all the keywords, and for each term in the vocabulary, a list of all the "places" in which the keyword appears in the collection is kept. Figure 2.4 shows a small inverted index, considering all words including stopwords. When querying, the lists are extracted from the inverted index and then merged. Queries are very fast because usually hashing in memory is used for the vocabulary, and the lists of occurrences are pre-sorted by some global relevance criteria.

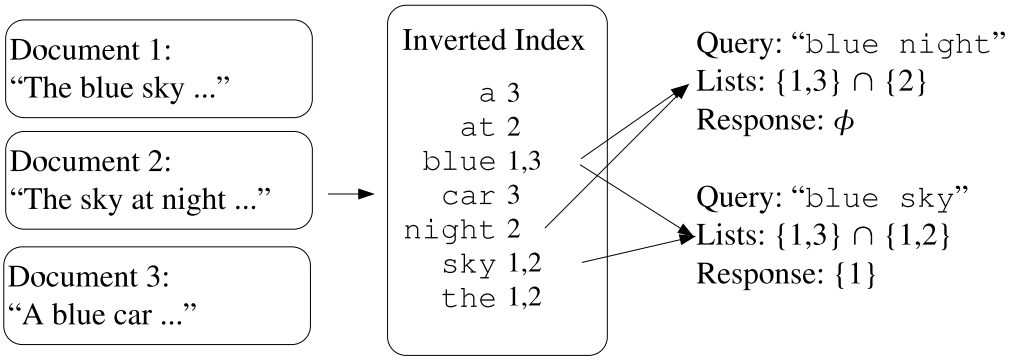


Figure 2.4: A sample inverted index with three documents. All tokens are considered for the purpose of this example, and the only text normalization operation is convert all tokens to lowercase. Searches involving multiple keywords are solved using set operations.

The granularity of the choice of the items in the list of occurrences determines the size of the index, and a small size can be obtained by storing only the document identifiers of the corresponding documents. If the search engine also stores the position where the term appears on each page the index is larger, but can be used for solving more complex queries such as queries for exact phrases, or proximity queries.

While the vocabulary grows sub-linearly with the collection size, the list of occurrences can be very large. The complete inverted index can take a significant fraction of the space occupied by the actual collection. An inverted index does not fit in main memory for a Web collection, so several partial indices are built. Each partial index represents only a subset of the collection and are later merged into the full inverted index.

In Figure 2.5 the main stages of the indexation process are depicted. During parsing, links are extracted to build a Web graph, and they can be analyzed later to generate link-based scores that can be stored along withe the rest of the metadata.

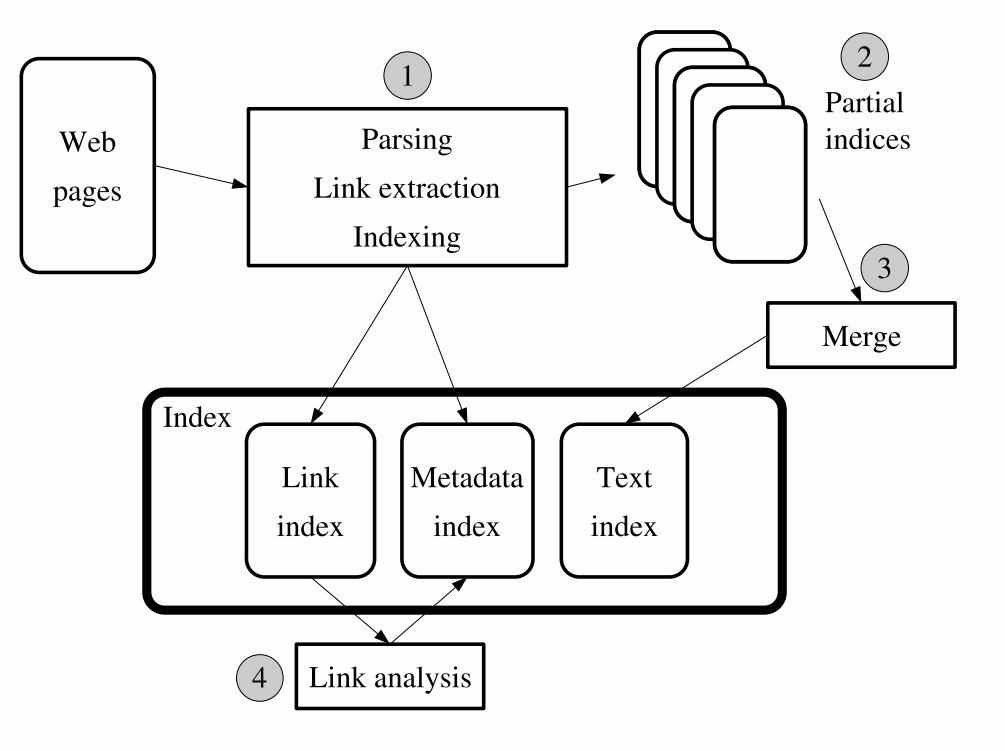


Figure 2.5: Indexing for Web search. (1) Pages are parsed and links and extracted. (2) Partial indices are written on disk when main memory is exhausted. (3) Indices are merged into a complete text index. (4) Off-line link analysis can be used to calculate static link-based scores.

2.2.2 Distributing query load

Query response time in today's search engines requires to be very fast, and should be done in a parallel way involving several machines. For parallelization, the inverted index is usually distributed among several physical computers. To partition the inverted index, two techniques are used: global inverted file and local inverted file [TGM93].

When using a global inverted file, the vocabulary is divided into several parts containing roughly the same amount of occurrences. Each computer is assigned a part of the vocabulary and all of its occurrences. Whenever a query is received, the query is sent to the computers holding the query terms, and the results are merged afterwards. Hence, load balancing is not easy.

When using a local inverted file, the document identifiers are divided, but each computer gets the full vocabulary. That is, step 3 in Figure 2.5 is omitted. A query is then broadcasted to all computers, obtaining good load balance. This is the architecture used in main search engines today, as building and maintaining a global index is hard.

Query processing involves a central "broker" that is assigned the task of distributing incoming queries and merging the results. As the results are usually shown in groups of 10 or 20 documents per page, the

broker does not need to request or merge full lists, only the top most results from each partial list.

Search engines exploit the fact that users seldom go past the first or second page of results. Search engines provide approximate result counts because they never perform a full merge of the partial result lists, so the total number of documents in the intersection can only be estimated. For this reason, when a user asks for the second or third page of results for a query, it is common that the full query is executed again.

2.2.3 Text-based ranking

The vector space model [Sal71] is the standard technique for ranking documents according to a query. Under this model, both a document and a query are seen as a pair of vectors in a space with as many dimensions as terms as the vocabulary. In a space defined in this way, the similarity of a query to a document is given by a formula that transforms each vector using certain weights and then calculates the cosine of the angle between the two weighted vectors:

$$sim_{(q,d)} = \frac{\sum_t w_{t,q} \times w_{t,d}}{\sqrt{\sum_t w_{t,q}^2} \times \sqrt{\sum_t w_{t,d}^2}}$$

In pure text-based information retrieval systems, documents are shown to the users in decreasing order using this similarity measure.

A weighting scheme uses statistical properties from the text and the query to give certain words more importance when doing the similarity calculation. The most used scheme is the TF-IDF weighting scheme [SB88], that uses the frequency of the terms in both queries and documents to compute the similarity.

TF stands for **term frequency**, and the idea is that a that if a term appears several times in a document it is better as for describing the contents of that document. The TF is usually normalized with respect to document length, that is, the parameter used is the frequency of term t divided by the frequency of the most frequent term in document d:

$$tf_{t,d} = \frac{freq_{t,d}}{\max_\ell freq_{\ell,d}}$$

IDF stands for **inverse document frequency** and reflects how frequent a term is in the whole collection. The rationale is that a term that appears in a few documents gives more information that a term that appears in many documents. If N is the number of documents and n_t if the number of documents containing the query term t, then $idf_t = \log \frac{N}{n_t}$.

Using these measures, the weight of each term in given by:

$$w_{t,q} = \left(\frac{1}{2} + \frac{1}{2} tf_{t,q}\right) idf_t \,, \qquad w_{t,d} = tf_{t,d}$$

The $1/2$ factor is added to avoid a query term having 0 weight. Several alternative weighting schemes have been proposed, but this weighting scheme is one of the most used and gives good results in practice.

2.3 Connectivity-based ranking

Web links provide a source of valuable information. In a context in which the number of pages is very large, and there are no trusted measures for asserting the quality of pages, Web links can be used as a collective, "emerging" measure of page quality.

It is known that Web pages sharing a link are more likely to be topically related that unconnected Web pages [Dav00]. The key assumption of connectivity-based ranking goes one step further, and asserts that a hyperlink from a page p' to a page p, means, in a certain way, that the content of page p is endorsed by the author of page p'.

Several algorithms for connectivity-based ranking based on this assumption are the subject of a survey by Henzinger [Hen01], and can be partitioned into:

- *Query-independent ranking*, that assign a fixed score to each page in the collection.
- *Query-dependent ranking*, or topic-sensitive ranking, that assign a score to each page in the collection in the context of a specific query.

2.3.1 Query-independent ranking

The first connectivity based query-independent ranking method was called Hyperlink Vector Voting (HVV) and was introduced by Li [Li98]. The HVV method uses the keywords appearing inside the links to a Web page to confer it a higher score on those keywords. Only the count of keyword-link pairs is used, so this ranking function is relatively easy to manipulate to get an undeserved ranking.

The Pagerank algorithm, introduced by Page *et al.* [PBMW98], is currently an important part of the ranking function used by the Google search engine [goo04]. The definition of Pagerank is recursive, stating in simple terms that "a page with high Pagerank is a page referenced by many pages with high Pagerank". Pagerank can be seen as a recursive HVV method.

To calculate the Pagerank, each page on the Web is modeled as a state in a system, and each hyperlink as a transition between two states. The Pagerank value of a page is the probability of being in a given page when this system reaches its stationary state.

A good metaphor for understanding this is to imagine a "random surfer", a person who visits pages at random, and upon arrival to each page, chooses an outgoing link uniformly at random from the links in that page. The Pagerank of a page is the fraction of time the random surfer spends at each page.

This simple system can be modeled by the following equation of a "simplified Pagerank". In this and the following equations, p is a Web page, $\Gamma^-(p)$ is the set of pages pointing to p, and $\Gamma^+(p)$ is the set of

pages p points to.

$$Pagerank'(p) = \sum_{x \in \Gamma^-(p)} \frac{Pagerank'(x)}{|\Gamma^+(x)|} \tag{2.5}$$

However, actual Web graphs include many pages with no out-links, which act as "rank sinks" as they accumulate rank but never distribute it to other pages. In stationary state, only they would have Pagerank > 0. These pages can be removed from the system and their rank computed later. Also, we would like pages not to accumulate ranking by using indirect self-references –self-links are easy to remove– not passing all of their score to other pages. For these reasons, most of the implementations of Pagerank add "random jumps" to each page. These random jumps are hyperlinks from every page to all pages in the collection, including itself, which provide a minimal rank to all the pages as well as a damping effect for self-reference schemes.

In terms of the random surfer model, we can state that when choosing the next step, the random surfer either chooses a page at random from the collection with probability ε, or chooses to follow a link from the current page with probability $1-\varepsilon$. This is the model used for calculating Pagerank in practice, and it is described by the following equation:

$$Pagerank(p) = \frac{\varepsilon}{N} + (1-\varepsilon) \sum_{x \in \Gamma^-(p)} \frac{Pagerank(x)}{|\Gamma^+(x)|} \tag{2.6}$$

N is the number of pages in the collection, and the parameter ε is typically between 0.1 and 0.2, based on empirical evidence. Pagerank is a global, static measure of quality of a Web page, very efficient in terms of computation time, as it only has to be calculated once at indexing time and is later used repeatedly at query time.

Note that Pagerank can also be manipulated and in fact there are thousands or millions of Web pages created specifically for the objective of deceiving the ranking function. Eiron *et al.* [EMT04] found that:

> "Among the top 20 URLs in our 100 million page Pagerank calculation using teleportation to random pages, 11 were pornographic, and they appear to have all been achieved using the same form of link manipulation. The specific technique that was used was to create many URLs that all link to a single page, thereby accumulating the Pagerank that every page receives from random teleportation, and concentrating it into a single page of interest."

Another paradigm for ranking pages based on a Markov chain is an absorbing model introduced by Amati *et al.* [AOP03, POA03]. In this model, the original Web graph is transformed adding, for each node, a "clone node" with no out-links. Each clone node p' is only linked from one node in the original graph p. When this system reaches stationary state, only the clone nodes have probabilities greater than zero. The probability of the clone node p' is interpreted as the score of the original node p. This model performs better than Pagerank for some information retrieval tasks.

A different paradigm for static ranking on the Web is the network flow model introduced by Tomlin [Tom03]. For ranking pages, a (sub)graph of the Web is considered as carrying a finite amount of fluid, and edges between nodes are pipes for this fluid. Using an entropy maximization method, two measures are obtained: a "TrafficRank" that is an estimation of the maximum amount of flow through a node in this network model, and a "page temperature", which is a measure of the importance of a Web page, obtained by solving the dual of this optimization problem. Both measures can be used for ranking Web pages, and both are independent of Pagerank.

The models presented in this section summarize each page on the Web with a single number, or a pair of numbers, but as the creators of Pagerank note, "the importance of a Web page is an inherently subjective matter that depends on readers interests, knowledge and attitudes" [PBMW98]; this is why query-dependent ranking is introduced to create ranking functions that are sensitive to user's needs.

2.3.2 Query-dependent ranking

In query-dependent ranking, the starting point is a "neighborhood graph": a set of pages that are expected to be relevant to the given query. Carriere and Kazman [CK97] propose to build this graph by starting with a set of pages containing the query terms; this set can be the list of results given by a full-text search engine. This *root set* is augmented by its "neighborhood" that comprises all (or a large sample) of the pages directly pointing to, or directly pointed by, pages in the root set. The construction procedure of the neighborhood set is shown in Algorithm 1.

Figure 2.6 depicts the process of creation of the neighborhood set. The idea of limiting the number of pages added to the neighborhood set by following back links was not part of the original proposal, but was introduced later [BH98].

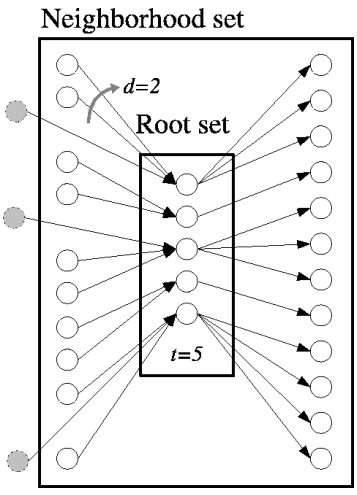


Figure 2.6: Expansion of the root set with $t = 5$ and $d = 2$. t is the number of pages in the root set, and d is the maximum number of back-links to include in the neighborhood set.

Algorithm 1 Creation of the neighborhood set S_σ of query σ

Require: σ query

Require: $t > 0$, size of root set.

Require: $d > 0$ number of back-links to include per page.

1: $R_\sigma \leftarrow$ top t results using a search engine.
2: $S_\sigma \leftarrow \emptyset$
3: **for all** $p \in R_\sigma$ **do**
4: Let $\Gamma^+(p)$ denote all pages p points to
5: Let $\Gamma^-(p)$ denote all pages pointed by p
6: $S_\sigma \leftarrow S_\sigma \cup \Gamma^+(p)$
7: **if** $|\Gamma^-(p)| \leq d$ **then**
8: $S_\sigma \leftarrow S_\sigma \cup \Gamma^-(p)$
9: **else**
10: $S_\sigma \leftarrow S_\sigma \cup$ an arbitrary set of d pages in $\Gamma^-(p)$
11: **end if**
12: **end for**
13: S_σ is the neighborhood set of query σ

It is customary that when considering links in the neighborhood set, only links in different Web sites are included, as links between pages in the same Web site are usually created by the same authors as the pages themselves, and do not reflect the relative importance of a page for the general community.

The most-cited algorithm, presented by Yuwono and Lee [YL96], is the simplest form of connectivity-based query-dependent ranking: after the neighborhood set has been built, each page p in it is assigned a score that is the sum of the number of query terms appearing in the pages pointing to p. This algorithm performed poorly when compared with pure content-based analysis, and its authors concluded that links by themselves are not a reliable indicator of semantic relationship between Web pages.

A more complex idea is the HITS algorithm presented by Kleinberg [Kle99] that is based on considering that relevant pages can be either "authority pages" or "hub pages". An authority page is expected to have relevant content for a subject, and a hub page is expected to have many links to authority pages.

The HITS algorithm produces two scores for each page, called "authority score" and "hub score". These two scores have a mutually-reinforcing relationship: a page with high authority score is pointed to by many pages with a high hub score, and a page with a high hub score points to many pages with a high authority score, as shown in Figure 2.7.

An iterative version of this algorithm is shown in Algorithm 2; in this version, the number of iterations is fixed, but the algorithm can be adapted to stop based on the convergence of the sequence of iterations.

The HITS algorithm suffers from several drawbacks in its pure form. Some of them were noted and

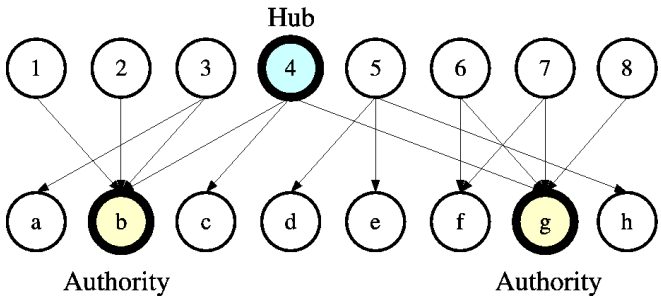


Figure 2.7: Hubs and authorities in a small graph. Node 4 is the best hub page, as it points to many authorities, and nodes *b* and *g* are the best authority pages.

Algorithm 2 Hub and authority score for each page in S_σ

Require: S_σ neighborhood set of query σ

Require: k number of iterations

1: $n \leftarrow |S_\sigma|$
2: Let z denote the vector $(1,1,1,...1) \in R^n$
3: $H_0 \leftarrow z$
4: $A_0 \leftarrow z$
5: **for** $j = 1$ to k **do**
6: **for** $i = 1$ to n **do**
7: $H_j[i] \leftarrow \sum_{x \in \Gamma^+(i)} A_{j-1}[x]$ $\{\Gamma^+(i)$ are pages i points to$\}$
8: $A_j[i] \leftarrow \sum_{x \in \Gamma^-(i)} H_j[x]$ $\{\Gamma^-(i)$ are pages pointing to $i\}$
9: **end for**
10: Normalize H_j and A_j so their components sum 1
11: **end for**
12: H_k is the vector of hub scores
13: A_k is the vector of authority scores

solved by Bharat and Henzinger [BH98]:

(a) Not all the documents in the neighborhood set are about the original topic ("topic drifting").

(b) There are nepotistic, mutually-reinforcing relationships between some hosts.

(c) There are many automatically generated links.

Problem (a) is the most important, as while expanding the root set, it is common to include popular pages that are highly-linked, but unrelated to the query topic. The solution is to use analysis of the contents of the documents when executing Algorithm 1, and pruning the neighborhood graph by removing the documents that are too different from the query. This is done using a threshold for the standard TF-IDF measure of similarity [SB88] between documents and queries.

Problems (b) and (c) can be avoided using the following heuristic: if there are k edges from documents on a host to documents in another host, then each edge is given a weight of $1/k$. This gives each document the same amount of influence on the final score, regardless of the number of links in that specific document.

A different variation of the HITS algorithm, designed specifically to avoid "topic drifting", was presented by Chakrabarti *et al.* [CDR+98]. In their approach, for each link, the text near it in the origin page and the full text of the destination page are compared. If they are similar, the link is given a high weight, as it carries information about semantic similarity between the origin and destination pages. As this heuristic keeps the pages in the neighborhood set more closely related, a more relaxed expansion phase can be done. The authors propose to follow two levels of links forward and backward from the root set, instead of just one.

Another approach to query-dependent ranking is topic-sensitive Pagerank, introduced by Haveliwala [Hav02], in this method, multiple scores for each page are pre-computed at indexing time, using an algorithm similar to Pagerank. Each score represents the importance of a page for each topic from a set of pre-defined topics. At query time, the ranking is done using the query to assign weights to the different topic-sensitive Pagerank scores of each page.

2.4 Web crawling issues

There are two important characteristics of the Web that generate a scenario in which Web crawling is very difficult: its large volume and its rate of change, as there is a huge amount of pages being added, changed and removed every day. Also, network speed has improved less than current processing speeds and storage capacities. The large volume implies that the crawler can only download a fraction of the Web pages within a given time, so it needs to prioritize its downloads. The high rate of change implies that by the time the crawler is downloading the last pages from a site, it is very likely that new pages have been added to the site, or that pages that have already been updated or even deleted.

Crawling the Web, in a certain way, resembles watching the sky in a clear night: what we see reflects the state of the stars at different times, as the light travels different distances. What a Web crawler gets is not a "snapshot" of the Web, because it does not represents the Web at any given instant of time [BYRN99]. The last pages being crawled are probably very accurately represented, but the first pages that were downloaded have a high probability of have been changed. This idea is depicted in Figure 2.8.

As Edwards *et al.* note, "Given that the bandwidth for conducting crawls is neither infinite nor free it is becoming essential to crawl the Web in a not only scalable, but efficient way if some reasonable measure of quality or freshness is to be maintained." [EMT01]. A crawler must carefully choose at each step which pages to visit next.

The behavior of a Web crawler is the outcome of a combination of policies:

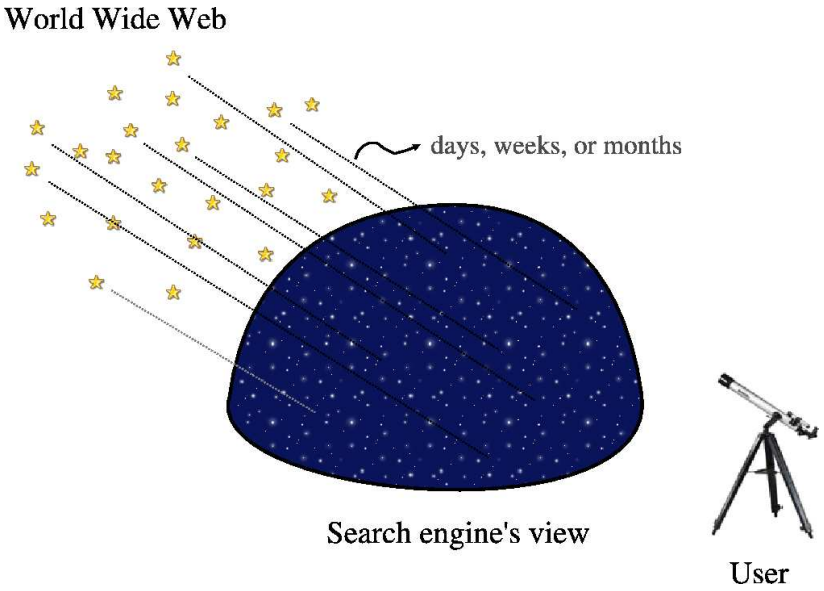


Figure 2.8: As the crawling process takes time and the Web is very dynamic, the search engine's view of the Web represents the state of Web pages at different times. This is similar to watching the sky at night, as the stars we see never existed simultaneously as we see them.

- A *selection policy* that states which pages to download.
- A *re-visit policy* that states when to check for changes to the pages.
- A *politeness policy* that states how to avoid overloading Web sites.
- A *parallelization policy* that states how to coordinate distributed Web crawlers.

2.4.1 Selection policy

Given the current size of the Web, even large search engines cover only a portion of the publicly available content; a study by Lawrence and Giles [LG00] showed that no search engine indexes more than 16% of the Web. As a crawler always downloads just a fraction of the Web pages, it is highly desirable that the downloaded fraction contains the most relevant pages, and not just a random sample of the Web.

This requires a metric of importance for prioritizing Web pages. The importance of a page is a function of its intrinsic quality, its popularity in terms of links or visits, and even of its URL (the latter is the case of vertical search engines restricted to a single top-level domain, or search engines restricted to a fixed Website). Designing a good selection policy has an added difficulty: it must work with partial information, as the complete set of Web pages is not known during crawling.

Cho *et al.* [CGMP98] made the first study on policies for crawling scheduling. Their data set was a 180,000-pages crawl from the `stanford.edu` domain, in which a crawling simulation was done with different strategies. The ordering metrics tested were breadth-first, backlink-count and partial Pagerank,

which are defined later in this article. One of the conclusions was that if the crawler wants to download pages with high Pagerank early during the crawling process, then the partial Pagerank strategy is the better, followed by breadth-first and backlink-count. However, these results are for just a single domain.

Najork and Wiener [NW01] performed an actual crawl on 328 million pages, using breadth-first ordering. They found that a breadth-first crawl captures pages with high Pagerank early in the crawl (but they did not compared this strategy against other strategies). The explanation given by the authors for this result is that "the most important pages have many links to them from numerous hosts, and those links will be found early, regardless of on which host or page the crawl originates".

Abiteboul *et al.* [APC03] designed a crawling strategy based on an algorithm called OPIC (On-line Page Importance Computation). In OPIC, each page is given an initial sum of "cash" which is distributed equally among the pages it points to. It is similar to a Pagerank computation, but it is faster and is only done in one step. An OPIC-driven crawler downloads first the pages in the crawling frontier with higher amounts of "cash". Experiments were carried in a 100,000-pages synthetic graph with a power-law distribution of in-links. However, there was no comparison with other strategies nor experiments in the real Web.

Boldi *et al.* [BSV04] used simulation on subsets of the Web of 40 million from the `.it` domain and 100 million pages from the WebBase crawl, testing breadth-first against random ordering and an omniscient strategy. The winning strategy was breadth-first, although a random ordering also performed surprisingly well. One problem is that the WebBase crawl is biased to the crawler used to gather the data. They also showed how bad Pagerank calculations carried on partial subgraphs of the Web, obtained during crawling, can approximate the actual Pagerank.

The importance of a page for a crawler can also be expressed as a function of the similarity of a page to a given query. This is called "focused crawling" and was introduced by Chakrabarti *et al.* [CvD99]. The main problem in focused crawling is that in the context of a Web crawler, we would like to be able to predict the similarity of the text of a given page to the query **before** actually downloading the page. A possible predictor is the anchor text of links; this was the approach taken by Pinkerton [Pin94] in a crawler developed in the early days of the Web. Diligenti *et al.* [DCL+00] propose to use the complete content of the pages already visited to infer the similarity between the driving query and the pages that have not been visited yet. The performance of a focused crawling depends mostly on the richness of links in the specific topic being searched, and a focused crawling usually relies on a general Web search engine for providing starting points.

2.4.2 Re-visit policy

The Web has a very dynamic nature, and crawling a fraction of the Web can take a long time, usually measured in weeks or months. By the time a Web crawler has finished its crawl, many events could have happened. We characterize these events as creations, updates and deletions [BYCSJ04]:

Creations When a page is created, it will not be visible on the public Web space until it is linked, so we assume that at least one page update –adding a link to the new Web page– must occur for a Web page creation to be visible.

A Web crawler starts with a set of starting URLs, usually a list of domain names, so registering a domain name can be seen as the act of creating a URL. Also, under some schemes of cooperation the Web server could provide a list of URLs without the need of a link, as shown in Chapter 6.

Updates Page changes are difficult to characterize: an update can be either *minor*, or *major*. An update is minor if it is at the paragraph or sentence level, so the page is semantically almost the same and references to its content are still valid. On the contrary, in the case of a major update, all references to its content are not valid anymore. It is customary to consider *any* update as *major*, as it is difficult to judge automatically if the page's content is semantically the same. Characterization of partial changes is studied in [LWP+01, NCO04].

Deletions A page is deleted if it is removed from the public Web, or if all the links to that page are removed. Note that even if all the links to a page are removed, the page is no longer visible in the Web site, but it will still be visible by the Web crawler. It is almost impossible to detect that a page has lost all its links, as the Web crawler can never tell if links to the target page are not present, or if they are only present in pages that have not been crawled.

Undetected deletions are more damaging for a search engine's reputation than updates, as they are more evident to the user. The study by Lawrence and Giles about search engine performance [LG00] reports that on average 5.3% of the links returned by search engines point to deleted pages.

Cost functions

From the search engine's point of view, there is a cost associated with not detecting an event, and thus having an outdated copy of a resource. The most used cost functions, introduced in [CGM00], are freshness and age.

Freshness This is a binary measure that indicates whether the local copy is accurate or not. The freshness of a page p in the repository at time t is defined as:

$$F_p(t) = \begin{cases} 1 & \text{if } p \text{ is equal to the local copy at time } t \\ 0 & \text{otherwise} \end{cases} \tag{2.7}$$

Age This is a measure that indicates how outdated the local copy is. The age of a page p in the repository,

at time t is defined as:

$$A_p(t) = \begin{cases} 0 & \text{if } p \text{ is not modified at time } t \\ t - \text{modification time of } p & \text{otherwise} \end{cases} \tag{2.8}$$

The evolution of these two quantities is depicted in Figure 2.9.

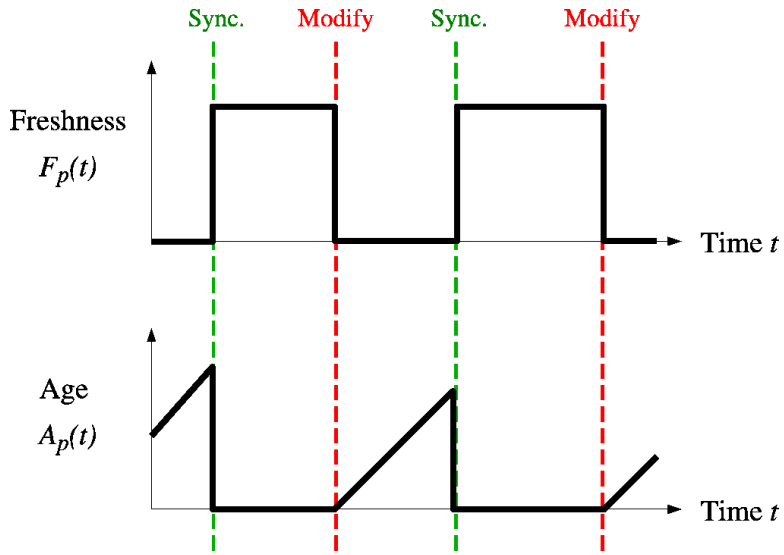


Figure 2.9: Evolution of freshness and age with time. Two types of event can occur: modification of a Web page in the server (event *modify*) and downloading of the modified page by the crawler (event *sync*).

Coffman *et al.* [EGC98] worked with a definition of the objective of a Web crawler that is equivalent to freshness, but use a different wording: they propose that a crawler must minimize the fraction of time pages remain outdated. They also noted that the problem of Web crawling can be modeled as a multiple-queue, single-server polling system, on which the Web crawler is the server and the Web sites are the queues. Page modifications are the arrival of the customers, and switch-over times are the interval between page accesses to a single Web site. Under this model, mean waiting time for a customer in the polling system is equivalent to the average age for the Web crawler.

Strategies

The objective of the crawler is to keep the average freshness of pages in its collection as high as possibly, or to keep the average age of pages as low as possible. These objectives are not equivalent: in the first case, the crawler is just concerned with *how many* pages are out-dated, while in the second case, the crawler is concerned with *how old* the local copies of pages are.

Two simple re-visiting policies were studied by Cho and Garcia-Molina [CGM03a]:

Uniform policy This involves re-visiting all pages in the collection with the same frequency, regardless of their rates of change.

Proportional policy This involves re-visiting more often the pages that change more frequently. The visiting frequency is directly proportional to the (estimated) change frequency.

In both cases, the repeated crawling order of pages can be done either at random or with a fixed order.

Cho and Garcia-Molina proved the surprising result that, in terms of average freshness, the *uniform policy* outperforms the *proportional policy* in both a simulated Web and a real Web crawl. The explanation for this result comes from the fact that, when a page changes too often, the crawler will waste time by trying to re-crawl it too fast and still will not be able to keep its copy of the page fresh. "To improve freshness, we should penalize the elements that change too often" [CGM03a].

The optimal re-visiting policy is neither the uniform policy nor the proportional policy. The optimal method for keeping average freshness high includes ignoring the pages that change too often, and the optimal for keeping average age low is to use access frequencies that monotonically (and sub-linearly) increase with the rate of change of each page. In both cases, the optimal is closer to the uniform policy than to the proportional policy: as Coffman *et al.* [EGC98] note, "in order to minimize the expected obsolescence time, the accesses to any particular page should be kept as evenly spaced as possible".

Explicit formulas for the re-visit policy are not attainable in general, but they are obtained numerically, as they depend on the distribution of page changes. Note that the re-visiting policies considered here regard all pages as homogeneous in terms of quality –all pages on the Web are worth the same– something that is not a realistic scenario, so further information about the Web page quality should be included to achieve a better crawling policy.

2.4.3 Politeness policy

As noted by Koster [Kos95], the use of Web robots is useful for a number of tasks, but comes with a price for the general community. The costs of using Web robots include:

- Network resources, as robots require considerable bandwidth, and operate with a high degree of parallelism during a long period of time.
- Server overload, especially if the frequency of accesses to a given server is too high.
- Poorly written robots, which can crash servers or routers, or which download pages they cannot handle.
- Personal robots that, if deployed by too many users, can disrupt networks and Web servers.

A partial solution to these problems is the robots exclusion protocol [Kos96] that is a standard for administrators to indicate which parts of their Web servers should not be accessed by robots. This standard does not include a suggestion for the interval of visits to the same server, even though this interval is the most effective way of avoiding server overload.

The first proposal for the interval between connections was given in [Kos93] and was 60 seconds. However, if we download pages at this rate from a Web site with more than 100,000 pages over a perfect connection with zero latency and infinite bandwidth, it would take more than 2 months to download only that entire Web site; also, we would be using a fraction of the resources from that Web server permanently. This does not seems acceptable.

Cho [CGM03b] uses 10 seconds as an interval for accesses, and the WIRE crawler [BYC02] uses 15 seconds as the default. The Mercator Web crawler [HN99] follows an adaptive politeness policy: if it took t seconds to download a document from a given sever, the crawler waits for $10 \times t$ seconds before downloading the next page. Dill *et al.* [DKM+02] use 1 second.

Anecdotal evidence from access logs shows that access intervals from known crawlers vary between 20 seconds and 3–4 minutes. It is worth noticing that even when being very polite, and taking all the safeguards to avoid overloading Web servers, some complaints from Web server administrators are received. Brin and Page note that:

> "... running a crawler which connects to more than half a million servers (...) generates a fair amount of email and phone calls. Because of the vast number of people coming on line, there are always those who do not know what a crawler is, because this is the first one they have seen." [BP98].

2.4.4 Parallelization policy

A parallel crawler is a crawler that runs multiple process in parallel. The goal is to maximize the download rate while minimizing the overhead from parallelization and to avoid repeated downloads of the same page.

To avoid downloading the same page more than once, the crawling system requires a policy for assigning the new URLs discovered during the crawling process, as the same URL can be found by two different crawling processes. Cho and Garcia-Molina [CGM02] studied two types of policy:

Dynamic assignment With this type of policy, a central server assigns new URLs to different crawlers dynamically. This allows the central server to, for instance, dynamically balance the load of each crawler.

With dynamic assignment, typically the systems can also add or remove downloader processes. The central server may become the bottleneck, so most of the workload must be transfered to the distributed crawling processes for large crawls.

There are two configurations of crawling architectures with dynamic assignment that have been described by Shkapenyuk and Suel [SS02]:

- A small crawler configuration, in which there is a central DNS resolver and central queues per Web site, and distributed downloaders.
- A large crawler configuration, in which the DNS resolver and the queues are also distributed.

Static assignment With this type of policy, there is a fixed rule stated from the beginning of the crawl that defines how to assign new URLs to the crawlers.

For static assignment, a hashing function can be used to transform URLs (or, even better, complete Web site names) into a number that corresponds to the index of the corresponding crawling process. As there are external links that will go from a Web site assigned to one crawling process to a Web site assigned to a different crawling process, some exchange of URLs must occur.

To reduce the overhead due to the exchange of URLs between crawling processes, the exchange should be done in batch, several URLs at a time, and the most cited URLs in the collection should be known by all crawling processes before the crawl (e.g.: using data from a previous crawl) [CGM02].

An effective assignment function must have three main properties: each crawling process should get approximately the same number of hosts (balancing property), if the number of crawling processes grows, the number of hosts assigned to each process must shrink (contra-variance property), and the assignment must be able to add and remove crawling processes dynamically. Boldi *et al.* [BCSV04] propose to use consistent hashing, which replicates the buckets, so adding or removing a bucket does not requires re-hashing of the whole table to achieve all of the desired properties.

2.5 Web crawler architecture

A crawler must have a good crawling strategy, as noted in the previous sections, but it also needs a highly optimized architecture. Shkapenyuk and Suel [SS02] noted that:

> "While it is fairly easy to build a slow crawler that downloads a few pages per second for a short period of time, building a high-performance system that can download hundreds of millions of pages over several weeks presents a number of challenges in system designed, I/O and network efficiency, and robustness and manageability."

Web crawlers are a central part of search engines, and details on their algorithms and architecture are kept as business secrets. When crawler designs are published, there is often an important lack of detail that prevents other from reproducing the work. There are also emerging concerns about "search engine

spamming", which prevent major search engines from publishing their ranking algorithms. The typical high-level architecture of Web crawlers is shown in Figure 2.10.

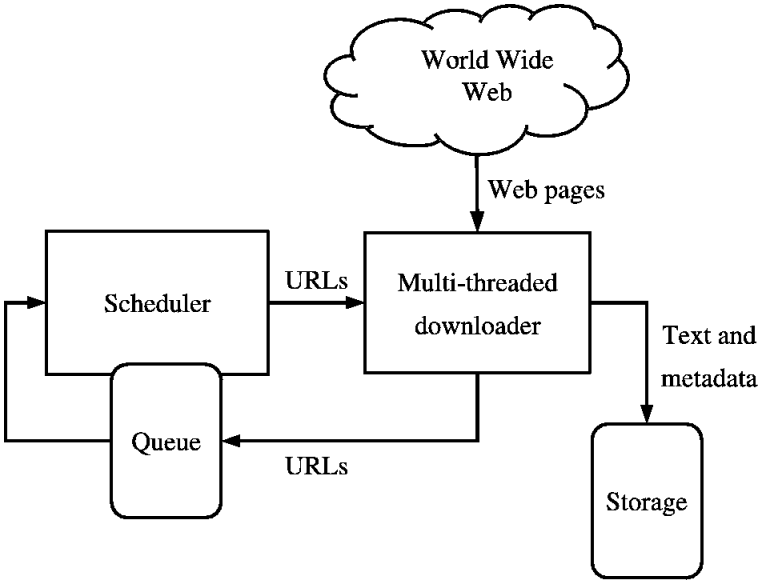


Figure 2.10: Typical high-level architecture of a Web crawler, involving a scheduler and a multi-threaded downloader. The two main data structures are the Web page (text) storage and the URL queue.

2.5.1 Examples of Web crawlers

The following is a list of published crawler architectures for general-purpose crawlers (excluding focused Web crawlers), with a brief description that includes the names given to the different components and outstanding features:

RBSE [Eic94] was the first published Web crawler. It was based on two programs: the first program, "spider" maintains a queue in a relational database, and the second program "mite", is a modified `www` ASCII browser that downloads the pages from the Web.

WebCrawler [Pin94] was used to build the first publicly-available full-text index of a sub-set of the Web. It was based on lib-WWW to download pages, and another program to parse and order URLs for breadth-first exploration of the Web graph. It also included a real-time crawler that followed links based on the similarity of the anchor text with the provided query.

World Wide Web Worm [McB94] was a crawler used to build a simple index of document titles and URLs. The index could be searched by using the `grep` UNIX command.

Internet Archive Crawler [Bur97] is a crawler designed with the purpose of archiving periodic snapshots of a large portion of the Web. It uses several process in a distributed fashion, and a fixed number

of Web sites are assigned to each process. The inter-process exchange of URLs is carried in batch with a long time interval between exchanges, as this is a costly process. The Internet Archive Crawler also has to deal with the problem of changing DNS records, so it keeps an historical archive of the hostname to IP mappings.

WebSPHINX [MB98] is composed of a Java class library that implements multi-threaded Web page retrieval and HTML parsing, and a graphical user interface to set the starting URLs, to extract the downloaded data and to implement a basic text-based search engine.

Google Crawler [BP98] is described in some detail, but the reference is only about an early version of its architecture, which was based in C++ and Python. The crawler was integrated with the indexing process, because text parsing was done for full-text indexing and also for URL extraction. There is an URL server that sends lists of URLs to be fetched by several crawling processes. During parsing, the URLs found were passed to a URL server that checked if the URL have been previously seen. If not, the URL was added to the queue of the URL server.

CobWeb [dSVG+99] uses a central "scheduler" and a series of distributed "collectors". The collectors parse the downloaded Web pages and send the discovered URLs to the scheduler, which in turns assign them to the collectors. The scheduler enforces a breadth-first search order with a politeness policy to avoid overloading Web servers. The crawler is written in Perl.

Mercator [HN99] is a modular Web crawler written in Java. Its modularity arises from the usage of interchangeable "protocol modules" and "processing modules". Protocols modules are related to how to acquire the Web pages (e.g.: by HTTP), and processing modules are related to how to process Web pages. The standard processing module just parses the pages and extract new URLs, but other processing modules can be used to index the text of the pages, or to gather statistics from the Web.

WebFountain [EMT01] is a distributed, modular crawler similar to Mercator but written in C++. It features a "controller" machine that coordinates a series of "ant" machines. After repeatedly downloading pages, a change rate is inferred for each page and a non-linear programming method must be used to solve the equation system for maximizing freshness. The authors recommend to use this crawling order in the early stages of the crawl, and then switch to a uniform crawling order, in which all pages being visited with the same frequency.

PolyBot [SS02] is a distributed crawler written in C++ and Python, which is composed of a "crawl manager", one or more "downloaders" and one or more "DNS resolvers". Collected URLs are added to a queue on disk, and processed later to search for seen URLs in batch mode. The politeness policy considers both third and second level domains (e.g.: `www.example.com` and `www2.example.com` are third level domains) because third level domains are usually hosted by the same Web server.

WebRACE [ZYD02] is a crawling and caching module implemented in Java, and used as a part of a more generic system called eRACE. The system receives requests from users for downloading Web pages, so the crawler acts in part as a smart proxy server. The system also handles requests for "subscriptions" to Web pages that must be monitored: when the pages changes, they must be downloaded by the crawler and the subscriber must be notified. The most outstanding feature of WebRACE is that, while most crawlers start with a set of "seed" URLs, WebRACE is continuously receiving new starting URLs to crawl from.

Ubicrawler [BCSV04] is a distributed crawler written in Java, and it has no central process. It is composed of a number of identical "agents"; and the assignment function is calculated using consistent hashing of the host names. There is zero overlap, meaning that no page is crawled twice, unless a crawling agent crashes (then, another agent must re-crawl the pages from the failing agent). The crawler is designed to achieve high scalability and to be tolerant to failures.

FAST Crawler [RM02] is the crawler used by the FAST search engine, and a general description of its architecture is available. It is a distributed architecture in which each machine holds a "document scheduler" that maintains a queue of documents to be downloaded by a "document processor" that stores them in a local storage subsystem. Each crawler communicates with the other crawlers via a "distributor" module that exchanges hyperlink information.

WIRE [BYC02, CBY02] is the crawler developed for this research, and is described in detail in Chapter 7 of this thesis.

In addition to the specific crawler architectures listed above, there are general crawler architectures published by Cho [CGM02] and Chakrabarti [Cha03].

A few Web crawlers have been released under the GNU public license: Larbin [Ail04], WebBase [Dac02], a free version of WebSPHINX [Mil04], GRUB [gru04] and HT://Dig [htd04]. For commercial products, see [SS04, bot04].

About practical issues of building a Web crawler, which is the subject of Appendix A, a list of recommendations for building a search engine was written by Patterson [Pat04].

2.5.2 Architectures for cooperation between Web sites and search engines

We study cooperation schemes for Web servers in Chapter 6. In this thesis, we only consider the cooperation between Web servers and crawlers, not between crawlers: this issue is studied in [McL02], using a crawler simulator and proving that crawlers can benefit from sharing information about last-modification date of pages. In this case, the cooperation between search engines occurs at crawling time, but search engines could also exchange information later, like in the "STARTS" proposal [GCGMP97].

There are several methods for keeping mirrors (replicas) of information services; these methods are not directly suitable for Web server cooperation because the crawler usually is interested in only a subset of the pages (the most interesting ones) and not in the entire site. Mirroring methods include RSYNC [TP03], that generates a series of fingerprints for "chunks" of data, and then compares those fingerprints to compress and send only the modified parts. CTM [Kam03] is a method for sending differences via e-mail, used to keep copies of source code for the Open BSD operating systems up-to-date.

A specific proposal for pushing last-modification data to Web crawlers is presented by Gupta and Campbell [GC01], including a cost model in which the meta-data is sent only if the Web site is misrepresented above a certain threshold in the search engine. A more general Internet notification system was presented by Brandt and Kristensen [BK97].

The Distribution and Replication Protocol (DRP) [vHGH+97] provides a protocol to distribute data using HTTP and data fingerprinting and index files. Another proposal that uses a series of files containing descriptions of Web pages, is presented in [BCGMS00].

DASL [RRDB02], the DAV searching and locating protocol, is a proposed extension to DAV that will allow searching the Web server using an HTTP query with certain extensions, but neither the query syntax nor the query semantics are specified by the protocol.

2.6 Conclusions

In this chapter, we have surveyed selected publications from the related work that are relevant for this thesis. We have focused in link analysis and Web crawling.

In the literature, we found that link analysis is an active research topic in the information retrieval community. The Web is very important today because it is the cornerstone of the information age, and is used by millions of persons every day, and it is natural that it provides opportunities for both business and research. Link analysis is, in a sense, the most important "new" component of the Web in relation to previous document collections and traditional information retrieval, and probably this explain why the field of link analysis has been so active.

On the contrary, the topic of Web crawling design is not represented so well in the literature, as there are few publications available. Web crawling research is affected by business secrecy because Web search engines, in a sense, mediate the interaction between users and Web sites and are the key for success of many Web sites. There is also secrecy involved because there are many concerns about search engine spamming, because there are no known ranking functions absolutely resilient to malicious manipulation, so ranking functions and crawling methods are usually not published.

The next chapter starts the main part of this thesis by presenting a new crawling model and architecture.

Chapter 3

A New Crawling Model and Architecture

Web crawlers have to deal with several challenges at the same time, and some of them contradict each other. They must keep fresh copies of Web pages, so they have to re-visit pages, but at the same time they must discover new pages, which are found in the modified pages. They must use the available resources such as network bandwidth to the maximum extent, but without overloading Web servers as they visit them. They must get many "good pages", but they cannot exactly know in advance which pages are the good ones.

In this chapter, we present a model that tightly integrates crawling with the rest of a search engine and gives a possible answer to the problem of how to deal with these contradictory goals, by means of adjustable parameters. We show how this model generalizes several particular cases, and propose a crawling software architecture that implements the model.

The rest of this chapter is organized as follows: Section 3.1 presents the problem of crawler scheduling, and Section 3.2 discusses the problems of a typical crawling model. Section 3.3 shows how to separate short-term and long-term scheduling, and Section 3.4 shows how to combine page freshness and quality to obtain an efficient crawling order. Section 3.5 introduces a general crawler architecture that is consistent with these observations.

Note: portions of this chapter have been presented in [CBY02, BYC02].

3.1 The problem of crawler scheduling

We consider a Web crawler that has to download a set of pages, with each page p having size S_p measured in bytes, using a network connection of capacity B, measured in bytes per second. The objective of the crawler is to download all the pages in the minimum time. A trivial solution to this problem is to download all the Web pages simultaneously, and for each page use a fraction of the bandwidth proportional to the size of each

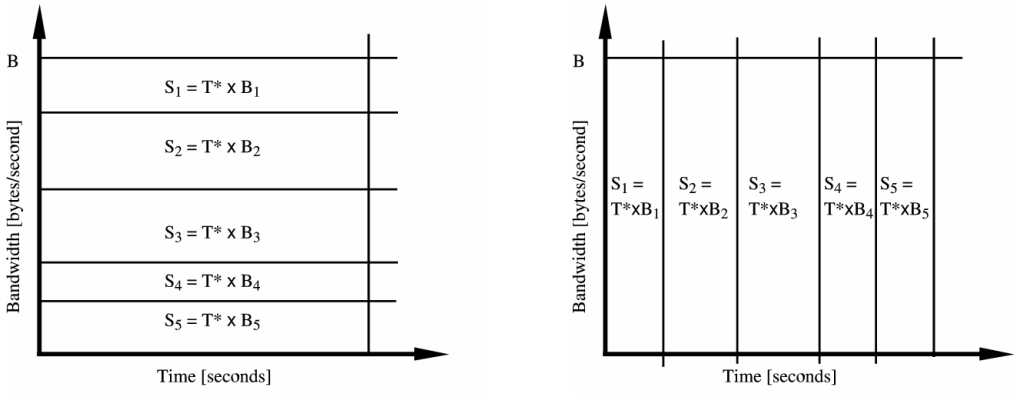
page. If B_p is the downloading speed for page p, then:

$$B_p = \frac{S_p}{T^*} \tag{3.1}$$

Where T^* is the optimal time to use all of the available bandwidth:

$$T^* = \frac{\Sigma_p S_p}{B} \tag{3.2}$$

This scenario is depicted in Figure 3.1a.



(a) Full parallelization (b) Full serialization

Figure 3.1: Two unrealistic scenarios for Web crawling: (a) parallelizing all page downloads and (b) serializing all page downloads. The areas represent page sizes, as s*ize* = s*peed* × t*ime*.

However, there are many restrictions that forbid this optimistic scenario. One restriction is that a scheduling policy must avoid overloading Web sites, enforcing a politeness policy as described in Section 2.4.3: a Web crawler should not download more than one page from a single Web site at a time, and it should wait several seconds between requests.

Instead of downloading all pages in parallel, we could also serialize all the requests, downloading only one page at a time at the maximum speed, as depicted in Figure 3.1b. However, the bandwidth available for Web sites B_i^{MAX} is usually lower than the crawler bandwidth B, so this scenario is not realistic either.

The presented observations suggest that actual download time lines are similar to the one shown in Figure 3.2. In the Figure, the optimal time T^* is not achieved, because some bandwidth is wasted due to limitations in the speed of Web sites (in the figure, B_3^{MAX}, the maximum speed for page 3 is shown), and to the fact that the crawler must wait between accesses to a Web site.

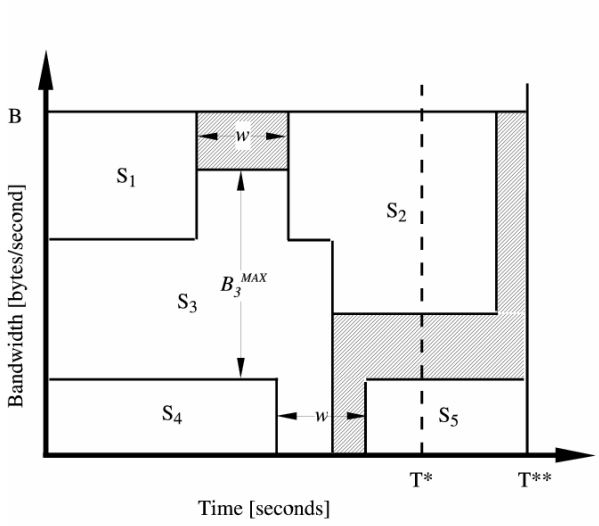


Figure 3.2: A more realistic download time line for Web crawlers. Pages $1-2$ and $4-5$ belong to the same site, and the crawler waits w seconds between them. The hatched portion is wasted bandwidth due to the constraints in the scheduling policy. The optimal time T^* is not achieved.

To overcome the problems shown in Figure 3.2, it is clear that we should try to saturate the network link, downloading pages from many different Web sites at the same time. Unfortunately, most of the pages are located in a small number of sites: the distribution of pages to sites, shown in Figure 3.3, is very bad in terms of crawler scalability. Thus, it is not possible to use productively a large number of robots and it is difficult to achieve a high utilization of the available bandwidth.

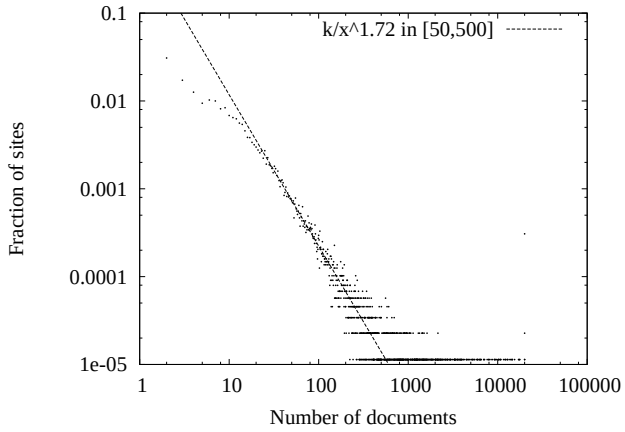


Figure 3.3: Distribution of site sizes in a sample of the Chilean Web. There are a few Web sites that are very large, and a large fraction of small Web sites. This poses a problem to Web crawlers, as they must download several pages from a small number of Web sites but at the same time they must avoid overloading them with requests.

There is another serious practical constraint: the HTTP request has latency, and the latency time can be over 25% of the total time of the request [LF98]. This latency is mainly the time it takes to establish the TCP

connection and it can be partially overcome if the same connection is used to issue several requests using the HTTP/1.1 “keep-alive” feature.

3.2 Problems of the typical crawling model

Crawling literature emphasizes on the words “crawler” and “spider”, and those words suggests walking through a directed graph. That is very far from what is really happening, because crawling is just automatic page downloading that does not need to follow a browsing-like pattern: in some cases a breadth-first approach is used, in other cases the crawling is done in a way that has not an obvious representation on the Web graph, and does not resembles a graph traversal.

The typical crawling algorithm comes from the early days of the World Wide Web, and it is given by Algorithm 3.

Algorithm 3 Typical crawling algorithm

Require: $p_1, p_2, ..., p_n$ starting URLs

1: $Q = \{p_1, p_2, ..., p_n\}$, queue of URLs to visit.
2: $V = \emptyset$, visited URLs.
3: **while** $Q \neq \emptyset$ **do**
4: Dequeue $p \in Q$, select p according to some criteria.
5: Do an asynchronous network fetch for p.
6: $V = V \cup \{p\}$
7: Parse p to extract text and extract outgoing links
8: $\Gamma^+(p) \leftarrow$ pages pointed by p
9: **for** each $p' \in \Gamma^+(p)$ **do**
10: **if** $p' \notin V \wedge p' \notin Q$ **then**
11: $Q = Q \cup \{p'\}$
12: **end if**
13: **end for**
14: **end while**

We consider that this algorithm can be improved, because during crawling it is not necessary to add the newly found URLs to Q each and every time a Web page is parsed. The new URLs can be added in groups or “batches”, because:

Indexing is done in batches. The crawling process adds information to a *collection* that will be *indexed*. The indexing process is done in batch, many megabytes of text at a time, and with current algorithms it is very inefficient to do it one document at a time, unless one can achieve an exact balance between the incoming stream of documents and the processing speed of the index [TLNJ01], and in this case,

the index construction becomes the bottleneck. Thus, in most search engines, the index is not updated continuously but completely at the same time. To the best of our knowledge, this is the case for most large search engines, and there is even a term coined for the update of Google's index ("Google dance"), when the new index is distributed to the different data centers [Sob03]. When the index is updated in batches, it is not important which URLs were transferred first.

Distributed crawlers exchange URLs in batches. If the crawler is distributed, then it has to send the results back to a central server, or it has to exchange results with other crawling processes. For better performance, it must send many URLs at a time, as the exchange of URLs generates an overhead that is mostly given by the context switches, not for the (relatively small) size of the URLs [CGM02]. This means that how the URLs are ordered *locally* should not impact the *global* crawling order.

The important URLs are seen earlier in the crawl. If some URL ordering is done and if this ordering is not based on text-similarity to a query, then in steady state a page that we have just seen is a very unlikely candidate to be downloaded in the near future: "good" pages are seen early in the crawling process [NW01]. Conversely, if a URL is seen for the first time in a late stage of the crawling process, there is a high probability that it is not a very interesting page. This is obviously true if Pagerank [PBMW98] is used, because it reflects the time a random surfer "spends" at the page and if a random surfer spends more time in a page, then probably the page can be reached from several links.

We have noticed that previous work tends to separate two similar problems and to mix two different problems:

- The two different problems that are usually mixed are the problem of short-term efficiency (maximizing the bandwidth usage and being polite with servers) and long-term efficiency (ordering the crawling process to download important pages first). We discuss why these two problems can be separated in Section 3.3.
- The two related problems that are usually treated as separate issues are the index freshness and the index intrinsic quality. We consider that it is better to think in terms of a series of scores related to different characteristics of the documents in the collection, *including freshness*, which should be weighted accordingly to some priorities that vary depending on the usage context of the crawler. This idea is further developed in Section 3.4.

3.3 Separating short-term from long-term scheduling

We intend to deal with long-term scheduling and short-term scheduling separately. To be able to do this, we must prove that both problems can be separated, namely, we must check if the intrinsic quality of a Web

page or a Web server is related to the bandwidth available to download that page. If that were the case, then we would not be able to select the most important pages first and later re-order pages to use the bandwidth effectively, because while choosing the important pages we would be affecting the network transfer speed.

We designed and ran the following experiment to validate this hypothesis. We took one thousand Chilean site names at random from the approximately 50,000 currently existing. We accessed the home page of these Web sites repeatedly each 6 hours during a 2-weeks period, and measured the connection speed (bytes/second) and latency (seconds). To get a measure of the network transfer characteristics and avoid interferences arising from variations in the connections to different servers, pages were accessed sequentially (not in parallel).

From the 1000 home pages, we were able to successfully measure 750 of them, as the others were down during a substantial fraction of the observed period, or did not answer our request with an actual Web page. In the analysis, we consider only Web sites that answered to the requests.

As a measure of the "importance" of Web sites, we used the number of in-links from different Web sites in the Chilean Web, as this is a quantitative measure of the popularity of the Web site among other Web site owners.

We measured the correlation coefficient r between the number of in-links and the speed ($r = -0.001$), and between the number of in-links and the latency ($r = -0.069$). The correlation between these parameters is not statistically significant. These results show that the differences in the network connections to "important" pages and "normal" pages are not relevant to long-term scheduling. Figure 3.4 shows a scatter plot of connection speed and number of in-links.

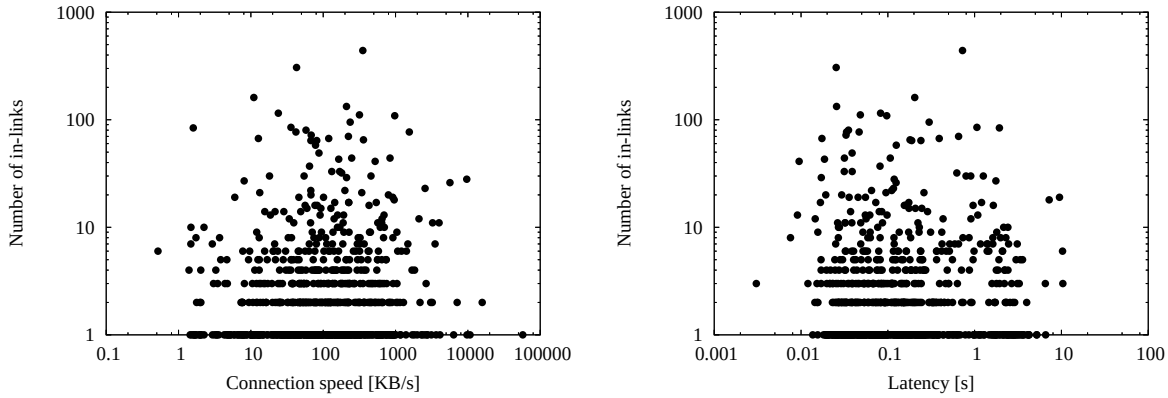


Figure 3.4: Scatter plot of connection speed (in Kilobytes per second) and latency (in seconds) versus popularity (measured as the number of in-links). In our experiments, we found no significant correlation between these variables. The results are averages of the measures obtained by connecting to 750 Web sites sequentially every 6 hours during a 2-weeks period.

For completeness, we also used the data gathered during this experiment to measure the correlation

between the connection speed and latency ($r = -0.645$), which is high, as shown in Figure 3.5. In the graph, we can see that Web sites with higher bandwidths tend to have low latency times.

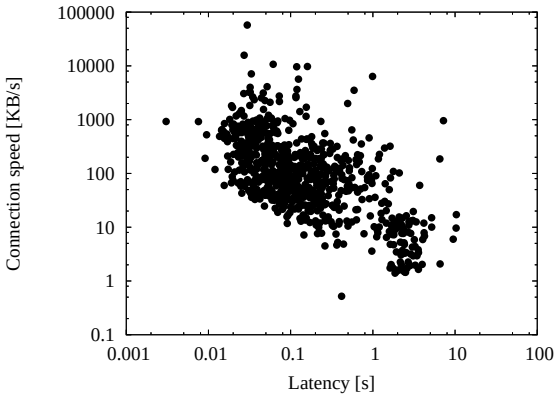


Figure 3.5: Scatter plot of connection speed versus latency. Web sites with low connection speeds tend to have also low latency.

Another interesting result that we obtained from this experiment was that connection speeds and latency times varied substantially during the observed period. We found on average a relative deviation of 42% for speed and 96% for latency, so these two quantities cannot be predicted based only on their observed mean values. The daily and weekly periodicity in Web server response time observed by Liu [Liu98] has to be considered for a more accurate prediction: Diligenti *et al.* [DMPS04] maintain several observed values for predicting connection speed, and group the observations by time of the day to account for the periodicity in Web server response time.

3.4 Combining page freshness and quality

A search engine's crawler is designed to create a collection of pages that is useful for the search engine's index. To be useful, the index should balance comprehensiveness and quality. These two goals compete, because at each scheduling step, the crawler must decide between downloading a new page, not currently indexed, or refreshing a page that is probably outdated in the index. There is a trade-off between quantity (more objects) and quality (more up-to-date objects).

In the following, we propose a function to measure the quality of the index of a search engine. The crawler's goal is to maximize this function.

We start by stating three factors that are relevant for the quality of a Web page in the index:

Intrinsic quality of the page. The index should contain a large number of Web pages that are *interesting* to the search engine's users. However, the definition of what will be interesting for users is a slippery one, and currently a subject of intense research. A number of strategies have been proposed [CGMP98,

DCL+00, NW01], usually relying in a ranking function for ordering the list of objects found by the search engine.

We cannot known in advance the interest that a Web page will have to users, but we can approximate it [CGMP98] using a ranking function that considers the partial information that the crawler has obtained so far during its process.

The intrinsic quality of a page can be estimated in many ways [CGMP98]:

- Link analysis (link popularity).
- Similarity to a given query.
- Accesses to the page on the index (usage popularity).
- Location-based, by the perceived depth (e.g. number of directories on the path to the Web object).
- Domain name, IP address or segment, geography, etc.

Representational quality of the page in the index. Every object in the index should *accurately represent* a real object in the Web. This is related to both the amount of data stored about the object (e.g.: it is not the same to index just the first 200 words than to index the full page) and to the rendering time of the object (e.g.: compression [WMB99] uses less space but may increase the rendering time).

The representational quality depends mainly on the quantity and format of the information being stored for every object. In the case of Web pages, we can order the representational quality from less to more:

- URL.
- URL + Index of text near links to that page.
- URL + Index of text near links to that page + Index of the full text
- URL + Index of text near links to that page + Index of the full text + Summary of text ("snippet") extracted using natural language processing techniques or simply by taking a few words from the beginning of the text
- URL + Index of text near links to that page + Index of the full text + Full text.

There are other possibilities, involving indexing just portions of the page using the HTML markup as a guide, i.e., indexing only titles, metadata and/or page headings.

Rendering time depends on the format, particularly if compression is used. Some adaptivity can be used, e.g.: text or images could be compressed except for those objects in which a large representational quality is required, because they are accessed frequently by the search engine.

At this moment, Google [goo04] uses only two values, either RepresentationalQuality$(p_i) = \mathit{high}$ and the Web page is stored almost completely, or RepresentationalQuality$(p_i) = \mathit{low}$ and only the URL

and the hyperlink anchor texts to that URL are analyzed. Note that in this case a page can be in the index without ever having been actually downloaded: the index for these pages is built using the URL and a few words that appeared in the context of links found towards that page. In the future, the page can be visited and its representational quality can increase, at the expense of more storage space and more network transfers.

There is no reason why this should be a binary variable. A selective index, which indexes partially certain pages and completely other pages can be a good solution for saving disk space in the future, especially if the distance between storage capacity and the amount of information available on the Web further increases.

Freshness of the page. Web content is very dynamic, and the rate of change of Web pages [DFKM97, BCS+00] is believed to be between a few months and one year, with the most popular objects having a higher rate of change than the others. We expect to maximize the probability of a page being fresh in the index, given the information we have about past changes: an estimator for this was shown in Section 2.4 (page 27).

Keeping a high freshness typically involves using more network resources to transfer the object to the search engine.

For the *value* of an object in the index, $V(p)$, a product function is proposed:

$$V(p) = \text{IntrinsicQuality}(p)^{\alpha} \times \text{RepresentationalQuality}(p)^{\beta} \times \text{Freshness}(p)^{\gamma} \tag{3.3}$$

The parameters α, β and γ are adjustable by the crawler's owner, and depend on the objective and policies of it. Other functions could be used, as long as they are increasing in the relevant quality measures, and allow to specify the relative importance between these values. We propose to use a product because the distribution of quality and rate of change are very skewed and we usually will be working with the logarithm of the ranking function for the intrinsic quality.

We propose that the *value* of an index $I = \{p_1, p_2, ..., p_n\}$ is the sum of the values of the objects p_i stored on the index:

$$V(I) = \sum_{i=1}^{n} V(p_i) \tag{3.4}$$

Depending on the application, other functions could be used to aggregate the value of individual elements into the value of the complete index, as long as they are non-decreasing on every component. For instance, a function such as $V(I) = \min_i V(p_i)$ could be advisable if the crawler is concerned with ensuring a baseline quality for all the objects in the index.

A good coverage, i.e., indexing a large fraction of the available objects, certainly increases the value of an index, but only if the variables we have cited: intrinsic quality, representational quality and freshness are considered. Coverage also depends on freshness, as new pages are usually found only on changed pages.

The proposed model covers many particular cases that differ on the relative importance of the measures described above. In Figure 3.6, different types of crawlers are classified in a taxonomy based on the proposed three factors.

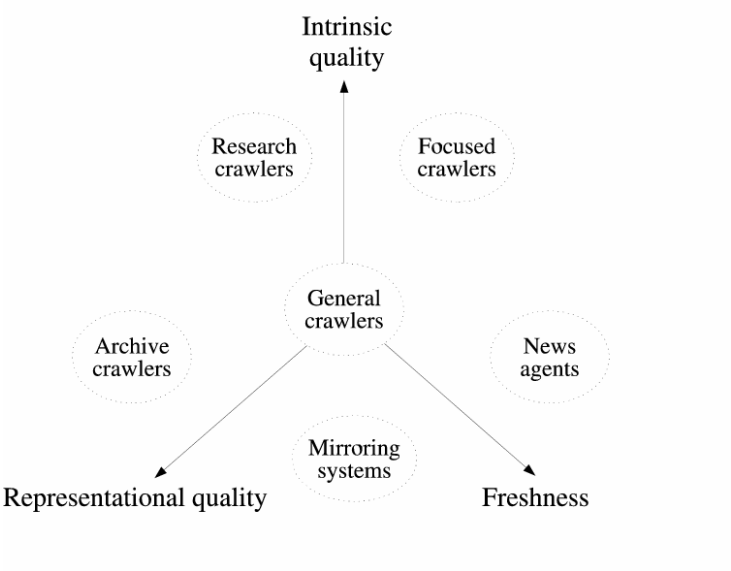


Figure 3.6: Different types of Web crawlers can be classified in our framework, based on the relative importance given to freshness, representational quality and intrinsic quality.

Research crawlers (e.g.: CiteSeer [cit04]) and focused crawlers are mostly interested in the intrinsic quality of the downloaded pages. Archive crawlers (e.g.: Internet Archive [arc04]) are mostly interested in keeping an accurate copy of the existing pages. News agents and mirroring systems are mostly interested in having fresh copies of the pages. General, large-scale crawlers are in the center of the graph, as they have to balance all the different aspects to have a good index.

3.5 A software architecture

The observations presented in the previous sections can be used to design a new crawling architecture. The objective of the design of this crawling architecture is to divide the crawling task into different tasks that will be carried efficiently by specialized modules.

A separation of tasks can be achieved with two modules, as shown in Figure 3.7. The *scheduler* calculates scores and assigns pages to several *downloader* modules that transfer pages through the network, parse their contents, extract new links and maintain the link structure.

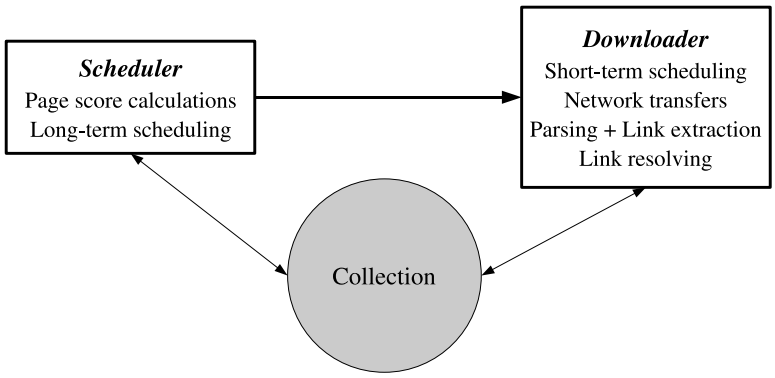


Figure 3.7: A software architecture with two modules. A "batch" of pages is generated by the *scheduler*, and downloaded and parsed by the *downloader*. Under this scheme, the scheduler requires read access to the collection, and the downloader read and write access.

There are some problems with this two-module architecture. One problem is that for the scheduler to work on the Web graph, during the calculations, the Web graph cannot change. So, the the process of modifying the Web graph should be as fast as possible, but parsing the pages can be slow and this could mean that we have to "lock" the Web graph during a long time. What can be done to overcome this is to parse all pages and accumulate links, and then add all the links found to the collection.

Another issue is that we could have different, optimized hardware architectures for the tasks of downloading and storing pages and for the task of parsing pages. Parsing pages can be expensive in terms of processing, while downloading pages requires mostly high network connectivity and fast disks. Moreover, if the network downloads must be carried with high parallelism, then each downloading task should be very lightweight. To solve these issues we divide the tasks of downloading, parsing and keeping the link structure, as shown in Figure 3.8. The following module names are used through the thesis:

Manager: page value calculations and long-term scheduling.

Harvester: short-term scheduling and network transfers.

Gatherer: parsing and link extraction.

Seeder: URL resolving and link structure.

Figure 3.9 introduces the main data structures that form the index of the search engine, and outlines the steps of the operation:

1. Efficient crawling order Long-term scheduling is done by the "manager" module, which generates the list with URLs that should be downloaded by the harvester in the next cycle (a "batch"). The objective of this module is to maximize the "profit" (i.e.: the increase in the index value) in each cycle.

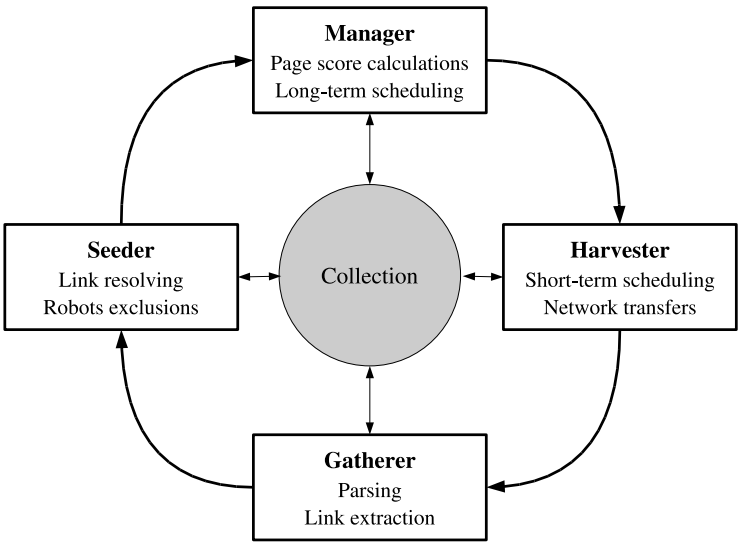


Figure 3.8: The proposed software architecture has a manager, that generates batches of URLs to be downloaded by the harvester. The pages then go to a gatherer that parses them and send the discovered URLs to a seeder.

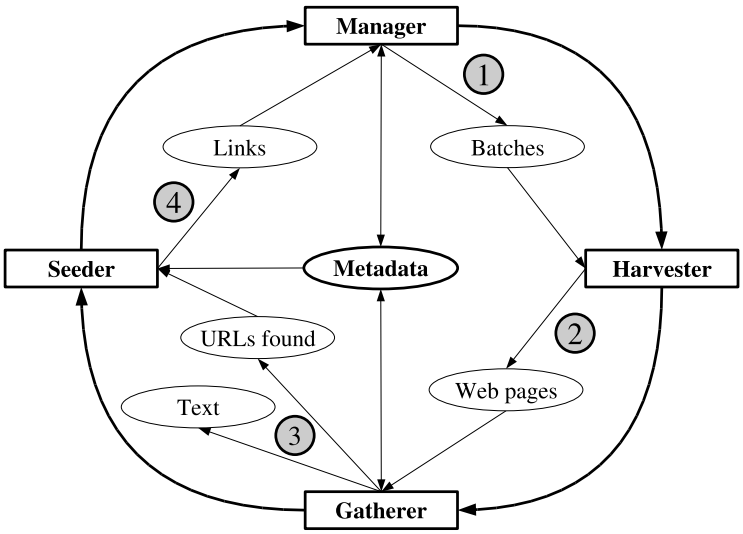


Figure 3.9: The main data structures and the operation steps of the crawler: (1) the manager generates a batch of URLs, (2) the harvester downloads the pages, (3) the gatherer parses the pages to extract text and links, (4) the seeder checks for new URLs and maintains the link structure.

2. Efficient network transfers Short-term scheduling is assigned to the "harvester" module. This module receives batches of URLs and its objective is to download the pages in the batch as fast as possible, using multiple connections and enforcing a politeness policy. The harvester generates a partial

collection, consisting mostly of raw HTML data.

3. **Efficient page parsing** The extraction of the text and links is assigned to the "gatherer" module. This module receives the partial collections downloaded by the harvester(s) and adds the text to the main collection. It also generates a list of found URLs that are passed to the seeder.

4. **Efficient URL manipulation** The URLs found are processed by a "seeder" module, which searches for new URLs that have not been seen before. This module also checks for URLs that should not be crawled because of the `robots.txt` exclusion protocol, described in Section 7.2 (page 114). The module maintains a data structure describing Web links.

The pattern of read and write accesses to the data structures is designed to improve the scalability of the crawler as, for instance, the pages can be downloaded and parsed while the Web graph is analyzed, and the analysis only must stop while the seeder is running.

The programs and data structures in Figure 3.9 are explained in detail in Chapter 7.

3.6 Conclusions

Web crawling is not only a trivial graph traversal problem. It involves several issues that arise from the distributed nature of the Web. First, Web crawlers must share resources with other agents, mostly with humans, and cannot monopolize Web sites' time –indeed, a Web crawler should try to minimize its impact on Web sites. Second, Web crawlers have to deal with an information repository which contains many objects of varying quality, including objects with very low quality created to lure the Web crawler and deceive ranking schemes.

We consider the problem of Web crawling as a process of discovering relevant objects, and one of the main problems is that a Web crawler always works with partial information, because it must infer the properties of the unknown pages based on the portion of the Web actually downloaded. In this context, the Web crawler requires access to as most information as possible about the Web pages.

While the model implies that all the portions of the search engine should know all the properties of the Web pages, the architecture introduced in this chapter is an attempt of separating these properties into smaller units (text, link graph, etc.) for better scalability. This architecture is implemented in the WIRE crawler and details on the implementation of the WIRE crawler is explained in Chapter 7.

Benchmarking this architecture requires a framework that allows comparisons in different settings of network, processor, memory and disk, and during this thesis we did not carry any benchmark of this type. However, the findings about scheduling strategies, stop criteria and Web characteristics presented in the following chapters are mostly independent of the chosen architecture.

There are other ways of dividing the tasks of a Web crawler that can use the same crawling model presented in sections 3.3 and 3.4 of this chapter; the modularization described here proves that the model can be implemented, and each task is simple enough so the entire system could be programmed and debugged during the duration of this thesis, but certainly there are other alternative architectures that could have been implemented.

In the context of today's Web, it is impossible to download all of the Web pages, furthermore, in Chapter 5 we argue that the number of Web pages is infinite, so the fraction of the Web that a crawler downloads should represent the most important pages. The next chapter studies algorithms for directing the crawler towards important pages early in the crawl.

Chapter 4

Scheduling Algorithms for Web Crawling

In the previous chapter, we described the general model of our Web crawler. In this chapter, we deal with the specific algorithms for scheduling the visits to the Web pages.

We started with a large sample of the Chilean Web that was used to build a Web graph and run a crawler simulator. Several strategies were compared using the simulator to ensure identical conditions during the experiments.

The rest of this chapter is organized as follows: Section 4.1 introduces our experimental framework and Section 4.2 the simulation parameters. Sections 4.3 and 4.4 compare different scheduling policies for long- and short-term scheduling. In Section 4.5 we test one of these policies using a real Web crawler, and the last section presents our conclusions.

Portions of this chapter were presented in [CMRBY04].

4.1 Experimental setup

We tested several scheduling policies in two different datasets corresponding to Chilean and Greek Web pages using a crawler simulator. This section describes how the dataset and how the simulator works.

4.1.1 Datasets: .cl and .gr

Dill *et al.* [DKM+02] studied several sub-sets of the Web, and found that the Web graph is self-similar in several senses and at several scales, and that this self-similarity is pervasive, as it holds for a number of different parameters. Top-level domains are useful because they represents pages sharing a common cultural context; we consider that they are more useful than large Web sites because pages in a Web site are more homogeneous. Note than a large sub-set of the whole Web (and any non-closed subset of the Web) is always biased by the strategy used to crawl it.

We worked with two datasets that correspond to pages under the `.cl` (Chile) and `.gr` (Greek) top-level domains. We downloaded pages using the WIRE crawler [BYC02] in breadth-first mode, including both static and dynamic pages. While following links, we stopped at depth 5 for dynamic pages and 15 for static pages, and we downloaded up to 25,000 pages from each Web site.

We made two complete crawls on each domain, in April and May for Chile, and in May and September for Greece. We downloaded about 3.5 million pages in the Greek Web and about 2.5 million pages in the Chilean Web. Some demographic information about the two countries is presented in Table 8.7 (page 145). Both datasets are comparable in terms of the number of Web pages, but were obtained from countries with wide differences in terms of geography, language, demographics, history, etc.

4.1.2 Crawler simulator

Using this data, we created a Web graph and ran a simulator by using different scheduling policies on this graph. This allowed us to compare different strategies under exactly the same conditions.

The simulator [1] models:

- The selected scheduling policy, including the politeness policy.
- The bandwidth saturation of the crawler Internet link.
- The distribution of the connection speed and latency from Web sites, which was obtained during the experiment described in Section 3.3 (page 43).
- The page sizes, which were obtained during the crawl used to build the Web graph.

We considered a number of scheduling strategies. Their design is based on a heap priority queue whose nodes represent sites. For each site-node we have another heap with the pages of the Web site, as depicted in Figure 4.1.

At each simulation step, the scheduler chooses the top Website from the queue of Web sites and a number of pages from the top of the corresponding queue of Web pages. This information is sent to a module that simulates downloading pages from that Website.

4.2 Simulation parameters

The parameters for our different scheduling policies are the following:

[1] The crawler simulator used for this experiment was implemented by Dr. Mauricio Marin and Dr. Andrea Rodriguez, and designed by them and the author of this thesis based on the design of the WIRE crawler. Details about the crawler simulator are not given here, as they are not part of the work of this thesis.

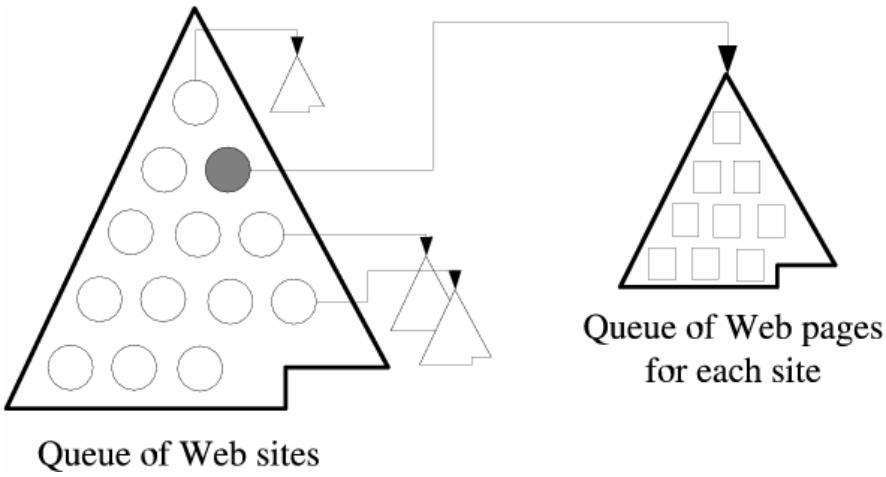


Figure 4.1: Queues used in the crawling simulator. We tested the scheduling policies using a structure with two levels: one queue for Web sites and one queue for the Web pages of each Web site.

- The policy for ordering the queue of Web sites, related to long-term scheduling.
- The policy for ordering the queues of Web pages, related to short-term scheduling.
- The interval w in seconds between requests to a single Web site.
- The number of pages c downloaded for each connection when re-using connections with the HTTP `Keep-alive` feature.
- The number r of maximum simultaneous connections, i.e.: the degree of parallelization. Although we used a large degree of parallelization, we restricted the robots to never open more than one connection to a Web site at a given time.

4.2.1 Interval between connections (w)

As noted in Section 2.4.3, a waiting time of $w = 60$ seconds is too large, as it would take too long to crawl large Web sites. Instead, we use $w = 15$ seconds in our experiments.

Liu *et al.* [Liu98] show that total time of a page download is almost always under 10 seconds. We ran our own experiments and measured that for sequential transfers (which are usually faster than parallel transfers) 90% of the pages were transfered in less than 1.5 seconds, and 95% of the pages in less than 3 seconds, as shown in Figure 4.2.

From the total time, latency is usually larger than the actual transfer time. This makes the situation even more difficult than what was shown in Figure 3.2, as the time spent waiting cannot be amortized effectively.

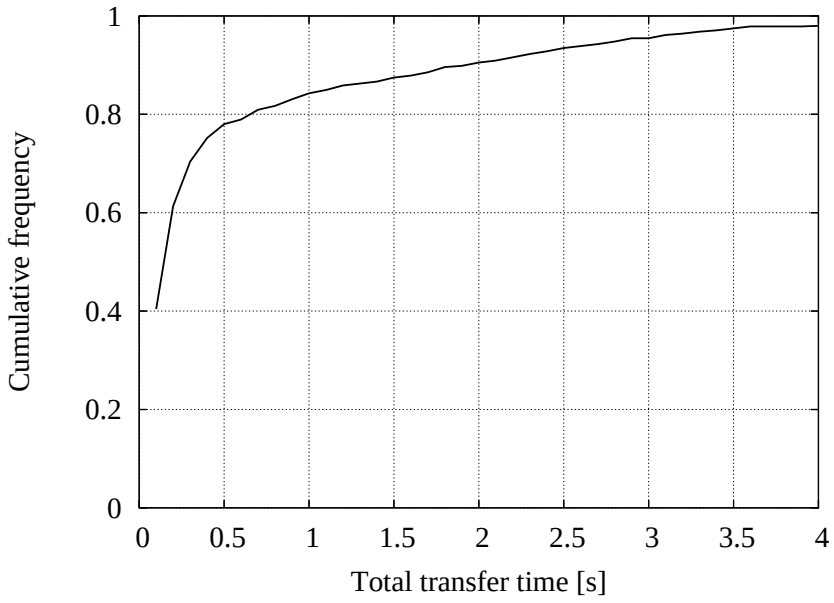


Figure 4.2: Total download time for sequential transfer of Web pages; this data provides from experiments in the Chilean Web and was used as an input for the Web crawler simulator.

4.2.2 Number of pages per connection (c)

We have observed the log files of several Web servers during this thesis. We have found that all the Web crawlers used by major search engines download only one page per each connection, and do not re-use the HTTP connection. We considered downloading multiple pages in the same connection to reduce latency, and measured the impact of this technique in the quality of the scheduling.

The protocol for keeping the connection open was introduced as the `Keep-alive` feature in HTTP/1.1 [FGM+99]; the configuration of the Apache Web server enables this feature by default and allows for a maximum of 100 objects downloaded per connection, with a timeout of 15 seconds between requests, so when using $c > 1$ in practice, we should also set $w \leq 15$ to prevent the server from closing the connection.

4.2.3 Number of simultaneous requests (r)

All of the robots currently used by Web search engines have a high degree of parallelization, downloading hundreds or thousands of pages at a given time. We used $r = 1$ (serialization of the requests), as a base case, $r = 64$ and $r = 256$ during the simulations, and $r = 1000$ during the actual crawl.

As we never open more than one connection to a given Web site, r is bounded by the number of Web sites available for the crawler, i.e.: the number of Web sites that have unvisited pages. If this number is too small, we cannot make use of a high degree of parallelization and the crawler performance in terms of pages per second drops dramatically.

The scarcity of large Web sites to download from is especially critical at the end of a large crawl, when we have already downloaded all the public pages from most of the Web sites. When downloading pages in batches, this problem can also arise by the end of a batch, so the pages should be carefully selected to include pages from as many Web sites as possible. This should be a primary concern when parallelism is considered.

4.3 Long-term scheduling

We tested different strategies for crawling pages in the stored Web graph. The complete crawl on the real Chilean or Greek Web takes about 8 days, so for testing many strategies it is much more efficient to use the crawler simulator. The simulator also help us by reproducing the exact scenario each time a strategy is tested.

Actual retrieval time for Web pages is simulated by considering the observed latency and transfer rate distribution, the observed page size for every downloaded page, and the saturation of bandwidth, which is related to the speed and number of active connections at a given time of the simulation.

For evaluating the different strategies, we calculated beforehand the Pagerank value of every page in the whole Web sample and used those values to calculate the cumulative sum of Pagerank as the simulated crawl goes by. We call this measure an "oracle" score since in practice it is not known until the complete crawl is finished. The strategies that are able to reach values close to the target total value faster are considered the most efficient ones.

There are other possible evaluation strategies for a Web crawler, but any strategy must consider some form of global ranking of the Web pages, to measure how fast ranking is accumulated. This global ranking could be:

- Number of page views, but this is hard to obtain in practice.
- Number of clicks on a search engine, but a search engine's result set represents only a small portion of the total Web pages.
- A combination of link and text analysis, but there are no established measures of quality that accomplish this without a specific query, and we want to assert overall quality, not quality for a specific topic.
- User votes or ranking.

However, we decided to use Pagerank as the global measure of quality because it can be calculated automatically and it has a non-zero value for each page.

We consider three types of strategies regarding how much information they can use: no extra information, historical information, and all the information. A random ordering can be considered a baseline for comparison. In that case, the Pagerank grows linearly with the number of pages crawled.

All of the strategies are bound to the following restrictions: $w = 15$ waiting time, $c = 1$ pages per connection, r simultaneous connections to different Web sites, and no more than one connection to each Web site at a time. In the first set of experiments we assume a situation of high-bandwidth for the Internet link, i.e., the bandwidth of the Web crawler B is larger than any of the maximum bandwidths of the Web servers B_i^{MAX}.

4.3.1 Strategies with no extra information

These strategies only use the information gathered during the current crawling process.

Breadth-first Under this strategy, the crawler visits the pages in breadth-first ordering. It starts by visiting all the home pages of all the "seed" Web sites, and Web page heaps are kept in such a way that new pages added go at the end. This is the same strategy tested by Najork and Wiener [NW01], which in their experiments showed to capture high-quality pages first.

Backlink-count This strategy crawls first the pages with the highest number of links pointing to it, so the next page to be crawled is the most linked from the pages already downloaded. This strategy was described by Cho *et al.* [CGMP98].

Batch-pagerank This strategy calculates an estimation of Pagerank, using the pages seen so far, every K pages downloaded. The next K pages to download are the pages with the highest estimated Pagerank. We used $K = 100,000$ pages, which in our case gives about 30 to 40 Pagerank calculations during the crawl. This strategy was also studied by Cho *et al.* [CGMP98], and it was found to be better than backlink-count. However, Boldi *et al.* [BSV04] showed that the approximations of Pagerank using partial graphs can be very inexact.

Partial-pagerank This is like *batch-pagerank*, but in between Pagerank re-calculations, a temporary pagerank is assigned to new pages using the sum of the Pagerank of the pages pointing to it divided by the number of out-links of those pages.

OPIC This strategy is based on OPIC [APC03], which can be seen as a weighted backlink-count strategy. All pages start with the same amount of "cash". Every time a page is crawled, its "cash" is split among the pages it links to. The priority of an uncrawled page is the sum of the "cash" it has received from the pages pointing to it. This strategy is similar to Pagerank, but has no random links and the calculation is not iterative – so it is much faster.

Larger-sites-first The goal of this strategy is to avoid having too many pending pages in any Web site, to avoid having at the end only a small number of large Web sites that may lead to spare time due to the "do not overload" rule. The crawler uses the number of un-crawled pages found so far as the priority for picking a Web site, and starts with the sites with the larger number of pending pages. This strategy was introduced in [CMRBY04] and was found to be better than breadth-first.

4.3.2 Strategies with historical information

These strategies use the Pagerank of a previous crawl as an estimation of the Pagerank in this crawl, and start in the pages with a high Pagerank in the last crawl. This is only an approximation because Pagerank can change: Cho and Adams [CA04] report that the average relative error for estimating the Pagerank four months ahead is about 78%. Also, a study by Ntoulas *et. al* [NCO04] reports that "the link structure of the Web is significantly more dynamic than the contents on the Web. Every week, about 25% new links are created". We explore a number of strategies to deal with the pages found in the current crawl which were not found in the previous one:

Historical-pagerank-omniscient New pages are assigned a Pagerank taken from an oracle that knows the full graph.

Historical-pagerank-random New pages are assigned a Pagerank value selected uniformly at random among the values obtained in previous crawl.

Historical-pagerank-zero New pages are assigned Pagerank zero, i.e., old pages are crawled first, then new pages are crawled.

Historical-pagerank-parent New pages are assigned the Pagerank of the parent page (the page in which the link was found) divided by the number of out-links of the parent page.

4.3.3 Strategy with all the information

Omniscient: this strategy can query an "oracle" which knows the complete Web graph and has calculated the actual Pagerank of each page. Every time the *omniscient* strategy needs to prioritize a download, it asks the oracle and downloads the page with the highest ranking in its frontier. Note that this strategy is bound to the same restrictions as the others, and can only download a page if it has already downloaded a page that points to it.

4.3.4 Evaluation

Our importance metric is Pagerank. Thus, for evaluating different strategies, we calculated the Pagerank value of every page in each Web graph and used those values to calculate the evolution of the Pagerank as the simulated crawl goes by.

We used three measures of performance: cumulative Pagerank, average Pagerank and Kendall's τ.

Cumulative Pagerank: we plotted the sum of the Pagerank of downloaded pages at different points of the crawling process. The strategies which are able to reach values close to the target total value 1.0 faster are considered the most efficient ones. A strategy which selects random pages to crawl will produce a diagonal line in this graph.

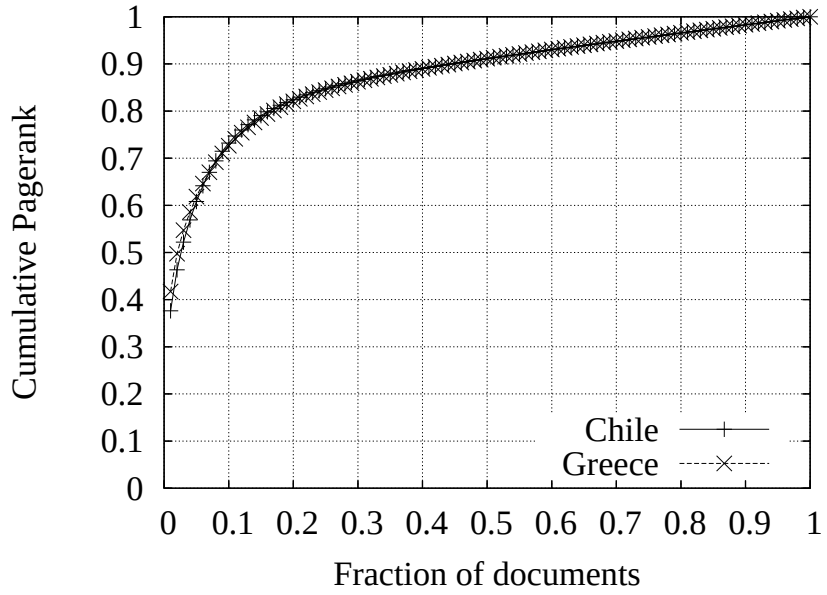


Figure 4.3: Cumulative Pagerank in the `.CL` and `.GR` domain, showing almost exactly the same distribution; these curves represents an upper bound on the cumulative Pagerank of any crawling strategy.

There is an upper bound on how well this can be done, and it is given by the distribution of Pagerank, which is shown in Figure 4.3.

The results for the different strategies are shown in Figures 4.4 and 4.5; in this simulation we are using $r = 1$, one robot at a time, because we are not interested in the time for downloading the full Web, but just in the crawling order.

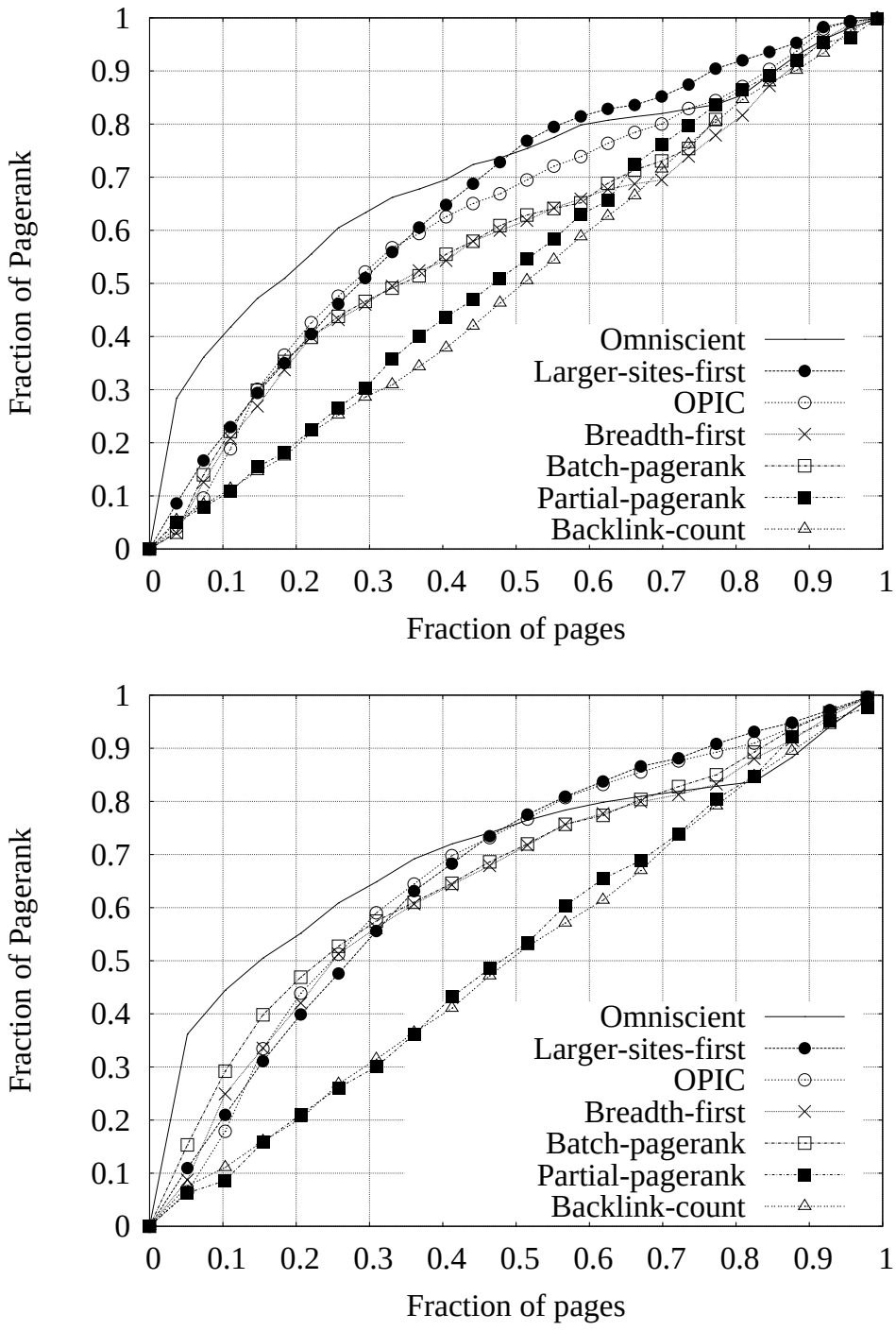


Figure 4.4: Comparison of cumulative Pagerank vs retrieved pages with the different strategies, excluding the historical strategies, in the Chilean sample during April and May 2005.

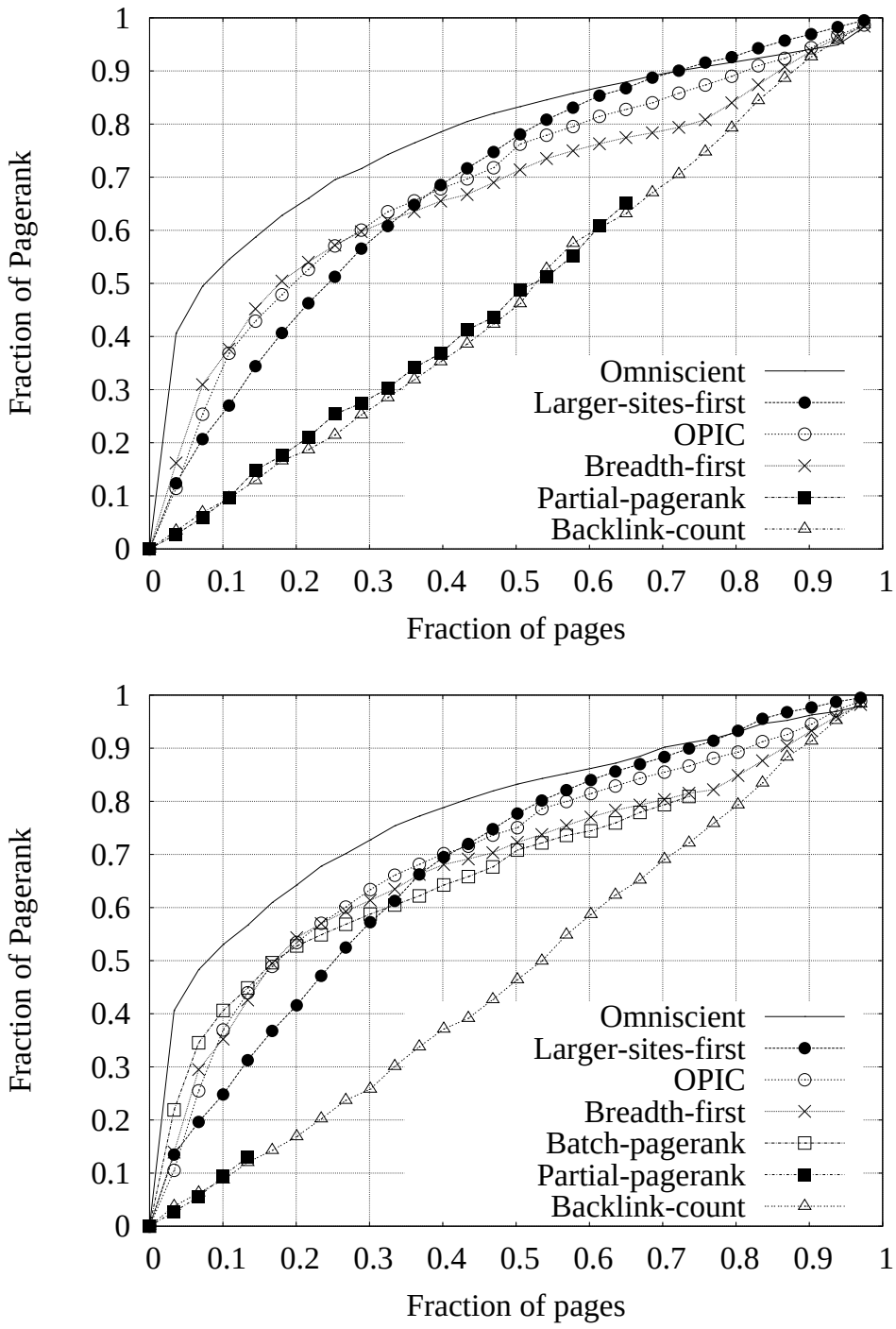


Figure 4.5: Comparison of cumulative Pagerank vs retrieved pages with the different strategies, excluding the historical strategies, in the Greek sample during May and September 2005.

Obviously the *omniscient* has the best performance, but it is in some sense too greedy because by the last stages of the crawl it performs close to random.

On the other end, *backlink-count* and *partial-pagerank* are the worst strategy according to cumulative Pagerank, and perform worser than a random crawl. They both tend to get stuck in pages that are locally optimal, and fail to discover other pages.

Breadth-first is close to the best strategies for the first 20-30% of pages, but after that it becomes less efficient.

The strategies *batch-pagerank*, *larger-sites-first* and *OPIC* have a better performance than the other strategies, with an advantage towards *larger-sites-first* when the desired coverage is high. These strategies can retrieve about half of the Pagerank value of their domains downloading only around 20-30% of the pages.

We tested the *historical-pagerank* strategies in the Greek Web graph of September, using the Pagerank calculated in May for guiding the crawl – we are using Pagerank that is 4 months old. We were able to use the Pagerank of the old crawl (May) for only 55% of the pages, as the other 45% of pages were new pages, or were not crawled in May.

Figure 4.6 shows results for a number of ways of dealing with the above 45% of pages along with results for the same Web graph but using the *OPIC* strategy for comparison. These results show that May Pagerank values are not detrimental to the crawl of September.

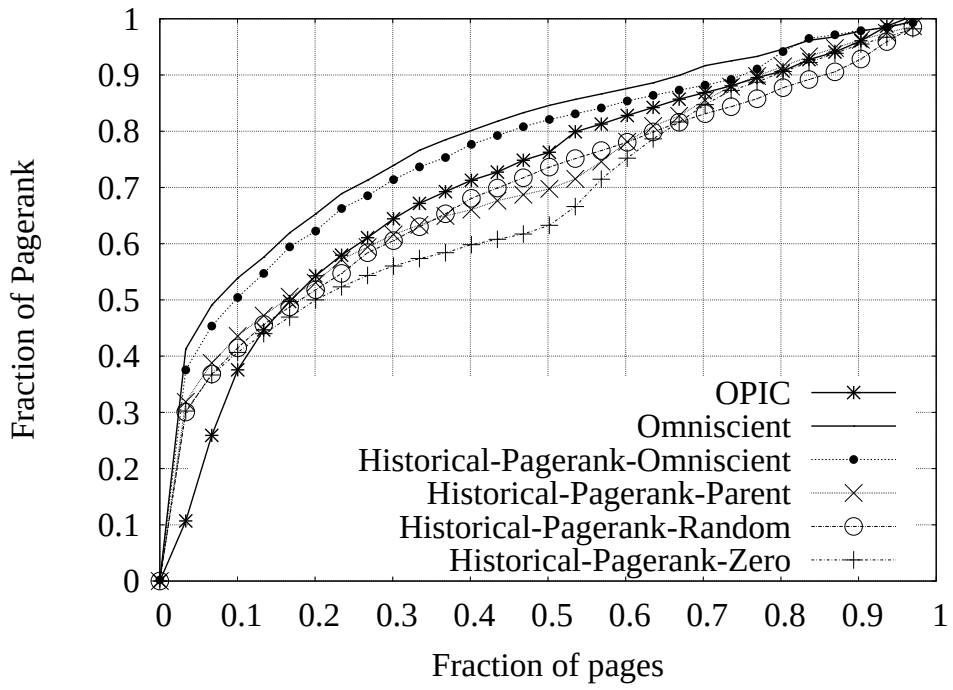


Figure 4.6: Comparison of cumulative Pagerank using the historical strategies against the *omniscient* and *OPIC* strategies, for a crawl of the Greek Web in September 2004, using Pagerank information from May 2004.

The *historical-pagerank-random* strategy has a good performance, despite of the fact that the Web graph is very dynamic [NCO04], and than on average it is difficult to estimate the Pagerank using historical information [CA04]. A possible explanation is that the ordering of pages by Pagerank changes more slowly and in particular the pages with high ranking have a more stable position in the ranking than the pages with low ranking, which exhibit a larger variability. Also, as Pagerank is biased towards old pages [BYSJC02], 55% of pages that already existed in May account for 72% of the total Pagerank in September.

Average Cumulative Pagerank: this is the average across the entire crawl. As we have normalized the cumulative Pagerank as a fraction of documents, it is equivalent to the area under the curves shown in Figures 4.4 and 4.5. The result is presented in Table 4.1, in which we have averaged the strategies across the four collections (note that the *historical-pagerank* strategies were tested in a single pair of collections, so they values are not averaged).

Kendall's Tau: this is a metric for the correlation between two ranked lists, which basically measures the number of pairwise inversions in the two lists [Ken70]. Two identical lists have $\tau = 1$, while two totally uncorrelated lists have $\tau = 0$ and reversed lists have $\tau = -1$. We calculated this coefficient for a 5000-page sample of the page ordering in each strategy, against a list of the same pages ordered by Pagerank. The results are shown in Table 4.1.

Table 4.1: Comparison of the scheduling strategies, considering average cumulative Pagerank during the crawl and Kendall's τ of the page ordering against the optimal ordering.

Strategy	Avg. Pagerank	τ
Backlink-count	0.4952	0.0157
Partial-pagerank	0.5221	0.0236
Breadth-first	0.6425	0.1293
Batch-pagerank	0.6341	0.1961
OPIC	0.6709	0.2229
Larger-sites-first	0.6749	0.2498
Historical-pagerank-zero	0.6758	0.3573
Historical-pagerank-random	0.6977	0.3689
Historical-pagerank-parent	0.7074	0.3520
Historical-pagerank-omni.	0.7731	0.6385
Omniscient	0.7427	0.6504

We attempted to measure precision, for instance, how many page downloads are necessary to get the top 10% of pages. However, this kind of measure is very sensitive to small variations, such as having a single high-quality page downloaded by the end of the crawl.

4.3.5 Multiple robots

The effect of increasing the number of robots to $r = 64$ and $r = 256$ is shown in Figure 4.7. Observing the rate of growth of the cumulative Pagerank sum, the results show that *larger-sites-first* is not affected by the number of robots; but *breadth-first* improves as the number of robots increases, because the crawler gathers information from many sources at the same time, and thus can find pages at a lower depth earlier.

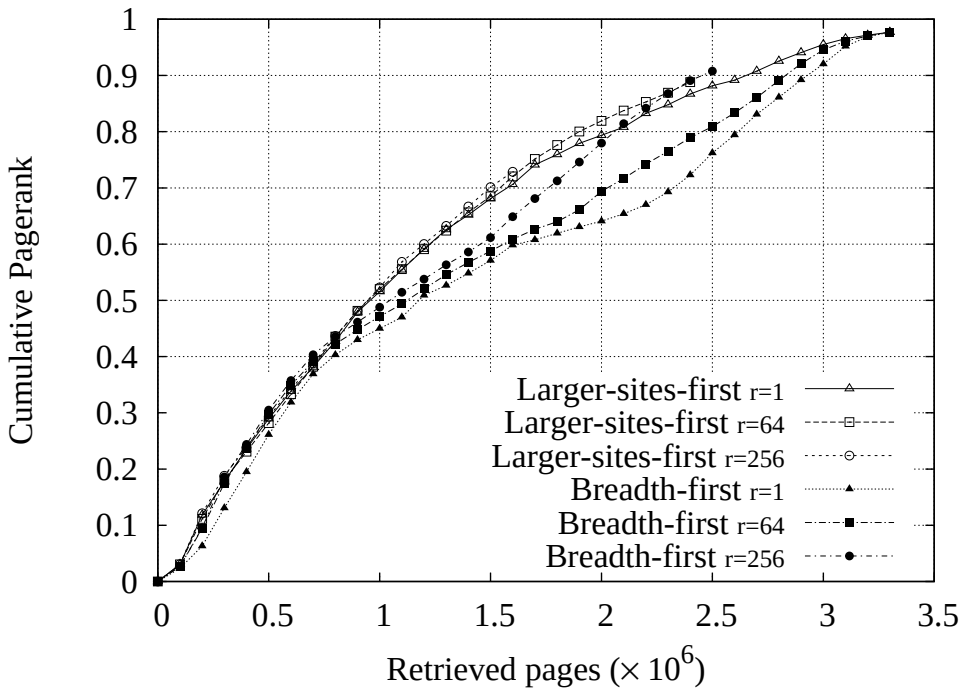


Figure 4.7: Cumulative sum of Pagerank values vs number of retrieved Web pages. Strategies *larger-sites-first* and *breadth-first*, case for $r = 1$, $r = 64$ and $r = 256$ robots.

Finally, Table 4.2 shows the effects in retrieval time when we increase the number of robots for different bandwidths, using the *larger-sites-first* strategy.

The results show that using more robots increases the rate of download of pages up to a certain point, and when bandwidth is saturated, it is pointless to use more robots (see Figure 4.8). Note that this result arises from a simulation that does not consider CPU processing time, and adding more robots increases the performance monotonically.

In a multi-threaded crawler, using more robots than necessary actually decreases the performance due to the load from context switches. This is not the case of the WIRE crawler, which is single threaded and uses an array of sockets, as explained in Section 7.2.2: there are no context switches, and handling even a large amount of sockets is not very demanding in terms of processing power. Also, idle sockets do not require processing.

Table 4.2: Predicted speed-ups for parallelism in the crawling process, using simulation and the *Larger-sites-first* strategy.

Bandwidth [bytes/second]	$r = 1$	$r = 64$	$r = 256$
200	0.2	1.6	3.0
2,000	0.7	3.8	16.0
20,000	1.0	27.0	83.3
200,000	1.0	43.0	114.1
2,000,000	1.0	54.3	204.3
20,000,000	1.0	54.6	220.1

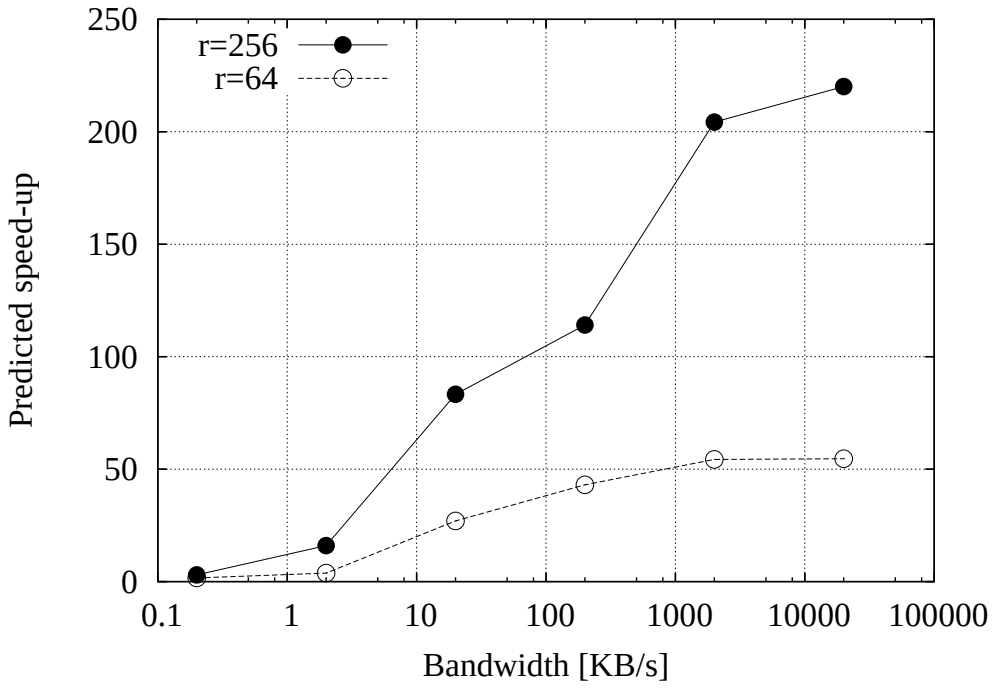


Figure 4.8: Predicted speed-up at different bandwidths, showing sub-linear growth and saturation. Note that the scale for the bandwidth is logarithmic.

4.4 Short-term scheduling

When crawling, especially in distributed crawling architectures, it is typical to work by downloading groups of pages, or to make periodic stops for saving a checkpoint with the current status of the crawler. These groups of pages or "batches" are fixed-size groups of K pages, chosen according to the long-term scheduling policy.

We have shown the distribution of pages on sites for the whole Web in Figure 3.3; on Figure 4.9 we show page distribution on sites for a typical batch, obtained at the middle of the crawl. The distribution is slightly less skewed than for the entire Web, as Web sites with very few pages are completed early in the crawl, but it is nevertheless very skewed.

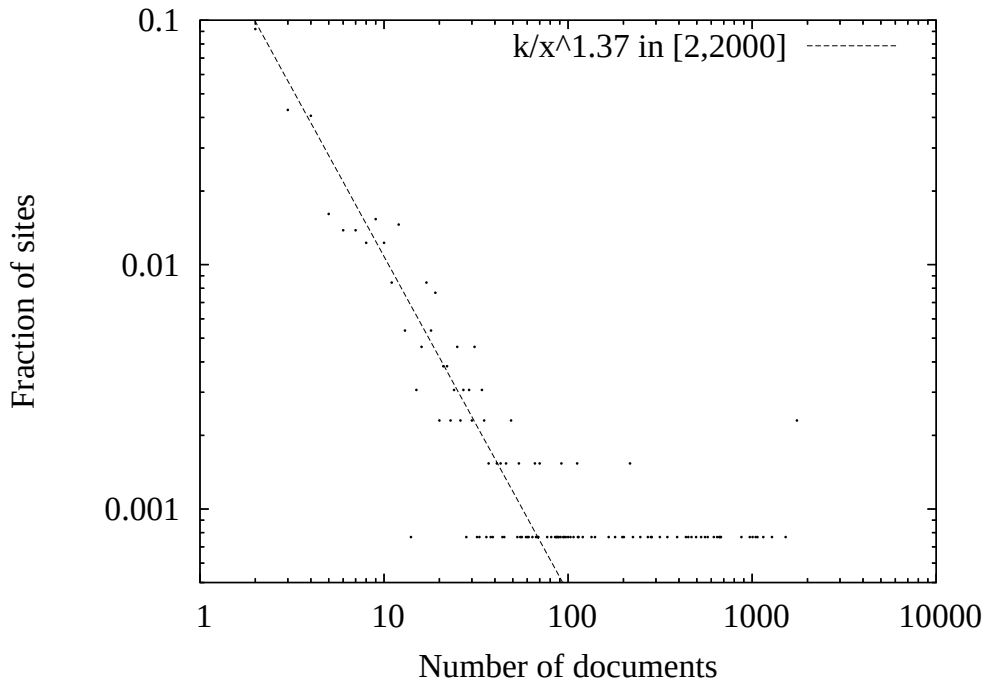


Figure 4.9: Distribution of Web pages to Web sites in a typical batch in the middle of the crawl using breadth-first crawling.

Even when a batch involves many Web sites, if a large fraction of those Web sites has very few pages available for the crawler, then quickly many of the robots will be idle, as two robots cannot visit the same Web site at the same time. Figure 4.10 shows how the effective number of robots involved in the retrieval of a batch drops dramatically as the crawl goes by. In this figure, the number of robots actually downloading pages varies during the crawl, as a robots must wait for w seconds before downloading the next page from a site, and if there are no other sites available, then that robot becomes inactive.

An approach to overcome this problem is to try to reduce waiting time. This can be done by increasing c and letting robots get more than one page every time they connect to a Web server. In figure 4.11 we show results for a case in which robots can download up to $c = 100$ pages per site in a single connection, using the HTTP/1.1 `Keep-alive` feature.

Downloading several pages per connection resulted in significant savings in terms of the total time needed for downloading the pages, as more robots are kept active for a longer part of the crawl. In the case of the small bandwidth scenario, the time to download a batch was reduced from about 33 to 29 hours, and in the case of a large bandwidth scenario, the time was reduced from 9 hours to 3 hours.

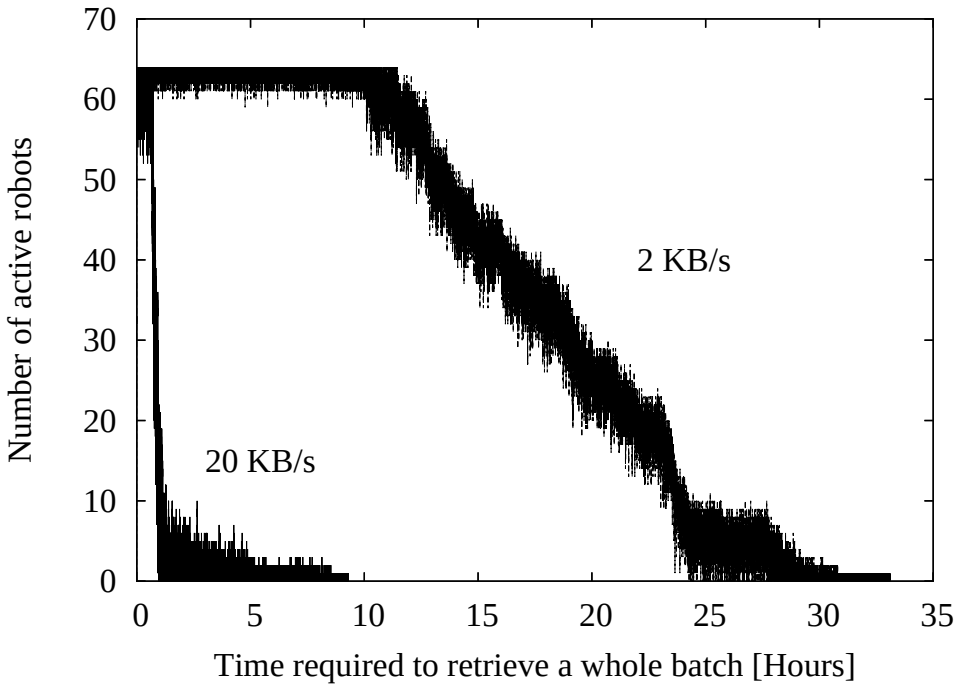


Figure 4.10: Number of active robots vs batch's total retrieval time. The two curves are for small (2 Kb/s) and large (20 Kb/s) bandwidth. In either case, most robots are idle most of the time, and the number of active robots varies as robots get activated and deactivated very often during the crawl.

Note that, as most of the latency of a download is related to the connection time, downloading multiple small pages with the same connection is very similar to downloading just a large Web page, therefore, *increasing the number of pages that are downloaded in the same connection is equivalent to reducing w, the waiting time between pages*. Reducing *w* in practice can be very difficult, because it can be perceived as a threat by Web site administrators, but increasing the number of pages downloaded by connection can be a situation in which both search engines and Web sites win.

Another heuristic that can be used in practice is monitoring the number of threads used while downloading pages, and stop the current crawl cycle if this number is too low. Pages that were not crawled are downloaded in the next batch. This also suggests preparing the next batch of pages in advance, and start the next batch before the current batch ends on when network usage drops below a certain threshold.

4.5 Downloading the real Web

In this section we describe two experiments for testing the *larger-sites-first* scheduling policy in a real Web crawler.

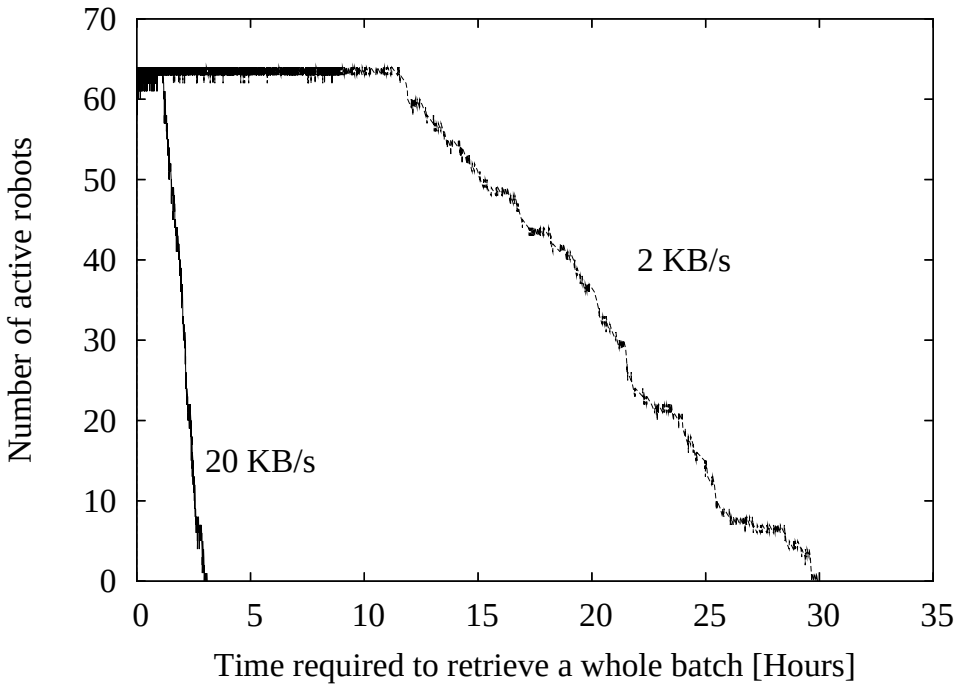


Figure 4.11: Number of active robots vs batch's total retrieval time. The two curves are for small (2 Kb/s) and large (20 Kb/s) bandwidth. In this case robots are allowed to request up to 100 pages with the same connection, that is the default maximum for the Apache Web server. In this case there is much less variability in the number of active robots.

4.5.1 Experiment 1

We started with a list of Web sites registered with the Chilean Network Information Center [nic04], and ran the crawler during 8 days with the *larger-sites-first* strategy. We visited 3 million pages in over 50,000 Web sites, downloading 57 GB of data.

We ran the crawler in batches of up to $K = 100,000$ pages, using up to $r = 1000$ simultaneous network connections, and we waited at least $w = 15$ seconds between accesses to the same Web site. The crawler used both the `robots.txt` file and meta-tags in Web pages according to the robot exclusion protocol [Kos95]. We did not use `Keep-alive` for this crawl, so $c = 1$.

We calculated the Pagerank of all the pages in the collection when the crawling was completed, and then measured how much of the total Pagerank was covered during each day. The results are shown in Figure 4.12.

We can see that by the end of the second day, 50% of the pages were downloaded, and about 80% of the total Pagerank was achieved; according to the probabilistic interpretation of Pagerank, this means we have downloaded pages in which a random surfer limited to this collection would spend 80% of its time. By the end of day four, 80% of the pages were downloaded, and more than 95% of the Pagerank, so in general this

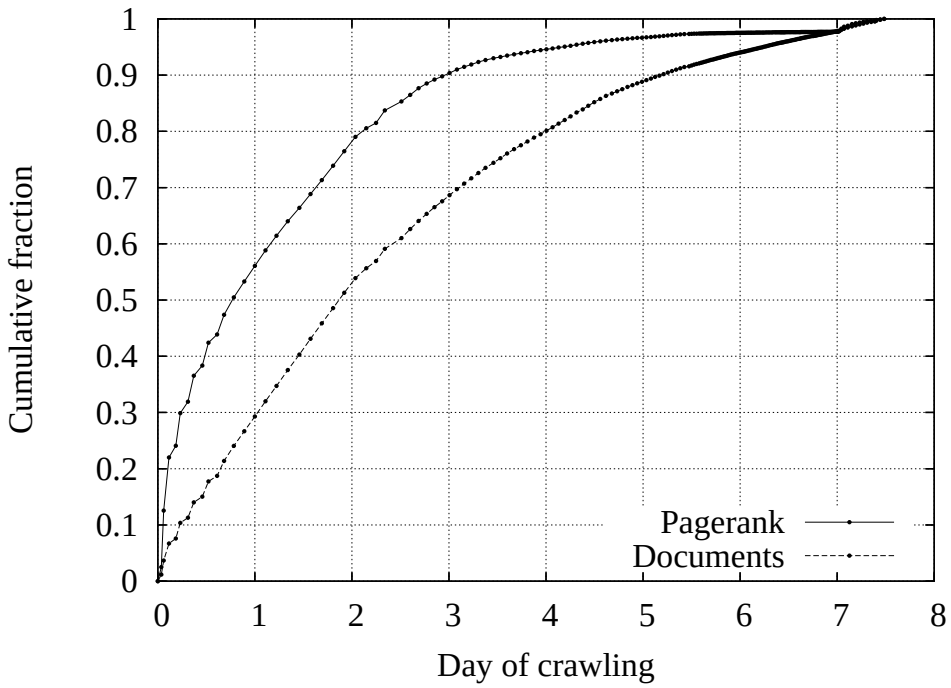


Figure 4.12: Cumulative sum of Pagerank values vs day of crawl, on an actual crawler using the *larger-sites-first* strategy. The fraction of retrieved Pagerank is larger than the fraction of retrieved documents during the entire crawl.

approach leads to "good" pages early in the crawl. In fact, the average Pagerank decreased dramatically after a few days, as shown in Figure 4.13, and this is consistent with the findings of Najork and Wiener [NW01].

It is reasonable to suspect that pages with good Pagerank are found early just because they are mostly home pages or are located at very low depths within Web sites. There is, indeed, an inverse relation between Pagerank and depth in the first few levels, but 3-4 clicks away from the home page the correlation is very low, as can be seen in Figure 4.14. There are many home pages with very low Pagerank as many of them have very few or no in-links: we were able to found those pages only by their registration under the `.cl` top-level domain database.

Regarding the relationship between the expected values of the simulation and the observed values, we plotted the cumulative Pagerank versus the number of pages downloaded, and obtained Figure 4.15. The results are consistent, and the actual crawl performed slightly better than the simulated crawl.

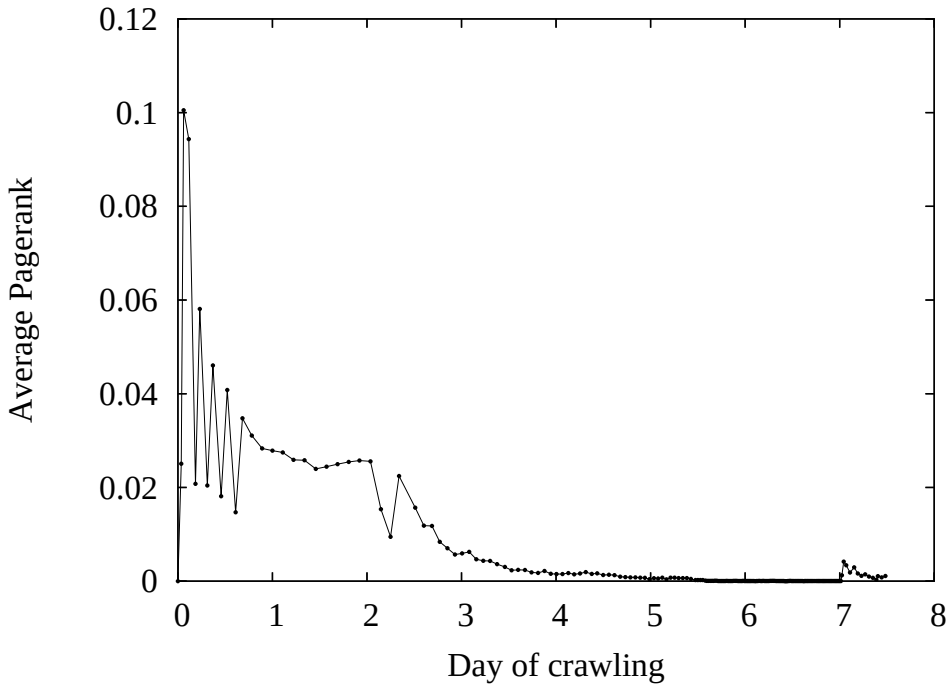


Figure 4.13: Average Pagerank per day of crawl using the *larger-sites-first* strategy.

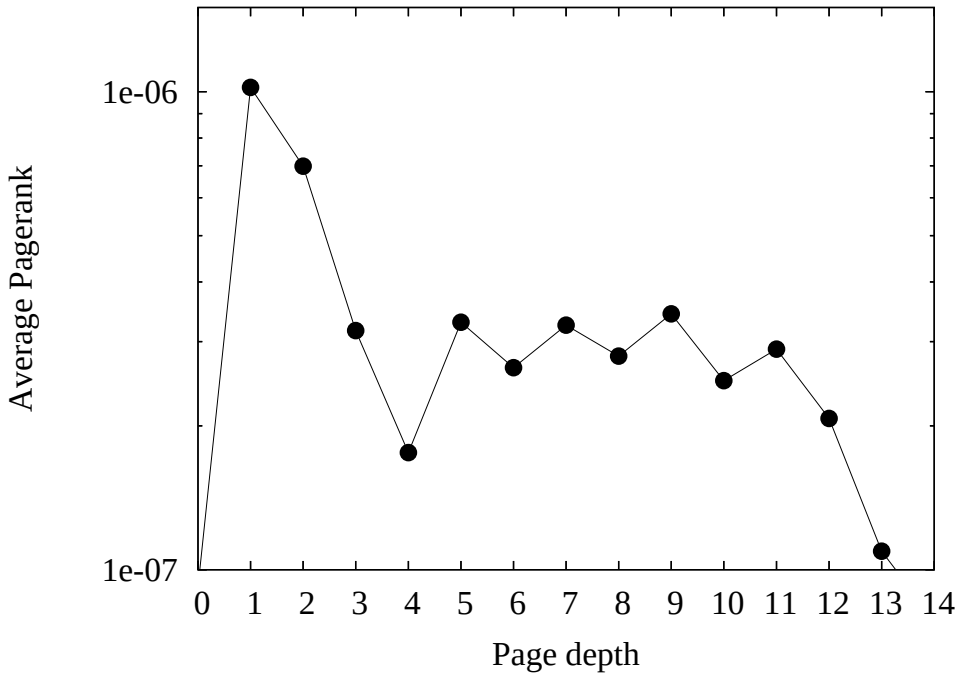


Figure 4.14: Average Pagerank versus page depth, showing that there is a correlation only in the first few levels.

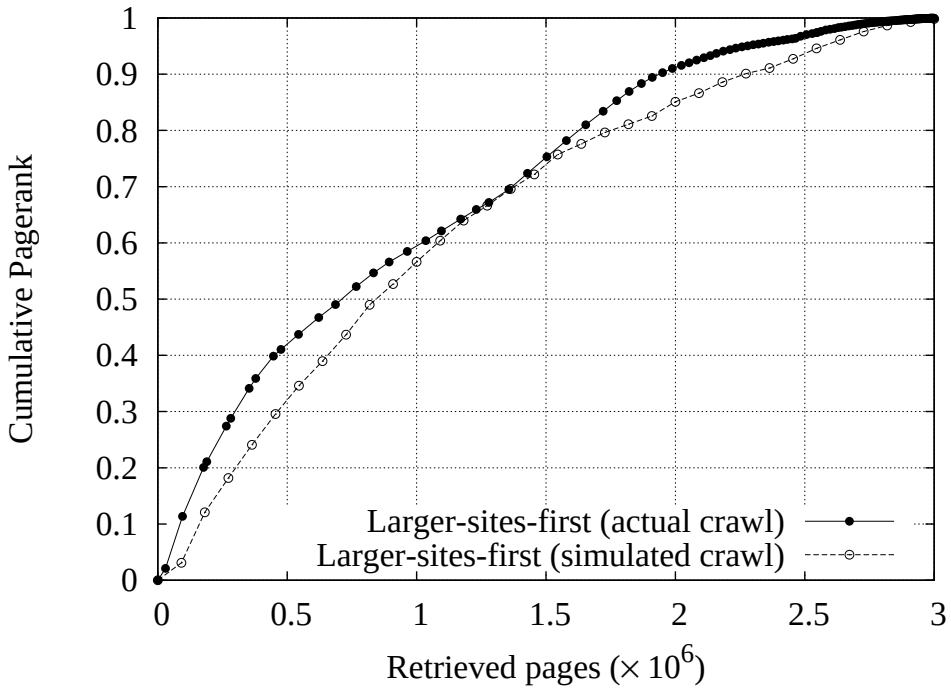


Figure 4.15: Cumulative sum of Pagerank values vs number of retrieved Web pages, on both actual and simulated Web crawls using *larger-sites-first* strategy.

4.5.2 Experiment 2

We also performed two actual crawls using WIRE [BYC02] in two consecutive weeks in the `.GR` domain, using *Breadth-first* and *Larger-sites-first*. We ran the crawler in a single Intel PC of 3.06GHz with 1Gb of RAM under Linux, in batches of up to 200,000 pages, using up to $r = 1000$ simultaneous network connections, with $w = 5$ seconds between accesses to the same Web site, and $w = 15$ for sites with less than 100 pages.

For this experiment, we focused in the time variable, as it is worthless to download pages in the right order if they cannot be downloaded fast. We calculated the Pagerank of all the pages in the collection when the crawling was completed and then measured how much of the total Pagerank was covered during each batch. The results are shown in Figure 4.16.

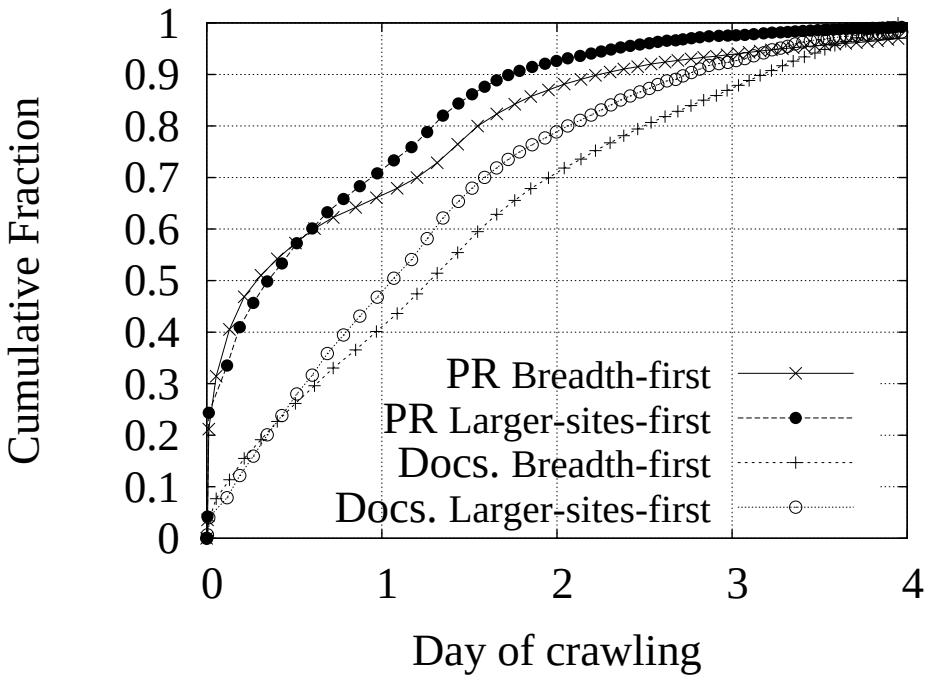


Figure 4.16: Cumulative Pagerank (PR) and cumulative fraction of documents (Docs.) of an actual crawl of the *.GR* domain using two strategies: *Breadth-first* and *Larger-sites-first*.

Both crawling strategies are efficient in terms of downloading the valuable pages early, but *larger-sites-first* is faster in both downloading documents and downloading good pages. This strategy "saves" several small Web sites for the middle and end part of the crawl and interleaves those Web sites with larger Web sites to continue downloading important pages at a fast pace.

4.6 Conclusions

Most of the strategies tested were able to download important pages first. As shown in [BSV04], even a random strategy can perform well on the Web, in terms that a random walk on the Web is biased towards pages with high Pagerank [HHMN00]. However, there are differences in how quickly high-quality pages are found depending on the ordering of pages.

The *historical-pagerank* family of strategies were very good, and in case of no historical information available, *OPIC* and *larger-sites-first* are our recommendations. *Breadth-first* has a bad performance compared with these strategies; *batch-pagerank* requires to do a full Pagerank computation several times during the crawl, which is computationally very expensive, and the performance is not better than simpler strategies.

Notice that the *larger-sites-first* strategy has practicals advantage over the *OPIC* strategy. First, requires less computation time, and also does not require knowledge of all in-links to a given page as *OPIC* does. The later is relevant when we think of distributed crawlers as no communication between computers is required to exchange these data during the crawling process. Thus *larger-sites-first* has better scalability making it more suitable for large scale distributed crawlers.

Also, our simulation results show that attempting to retrieve as many pages from a given site ($c >> 1$), allows the crawler to effectively amortize the waiting time w before visiting the same site again. This certainly helps to achieve a better utilization of the available bandwidth, and is good for both the search engine and the Web site administrator.

Experiments with a real crawl using the *larger-sites-first* strategy on the ever-changing Web validated our conclusions whereas simulation was the only way to ensure that all strategies considered were compared under the same conditions.

We verified that after a few days, the quality of the retrieved pages is lower than at the beginning of the crawl. At some point, and with limited resources, it could be pointless to continue crawling, but, when is the right time to stop a crawl? The next chapter deals with this subject through models and actual data from Web usage.

Chapter 5

Crawling the Infinite Web

We have seen in Chapter 4 several scheduling policies for ordering pages during Web crawling. The objective of those policies is to retrieve "good" pages early in the crawl. We have considered that the Web is bounded, but a large amount of the publicly available Web pages are generated dynamically upon request, and contain links to other dynamically generated pages. This usually results in Web sites that can be considered to have arbitrarily many pages.

This poses a problem to Web crawling, as it must be done in such a way that it stops downloading pages from each Web site at some point. But how deep must the crawler go?

In this chapter:

- We propose models for random surfing inside a Web site when the number of pages is *unbounded*. For that, we take the tree induced by the Web graph of a site, and study it by levels.
- We analyze these models, focusing on the question of how "deep" users go inside a Web site.
- We validate these models using actual data from Web sites, as well as using a link analysis measure such as Pagerank [PBMW98].

The next section outlines the motivation of this work, namely, the existence of dynamic pages. In Section 5.2, three models of random surfing in dynamic Web sites are presented and analyzed; in Section 5.3, these models are compared with actual data from the access log of several Web sites. The last section concludes with some final remarks and recommendations for practical Web crawler implementations.

Portions of this chapter were presented in [BYC04].

5.1 Static and dynamic pages

Most studies about the Web refer only to the "publicly indexable portion" [LG98], excluding a portion of the Web that has been called "the hidden Web" [RGM01] or the "deep Web" [Ber01, GA04]. The non-indexable portion is characterized as all the pages that normal users could eventually access, but automated agents such as the crawlers used by search engines can not.

Certain pages are not indexable because they require previous registration or some special authorization such as a password, or are only available when visited from within a certain network, such as a corporate intranet. Others are *dynamic pages*, generated after the request has been made. Some times they are not indexable because they require certain parameters as input, e.g. query terms, and those query terms are unknown at crawling time. The different portions of the Web are depicted in Figure 5.1.

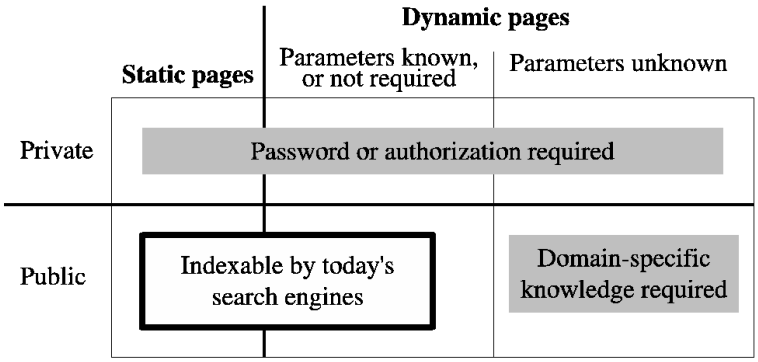


Figure 5.1: The Web can be divided into password-protected and publicly available, and into dynamic and static pages.

However, many dynamic pages are indexable, as the parameters for creating them can be found by following links. This is the case of, e.g. typical product catalogs in Web stores, in which there are links to navigate the catalog without the user having to pose a query.

The Web is usually considered as a collection of pages, in the same sense as in traditional Information Retrieval collections, but much larger. Under this assumption, the Web graph has a finite number of nodes in which measures such as diameter are well defined. This is fundamentally wrong. The amount of information in the Web at any given time is certainly finite, but when a dynamic page leads to another dynamic page, *the number of pages can be potentially infinite*. Take for instance a dynamic page that implements a calendar, you can always click on "next month" and from some point on there will be no more data items in the calendar; humans can be reasonably sure that it is very unlikely to find events scheduled 50 years in advance, but a crawler can not. A second example would be a calculator, such as a dynamic page that calculates approximations of π using an iterative method. A crawler cannot tell when two pages reflect the same information. There are many more examples of "crawler traps" that involve loops and/or near-duplicates that can be detected afterwards, but we want to avoid downloading them.

Also, personalization is a source of a large number of pages; if you go to `www.amazon.com` and start browsing your favorite books, soon you will be presented with more items about the same topics and automatically generated lists of recommendations, as the Web site assembles a vector of preferences of the visitor. The visitor is, in fact, creating Web pages as it clicks on links, and an automated agent such as a Web crawler generates the same effect. This is a case of uncertainty, in which the instrument, the Web crawler, affects the object it is attempting to measure.

This poses a problem to Web crawling, as it must be done in such a way that it stops downloading pages from each Web site at some point. Most researchers usually take one of the following approaches to this:

Download only static pages A common heuristic to do so is to avoid downloading URLs containing a question mark, but this heuristic can fail as there are many URLs which are dynamically generated but do not use the CGI standard, encoding the parameters in the rest of the URL. Also, a valuable fraction of the publicly available Web pages is generated dynamically upon request, and it is not clear why those pages should be penalized in favor of static pages.

Download dynamic pages only with one set of parameters When doing this, dynamic pages are either downloaded with the set of parameters of the first time they are found, or with an empty set of parameters. The obvious drawback is that dynamic pages could query a database and a single set of parameters cannot represent the contents of the database.

Download up to a maximum amount of pages This creates a data set that is highly dependent on the crawling strategy. Moreover, this cannot be used to compare, for instance, the amount of information on different domains.

Download up to a certain amount of pages per domain name As a small sum has to be paid for registering a domain name, there is a certain effort involved in creating a Web site under a domain name. However, there are certain domain names such as ".co.uk" which are very large and might require special rules.

Download up to a certain amount of levels per Web site Starting from the home page of each Web site, follow links up to a certain depth. This is the approach we consider in this paper, and the natural question is: how deep must the crawler go?

The Web of dynamically generated content is crawled superficially by many Web crawlers, in some cases because the crawler cannot tell a dynamic URL from a static one, and in other cases purposefully. However, few crawlers will go deeper, unless they know when to stop and how to handle dynamic pages with links to more dynamic pages. In our previous experiences with the WIRE crawler [BYC02], we usually limit the depth at which pages are explored, typically to 5 links in dynamic pages and 15 links in static pages. When we plot the number of pages at a given depth, a profile as the one shown in Figure 5.2 is obtained.

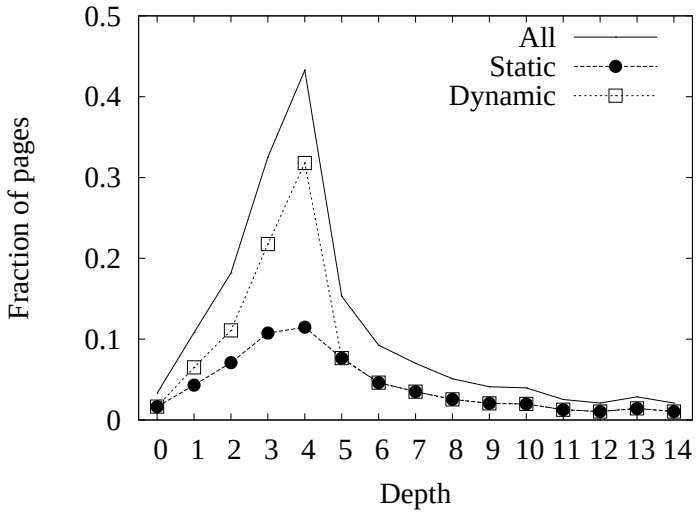


Figure 5.2: Amount of static and dynamic pages at a given depth. Dynamic pages were crawled up to 5 levels, and static pages up to 15 levels. At all depths, static pages represent a smaller fraction of the Web than dynamic pages.

Notice that here we are not using the number of slashes in the URL, but using the real shortest distance in links with the start page(s) of the Web site. The dynamic pages grow with depth, while the static pages follow a different shape, with the maximum number of pages found around 2 or 3 links deep; this is why some search engines use the heuristic of following links to URLs that seems to hold dynamically generated content only from pages with static content. This heuristic is valid while the amount of information in static pages continues to be large, but that will not be the case in the near future, as large Web sites with only static pages are very hard to maintain.

We deal with the problem of capturing a relevant portion of the *dynamically generated content with known parameters*, while avoiding the download of too many pages. We are interested in knowing if a user will ever see a dynamically generated page. If the probability is too low, should a search engine like to retrieve that page? Clearly, from the Web site or the searcher's point of view, the answer should be yes, but from the search engine's point of view, the answer might be no.

5.2 Random surfer models for an infinite Web site

We will consider a Web site $S = (Pages, Links)$ as a set of pages under the same host name that forms a directed graph. The nodes are $Pages = \{P_1, P_2, \dots\}$ and the arcs are $Links$ such that $(P_i, P_j) \in Links$ iff there exists a hyperlink from page P_i to page P_j in the Web site.

Definition (User session) We define a user session $\mathbf{u}$ as a finite sequence of page views $\mathbf{u} = (P_1, P_2, \ldots, P_n)$, with $P_i \in \mathit{Pages}$, and $(P_i, P_{i+1}) \in \mathit{Links}$. The first request u_0 does not need to be the start page located at the root directory of the server, as some users may enter to Web site following a link to an internal page, e.g., if they come from a search engine.

Definition (Page depth) For a page P_i and a session $\mathbf{u}$, we define the depth of the page in the session, $depth(P_i, \mathbf{u})$ as:

$$depth(P_i, \mathbf{u}) = \begin{cases} 0 & \text{if } P_i = u_0 \\ \min depth(P_j, \mathbf{u}) + 1 & P_j \in \mathbf{u}, j < i, (P_j, P_i) \in \mathit{Links} \end{cases}$$

The depth is basically the length of the shortest path from the start page through the pages actually seen during a session. Note that the depth of a page is not only a function of the Web site structure, it is the *perceived* depth during a particular session $\mathbf{u}$.

Definition (Session depth) We define the depth of session $\mathbf{u}$ as $\max depth(P_i, u)$ with $P_i \in u$. We are interested in this variable as its distribution is relevant from the point of view of search engines.

For random surfing, we can model each page in *Pages* as a state in a system, and each hyperlink in *Links* as a possible transition. This kind of model has been studied by Huberman *et al.* [HPPL98, AH00]. We propose to use a related model that collapses multiple pages at the same level as a single node, as shown in Figure 5.3. That is, the Web site graph is collapsed to a sequential list.

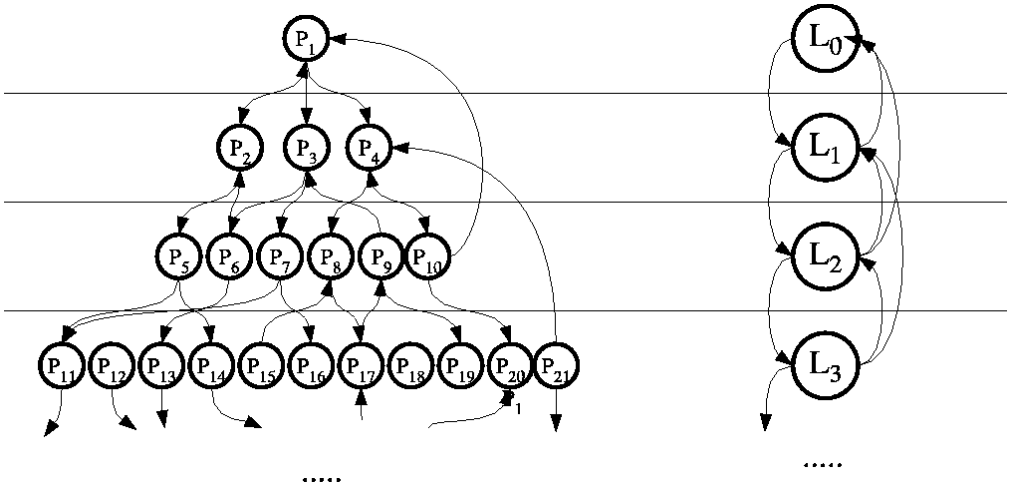


Figure 5.3: A Web site and a sequence of user actions can be modeled as a tree (left). If we are concerned only with the depth at which users explore the Web site, we can collapse the tree to a linked list of levels (right).

The advantage of modeling the Web site graph as a sequential list as that we do not need to model exactly which page a user is visiting, because we do not need this information as our main concern is at what

depth the user is inside a Web site. Also, different Web sites have varying degrees of connectivity, so for considering the entire Web site we would need to model both the number of out-links of each page and the distribution of the overlap of out-links between pages.

At each step of the walk, the surfer can perform one of the following atomic actions: go to the next level (action *next*), go back to the previous level (action *back*), stay in the same level (action *stay*), go to a different previous level (action *prev*), go to a different deeper level (action fwd), go to the start page (action *start*) or jump outside the Web site (action $jump$).

For action $jump$ we add an extra node `EXIT` to signal the end of a user session (closing the browser, or going to a different Web site) as shown in Figure 5.4. Regarding this Web site, after leaving, users have only one option: start again in a page with depth 0 (action *start*).

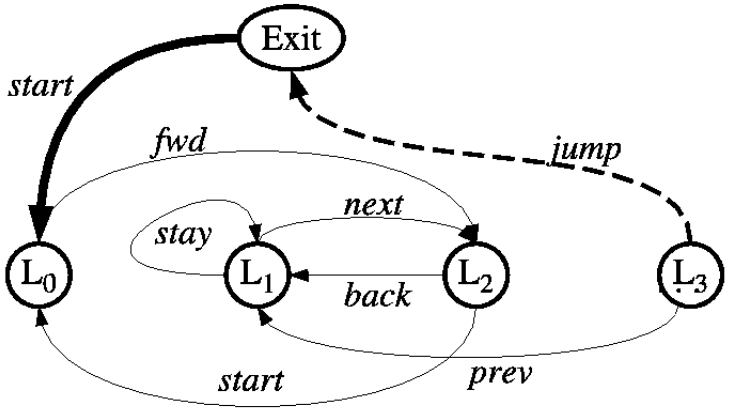


Figure 5.4: Representation of the different actions of the random surfer. The EXIT node represents leaving the Web site, and the transition between that node and the start level has probability one.

As this node `EXIT` has a single out-going link, it does not affect the results for the other nodes if we remove the node `EXIT` and change this by transitions going to the start level L_0. Another way to understand it is that as this process has no memory, *going back to the start page or starting a new session are equivalent*, so actions $jump$ and *start* are indistinguishable in terms of the resulting probability distribution for the other nodes. As a response to the same issue, Levene *et al.* [LBL01] proposed to use an absorbing state representing leaving the Web site; but we cannot use this idea because we want to calculate and compare stationary probability distributions.

The set of atomic actions is $\mathcal{A} = \{next, start/jump, back, stay, prev, fwd\}$ and the probabilities if the user is currently at level ℓ, are:

- $Pr(next|\ell)$: probability of advancing to the level $\ell+1$.
- $Pr(back|\ell)$: probability of going back to the level $\ell-1$.

- $Pr(stay|\ell)$: probability of staying at the same level ℓ.
- $Pr(start, jump|\ell)$: probability of going to the start page of this session, when it is not the previous two cases; this is equivalent in our model to begin a new session,
- $Pr(prev|\ell)$: probability of going to a previous level that is neither the start level nor the immediate preceding level.
- $Pr(fwd|\ell)$: probability of going to a following level that is not the next level.

As they are probabilities, $\sum_{action \in \mathcal{A}} Pr(action|\ell) = 1$. The probability distribution of all levels at a given time is the vector $\mathbf{x}(t)$. When there exists a limit, we will call this $\lim_{t\to\infty} \mathbf{x}(t) = \mathbf{x}$.

In this paper, we study three models with $Pr(next|\ell) = q$ for all levels, i.e.: the probability of advancing to the next level is constant. Our purpose is to predict how far will a real user go into a dynamically generated Web site. If we know that, e.g.: $x_0 + x_1 + x_2 \geq 0.9$, then the crawler could decide to crawl just those three levels.

The models we analyze were chosen to be as simple and intuitive as possible, though without sacrificing correctness. We seek more than just fitting the distribution of user clicks, we want to understand and explain user behavior in terms of simple operations.

Our models are "birth-and-death" processes, because they have an interpretation in terms of each level being a number representing the population of a certain species, and each transition between two levels represents either a birth of a death of a member. In this context, we note in advance that any given model in which from a certain point over the rate of death (going back to the first levels) exceeds the rate of birth (going deeper), then the population will be bounded (the visits will be found mostly in the first levels).

5.2.1 Model A: back one level at a time

In this model, with probability q the user will advance deeper, and with probability $1-q$ the user will go back one level, as shown in Figure 5.5.

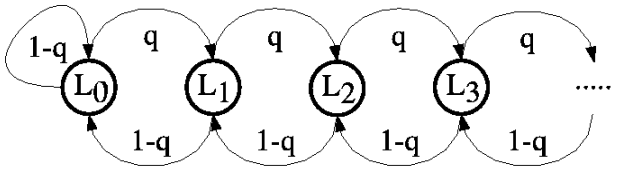


Figure 5.5: Model A, the user can go forward or backward one level at a time.

Transition probabilities are given by:

- $Pr(next|\ell) = q$

- $Pr(back|\ell) = 1-q$ for $\ell \geq 1$
- $Pr(stay|\ell) = 1-q$ for $\ell = 0$
- $Pr(start, jump|\ell) = 0$
- $Pr(prev|\ell) = Pr(fwd|\ell) = 0$

A stable state **x** is characterized by:

$$\begin{aligned} x_i &= qx_{i-1} + (1-q)x_{i+1} \quad (\forall i \geq 1) \\ x_0 &= (1-q)x_0 + (1-q)x_1 \end{aligned}$$

The solution to this recurrence is:

$$x_i = x_0 \left(\frac{q}{1-q}\right)^i \quad (\forall i \geq 1) \ .$$

If $q \geq 1/2$ then the solution is $x_i = 0$, and $x_\infty = 1$, so we have an asymptotic absorbing state. In our framework this means that no depth boundary can ensure a certain proportion of pages visited by the users. When $q < 1/2$ and we impose the normalization constraint, $\sum_{i\geq 0} x_i = 1$, we have a geometric distribution:

$$x_i = \left(\frac{1-2q}{1-q}\right)\left(\frac{q}{1-q}\right)^i$$

The cumulative probability of levels $0 \ldots k$ is:

$$\sum_{i=0}^{k} x_i = 1 - \left(\frac{q}{1-q}\right)^{k+1}$$

This distribution is shown in Figure 5.6. We also calculate the session length, if we consider that a session ends when the user returns to level zero, as actions *start* and *jump* are equivalent. This is equivalent to the average return time to the origin in a Markov chain, which is $1/x_0$ [MT93]. Hence, $E(|\mathbf{u}|) = \frac{1-q}{1-2q}$.

5.2.2 Model B: back to the first level

In this model, the user will go back to the start page of the session with probability $1-q$. This is shown in Figure 5.7.

The transition probabilities are given by:

- $Pr(next|\ell) = q$
- $Pr(back|\ell) = 1-q$ if $\ell = 1$, 0 otherwise
- $Pr(stay|\ell) = 1-q$ for $\ell = 0$

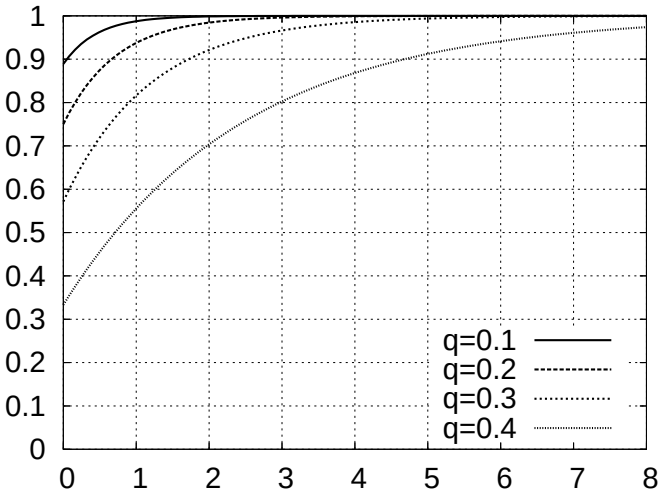


Figure 5.6: Distribution of visits per depth predicted by model A.

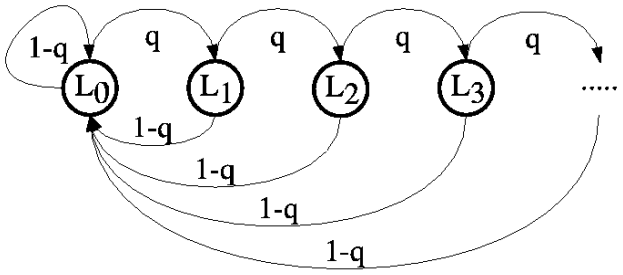


Figure 5.7: Model B, users can go forward one level at a time, or they can go back to the first level either by going to the start page, or by starting a new session.

- $Pr(start, jump|\ell) = 1-q$ for $\ell \geq 2$
- $Pr(prev|\ell) = Pr(fwd|\ell) = 0$

A stable state **x** is characterized by:

$$\begin{aligned} x_0 &= (1-q)\sum_{i\geq 0} x_i = (1-q) \\ x_i &= qx_{i-1} \quad (\forall i \geq 1) \end{aligned}$$

and $\sum_{i\geq 0} x_i = 1$.

As we have $q < 1$ we have another geometric distribution:

$$x_i = (1-q)q^i$$

The cumulative probability of levels $0..k$ is:

$$\sum_{i=0}^{k} x_i = 1 - q^{k+1}$$

This distribution is shown in Figure 5.8. In this case we have $E(|\mathbf{u}|) = \frac{1}{1-q}$.

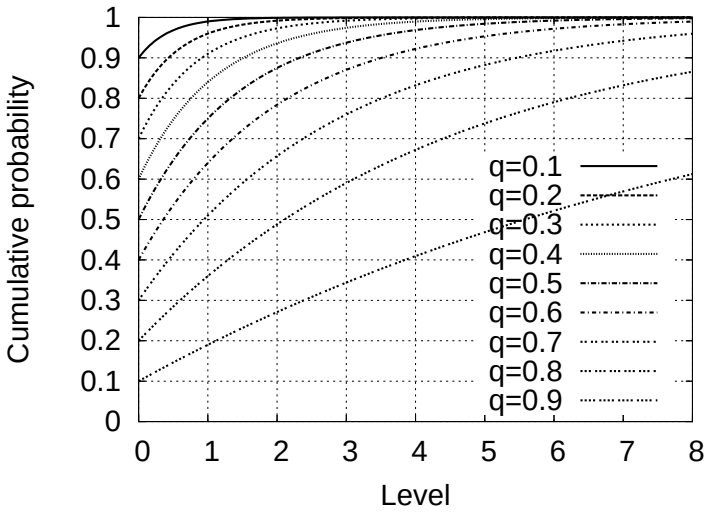


Figure 5.8: Distribution of visits per depth predicted by model B.

5.2.3 Model C: back to any previous level

In this model, the user can either discover a new level with probability q, or go back to a previous visited level with probability $1-q$. If a user decides to go back to a previously seen level, the level will chosen uniformly from the set of visited levels (including the current one), as shown in the Figure 5.9.

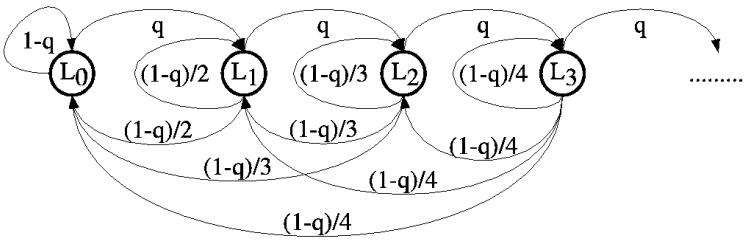


Figure 5.9: Model C: the user can go forward one level at a time, and can go back to previous levels with uniform probability.

The transition probabilities are given by:

- $Pr(next|\ell) = q$
- $Pr(back|\ell) = 1 - q/(\ell+1)$ for $\ell \geq 1$
- $Pr(stay|\ell) = 1 - q/(\ell+1)$
- $Pr(start, jump|\ell) = 1 - q/(\ell+1)$ for $\ell \geq 2$

- $Pr(prev|\ell) = 1 - q/(\ell+1)$ for $\ell \geq 3$
- $Pr(fwd|\ell) = 0$

A stable state **x** is characterized by:

$$\begin{aligned} x_0 &= (1-q)\sum_{k\geq 0}\frac{x_k}{k+1} \\ x_i &= qx_{i-1} + (1-q)\sum_{k\geq i}\frac{x_k}{k+1} \quad (\forall i > 1) \end{aligned}$$

and $\sum_{i\geq 0} x_i = 1$.

We obtain a solution of the form:

$$x_i = x_0\,(i+1)\,q^i$$

Imposing the normalization constraint, this yields:

$$x_i = (1-q)^2\,(i+1)\,q^i$$

The cumulative probability of levels $0..k$ is:

$$\sum_{i=0}^{k} x_i = 1 - (2+k-(k+1)\,q)\,q^{k+1}$$

This distribution is shown in Figure 5.10. In this case we have $E(|\mathbf{u}|) = \frac{1}{(1-q)^2}$.

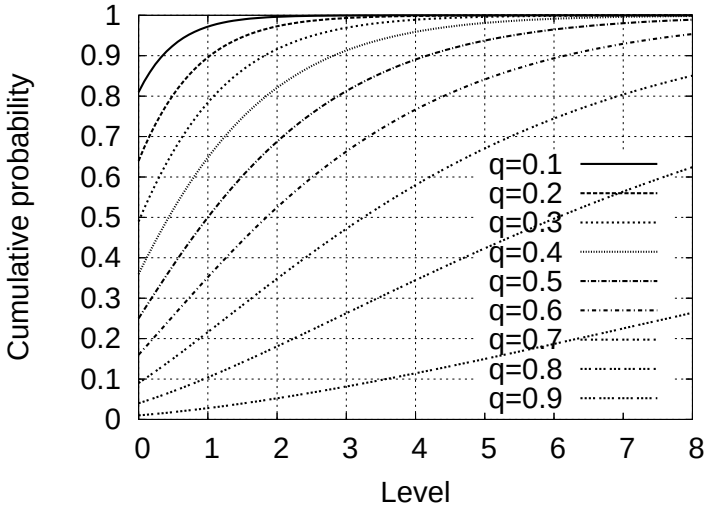


Figure 5.10: Distribution of visits per depth predicted by model C.

5.2.4 Model comparison

We can see that if $q \leq 0.4$, then in these models there is no need for the crawler to go past depth 3 or 4 to capture more than 90% of the pages a random surfer will actually visit, and if q is larger, say, 0.6, then the crawler must go to depth 6 or 7 to capture the same amount of page views.

Note that the cumulative distribution obtained with model A ("back one level") using parameter q_A, and model B ("back to home") using parameter q_B are equivalent if:

$$q_A = \frac{q_B}{1+q_B} \ .$$

So, as the distribution of session depths is equal, except for a transformation in the parameter q, we will consider only model B for charting and fitting the distributions of session depth.

It is worth noticing that a good model should approximate both the distribution of session depth and the distribution of session length. Table 5.1 shows the predicted session lengths.

Table 5.1: Predicted average session length for the models, with different values of q.

q	Model A	Model B	Model C
0.1	1.13	1.11	1.23
0.2	1.33	1.25	1.56
0.3	1.75	1.43	2.04
0.4	3.00	1.67	2.78
0.5	–	2.00	4.00
0.6	–	2.50	6.25
0.7	–	3.34	11.11
0.8	–	5.00	25.00
0.9	–	10.00	100.00

In Table 5.1 we can see that although the distribution of session depth is the same for models A and B, model B predicts shorter sessions. Observed average session lengths in the studied Web sites are mostly between 2 and 3, so reasonable values for q lie between 0.4 and 0.6.

5.3 Data from user sessions in Web sites

We studied real user sessions on 13 different Web sites in the US, Spain, Italy and Chile, including commercial and educational sites, non-governmental organizations, and sites in which collaborative forums play a major role, also known as "Blogs".

We obtained access logs with anonymous IP addresses from these Web sites, and processed them to obtain user sessions:

- Sort the logs by IP address of the client, then by access time stamp.
- Consider only `GET` requests for static and dynamic HTML pages or documents such as Word, PDF or Postscript.
- Consider that a session expires after 30 minutes of inactivity, as this is common in log file analysis software, and is based on empirical data [CP95].
- Consider that a session expires if the `User-Agent` changes [CMS99], as a way of overcoming the issue that multiple clients can be behind the same IP address.
- Consider multiple consecutive hits to the same page (page reload) as a single page view.
- In pages with frames, consider all the frames as a single page, this required manual inspection of pages with frames.
- Ignore hits to Web applications such as e-mail or content management systems, as they neither respond to the logic of page browsing, nor are usually accessible by Web crawlers.
- Expand a session with missing pages (e.g.: if the user clicks "back" in his browser, and then follow a link). This information is obtained from the `Referrer` field, and is a way of partially overcoming the issue of caching. Note that, as re-visits are not always recorded because of caching [TG97], data from log files *overestimates the depth at which users spent most of the time*, so user visits could be actually even less deep.

Additionally, manual inspection of the data led to the following heuristics to discard automated agents:

- Identify robots by their accesses to the `/robots.txt` file, as suggested by Tan and Kumar [TK02].
- Identify robots by known `User-Agent` fields.
- Ignore malicious hits searching for security holes, which are usually a sequence of requests searching for buffer overflows or other software bugs. These requests are usually done by automated agents like Nessus [Der04].

5.3.1 General characteristics of user sessions

The characteristics of the sample, as well as the results of fitting models B and C to the data are summarized in Table 5.2. The names of the Web sites are not public because some of the log files, specially those of commercial entities, were obtained under the condition of publishing only the statistical results.

Table 5.2: Characteristics of the studied Web sites. The number of user sessions does not reflect the relative traffic of the Web sites, as it was obtained in different time periods. The average number of page views per session is larger in Blogs. "Root entry" is the fraction of sessions starting in the home page.

Code	Type	Country	Recorded sessions	Average session length	Average max. depth	Root entry
E1	Educational	Chile	5,500	2.26	0.98	84%
E2	Educational	Spain	3,600	2.82	1.41	68%
E3	Educational	US	71,300	3.10	1.31	42%
C1	Commercial	Chile	12,500	2.85	0.99	38%
C2	Commercial	Chile	9,600	2.12	1.01	32%
R1	Reference	Chile	36,700	2.08	0.95	11%
R2	Reference	Chile	14,000	2.72	1.21	22%
O1	Organization	Italy	10,700	2.93	1.97	63%
O2	Organization	US	4,500	2.50	1.13	1%
OB1	Organization + Blog	Chile	10,000	3.73	1.89	31%
OB2	Organization + Blog	Chile	2,000	5.58	2.48	84%
B1	Blog	Chile	1,800	9.72	3.56	39%
B2	Blog	Chile	3,800	10.39	2.31	21%

By inspecting Table 5.2, we observe that the average session length involves about 2 to 3 pages, and user sessions in Blogs are larger than in the other Web sites. This is reasonable as Web postings are very short, so a user reads several of them during a session.

5.3.2 Distribution of visits per depth

Figure 5.11 shows the cumulative distribution of visits per page depth to Web sites. We can see that at least 80%-95% of the visits occur at depth ≤ 4 (this is, no more than four "clicks" away from the entry page). It is also noticeable that about 30%-50% of the sessions include only the start page.

The distribution of visits per depth follows a power law, as shown in Figure 5.12. We only selected the log files with more than 10,000 sessions recorded in order to have enough sessions across the entire range of the figure, which is 30 levels.

An interesting observation about the distribution session lengths is that although they are longer in Blogs, they are not much deeper than in the other Web sites, as shown in Table 5.2. This led us to study the relationship between session length and session depth. The result is shown in Figure 5.13, which uses

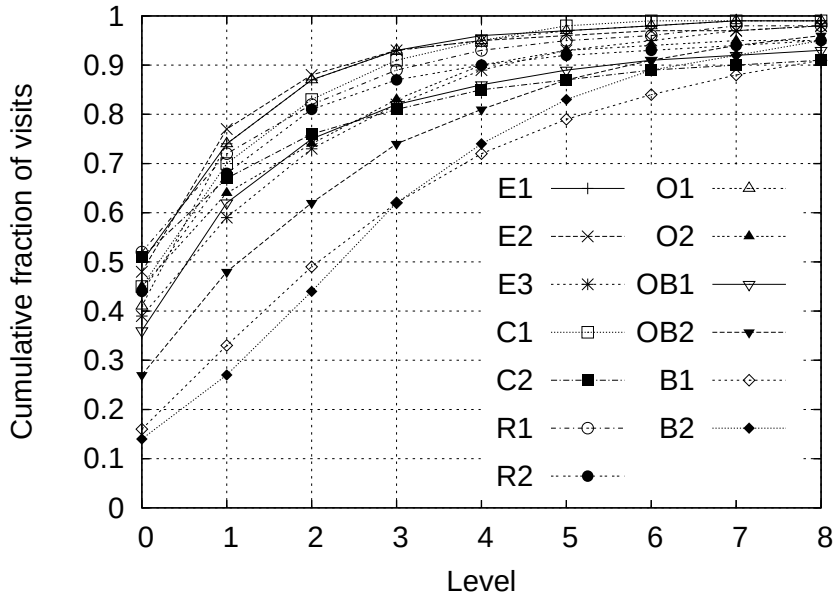


Figure 5.11: Cumulative distribution of visits per level, from access logs of Web sites. E=educational, C=commercial, O=non-governmental organization, OB=Organization with on-line forum, B=Blog (Web log or on-line forum).

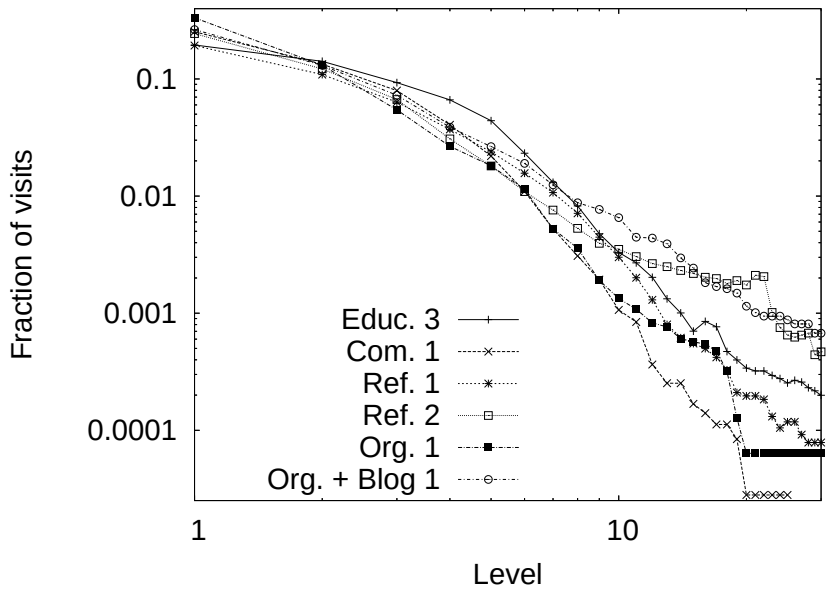


Figure 5.12: Distribution of visits per level. In this figure we only selected the log files with more than 10,000 sessions recorded.

information from all our samples including Blogs. Session depth grows slower than session length, and even long sessions, which are very rare, are not so deep. User browsing is certainly not depth-first.

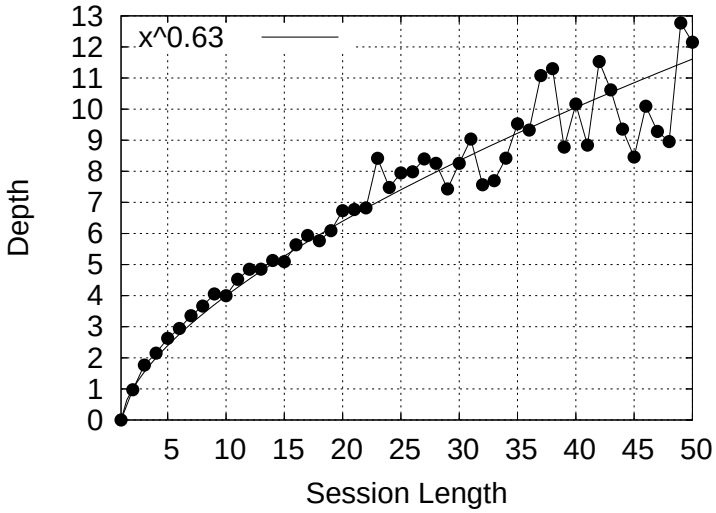


Figure 5.13: Session length vs. average session depth in the studied user sessions. Even very long sessions, which are rare, are not very deep.

The discrepancy between session length and depth is important from the point of view of an alternative model. Suppose the user chooses a session length at random before entering the Web site (this session length could reflect that the user has a certain amount of time or interest in the topic). In this model, the average session depth could be overestimated if we do not account for the fact that the browsing pattern is not depth-first. Figure 5.14 shows the session length distribution, which follows a power law with parameter almost -2. This differs from the results of Huberman that had parameter -3/2 [HPPL98].

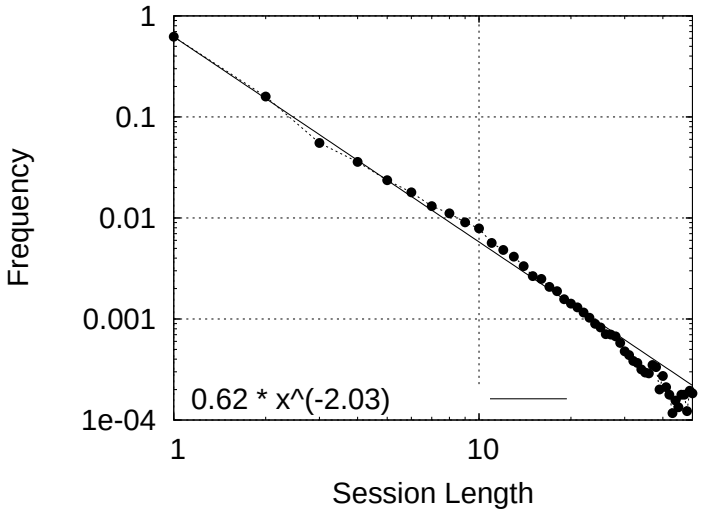


Figure 5.14: Session length distribution.

Table 5.3: Results of fitting models B (equivalent to model A) and C to the distribution of visits per depth in the studied Web sites. The minimum fitting error for each Web site is shown in bold face.

	Model B		Model C	
Code	q	Error	q	Error
Educ. 1	0.51	**0.88%**	0.33	3.69%
Educ. 2	0.51	**2.29%**	0.32	4.11%
Educ. 3	0.64	**0.72%**	0.45	3.65%
Com. 1	0.55	**0.39%**	0.36	2.90%
Com. 2	0.62	**5.17%**	0.41	10.48%
Ref. 1	0.54	**2.96%**	0.34	6.85%
Ref. 2	0.59	**2.75%**	0.39	6.11%
Org. 1	0.54	2.36%	0.35	**2.27%**
Org. 2	0.62	**2.31%**	0.42	5.95%
Org. + Blog 1	0.65	**2.07%**	0.46	5.20%
Org. + Blog 2	0.72	**0.35%**	0.54	2.00%
Blog 1	0.79	0.88%	0.63	**0.70%**
Blog 2	0.78	1.95%	0.63	**1.01%**

5.4 Model fit

We fitted the models of cumulative depth to the data from Web sites. The results are presented in Table 5.3 and Figure 5.18. In general, the curves produced by model B (and model A) are a better approximation to the user sessions than the distribution produced by model C, except for Blogs, as seen in Figure 5.19. The approximation is good for characterizing session depth, with error in general lower than 5%.

We also studied the empirical values for the distribution of the different actions at different levels in the Web site. We averaged this distribution across all the studied Web sites at different depths. The results are shown in Table 5.4, in which we consider all the Web sites except for Blogs.

Inspecting Table 5.4, we can see that the actions *next*, *jump* and *back* are the more important ones, which is evidence for the adequacy of models A (back one level) and model B (back to start level).

We can see in Figure 5.15 that $Pr(next|\ell)$ does not vary too much, and lies between 0.45 and 0.6, increasing as ℓ grows. This is reasonable as a user that already have seen several pages is more likely to follow a link. From the point of view of our models, it is certainly not constant, but is almost constant for the first five levels which are the relevant ones. On the other hand, *prev* and *back* are closer to constant.

Table 5.4: Average distribution of the different actions in user sessions of the studied Web sites, except for Blogs. Transitions with values greater than 0.1 are shown in boldface.

Level	Observations	Next	Start	Jump	Back	Stay	Prev	Fwd
0	247985	**0.457**	–	**0.527**	–	0.008	–	0.000
1	120482	**0.459**	–	**0.332**	**0.185**	0.017	–	0.000
2	70911	**0.462**	**0.111**	**0.235**	**0.171**	0.014	–	0.001
3	42311	**0.497**	0.065	**0.186**	**0.159**	0.017	0.069	0.001
4	27129	**0.514**	0.057	**0.157**	**0.171**	0.009	0.088	0.002
5	17544	**0.549**	0.048	**0.138**	**0.143**	0.009	**0.108**	0.002
6	10296	**0.555**	0.037	**0.133**	**0.155**	0.009	**0.106**	0.002
7	6326	**0.596**	0.033	**0.135**	**0.113**	0.006	**0.113**	0.002
8	4200	**0.637**	0.024	**0.104**	**0.127**	0.006	0.096	0.002
9	2782	**0.663**	0.015	**0.108**	**0.113**	0.006	0.089	0.002
10	2089	**0.662**	0.037	0.084	**0.120**	0.005	0.086	0.003
11	1649	**0.656**	0.020	0.076	**0.119**	0.018	**0.105**	0.004
12	1273	**0.687**	0.040	0.091	0.091	0.007	0.082	0.001
13	1008	**0.734**	0.015	0.058	**0.112**	0.005	0.054	0.019
14	814	**0.716**	0.005	0.051	**0.113**	0.015	0.080	0.019
15	666	**0.762**	0.025	0.056	0.091	0.008	0.041	0.017

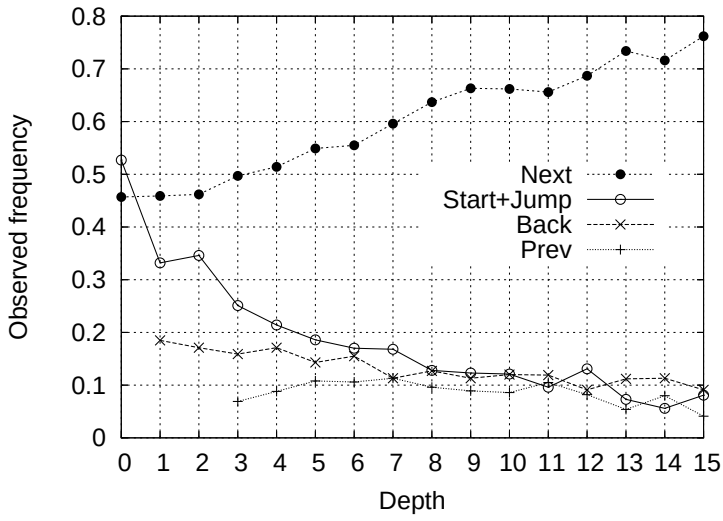


Figure 5.15: Experimental values for our atomic actions.

Actions *start*, *stay* and fwd are not very common. These actions include visits to pages that have been already seen, but it seems that pages are only re-visited by going back one level.

5.5 Conclusions

The models and the empirical data presented lead us to the following characterization of user sessions: they can be modeled as a random surfer that either advances one level with probability q, or leaves the Web site with probability $1-q$. In general $q \approx 0.45 - 0.55$ for the first few levels, and then $q \approx 0.65 - 0.70$. This simplified model is good enough for representing the data for Web sites, but:

- We could also consider Model A (back one level at a time), which is equivalent in terms of cumulative probability per level, except for a change in the parameters. Based on the empirical data, we observe that users at first just leave the Web site while browsing (Model B), but after several clicks, they are more likely to go back one level (Model A).
- A more complex model could be derived from empirical data, particularly one that considers that q depends on ℓ. We considered that for deciding when to stop while doing Web crawling, the simple model is good enough.
- Model C appears to be better for Blogs. A similar study to this one, focused only in the access logs of Blogs seems a reasonable thing to do since Blogs represent a growing portion of on-line pages.

In all cases, the models and the data show evidence of a distribution of visits that is strongly biased to the first few levels of the Web site. According to this distribution, more than 90% of the visits are closer than 3 to 4 clicks away from the entry page in most of the Web sites. In the case of Blogs, we observed deeper user sessions, with 90% of the visits within 6 to 7 clicks away from the entry page. Although our models do not fit well for deep sessions, they are accurate for the first five relevant levels. Also, we would need much more data to get significant results for over six levels.

In theory, as internal pages can be starting points, it could be concluded that Web crawlers must always download entire Web sites. However, entry pages are usually only in the first few levels of a Web site. If we consider the physical page depth in the directory hierarchy of a Web site, we observe that the frequency of surfing entry points per level rapidly decreases, as shown in Figure 5.16. This is consistent with the findings of Eiron *et al.*; they observed that "when links are external to a site, they tend to link to the top level of the site" [EMT04].

Link analysis, specifically Pagerank, provides more evidence for our conclusions. We asked, what fraction of the total Pagerank score is captured by the pages on the first ℓ levels of the Web sites? To answer this, we crawled a large portion of the Chilean Web (.cl) obtaining around 3 million pages in April of

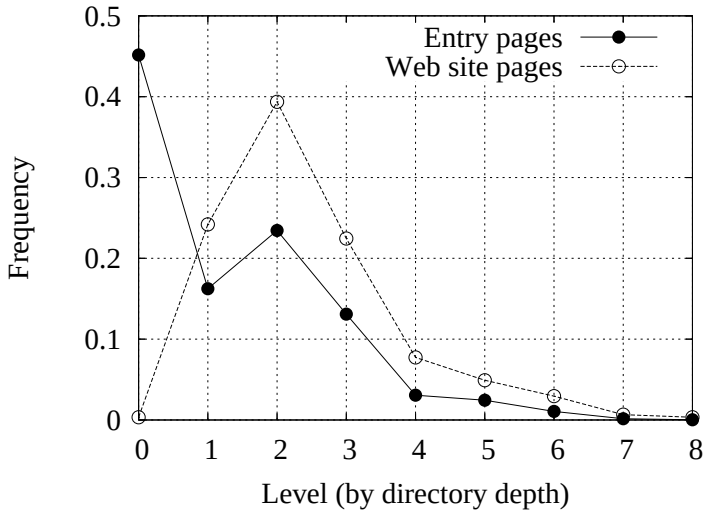


Figure 5.16: Fraction of different Web pages seen at a given depth, and fraction of entry pages at the same depth, considering the directory structure, in the studied Web sites. Frequencies are normalized relative to all pages.

2004, using 150 thousand seed pages that found 53 thousand Web sites. Figure 5.17 shows the cumulative Pagerank score for this sample. Again, the first five levels capture more than 80% of the best pages. Note that the levels here are obtained in terms of the global Web structure, considering internal and external links, not user sessions. These results are consistent with the findings by Najork and Wiener [NW01].

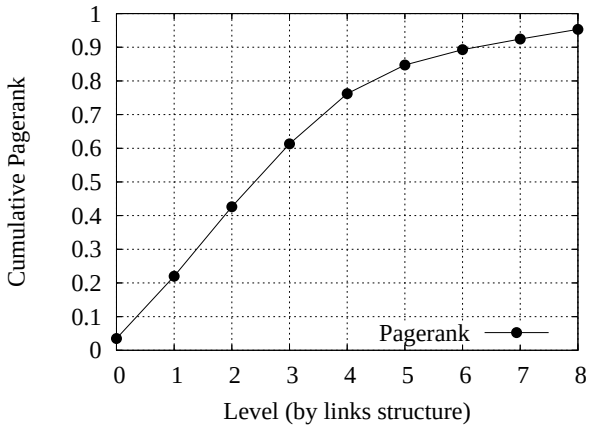


Figure 5.17: Cumulative Pagerank by page levels in a large sample of the Chilean Web.

These models and observations could be used by a search engine, and they can also account for differences in Web sites. For instance, if the search engine's crawler performs a breadth-first crawling and can measure the ratio of new URLs from a Web site it is adding to its queue vs. seen URLs, then it should be able to infer how deep to crawl that specific Web site. The work we presented in this article provides a framework

for that kind of adaptivity.

An interesting enhancement of the models shown here is to consider the content of the pages to detect duplicates and near-duplicates. In our model, downloading a duplicate page should be equivalent to going back to the level at that we visited that page for the first time. A more detailed analysis could also consider the distribution of terms in Web pages and anchor text as the user browses through a Web site.

A different class of models for user browsing, including models based on economical decisions could be used, but those models should be able to fit both, the distribution of session length and the expected session depth.

As the amount of on-line content that people, organizations and business are willing to publish grows, more Web sites will be built using Web pages that are dynamically generated, so those pages cannot be ignored by search engines. Our aim is to generate guidelines to crawl these new, practically infinite, Web sites.

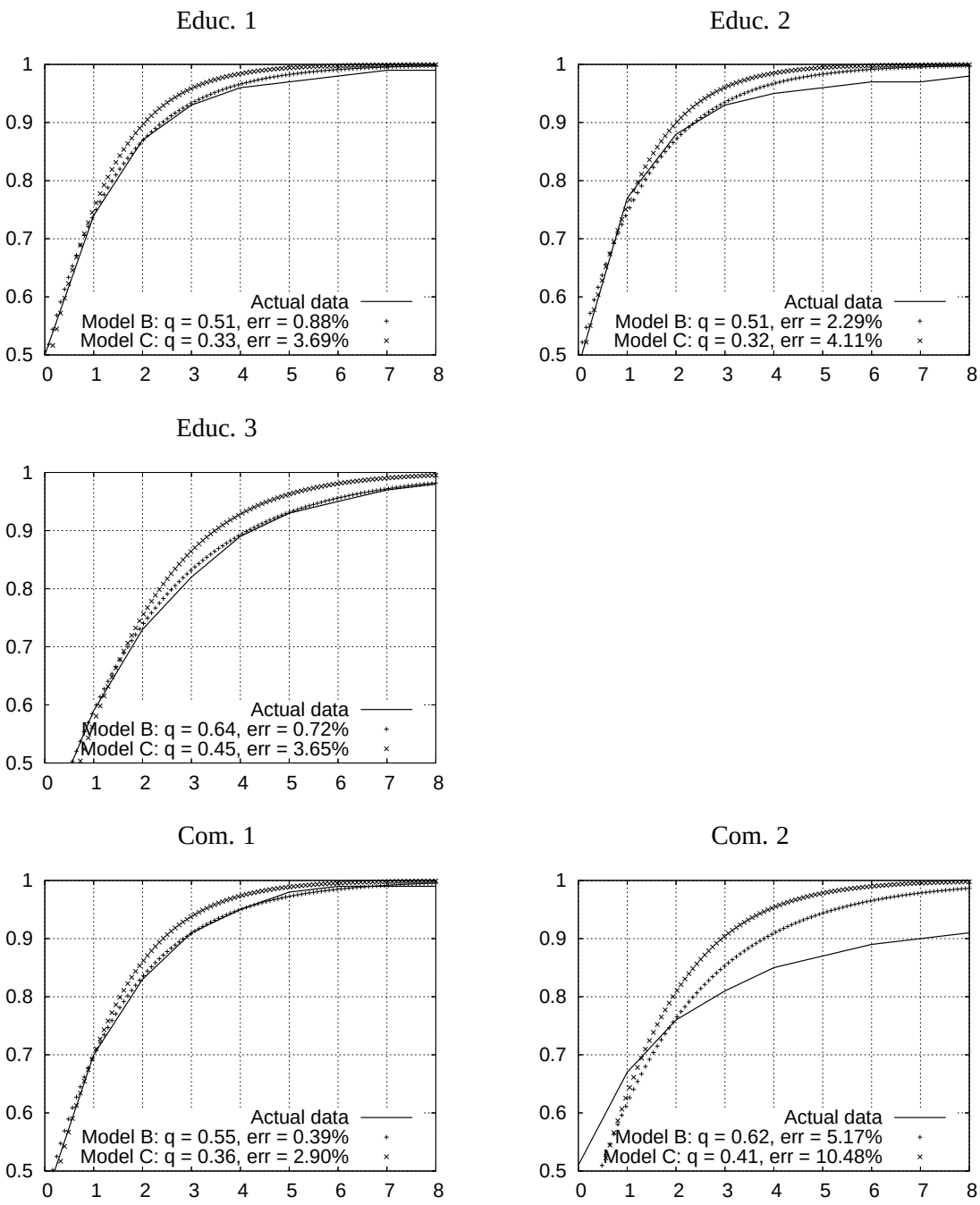


Figure 5.18: Fit of the models to actual data, except for Blogs and non-governmental organizations with Blogs. Model B (back to start level), has smaller errors for most Web sites. The asymptotic standard error for the fit of this model is 5% in the worst case, and consistently less than 3% for all the other cases. Note that we have zoomed in into the top portion of the graph (continues on the next page).

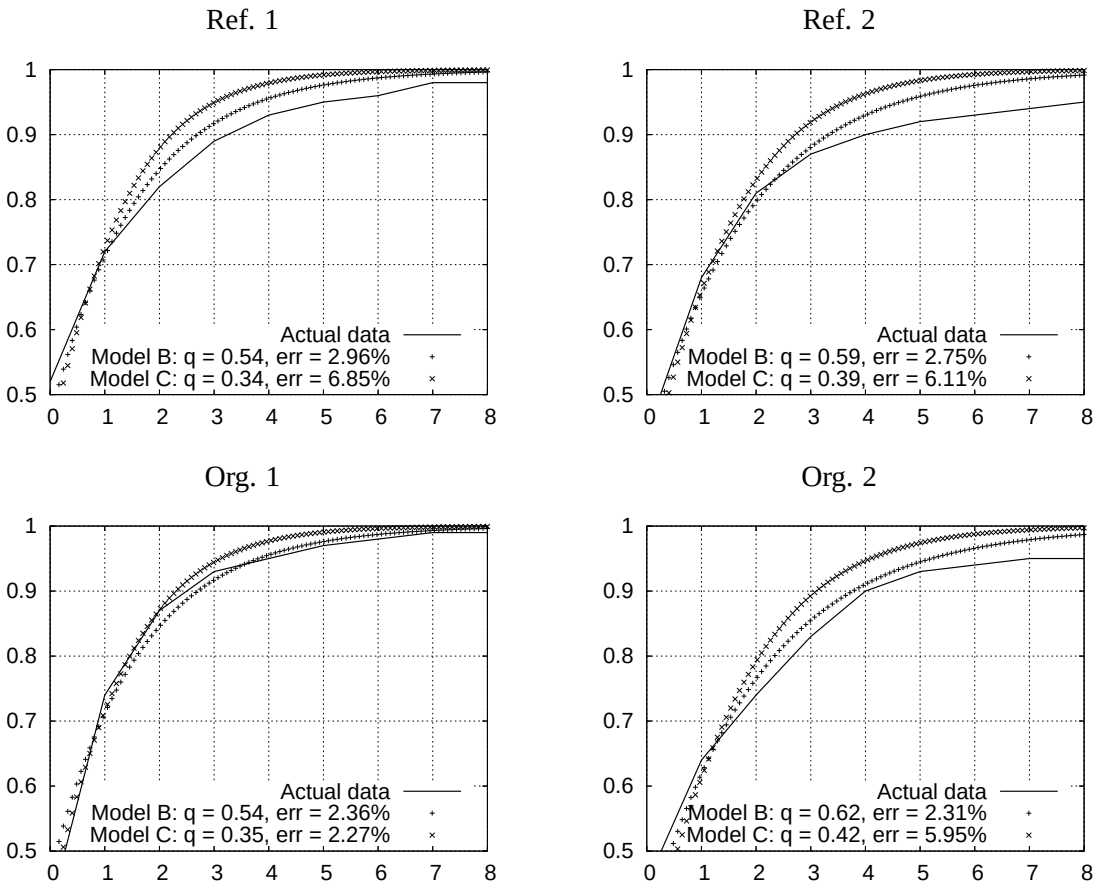


Figure 5.18 (cont.)

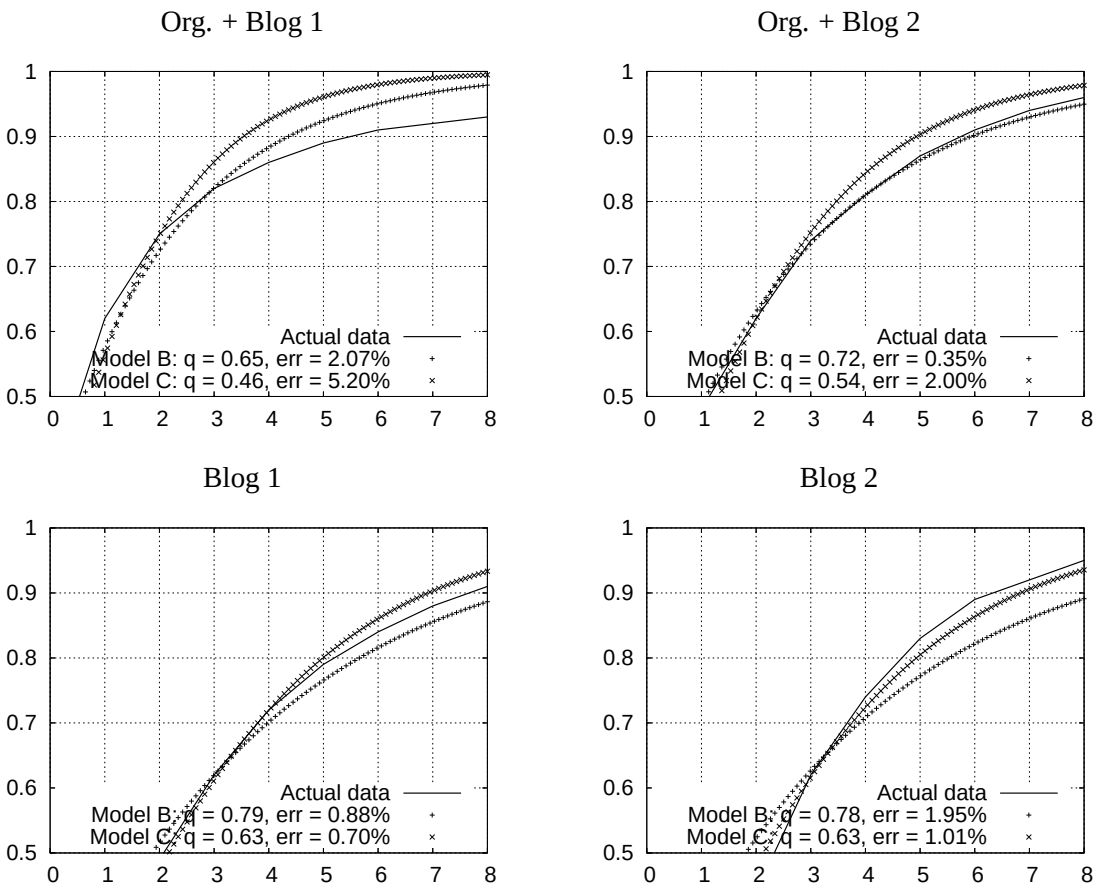


Figure 5.19: Fit of the models to actual data in the case of Blogs. In this case user sessions tend to go deeper inside the Website because more pages are visited per session, probably because Blog postings tend to be short.

Chapter 6

Proposals for Web Server Cooperation

As the number of publicly available Web pages increases, the problem of keeping a search engine index up-to-date with changes becomes increasingly more difficult [LG99], and it is common to find several pages outdated in current search engines [SS04]. This makes things more difficult for the user who seeks information and affects the image of the search engine, but this also has costs for the Web sites that are misrepresented, if we consider the whole user experience.

If a user searches for a keyword on a search engine, and then chooses a page from the search results that no longer exists, or that contains material that is currently irrelevant for the user's information need, the user will be frustrated with both the search engine *and the Web site* for not finding this information. There is also an opportunity cost related to these visitors: maybe the information they want was moved to another page in the same website and the search engine was not aware of the change. In this case, it would be better for the Web site to inform the search engine of the update.

Web crawlers can use an important amount of network and processor resources from the Web server, especially if they do not follow existing rules of "good behavior" [Kos95]. Web crawlers tend to visit many more pages than humans, and they request them very fast, normally with 10 to 30 seconds between visits; so they are believed to account for at least 16% of the requests [MAR+00]. Many of the requests are to unmodified resources, and can be avoided in certain schemes if the server informs the crawler about which resources have not been modified since its last visit.

Hence, an Web site administrator has incentives to improve the representation of his Web site in the search engine's index and to prevent unnecessary visits from crawlers. The mechanism for accomplishing this is what we call a *cooperation* scheme between the Web server and the Web crawler.

In this chapter we show some existing techniques for cooperation, and we propose new ones; some of the techniques shown here were not designed for this specific purpose but they can be adapted. We also present a discussion about the relative merits of each technique. Finally, we implement one of them in the WIRE crawler.

The next section presents general aspects about cooperation schemes, Section 6.2 presents polling-based schemes, and Section 6.3 presents interruption-based schemes. Section 6.4 shows a relative comparison of costs. Section 6.5 describes how a cooperation scheme was implemented in the WIRE crawler. The last section presents the conclusions and future work.

Portions of this chapter were presented in [Cas03].

6.1 Cooperation schemes

The standard HTTP transaction follows the request-response paradigm: a client requests a page from a Web server, and the Web server responds with the page, as depicted in Figure 6.1. This transaction involves meta-data (information about the page) that is downloaded along with the actual page contents.

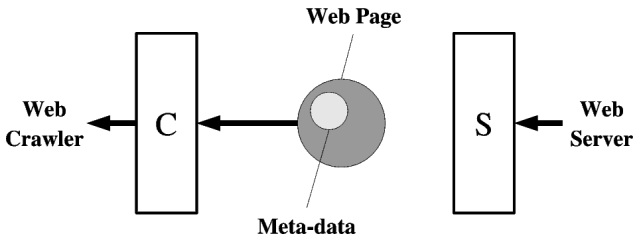


Figure 6.1: Schematic drawing of a normal transaction between a Web crawler ("C" on the left) and a Web server ("S" on the right). The large, dark circle in the middle represents a Web page and the small, light circle represents its meta-data. The arrow from the Web page to the Web crawler indicates that the Web crawler initiates the connection. We use this pictorial representation in the next two figures.

The cooperation schemes we study in this thesis can be divided into two groups: *polling* and *interrupt*.

Polling (or *pull*) schemes: the Web crawler periodically requests data from the Web server based on a scheduling policy. These requests check if a page have been changed, and then download the page.

Interrupt (or *push*) schemes: in this case the Web server begins a transaction with the search engine whenever there is an event. These events can happen when one or multiple pages are updated, based on server policies. The search engines must subscribe to the servers from which they want to receive events. This is similar to the relationship between the main processor and a hardware device (network card, scanner, etc.) in a modern computer.

Note that polling becomes equivalent to an interrupt when the polling period tends to zero; but the usage of resources at both ends of the polling line increase at the same time.

In this thesis, we study several cooperation schemes, which are summarized on Table 6.1.

Transfered data	Polling version	Interrupt version
Meta-data	Serve meta-data of content	Send meta-data of updates
Differences of site	Serve differences of content	Send differences of content
Pages	Serve pages only if modified	Send changed pages
Batches of pages	Serve many pages per request	Send batch update
Site	Serve entire site compressed	Send entire site
Mixed	Filtering interface	Remote agent

Table 6.1: List of cooperation schemes analyzed in this thesis. All of them have two versions: polling (poll) and interrupt (push).

Before we get into the details of each scheme, there are some issues we must mention that are almost independent of the scheme used:

Compression can be used for decreasing transmission cost at the expense of using more processing power on the server side. On the current Web, compression is used for transferring most images –because they are usually stored in a compressed format– but normally it is not applied for textual Web pages. The HTTP/1.0 protocol considers requests of compressed bodies using the `Accept-encoding` header, so compressing Web-pages can be used but it is usually left to the communication between the ISP and the final user. Compression can be used with complete Web pages, bundles of Web pages and their resources, or Web page differences [SS03] (more details in Section 6.2).

Privacy issues arise when the crawler has access to information in the server that was not meant to be public. This may sound strange, but in practice when using a Web crawler it is possible to download files that are linked by mistake or private directories that allow a virtual listing; we have even found complete plain-text password files!. Many Web site administrators mistakenly believe that by not publishing the URL of a Web page they can keep the page private. This is a common practice, so most Web site administrators are very reluctant to provide access for clients to list the contents of directories. Note that if users follow an external link from one of these "private" pages, their browser will inform the referrer to the next Web site, and this referrer could be logged and inspected so it is pointless to try to keep unknown URLs private.

Index update capabilities are very reduced in global-scale Web search engines: constraints in terms of disk space are the most important limitation, so it is not always possible to store a complete copy of the downloaded pages. This can lead to some difficulties; for instance, on a standard inverted index, removing a page or updating a paragraph without having the complete text can be impossible or very time-consuming. Also, in many cases updating batches of pages is preferred to updating single pages to reduce the overall processing cost.

Search engine "spamming" occurs whenever Web site administrators try to get undeserved high rat-

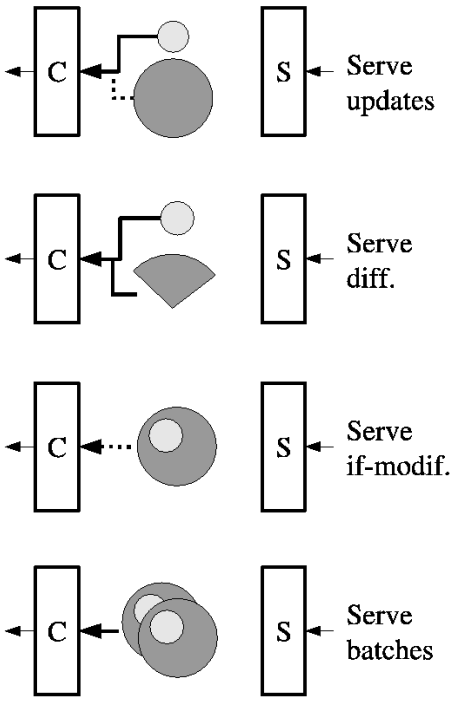


Figure 6.2: Diagrams of polling-based cooperation. The arrow between the Web crawler "C" and the Web server "S" represents who initiates the connection. The small, white circle represents meta-data and the large, gray circle represents the contents of the page.

ings in search engines. Data provided by Web servers cannot be trusted completely, for instance, in terms of self-asserted frequency of updates of the pages or local page importance. Web crawlers used by most search engines are interested only in some of the pages of a Web site, and the decision of which pages must be added to the index should be left to the search engine. In notification schemes, a degree of trust can be established, e.g.: if a Web site sends update notifications, but when pages are inspected by the Web crawler they have not changed, then the search engine can ignore further notifications from that Web site for a period of time.

Structure and HTML markup used in a Web site affects the visibility of its pages by search engine crawlers. Information that is accessible only through forms is, in general, difficult to gather for Web crawlers; this is called the "hidden Web" [RGM01]. Web sites could attract more visitors if they provide a crawler-friendly interface for this data to be indexed by search engines.

6.2 Polling-based cooperation

In all these cases, the Web crawler queries the Web server with certain periodicity; the cooperation schemes discussed in this section are depicted in Figure 6.2.

Serve meta-data of updates. A file containing last-modification data (and probably file size and path) is served. This file can contain a description of many pages on the Web site. In the case of single files, the HTTP protocol provides `HEAD` requests that are responded with meta-data about the requested object. Multiple `HEAD` requests can be pipelined, but this is not as efficient as serving a concise file. Examples: the Distribution and Replication Protocol (DRP) [vHGH+97], the proposal by Brandman *et al.* [BCGMS00] in which files containing information about changes are served, and the proposal by Buzzi [Buz03] that includes information obtained from the access log files. RDF [LS99] also includes the possibility of informing time-related data about the resources.

Finally, the HTTP `Expires:` header presents a way of informing the crawler of the next change in a Web page, but this involves prediction and therefore is not widely used.

Serve differences of content. The Web server provides a series of differences between a base version and newer versions. In the most simple case, the difference is only between the last and the current version. Examples: the HTTP Delta responses proposed by Mogul *et al.* [MDFK97] that use the `Content-encoding` field of HTTP responses, however, a disadvantage is that servers must retain potentially many different versions of their Web pages and that it can only be used for Web pages that have already been visited. Another disadvantage is that re-visits account for a small fraction of the total downloads, so this cannot be used for all pages.

Savant and Suel [SS03] study the possibility of delta-encoding Web pages with respect to other similar Web pages in the same server that are already in a client's cache, which gives a lower compression ratio but imposes less workload on the Web server and can be used for a larger fraction of the accesses. In their approach these differences are also compressed. See the survey by Suel and Memon [SM02] for a summary of techniques for delta compression for remote synchronization of files.

An example of delta compression being used in practice is that source code for popular free software can be updated using the `patch` [WEDM00] program, with servers providing differences between the original version and the new version. For Web sites, differences in terms of structural changes in the links, can be encoded using tables as in *WHOWEDA* [BMN03].

Serve pages only if modified. The Web crawler can avoid downloading a file if the file has not been modified. Examples: in HTTP/1.0 this is done using a date the crawler provides (usually the last visit to the same page) in an `If-Modified-Since` header; these headers are used only by a minority of crawlers [BCGMS00], although they are supported by most Web servers. In HTTP/1.1, an *entity-tag* (E-Tag) can be provided: this is a hash function of the text of the document.

Serve multiple pages on one request. The overhead arising from multiple TCP connections can be avoided by requesting a series of pages in the same connection. Example: this is usual for modern Web browsers,

and is implemented using the `Connection` header with the `keep-alive` value in HTTP/1.1 [FGM+99]; in this case, pipelining of the requests can also be used. With pipelining, the user agent requests multiple pages without waiting for a response, and then receives multiple responses in the same order as they were requested.

Serve entire site in a large file. This is suitable only if the Web changes occur in many pages, or if the Web site is composed of many small files. Example: typically, Linux distributions are distributed in whole CD- or DVD-sized disk images and not on a file-by-file basis.

Filtering interfaces. This is a standard method for answering queries from the crawler. The typical query a Web crawler could ask is "give me all the pages that have changed since this date". A more powerful filtering interface could also include requests for differences, or querying about other characteristics such as page sizes or local importance. Examples: DASL [RRDB02] for searching Web servers, RSYNC [TP03] for mirroring content, and the Common Index Protocol (CIP) [AM99]. In WebDAV [web04a], the `PROPFIND` method allows to query for properties of a document, and a proposed extension `BPROPFIND` for querying about groups of documents. A generic filtering interface could also be implemented using a Web Service [CCMW01].

6.3 Interruption-based cooperation

In all these cases, the Web server sends data to the Web server whenever there is an update (page change, deletion or new page). The Web crawlers must subscribe with the Web server to start receiving notifications, and when they receive them, they can choose to process, enqueue or ignore them.

The following cooperation schemes are depicted in Figure 6.3:

Send meta-data of updates. The notification includes only meta-data about the update, as a minimum, the URL of the resource and a timestamp of the event would be required. Examples: the Keryx [BK97] notification service, developed during the apogee of push-based content delivery, and the Fresh Flow proposal for Web server cooperation [GC01].

Send differences of content. Whenever an event happens, the Web server sends a file containing the difference between the last version and the current one (if the changes are major, the Web server may send the entire file). This exploits the fact that most page changes are minor, e.g.: after one year, 50% of changed pages have changed less than 5% [NCO04]. Example: CTM [Kam03]: in this case, differences on a repository of source code are sent to interested users by electronic mail and the receivers automatically execute

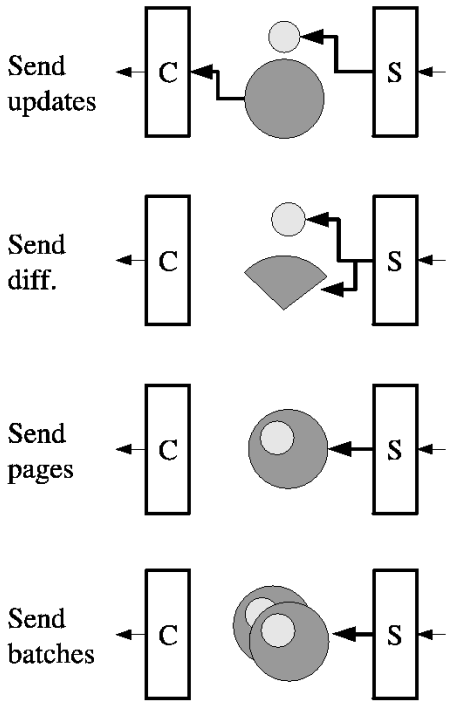


Figure 6.3: Diagrams of interruption-based schemes of cooperation. Connections are initiated by the server-side (right) and handled by the crawler (left).

the `patch` [WEDM00] program to apply the update to the local copy. In CTM, every 50 relative "deltas", a complete base set is sent.

Send changed pages. The Web server sends the complete text of each updated or new page when the modifications are made. Examples: this was typical in push technologies [KNL98], and was implemented in services like "Marimba Castanet" and "Pointcast" in the early days of the Web. Currently, it is being used in wireless devices [Cha02].

Send multi-pages updates. The Web server sends batches of modified pages according to some schedule. This can be useful if the updates are regular and involve several pages; for example, in the Web site of a daily or weekly newspaper. This is the idea behind the protocol used for keeping pages updated in mobile devices used by AvantGO [ava04], in which a single device receives a compressed bundle of pages from several Web sites.

Send entire site. The Web server sends the entire site. This is useful, for instance, for uploading an entire Web site when the site is publicly available for the first time, or if there is a major modification involving most of the pages, as an extension of the previous scheme.

Strategy	Network cost	Processing (server)	Processing (crawler)	Freshness improvement
Send meta-data of updates	+	+		High
Send differences of content	– –	+ +	+	High
Send changed pages	–	+		High
Send batch update	+	+		High
Send entire site	+ +	+		High
Remote agent	– –	+ +	–	High
Serve meta–data of content	+	+		Normal
Serve differences of content	– –	+ +	+	Normal
Serve pages only if modified	–			Normal
Serve many pages in one request	–			Normal
Serve entire site compressed	+ +	+		Normal
Filtering interface	– –	+ +	–	High

Table 6.2: Relative costs of server cooperation schemes discussed, against the base case where no cooperation exists: "+" means more cost, "−" means less cost. The last column is the expected improvement in freshness for the search engine's collection

Remote agent. The Web server executes software provided by the search engine; this software includes instructions to identify important pages and to detect changes in the pages that are relevant for the search engine. Important pages can be identified based on local connectivity, textual information, or log file analysis. Changes can be detected using a custom algorithm that varies depending on the search engine's characteristics. When there is a change, the agent sends back some data to the search engine. This can be meta-data, complete pages, or differences. This is a typical application for a mobile agent [LO99], and the cooperation can be taken one step further, as in some cases the agent could help the search engine by fetching data from "near" servers, as proposed by Theilmann and Rothermel [TR99].

6.4 Cost analysis

6.4.1 Costs for the Web server

We will consider unitary (per-page) costs and benefits:

- b: Benefit from a user viewing one page, from advertising revenues or from other sources.
- c_n: Network cost for serving one page, i.e.: bandwidth cost.

- c_p: Processing cost for serving one page, i.e.: servers cost.

A simple observation is that we should have (in theory) $b \geq c_n + c_p$, otherwise, the Web site would not be able to pay the operation costs. However, we should note that some Web sites can be financed by revenues from other sources. Another observation is that in general processing capacity is cheaper than network connectivity, so in general $c_n > c_p$.

Estimates: We cannot measure these quantities, but we can make some estimates: as of 2004, the cost per page-view of an advertising campaign on the Web is about US$ 0.05, so it is likely that $b \geq 0.05$. On the other end, having a Web server costs about US$ 10 for 5 gigabits of traffic, or 625Mb; if each page including images is 40Kb on average, this is enough for 15,000 page-views; notice that network bandwidth is usually "overbooked" in popular virtual servers, probably by a factor or 2 or 3, so an estimate of the cost is: $c_n + c_p \leq 0.002$. Serving pages to a Web crawler is cheaper because the Web crawler does not download the images.

This is a very rough estimate, but it reveals something interesting: if we only account for Web server usage, serving a Web page costs at most $1/25$ of the benefit, and this is probably the biggest cause of the huge success of the World Wide Web as a platform for publishing information. The main source of cost when keeping a large Web site is not the Web hosting, but rather the cost of producing and maintaining its contents.

In Table 6.2 we provide a rough estimation of relative costs associated with these cooperation schemes. Network bandwidth savings are the product of not dealing with unnecessary requests from the crawlers, and costs, from sending more than is necessary. Processing costs involve keeping meta-data, calculating differences, or more complex processing. Benefits arise from increased freshness on the Web search engine, and are higher if an interruption (*push*) is involved.

Which is the best strategy for the Web server? This will depend on the price the Web server is willing to pay; if this is minor, then using server software that correctly implements HTTP/1.1 is the best option. If the price is moderate, then serving and sending meta-data of updates is the best option. If the server wishes to invest more resources, it can benefit from providing content differences and/or a filtering interface for the crawlers.

6.4.2 Costs for the crawler

The main costs for the crawler for each page are:

- Polling or receiving an interrupt. We will consider that in terms of network and processing, generating a connection or handling an interrupt are the same.

- Downloading the page.
- Processing the downloaded page.

An estimation of these costs is due to Craswell *et al.* [CCHM04], and it is close to US $1.5 Million for an entire crawl of the Web, or about US$ 0.002 per page. Remarkably, this is exactly our estimation of the costs for the Web server, and both figures were obtained independently.

The freshness of the repository is higher in interrupt-based strategies, as there is no significant delay between the server update and the search engine syncing of the page. The costs for the crawler are summarized in Table 6.2. Network cost for the crawler is the same as for the server, as each transfer in these schemes involves one crawler and one server.

However, interrupt-based strategies have to be implemented carefully, because if too many Web servers are sending interrupts to the Web crawler at whatever time they choose, then the search engine risks being overloaded by these requests, loosing control over the crawling process. It is likely that interrupt-based strategies can only be deployed for a small group of Web sites.

Which is the best strategy for the crawler? A remote agent or filtering interface can help to distribute the workload of the search engine, especially if servers cooperate in pre-processing documents or in generating partial indexes. The remote agent can be used for the more important Web sites (such as news sources) if the crawler can process interrupts as they arrive, probably by keeping a small index for the most changing data.

An extreme case of using an agent could be when the Web server generates the (partial) inverted index and then sends it to the search engine, which only needs to perform a merge. In this case, the crawling problem is simplified, and is transformed into polling or pushing of indexes.

6.4.3 Overall cost

It is very important to consider that not all Web servers are equal, and the distribution of "quality" on the Web is, by all measures, very skewed: most of the important Web pages are on a few Web servers, as shown in Figure 8.11 (page 133). The best servers are not necessarily the larger ones, in Figure 6.4 we compare average Pagerank with site size and find no correlation.

By inspecting Table 6.2, a noticeable fact is that the schemes that do not require extra cost for the Web server are already implemented in HTTP (`keep-alive` and `if-modified-since` features).

It is clear that if the request–response paradigm is enforced strictly, the scheme that can provide the best benefits in terms of freshness is a filtering interface. Pushing or pulling differences of content are probably the most balanced schemes, because the server gains in less bandwidth usage. These schemes are more useful if many clients can benefit from differences: not only the Web crawlers of search engines, but also the general public using enabled browsers or cache services.

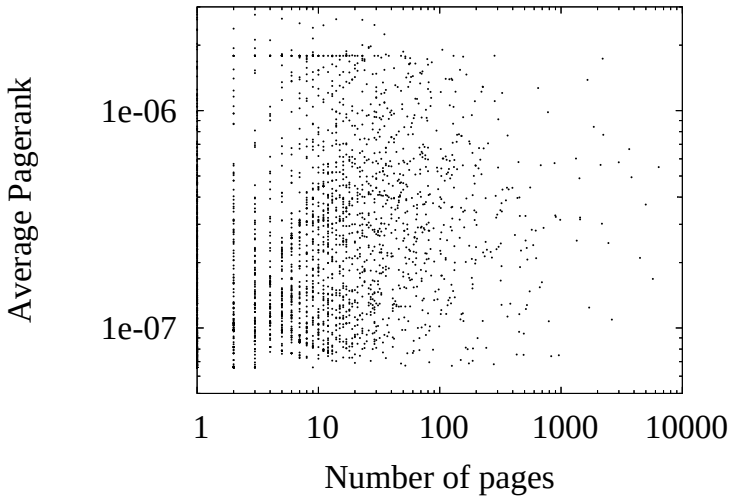


Figure 6.4: Average Pagerank versus number of pages, for a sample of 25,000 Web sites in the Chilean Web. The size of a Web site does not seems to be correlated with the quality of its pages according to this metric.

6.5 Implementation of a cooperation scheme in the WIRE crawler

The WIRE crawler supports a cooperation scheme based on serving meta-data of updates. The Web server provides an XML file containing a description of the documents provided by the Web server.

We wanted to use publicly-available XML name spaces to conform to existent definitions. We used the following XML applications (languages):

RSS RDF Site Summary, also called "Rich Site Summary" or "Really Simple Syndication" is an extension of RDF. It was conceived as a simplification of RDF to be able to aggregate multiple Web sites in a single interface for the "My Netscape" service [Lib00]. Nowadays, it is widely used by news sources to provide a short list of the latest news histories to be used by news aggregators.

DC The Dublin Core is a simple set of metadata elements to describe electronic documents. It is designed to provide a minimal set of descriptive elements for Web pages [dc04], including date, type, format, copyright status, etc.

The Web server periodically generates a file `robots.rdf`, located at the root of the Web site, containing the last-modification time of all the URLs in its public space. An example file is shown in Figure 6.5.

Currently the file contains only the URL and the last-modification time, which is the information the WIRE crawler can use, but in the future it could include more information such as page size, format, number of accesses, etc.

```
<?xml version="1.0"?>
<rdf:rdf
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns"
 xmlns:dc="http://purl.org/dc/elements/1.1/"
 xmlns="http://purl.org/rss/1.0/">
   <channel rdf:about="http://www.example.com/">
     <items><rdf:Seq>
       <rdf:li rdf:resource="http://www.example.com/one.html"/>
       <rdf:li rdf:resource="http://www.example.com/two.html"/>
     </rdf:Seq></items>
   </channel>
  <item rdf:about="http://www.example.com/one.html">
    <dc:modified>2004-10-01T12:05+02:00</dc:modified>
  </item>
  <item rdf:about="http://www.example.com/one.html">
    <dc:modified>2004-11-21T09:01+02:00</dc:modified>
  </item>
</rdf:rdf>
```

Figure 6.5: Example of a `robots.rdf` file.

The implementation of this scheme has two parts: a server-side script that generates the file, and an interpreter in the Web crawler.

6.5.1 Web server implementation

On the server-side, a Perl script is provided for generating the RSS file. This script requires two parameters:

- The root directory of pages in the Web site.
- The base URL of the home page of the Web site.

Optional parameters are:

- Patterns to include pages. The default is to include all pages that include the substring "`.htm`".
- Patterns to reject pages, to exclude private directories.
- The maximum number of pages to include in the file.

A typical example of usage is:

```
% wire-rss-generate --docroot /home/httpd/html --base http://www.example.com/
    > /home/httpd/html/robots.rdf
```

This program is executed periodically using `crontab`. The frequency of updates should be related to the frequency of update of the Web site, but generating the file on a daily basis seems acceptable.

The `.rdf` extension was chosen because it is usually a registered file type in the Apache Web servers, and therefore the response included the corresponding `application/xml+rdf` content-type.

6.5.2 Web crawler implementation

The WIRE crawler handles the download of this file similarly to the `robots.txt` file [Kos95]. A setting in the crawler configuration file controls the frequency at which this file is checked for changes.

The crawler parses the `robots.rdf` file and for each item found, it checks the last modification time of the file. This timestamp is entered into the equation for estimating the probability of the object being outdated, as shown in Section 2.4 (page 27).

6.5.3 Testing

We tested our implementation to gather insights about how it works in practice. This is a first step that is necessary to learn about the system before a large-scale studied is carried.

We tested our implementation over a month with a Web site containing medical information; this Web site has 249 pages. Issuing a `HEAD` request for each page, just to check for the last-modification timestamp, generates 108,777 bytes of traffic, with an average of 434 bytes per page. It takes about 5 minutes to sequentially make all of these requests, even if we do not wait between them.

When using cooperation, the generated `robots.rdf` file is about 61,504 bytes, with an average is 247 bytes per page; this is more than 40% of savings in bandwidth, and with an important advantage: everything is done in just one request in less than 5 seconds.

An unexpected benefit of this implementation is that Web pages are usually discovered slowly, level by level as the crawler must parse the Web pages to find links. When the page listing is found on a single file, the Web crawler can download the entire site in just one session, without having to parse data to discover pages.

We learned that if the updates involve only just one page, then if the `robots.rdf` file is too large this scheme can waste network resources because the complete file with the metadata is transfered each time. The `robots.rdf` could be divided into several parts for very large Web sites, and these parts could be chosen in

such a way that the most important pages are found in a small part –for instance, by dividing the Web site by levels.

We also learned that for a good scheduling using a file with meta-data, the important search engine parameter is not the minimum re-visiting period, but the maximum acceptable outdated probability; otherwise, bandwidth can be wasted by requesting the meta-data file more often that it is necessary, especially if only a few Web sites are involved and they do not change too often.

6.6 Conclusions

How probable is the wide adoption of a cooperation strategy? The basic HTTP protocol is not completely implemented in the same way across different servers, and the minimum-common-denominator is quite poor in terms of functionalities, as explained in Appendix A.

On the other hand, Web site administrators can cooperate if it is not too costly and means an important benefit. This benefit should come mostly in terms of being better represented on the Web search engines. We consider that the reductions on load for the Web server are probably not enough by themselves to justify the adoption of a cooperation strategy.

There are also some specific applications that can use a cooperation strategy: most general search engines offer specialized (paid) search services for specific Web sites. These search services could be improved if software for cooperating with the search service is installed in the server-side.

With the emergence of Web services, filtering strategies could be an interesting possibility for the near future, as they can help crawlers and other autonomous agents to interact with Web servers at a more meaningful level.

Chapter 7

Our Crawler Implementation

We developed a Web crawler that implements the crawling model and architecture presented in Chapter 3, and supports the scheduling algorithms presented in Chapter 4. This chapter presents the implementation of the Web crawler in some detail. Source code and technical documentation, including a user manual are available at `http://www.cwr.cl/projects/WIRE/`.

The rest of this chapter is organized as follows: section 7.1 presents the programming environment used. Section 7.2 details the main programs, section 7.3 the main data structures and section 7.4 the configuration variables.

7.1 Programming environment and dependencies

The programming language used was C for most of the application. We also used C++ to take advantage of the C++ Standard Template Library to shorten development time; however, we did not use the STL for the critical parts of our application (e.g.: we developed a specialized implementation of a hash table for storing URLs). The crawler currently has approximately $25,000$ lines of code.

For building the crawler, we used the following software packages:

ADNS [Jac02] Asynchronous Domain Name System resolver, replaces the standard DNS resolver interface with non-blocking calls, so multiple host names can be searched simultaneously. We used ADNS in the "harvester" program.

LibXML2 [lib02] An XML parser developed in C for the Gnome project. It is very portable, and it is also an efficient and very complete specification of the XPath language. We used XPath for the configuration file of the crawler, and for parsing the "robots.rdf" file used for Web server cooperation during the crawl, as shown in Chapter 6.

We made extensive use of the `gprof` utility to improve the speed of the application.

7.2 Programs

In this section, we will present the four main programs: manager, harvester, gatherer and seeder. The four programs are run in cycles during the crawler's execution, as shown in Figure 3.8.

7.2.1 Manager: long-term scheduling

The "manager" program generates the list of K URLs to be downloaded in this cycle (we used $K = 100,000$ pages by default). The procedure for generating this list is outlined below.

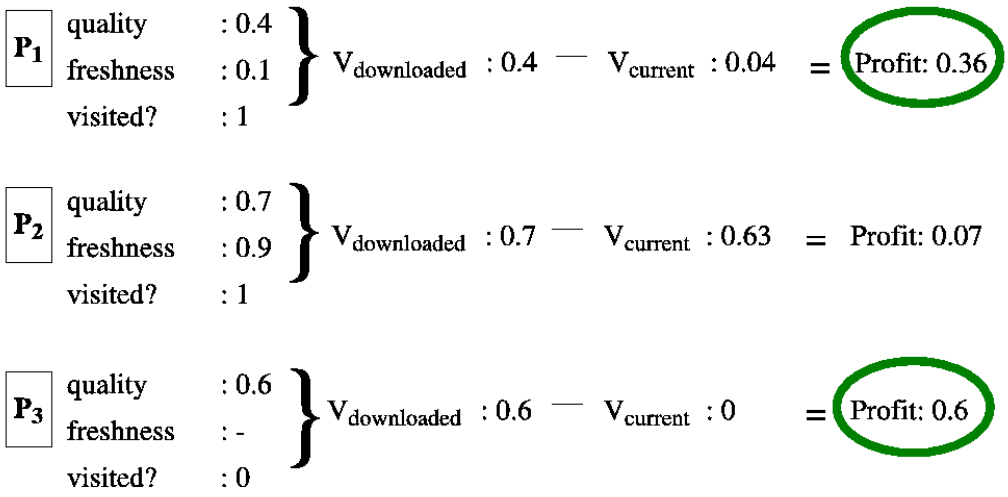


Figure 7.1: Operation of the manager program with $K = 2$. The two pages with the highest expected profit are assigned to this batch.

The current value of a page is IntrinsicQuality$(p) \times Pr($Freshness$(p) = 1) \times$ RepresentationalQuality(p), where RepresentationalQuality(p) equals 1 if the page has been visited, 0 if not. The value of the downloaded page is IntrinsicQuality$(p) \times 1 \times 1$. In Figure 7.1, the manager should select pages P_1 and P_3 for this cycle.

1. Filter out pages that were downloaded too recently In the configuration file, a criteria for the maximum frequency of re-visits to pages can be stated (e.g.: no more than once a day or once a week). This criteria is used to avoid accessing only a few elements of the collection, and is based on the observations by Cho and Garcia-Molina [CGM03a].

2. Estimate the intrinsic value of Web pages The manager program calculates the value of all the Web pages in the collection according to a ranking function. The ranking function is specified in the con-

figuration file, and it is a combination of one or several of the following: Pagerank [PBMW98], static hubs and authority scores [Kle99], weighted link rank [Dav03, BYD04], page depth, and a flag indicating if a page is static or dynamic. It can also rank pages according to properties of the Web sites that contain the pages, such as "Siterank" (which is like Pagerank, but calculated over the graph of links between Web sites) or the number of pages that still have not been downloaded from that specific Web site, this is, the strategy presented in Chapter 4.

3. **Estimate the freshness of Web pages** The manager programs estimates $Pr(Freshness_p = 1)$ for all pages that have been visited, using the information collected from past visits and the formulas presented in Section 2.4 (page 27).

4. **Estimate the profit of retrieving a Web page** The program considers that the representational quality of a Web page is either 0 (page not downloaded) or 1 (page downloaded). Then it uses the formula given in Section 3.4 (page 45) with $\alpha = \beta = \gamma = 1$ to obtain the profit, in terms of the value of the index, obtained by downloading the given page. This is high, e.g.: if the intrinsic value of the page is high, and the page copy is not expected to be fresh, so important pages are crawled more often.

5. **Extract top K pages according to expected profit** Or less than K pages if there are fewer URLs available. Pages are selected according to how much their value in the index will increase if they are downloaded now.

An hypothetical scenario for the manager program with $K = 2$ is depicted in Figure 7.1. The manager objective is to maximize the profit in each cycle.

For parallelization, the batch of pages generated by the manager is stored in a series of files that include all the URLs and metadata of the required Web pages and Web sites. It is a closed, independent unit of data that can be copied to a different machine for distributed crawling, as it includes all the information the harvester needs. Several batches of pages can be generated during the same cycle by taking more URLs from the top of the list.

7.2.2 Harvester: short-term scheduling

The "harvester" programs receives a list of K URLs and attempts to download them from the Web.

The politeness policy chosen is to never open more than one simultaneous connection to a Website, and to wait a configurable amount of seconds between accesses (default 15). For the larger Websites, over a certain quantity of pages (default 100), the waiting time is reduced (to a default of 5 seconds). This is because by the end of a large crawl only a few Web sites remain active, and the waiting time generates inefficiencies in the process that are studied in Chapter 4.

As shown in Figure 7.2, the harvester maintains a queue for each Web site. At a given time, pages are being transferred from some Web sites, while other Web sites are idle to enforce the politeness policy. This is implemented using a priority queue in which Web sites are inserted according to a timestamp for their next visit.

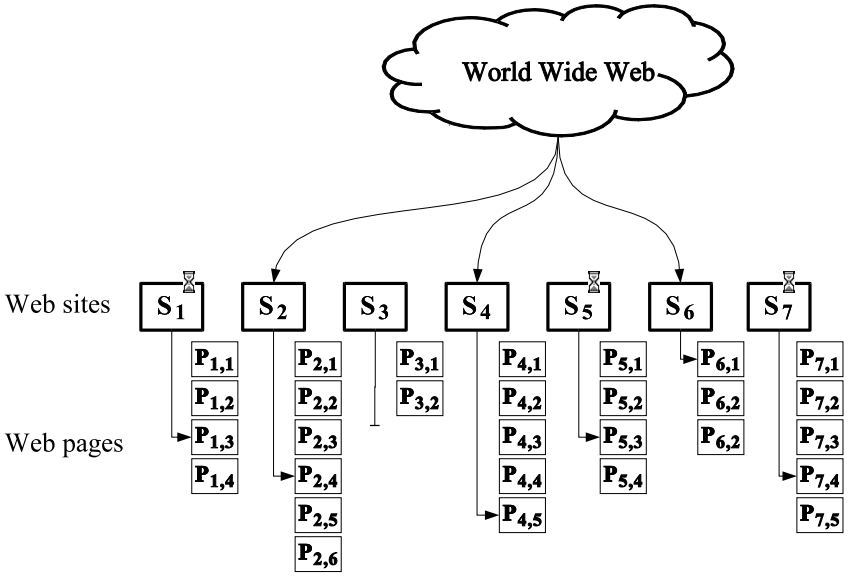


Figure 7.2: Operation of the harvester program. This program creates a queue for each Web site and opens one connection to each active Web site (sites 2, 4, and 6). Some Web sites are "idle", because they have transfered pages too recently (sites 1, 5, and 7) or because they have exhausted all of their pages for this batch (3).

Our first implementation used Linux threads [Fal97] and did blocking I/O on each thread. It worked well, but was not able to go over 500 threads even in PCs with processors of 1GHz and 1GB of RAM. It seems that entire thread system was designed for only a few threads at the same time, not for higher degrees of parallelization.

Our current implementation uses a single thread with non-blocking I/O over an array of sockets. The `poll()` system call is used to check for activity in the sockets. This is much harder to implement than the multi-threaded version, as in practical terms it involves programming context switches explicitly, but the performance was much better, allowing us to download from over 1000 Web sites at the same time with a very lightweight process.

The output of the harvester is a series of files containing the downloaded pages and metadata found (e.g.: server response codes, document lengths, connection speeds, etc.). The response headers are parsed to obtain metadata, but the pages themselves are not parsed at this step.

7.2.3 Gatherer: parsing of pages

The "gatherer" program receives the raw Web pages downloaded by the harvester and parses them. In the current implementation, only `text/plain` and `text/html` pages are accepted by the harvester, so these are the only MIME types the gatherer has to deal with.

The parsing of HTML pages is done using an events-oriented parser. An events-oriented parser (such as SAX [Meg04] for XML) does not build an structured representation of the documents: it just generates function calls whenever certain conditions are met, as shown in Figure 7.3. We found that a substantial amount of pages were not well-formed (e.g.: tags were not balanced), so the parser must be very tolerant to malformed markup.

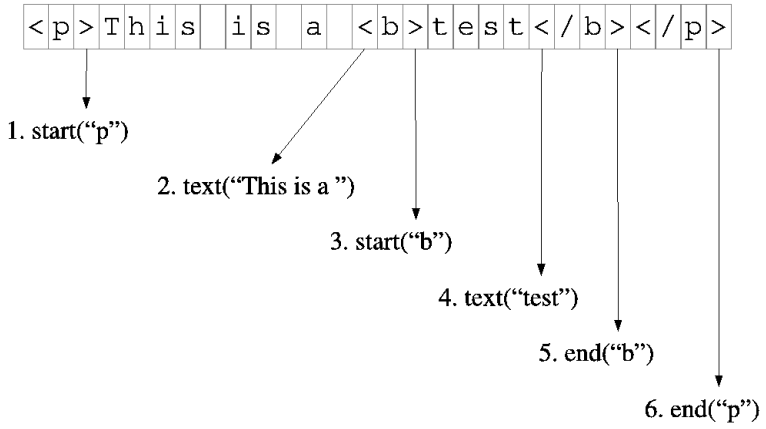


Figure 7.3: Events-oriented parsing of HTML data, showing the functions that are called while scanning the document.

During the parsing, URLs are detected and added to a list that is passed to the "seeder" program. At this point, exact duplicates are detected based on the page contents, and links from pages found to be duplicates are ignored to preserve bandwidth, as the prevalence of duplicates on the Web is very high [BBDH00].

The parser does not remove all HTML tags. It cleans superfluous tags and leaves only document structure, logical formatting, and physical formatting such as bold or italics. Information about colors, backgrounds, font families, cell widths and most of the visual formatting markup is discarded. The resulting file sizes are typically 30% of the original size and retain most of the information needed for indexing. The list of HTML tags are kept or removed is configurable by the user.

7.2.4 Seeder: URL resolver

The "seeder" program receives a list of URLs found by the gatherer, and adds some of them to the collection, according to a criteria given in the configuration file. This criteria includes patterns for accepting, rejecting,

and transforming URLs.

Accept Patterns for accepting URLs include domain name and file name patterns. The domain name patterns are given as suffixes (e.g.: `.cl`, `.uchile.cl`, etc.) and the file name patterns are given as file extensions. In the later case, accepted URLs can be enqueued for download, or they can be just logged on a file, which is the current case for images and multimedia files.

Reject Patterns for rejecting URLs include substrings that appear on the parameters of known Web applications (e.g. `login`, `logout`, `register`, etc.) that lead to URLs which are not relevant for a search engine. In practice, this manually-generated list of patterns produces significant savings in terms of requests for pages with no useful information.

Transform To avoid duplicates from session ids, which are discussed in Section A.6.1 (page 163), we detect known session-id variables and remove them from the URLs. Log file analysis tools can detect requests coming from a Web crawler using the "user-agent" request header that is provided, so this should not harm Web server statistics.

The seeder also processes all the "robots.txt" and "robots.rdf" files that are found, to extract URLs and patterns:

robots.txt This file contains directories that should not be downloaded from the Web site [Kos96]. These directories are added to the patterns for rejecting URLs in a per-site basis.

robots.rdf This file contains paths to documents in the Web site, including their last-modification times. It is used for server cooperation, as explained on Chapter 6.

The seeder also recognizes filename extensions for known programming languages used for the Web (e.g. `.php`, `.pl`, `.cfm`, etc.) and mark those URLs as "dynamic pages". Dynamic pages may be given higher or lower scores during long term scheduling.

To initialize the system, before the first batch of pages is generated by the manager, the seeder program is executed with a file providing the starting URLs for the crawl.

7.3 Data structures

7.3.1 Metadata

All the metadata about Web pages and Web sites is stored in files containing fixed-sized records. The records contain all the information about a Web page or Web site except for the URL and the contents of the Web page.

There are two files: one for metadata about Web sites, sorted by site-id, and one for metadata about Web pages, sorted by document-id. Metadata currently stored for a Web page includes information about:

Web page identification Document-id, which is an unique identifier for a Web page, and Site-id, which is an unique identifier for Web sites.

HTTP response headers HTTP response code and returned MIME-type.

Network status Connection speed and latency of the page download.

Freshness Number of visits, time of first and last visit, total number of visits in which a change was detected and total time with no changes. These are the parameters needed to estimate the freshness of a page.

Metadata about page contents Content-length of the original page and of the parsed page, hash function of the contents and original doc-id if the page is found to be a duplicate.

Page scores Pagerank, authority score, hub score, etc. depending on the scheduling policy from the configuration file.

Metadata currently stored for a Web site includes:

Web site identification Site-id.

DNS information IP-address and last-time it was resolved.

Web site statistics Number of documents enqueued/transfered, dynamic/static, erroneous/OK, etc.

Site scores Siterank, sum of Pagerank of its pages, etc. depending on the configuration file.

In both the file with metadata about documents, and the file with metadata about Web sites, the first record is special, as it contains the number of stored records. There is no document with doc-id$=0$ nor Web site with site-id$=0$, so identifier 0 is reserved for error conditions and record for document i is stored at offset $\text{sizeof}(\text{docid}) \times i$.

7.3.2 Page contents

The contents of Web pages are stored in variable-sized records indexed by document-id. Inserts and deletions are handled using a free-space list with first-fit allocation.

This data structure also implements duplicate detection: whenever a new document is stored, a hash function of its contents is calculated. If there is another document with the same hash function and length, the contents of the documents are compared. If they are equal, the document-id of the original document is returned, and the new document is marked as a duplicate.

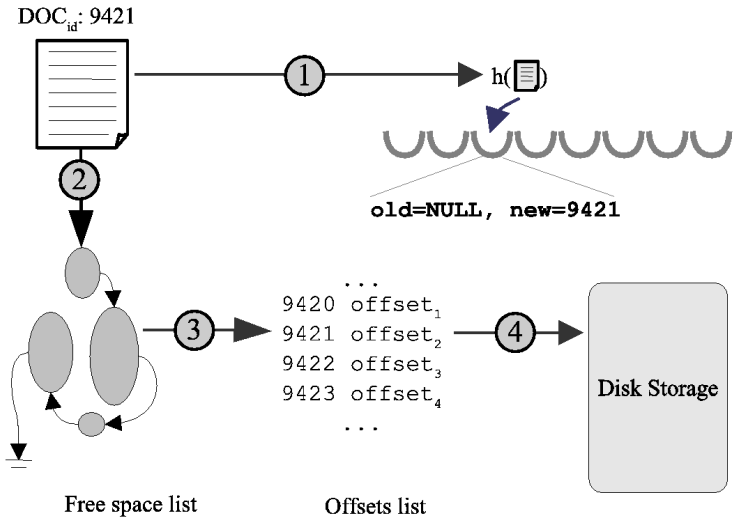


Figure 7.4: Storing the contents of a document requires to check first if the document is a duplicate, then searching for a place in the free-space list, and then writing the document to disk.

The process for storing a document, given its contents and document-id, is depicted in Figure 7.4:

1. The contents of the documents are checked against the content-seen hash table. If they have been already seen, the document is marked as a duplicate and the original doc-id is returned.
2. A free space is searched in the free-space list. This returns a document offset in the disk pointing to an available position with enough free space.
3. This offset is written to the index, and will be the offset for the current document.
4. The document contents are written to the disk at the given offset.

This module requires support to create large files, as for large collections the disk storage grows over 2GB, and the offset cannot be provided in a variable of type "`long`". In Linux, the LFS standard [Jae04] provides offsets of type "`long long`" that are used for disk I/O operations. The usage of continuous, large files for millions of pages, instead of small files, can save a lot of disk seeks, as noted also by Patterson [Pat04].

7.3.3 URLs

The structure that holds the URLs is highly optimized for the most common operations during the crawling process:

- Given the name of a Web site, obtain its site-id.
- Given the site-id of a Web site and a local link, obtain the doc-id for the link.
- Given a full URL, obtain both its site-id and doc-id.

The implementation uses two hash tables: the first for converting Web site names into site-ids, and the second for converting "site-id + path name" to a doc-id. The process for converting a full URL is shown in Figure 7.5.

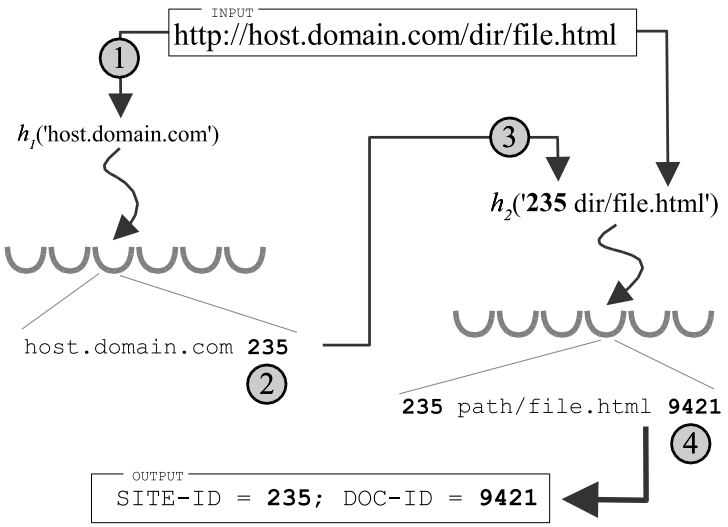


Figure 7.5: For checking a URL: (1) the host name is searched in the hash table of Web site names. The resulting site-id (2) is concatenated with the path and filename (3) to obtain a doc-id (4).

This process is optimized to exploit the locality on Web links, as most of the links found in a page point to other pages co-located in the same Web site.

7.3.4 Link structure

The link structure is stored on disk as an adjacency list of document-ids. This adjacency list is implemented on top of the same data structure used for storing the page contents, except for the duplicate checking. As only the forward adjacency list is stored, the algorithm for calculating Pagerank cannot access efficiently the list of back-links of a page, so it must be programmed to use only forward links. This is not difficult to do, and Algorithm 4 illustrates how to calculate Pagerank without back-links; the same idea is also used for hubs and authorities.

Our link structure does not use compression. The Web graph can be compressed by exploiting the locality of the links, the distribution of the degree of pages, and the fact that several pages share a substantial

Algorithm 4 Calculating Pagerank without back-links

Require: G Web Graph.

Require: q dampening factor, usually $q \approx 0.15$

$N \leftarrow |G|$
for each $p \in G$ **do**
  $\text{Pagerank}_p = \frac{1}{N}$
  $\text{Aux}_p = 0$
end for
while Pagerank not converging **do**
  for each $p \in G$ **do**
    $\Gamma^+(p) \leftarrow$ pages pointed by p
    for each $p' \in \Gamma^+(p)$ **do**
      $\text{Aux}_{p'} = \text{Aux}_{p'} + \frac{\text{Pagerank}_p}{|\Gamma^+(p)|}$
    end for
  end for
  for each $p \in G$ **do**
    $\text{Pagerank}_p = \frac{q}{N} + (1-q)\text{Aux}_p$
    $\text{Aux}_p = 0$
  end for
  Normalize Pagerank: $\sum \text{Pagerank}_p = 1$
end while

portion of their links [SY01]. Using compression, a Web graph can be represented with as few as 3-4 bits per link [BV04].

7.4 Configuration

Configuration of the crawling parameters is done with a XML file. Internally, there is a mapping between XPath expressions (which represent parts of the XML file) and internal variables with native data types such as integer, float or string. When the crawler is executed, these internal variables are filled with the data given in the configuration file.

Table 7.1 shows the main configuration variables with their default values. For a detail of all the configuration variables, see the WIRE documentation at `http://www.cwr.cl/projects/WIRE/doc/`.

XPath expression	Default value	Description
collection/base	/tmp/	Base directory for the crawler
collection/maxdoc	10 Mill.	Maximum number of Web pages.
collection/maxsite	100,000	Maximum number of Web sites.
seeder/max-urls-per-site	25,000	Max. pages to download from each Web site.
seeder/accept/domain-suffixes	.cl	Domain suffixes to accept.
seeder/ext/download/static	*	Extensions to consider as static.
seeder/ext/download/dynamic	*	Extensions to consider as dynamic.
seeder/ext/log/group	*	Extensions of non-html files.
seeder/sessionids	*	Suffixes of known session-id parameters.
manager/maxdepth/dynamic	5	Maximum level to download dynamic pages.
manager/maxdepth/static	15	Maximum level to download static pages.
manager/batch/size	100,000	URLs per batch.
manager/batch/samesite	500	Max. number of URLs from the same site.
manager/score	*	Weights for the different score functions.
manager/minperiod	*	Minimum re-visiting period.
harvester/resolvconf	127.0.0.1	Address of the name server(s).
harvester/blocked-ip	127.0.0.1	IPs that should not be visited.
harvester/nthreads/start	300	Number of simultaneous threads or sockets.
harvester/nthreads/min	10	Minimum number of active sockets.
harvester/timeout/connection	30	Timeout in seconds.
harvester/wait/normal	15	Number of seconds to wait (politeness).
harvester/maxfilesize	400,000	Maximum number of bytes to download.
gatherer/maxstoredsize	300,000	Maximum number of bytes to store.
gatherer/discard	*	HTML tags to discard.
gatherer/keep	*	HTML tags to keep.
gatherer/link	*	HTML tags that contain links.

Table 7.1: Main configuration variables of the Web crawler. Default values marked "*" can be seen at `http://www.cwr.cl/projects/WIRE/doc/`

7.5 Conclusions

This chapter described the implementation of the WIRE crawler, which is based on the crawling model developed for this thesis. The Web as an information repository is very challenging, especially because of its dynamic and open nature; thus, a good Web crawler needs to deal with some aspects of the Web that become visible only while running an extensive crawl, and there are several special cases, as shown in Appendix A.

There are a few public domain crawling programs listed under Section 2.5.1 (page 35). We expect to benchmark our crawler against some of them in the future, but there is still work to do to get the most out of this architecture. The most important task is to design a component for coordinating several instances of the Web crawler running in different machines, or to be able to carry two parts of the process at the same time, such as running the harvester while the gatherer is working on a previous batch. This is necessary because otherwise the bandwidth is not used while parsing the Web pages.

Our first implementation of the Web crawler used a relational database and threads for downloading the Web pages, and the performance was very low. Our current implementation, with the data structures presented in this chapter, is powerful enough for downloading collections in the order of tens of millions of pages in a few days, which is reasonable for the purposes of creating datasets for simulation and analysis, and for testing different strategies. There is plenty of room for enhancements, especially in the routines for manipulating the Web graph –which is currently not compressed, but should be compressed for larger datasets– and for calculating link-based scores.

Also, for scaling to billions of Web pages, some data structures should be analyzed on disk instead of in memory. This development is outside the scope of this thesis, but seems a natural continuation of this work.

The next two chapters present a study of a Web collection and the practical problems found while performing this large crawl.

Chapter 8

Characterization of the Chilean Web

As an application of the crawler implementation presented in Chapter 7, we downloaded and studied the pages under the `.CL` top-level domain.

The WIRE crawler includes a module for generating statistics and reports about its collection. In this chapter, we present several characteristics of the Chilean Web, and we compare some of them with the characteristics of the Greek Web.

8.1 Reports generated by WIRE

The procedure for generating reports has the following steps:

1. Analysis of the metadata that is kept in the data structures described in Section 7.3 (page 118), and generation of statistics as plain text files.
2. Generation of `gnuplot` scripts to generate graphs, and invocation of `gnuplot`.
3. Generation of the report using LaTeX.

This procedure makes maintenance of the reports easier, as the data is separated from the representation. The exact commands for generating reports are detailed in the user manual that is available on-line at `http://www.cwr.cl/projects/WIRE/doc/`.

The generated reports include:

- A report about characteristics of the pages that were downloaded.
- A report about links found in those pages.
- A report about languages.

- A report about Web sites.
- A report about links in the Web site graph.

In this chapter, most of the data tables and graphics (except for pie charts, due to a limitation of the GNU plot program) were generated using the WIRE report generator.

8.2 Collection summary

Table 8.1 summarizes the main characteristics of the collection, which was obtained in May 2004.

Table 8.1: Summary of the characteristics of the studied collection from the Chilean Web.

Downloaded Web pages	3,313,060	
Static	2,191,522	66.15%
Dynamic	1,121,538	33.85%
Unique	3,110,205	93.88%
Duplicates	202,855	6.12%

Downloaded Web sites	49,535
Static pages per site	40.40
Dynamic pages per site	26.73
Pages per site	67.13

We downloaded up to five levels of dynamic pages and also up to 15 levels of static pages. We also limited the crawl to HTML pages, downloading at most 300 Kb of data per URL, with a maximum of 20,000 pages per Web site.

8.3 Web page characteristics

8.3.1 Status code

Figure 8.1 shows the distribution of the HTTP response code. In the figure, we have merged several HTTP response codes for clarity:

- **OK** includes requests that lead to a page transfer: `OK` (200) and `PARTIAL CONTENT` (206) responses.
- **MOVED** includes all the redirects to other pages: `MOVED` (301), `FOUND` (302) and `TEMPORARY REDIRECT` (307).
- **SERVER ERROR** includes all failures on the server side: `INTERNAL SERVER ERROR` (500), `BAD GATEWAY` (502), `UNAVAILABLE` (503), and `NO CONTENT` (204).

- **FORBIDDEN** includes all the requests that are denied by the server: `UNAUTHORIZED` (401), `FORBIDDEN` (403) and `NOT ACCEPTABLE` (406).

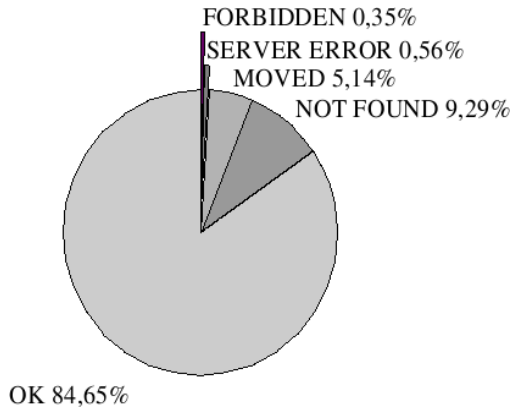


Figure 8.1: Distribution of HTTP response code.

In all our experiments, we usually have had between 75% and 85% of responses leading to an actual transfer of data. From the point of view of Web crawler, the fraction of failed requests is significant, and it should be considered in short-term scheduling strategies when "overbooking" the network connection.

The fraction of broken links, over 9%, is very significative. This means that the Web is dynamic, and quality control on existent Web sites is neither meticulous nor frequent enough.

8.3.2 Content length

To save bandwidth, we downloaded only the first 300 KB of the pages. The center of the distribution of page sizes follows a Zipf's law of parameter -3.54, as shown in Figure 8.2. Close to 300 KB the number of pages looks higher than expected because of the way in which we enforced the download limit.

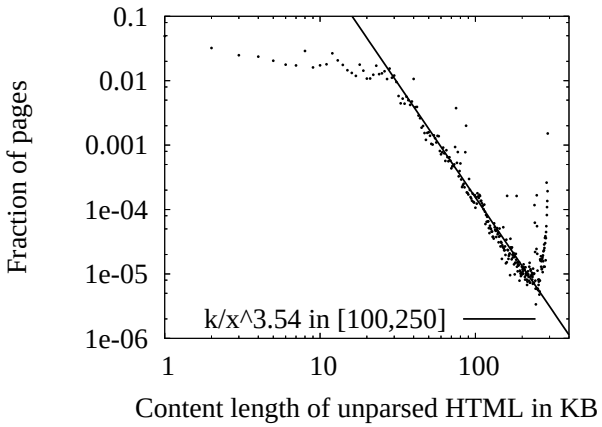


Figure 8.2: Distribution of content length of pages.

We observe that below 12 Kilobytes, there are fewer pages than predicted by the Zipf's law. This is because of a limit of HTML coding: the markup is not designed to be terse and even a short text requires a certain amount of markup. As HTML is used as a presentational language, controlling the formatting attributes of the pages, it generates a significant overhead over text size, especially for complex designs.

For a Web crawler, 300Kb seems to be a safe limit for HTML pages, as there are very few Web pages with more than this amount of data.

8.3.3 Document age

We observed the last-modification date returned by Web servers, and applied the heuristic described in Section A.3.5 (page 160) to discard wrong dates. We found that 83% of the Web sites studied returned valid last-modified dates for their pages. The distribution of page age in terms of months and years is shown in Figure 8.3.

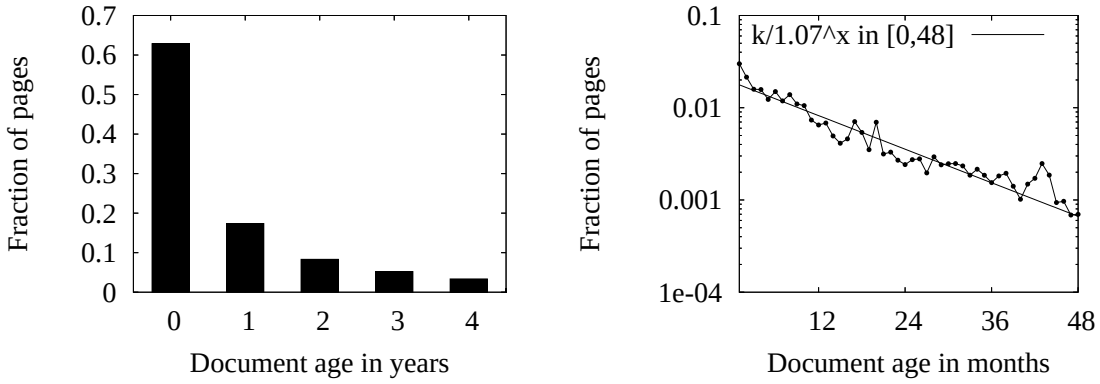


Figure 8.3: Distribution of page age. Note that for the graph of Page age in months the scale is semi-log.

Page changes exhibit an exponential distribution, as seen in the graphic of document age in months. Note that more than 60% of pages have been created or modified in the last year, so the Chilean Web is growing at a fast pace.

8.3.4 Page depth

We limited the browser to download only five levels of dynamic pages, and up to 15 levels of static pages. The distribution of pages by depth is shown in Figure 8.4.

The distribution of static pages follows a shape whose maximum is in the fifth level, but the distribution of dynamic pages tends to grow without bounds. This is because dynamic pages have links to other dynamic pages, as discussed in Chapter 5.

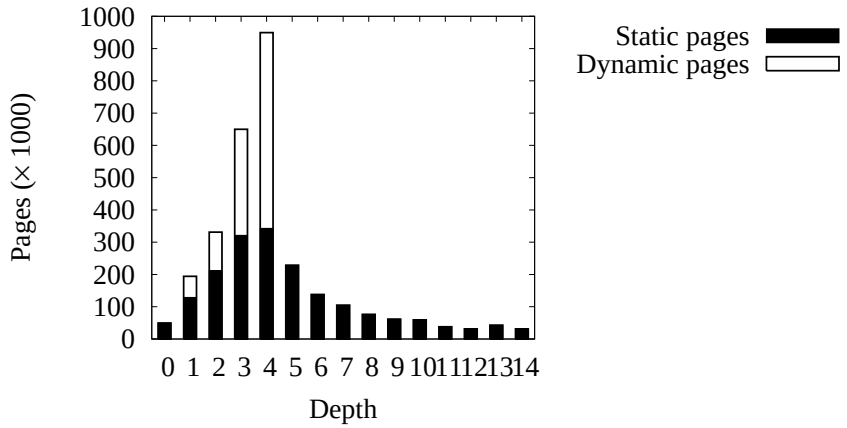


Figure 8.4: Distribution of pages at different depths.

8.3.5 Languages

We took a sample of 5,000 pages, and analyzed their contents to compare their word lists against a series of lists of stop words in several languages on the Chilean Web. We found about 71% of the pages in Spanish, and 27% in English. Other languages appeared with much less frequency.

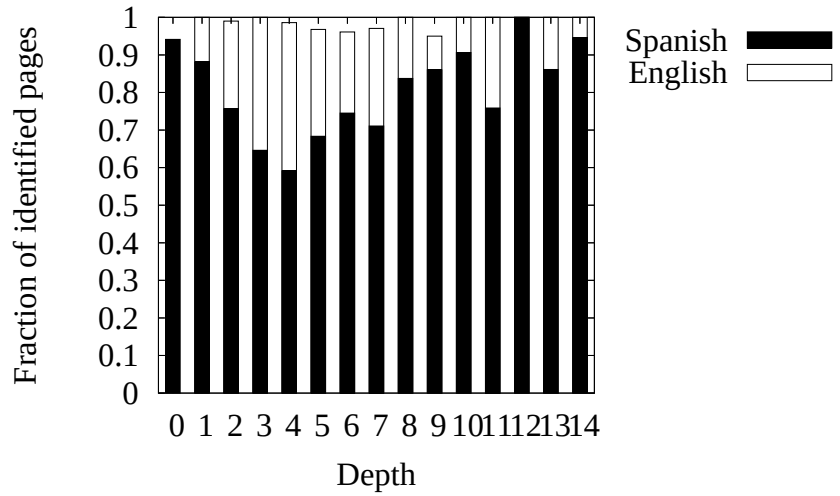


Figure 8.5: Distribution of languages by page depth.

There are large variations in this distribution if we check Web sites at different levels, as shown in Figure 8.5. For instance, over 90% of home pages are mostly in Spanish, but this figure goes as low as 60% if we take pages at depth 5, as shown in Figure 8.5. By inspecting a sample of the English pages at deeper levels, we found that their are mostly mirrors of Web sites such as the Linux Documentation Project, or the TuCows shareware repository.

8.3.6 Dynamic pages

About 34% of the pages we downloaded were dynamically generated. The most used application was PHP [php04], followed by ASP [asp04] and pages generated using Java (`.jhtml` and `.jsp`). The distribution is shown in Figure 8.6.

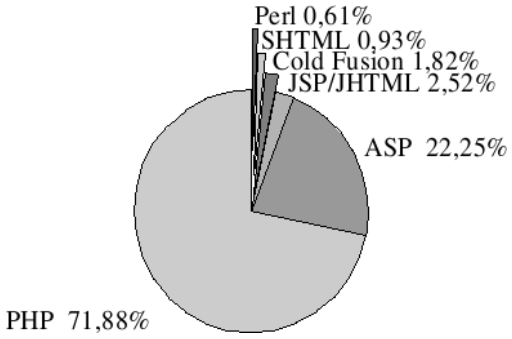


Figure 8.6: Distribution of links to dynamic pages.

PHP, an Open Source technology clearly dominates the market. Dynamic pages are mostly built using hypertext pre-processing (PHP, ASP, JHTML, ColdFusion), in which commands for generating dynamic content, such as accessing a database, are embedded in documents that are mostly HTML code. It must be considered also that some dynamic pages use HTML extension, and that some of the static pages in HTML are generated automatically using batch processing with content management systems, so there are other technologies for dynamic pages that could be missing from this analysis.

8.3.7 Documents not in HTML

We found 400,000 links to non-HTML files containing extensions used for documents. Portable Document Format (PDF) is the most widely used format and the de facto standard, followed by plain text and Microsoft Word. The distribution is shown in Figure 8.7.

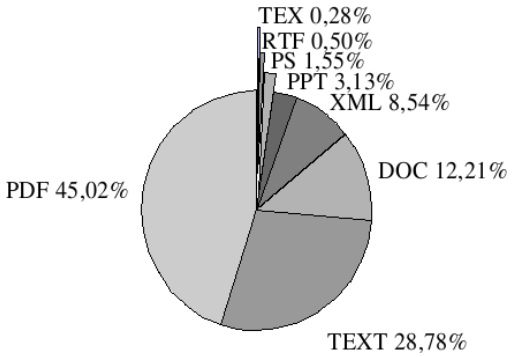


Figure 8.7: Distribution of links to documents found on Chilean Web pages, excluding links to HTML files.

Despite the fact that Microsoft Windows is the most used operating system, file types associated with Microsoft Office applications such as Word or Powerpoint are not used as much as we could expect, probably because of concerns of viruses or lost of formatting.

There are over 30,000 XML files in the Chilean Web, including files with the extensions DocBook, SGML, XML and RDF. In our opinion, this amount of links suggest that it is worth to download those XML files and analyze them, as we could start searching on them.

8.3.8 Multimedia

There are several links to multimedia files, including over 80 million links to images, 50,000 links to audio files, and 8,000 links to video files. The distribution of file formats is shown in Figure 8.8.

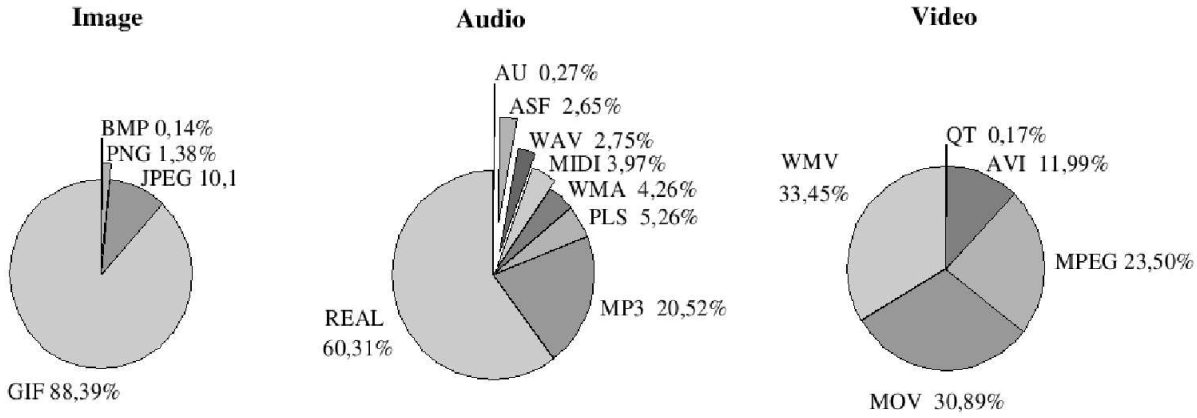


Figure 8.8: Distribution of links to multimedia files found on Chilean Web pages.

Compuserve `GIF` is the most used file format for images, followed by `JPEG`. The Open Source PNG format, which was conceived as a replacement of GIF is still rarely used. The contents of these images was analyzed in the context of a face detection is analyzed in [BYdSV+04].

Realnetwork's `Realaudio` and `MP3` are the most used file formats for audio, and are mostly used for streaming in Internet radios. In the case of video, there is no clear dominant format and there are relative few video images on the Web (1/1000 of the quantity of image files). We also found over 700,000 links to Flash animations, mostly in the home pages of Web sites.

We found that about 1/3 of the links to multimedia files from home pages were not unique, and that this fraction falls to 1/10 of the links when internal pages are considered. This suggests that Web site designers usually have a small set of images that are used across their entire Web sites.

8.3.9 Software and source code

We found links to 30,000 files with extensions used for source code, and 600,000 files with extensions used for software. The later does not count software that is distributed in compressed files such as `.tar` or `.zip`. The distribution of the links found is shown in Figure 8.9.

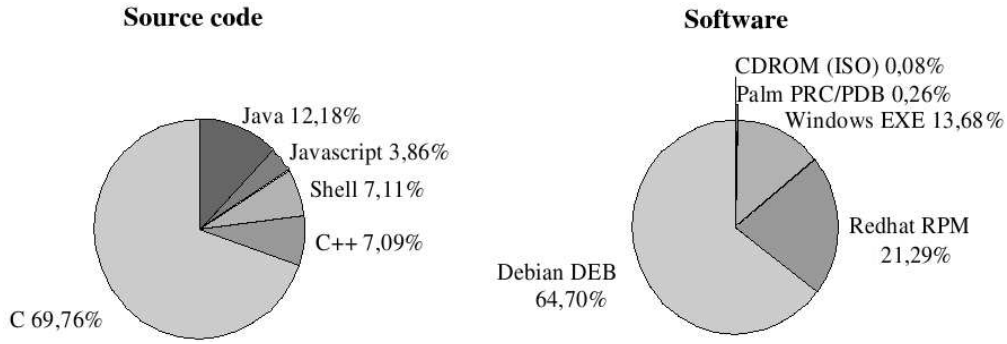


Figure 8.9: Distribution of links to source code and software.

Note that the number of files containing software packages for Linux distributions doubles the one for Windows software; the explanation is that in Linux an application is usually comprised of several packages. Nevertheless, this reflects a comparable level of availability of software packages for both platforms.

Software repositories are usually mirrored at several locations, and the prevalence of mirrors on the Web is in general very high, but the method for avoiding duplicates explained in Section A.5.2 (page 163) worked very well in removing these mirrors, as we only have 6% of duplicate pages.

8.3.10 Compressed files

We found links to 370,000 files with extensions used for packed or compressed files, and their distribution is shown in Figure 8.10.

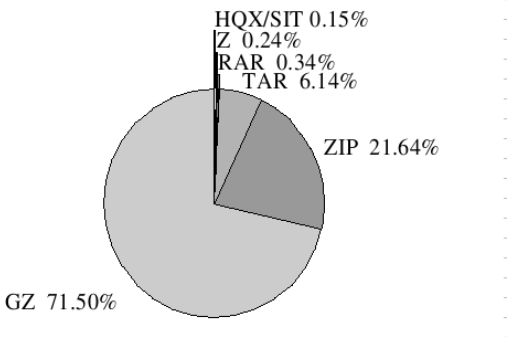


Figure 8.10: Distribution of links to compressed files.

The `GZ` extension, used in the GNU `gzip` program, is the most common extension. Note that in this

case these files probably include software packages that are not counted in Figure 8.9.

8.4 Web site characteristics

8.4.1 Number of pages

We observed an average of 67.1 pages per Web site, but the mode is much smaller than that. The distribution of the number of Web pages on Web sites is very skewed, as shown in Figure 8.11, following a Zipf's law of parameter -1.77.

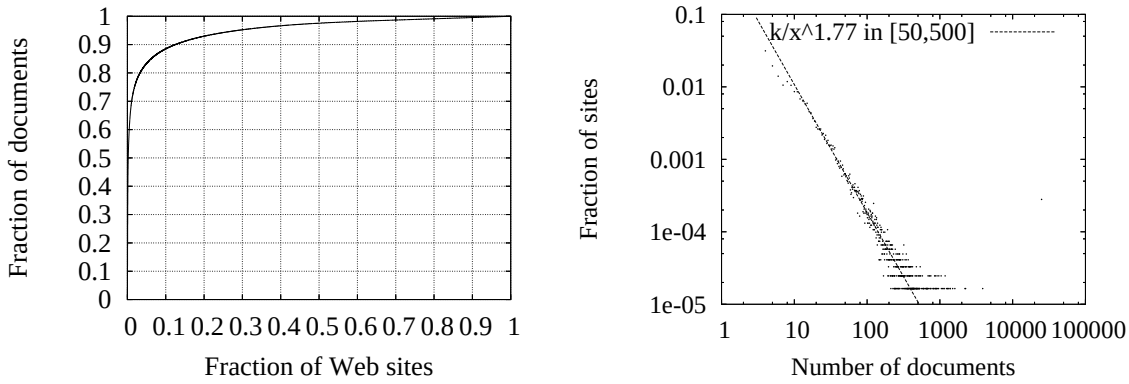


Figure 8.11: Pages per Web site.

.

There are many domain names that are registered with the sole purpose of reserving the name for later use. For instance, only half of the registered domain names under `.cl` have a Web site, and from those, about half have only one page, so only about a quarter of the Web sites are proper Web sites with at least two pages. Although the number of Web sites on the Chilean Web has doubled in the last three years, the fraction of Web sites with just one page remains constant.

On the other end, there are very large Web sites. The top 10% of the Web sites contain over 90% of the Web pages.

8.4.2 Page size

The average size of a complete Web site, considering only HTML pages, is about 1.1 Megabytes (we do not know the total amount of information on the Web site, as we did not download multimedia files). The distribution of the size of Web sites in terms of bytes is also very skewed, as can be seen on Figure 8.12. It is even more skewed than the distribution of the number of pages, as the top 10% of Web sites contain over 95% of the total page contents in bytes.

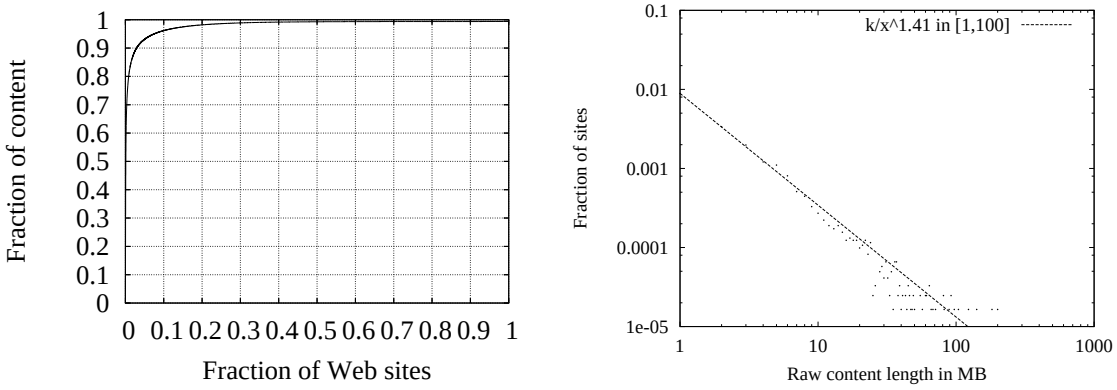


Figure 8.12: Page contents per Web site.

.

The distribution of the page sizes suggests that using the schemes for server cooperation, presented in Chapter 6, with just a few large Web sites could be very efficient.

8.4.3 Maximum depth

As defined in Chapter 5, the home page of a Web site is at level 0, and the level of a page is the shortest path from that page to the home page of its Web site.

Most of the Web sites we studied are very shallow, as shown in Figure 8.13. The average maximum depth of a Web site is 1.5.

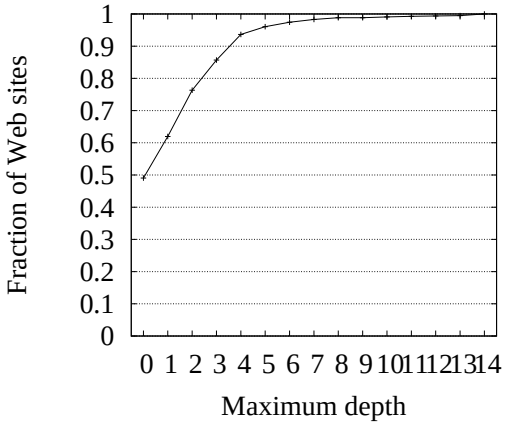


Figure 8.13: Cumulative Web site maximum depth.

.

The distribution of the maximum depth of Web sites is further evidence in favor of what is proposed in

Chapter 5, namely, downloading just a few levels per Web site.

8.4.4 Age

We measured the age of Web sites, observing the age of the oldest and newest page, as well as the average age. The age of the oldest page is a upper bound on how old the Web site is, and the age of the newest page is a lower bound on how often the Web site is updated. The results are shown in Figure 8.14.

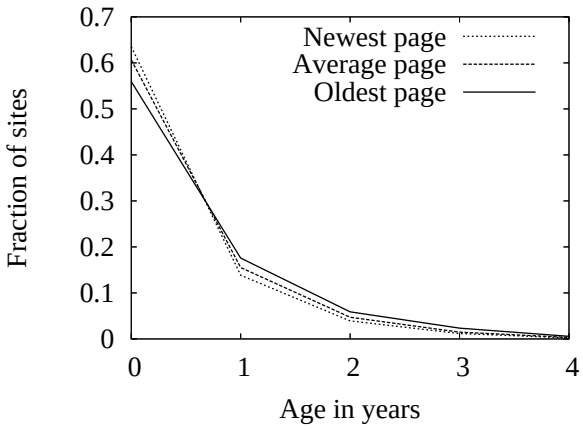


Figure 8.14: Web site age.

.

According to this figure, about 55% of the Web sites were created this year, and about 3/4 of the Web sites in the last 2 years. This implies that for obtaining a large coverage in a national top-level domain, it is necessary to obtain the most recently registered domain names frequently.

8.5 Links

8.5.1 Degree

The distribution of links is skewed, with very few pages having large amounts of links. The distribution of in-degree is much more skewed than the distribution of out-degree, as shown in Figure 8.15: having a Web page with a large in-degree is much more difficult than having a page with a large out-degree.

The distribution of out-degree is similar to the distribution of page-sizes, and there is indeed a correlation between both, as a page cannot have too many links if it is too small, as shown in Figure 8.16.

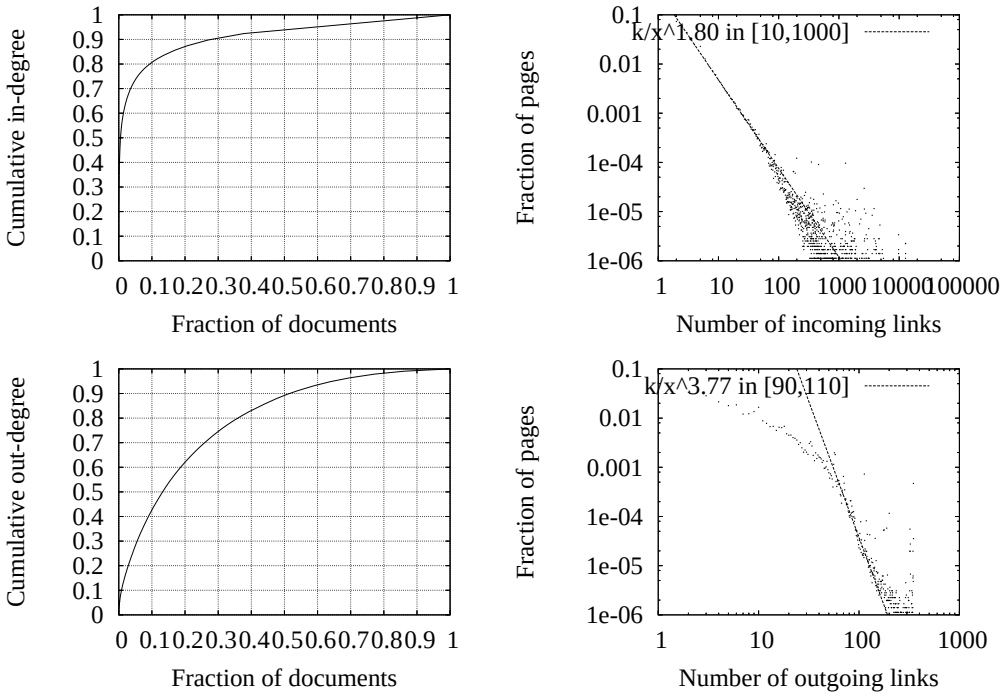


Figure 8.15: Distribution of in- and out-degree.

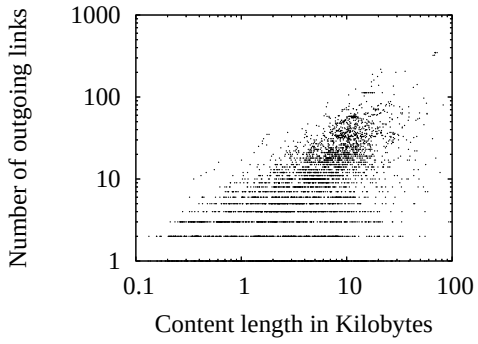


Figure 8.16: Content length vs number of outgoing links.

8.5.2 Link scores

We compared the distributions of Pagerank [PBMW98], and a variation of the HITS algorithm [Kle99], in which we use the entire Web as the expanded root set (this can be seen as a static version of HITS).

The distribution of link scores is shown in Figure 8.17.

As Pagerank is calculated using random jumps to other pages, even pages with very few links have a "parasitic" Pagerank value (no page has zero probability) that is lower than $1/N$, where N is the number of pages.

On the other hand, a page needs "good" links (out-links to authorities in the case of hubs, in-links from

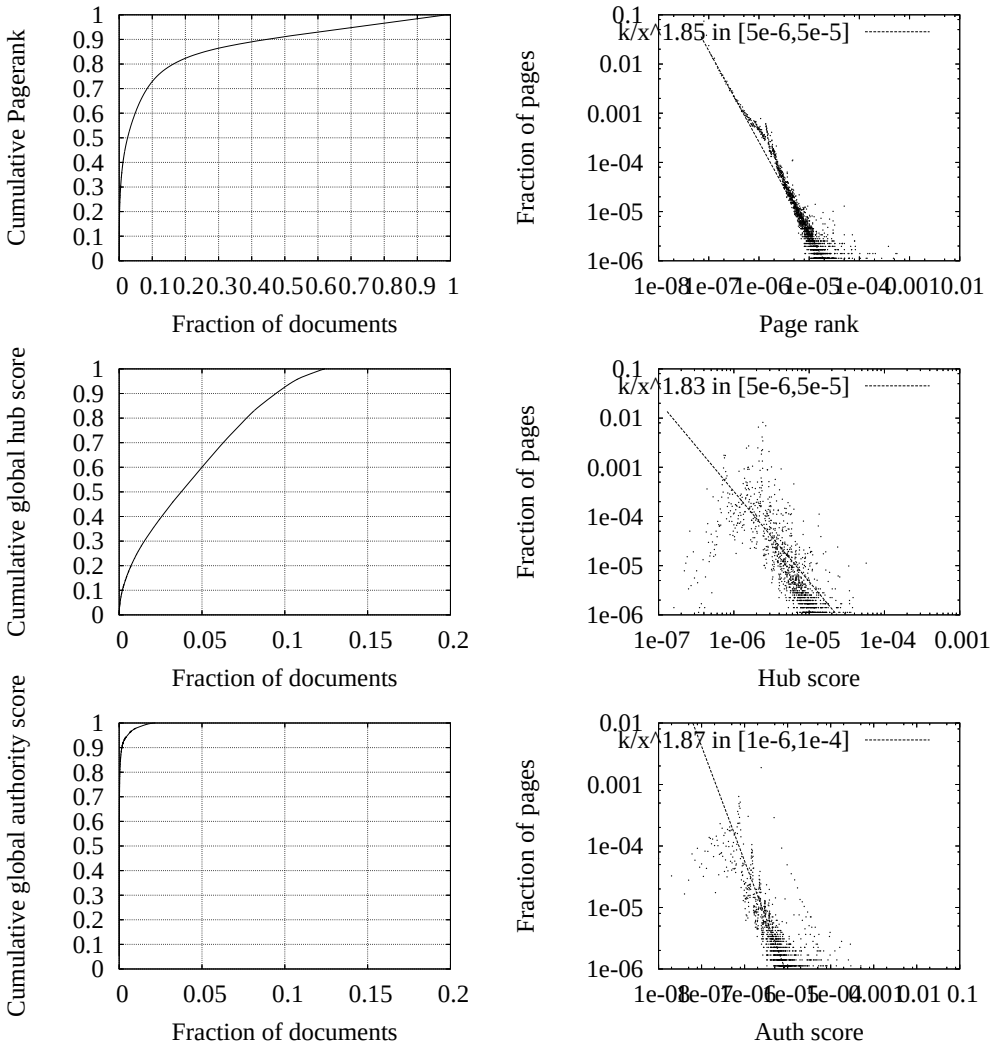


Figure 8.17: Distribution of Pagerank, global Hubs and Authority scores.

hubs in the case of authorities) to have a non-zero value. Only 12% of pages have a non-zero hub value and only 3% of pages have a non-zero authority value.

8.5.3 Links to external domains

We found 2.9 million outgoing external links, i.e.: links that include a host name part. After discarding links to other hosts in .CL we obtained 700,000 links. The distribution of links into domains for the top 20 domains is shown in Table 8.2.

Top level domain	Percent of links	Top level domain	Percent of links
COM	65.110%	PE - Peru	0.558%
ORG	11.806%	ES - Spain	0.494%
NET	8.406%	FR - France	0.464%
DE - Germany	1.621%	JP - Japan	0.462%
MX - Mexico	1.059%	NL - Netherlands	0.444%
BR - Brazil	0.977%	IT - Italy	0.431%
AR - Argentina	0.846%	VE - Venezuela	0.400%
CO - Colombia	0.809%	TW - Taiwan	0.382%
UK - United Kingdom	0.644%	SG - Singapur	0.371%
EDU	0.609%	KR - Korea	0.370%

Table 8.2: Fraction of links to external domains, top 20 domains

Most of the countries in the table are Latin American countries, but there are also links to large domains such as .COM or .DE. We took data from the exports promotion bureau of Chile, "ProChile" [Pro04], regarding the volume of exports from Chile to other countries, and we compared this with the number of links found. We took the top 50 countries that receive more exports from Chile. The USA –which is the largest destination of Chilean exports– was taken as the .COM domain. The results are shown in Figure 8.18.

There is a relationship between number of outgoing links and exports volume. The most important outliers are Asian countries, the outliers above the line have a high volume of exports but few links, probably because of a language barrier.

8.6 Links between Web sites

In the following, we consider links between Web sites. A link between two Web sites represents one or many links between their pages, preserving direction. Several links between pages are collapsed to a single link between Web sites, and self-links are not considered.

A summary of the characteristics of the links found is presented in Table 8.3.

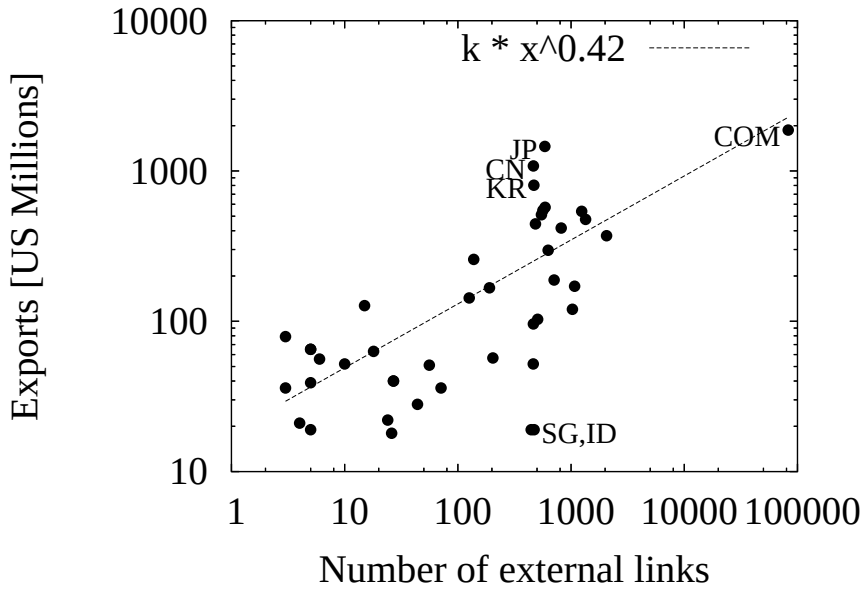


Figure 8.18: Relationship between number of external links from Chilean Web site and exports from Chilean companies to the top 50 destinations of Chilean exports.

Table 8.3: Summary of characteristics of links between Web sites.

Downloaded Web Sites	49,535	
At least one in-link	17,738	36%
At least one out-link	13,820	28%
At least one in-link or out-link	23,499	47%

8.6.1 Degree in the Web site graph

The distribution of in- and out-degree also reveals a scale-free network, as shown in Figure 8.19. The cumulative graphs consider only the Web sites with at least one in- or out-link respectively.

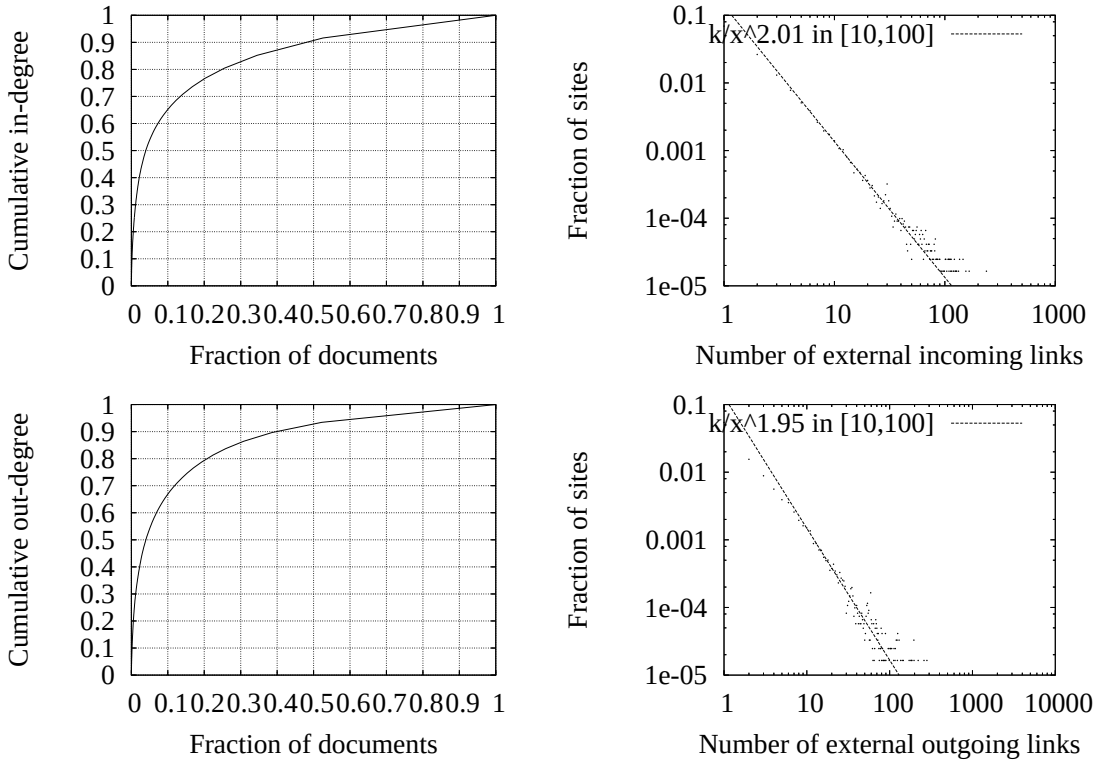


Figure 8.19: Distribution of in- and out-degree in Web sites.

8.6.2 Sum of link scores

We studied the link scores presented in Figure 8.17, and summed them by Web sites. The result is shown in Figure 8.20.

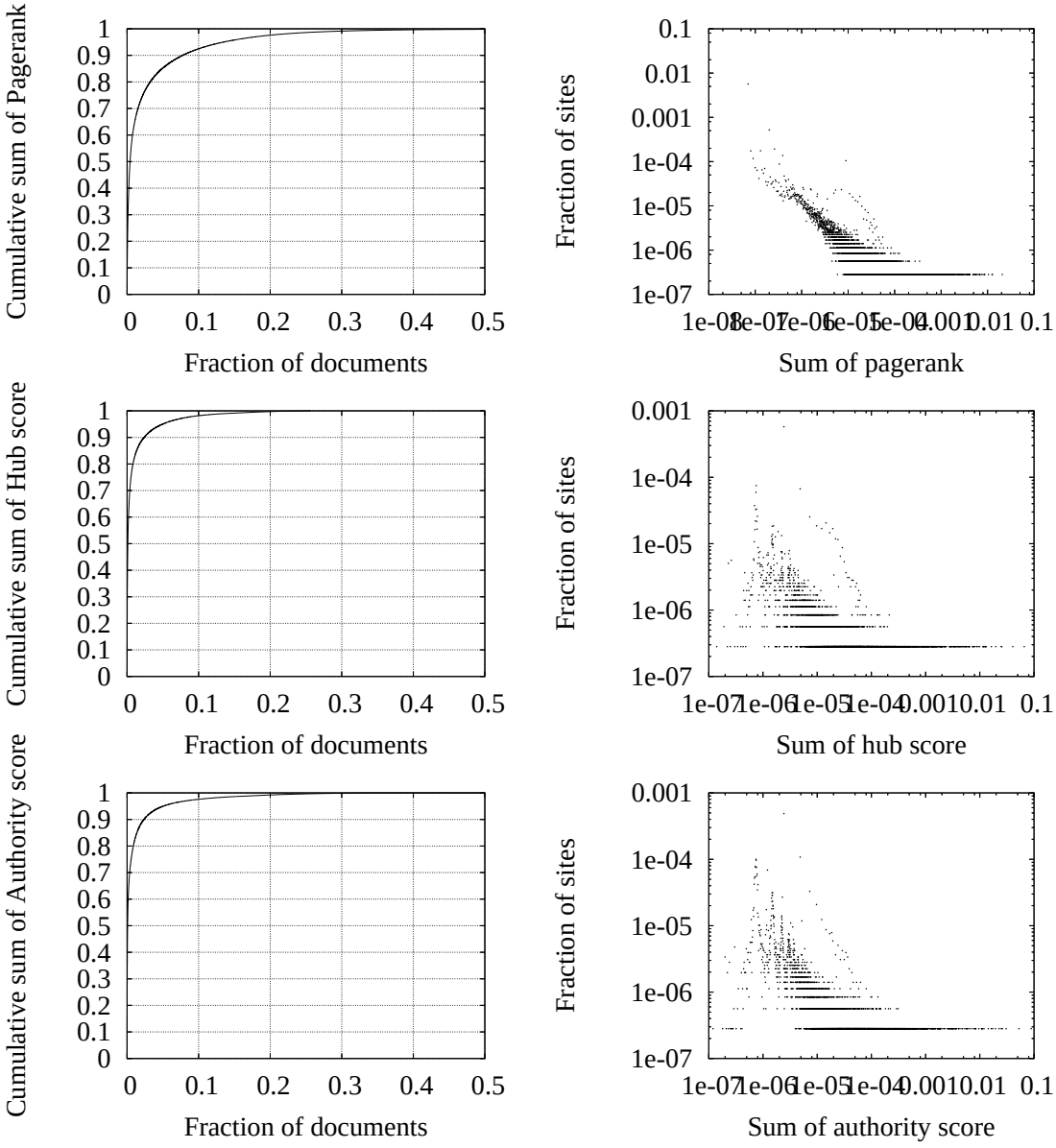


Figure 8.20: Distribution of Pagerank, global hubs and authority score in the graph of Web sites.

8.6.3 Most linked Web sites

The most linked Web sites on the Chilean Web are listed in table 8.4. There is a very strong presence of government-related Web sites in the top places, as well as universities.

Site name	Site type	Number of links
hits.e.cl	Access counter	675
www.sii.cl	Government (internal revenue service)	647
www.uchile.cl	University	595
www.mineduc.cl	Government (education)	513
www.meteochile.cl	Meteorology Service	490
www.emol.com	Newspaper	440
www.puc.cl	University	439
www.bcentral.cl	Government (bank)	404
www.udec.cl	University	366
www.corfo.cl	Government (industry)	354

Table 8.4: Most referenced Web sites, by number of in-links in the graph of Web site links.

8.6.4 Strongly connected components

We studied the distribution of the sizes of strongly connected components (SCC) on the graph of Web sites. A giant strongly connected component appears, as observed by Broder *et al.* [BKM$^+$00]. This is a typical signature of a scale-free network. The distribution of SCC sizes is presented in Table 8.5 and Figure 8.21, here we are considering only Web sites with links to other Web sites.

Component size	Number of components
1	17,393
2	283
3	54
4	20
5	4
6	3
7	2
8	2
9	2
10	1
5,202	**(Giant SCC) 1**

Table 8.5: Size of strongly connected components.

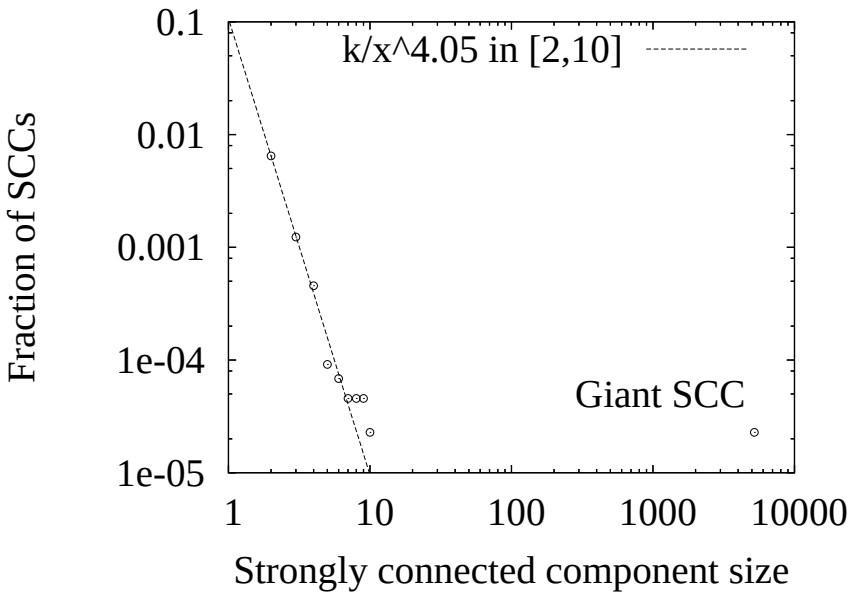


Figure 8.21: Distribution of strongly connected components.

8.6.5 Web links structure

In [BYC01] we extended the notation introduced by Broder *et al.* [BKM+00] for analyzing Web structure, by dividing the MAIN component into four parts:

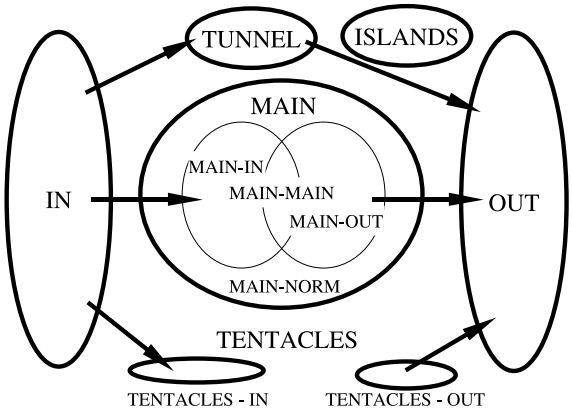
(e) MAIN-MAIN, which are sites that can be reached directly from the IN component and can reach directly the OUT component;

(f) MAIN-IN, which are sites that can be reached directly from the IN component but are not in MAIN-MAIN;

(g) MAIN-OUT, which are sites that can reach directly the OUT component, but are not in MAIN-MAIN;

(h) MAIN-NORM, which are sites not belonging to the previously defined subcomponents.

Note that the Web sites in the ISLANDS component are found only by directly accessing the home page of those Web sites. This is possible because we had a complete list of the registered domains under `.cl` at the time of our studies. The distribution of Web sites into components is shown in Figure 8.22. This structure evolves over time, as studied in [BYP03, BYP04].



Component name	Size
MAIN_NORM	2.89%
MAIN_MAIN	3.16%
MAIN_IN	1.20%
MAIN_OUT	3.26%
IN	7.23%
OUT	18.15%
TENTACLES-IN	2.75%
TENTACLES-OUT	4.23%
TUNNEL	0.33%
ISLAND	56.81%

Figure 8.22: Macroscopic structure of the Web.

.

8.7 Comparison with the Greek Web

We seek to understand to what extent the studies of the Chilean Web represent other subsets of the Web. Dill *et al.* [DKM+02] have shown that the Web graph is self-similar in a pervasive and robust sense. We compared some characteristics of the Chilean and the Greek Web, including the Web graphs but also other properties such as size or number of pages. The pages for this study were obtained simultaneously on the Greek and Chilean Web domains during January 2004.

We downloaded pages using a breadth-first scheduler for up to 5 levels for dynamically generated pages, and up to 15 levels for static, HTML pages. We limited the crawler to 20,000 pages per website; and considered only pages under the `.gr` and `.cl` domains.

Both countries are comparable in terms of the number of pages, but have many differences in terms of language, history, economy, etc. Table 8.7 summarizes information about the page collection, as well as some demographic facts that provide the context for this section.

	Greece	Chile
Population [Uni02]	10.9 Million	15.2 Million
Gross Domestic Product [The02]	133 US$ bn.	66 US$ bn.
Per-capita GDP, PPP [The02]	17,697 US$	10,373 US$
Human development rank [Uni03]	24^{th}	43^{th}
Web servers contacted	28,974	36,647
Pages downloaded	4.0 Million	2.7 Million
Pages with HTTP OK	77.8%	78.3%

Table 8.6: Summary of characteristics.

8.7.1 Web pages

Figure 8.23 shows the depth at which the pages of the collection were found; note that 5 is the limit we set for dynamic pages, as dynamic pages grows exponentially with depth. The distribution is almost identical.

Figure 8.24 shows the distribution of HTML page sizes, not considering images, showing a peak between 10 and 15 Kilobytes. The right-tail follows a power-law similar to previous results, and both distributions are very similar.

Figure 8.25 plots the number of pages per website. This has a very skewed distribution, as few websites account for a large portion of the total web; so we have plotted this in log-log scale.

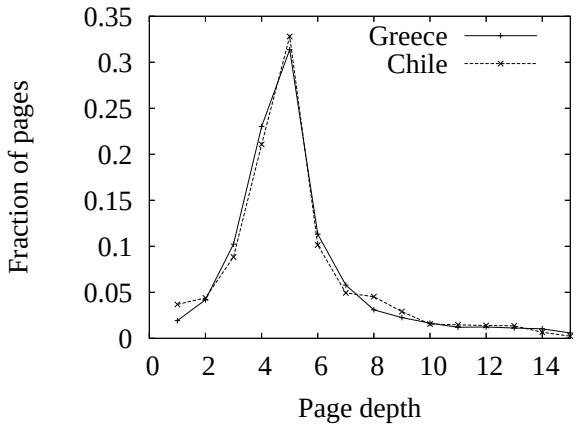


Figure 8.23: Comparison of page depth, 1 is the page at the root of the server.

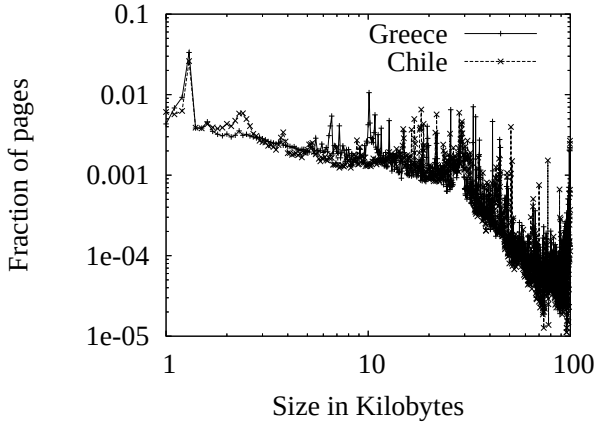


Figure 8.24: Comparison of the distribution of page size in Kilobytes.

8.7.2 Links

Figure 8.26 shows that these two sub-graphs of the web have these characteristics, revealing the existence of self-similarities. The power law parameter depends a lot on the range of data used. Taking degrees of at most 350, we obtain -2.02 and -2.11 for in-degree, and -2.17 and -2.40 for out-degree; for .GR and .CL, respectively. Discarding degrees smaller than 50, the parameters are closer to -2.3 and -2.8 for in-degree and out-degree. This should be compare with the results in [KRR$^+$00] that found -2.1 and -2.7, respectively, for 200 million pages in 1999.

The distribution of out-degree is different, as the in-degree in many cases reflects the popularity of a web page, while the out-degree reflects a design choice of the page maintainer. Also, it is much easier to have a page with many outgoing links than one with many incoming links.

For the graph components, we use the bow-tie structure proposed by Broder et al. [BKM$^+$00]; but we

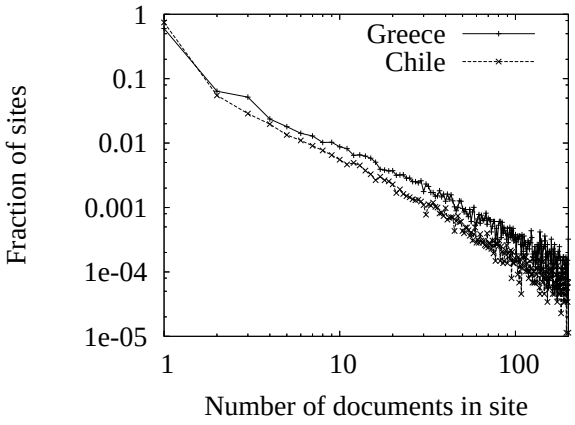


Figure 8.25: Comparison of the distribution of the number of pages per website.

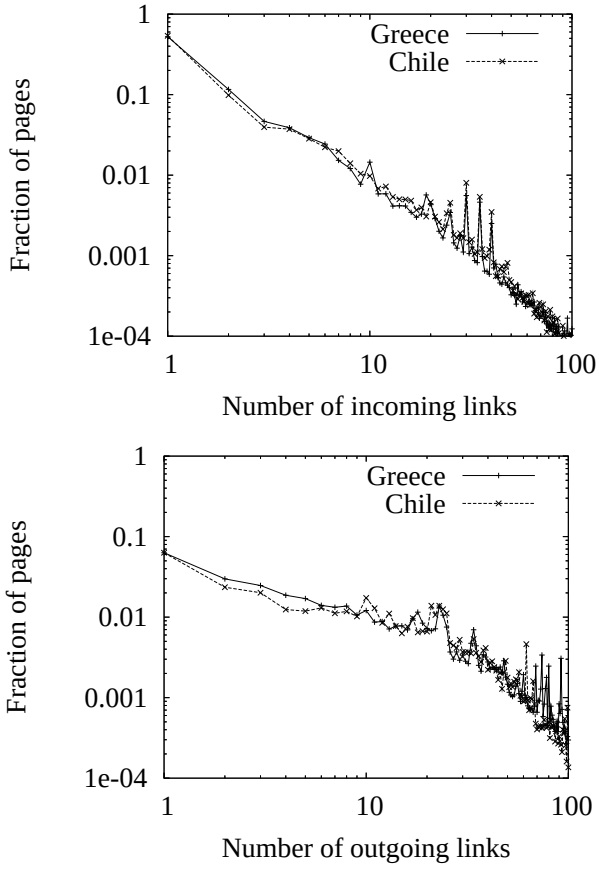


Figure 8.26: Comparison of the distributions of in-degree and out-degree.

considered only links between different websites, collapsing all the pages of a website to a single node of the graph. We show the relative size of components in Figure 8.27.

Note that that the MAIN (the giant strongly connected component) seems to be larger in the Greek web in expense of the ISLAND component - this can be an indicator of a better connected Web, although the seeds for the Chilean crawling had more islands.

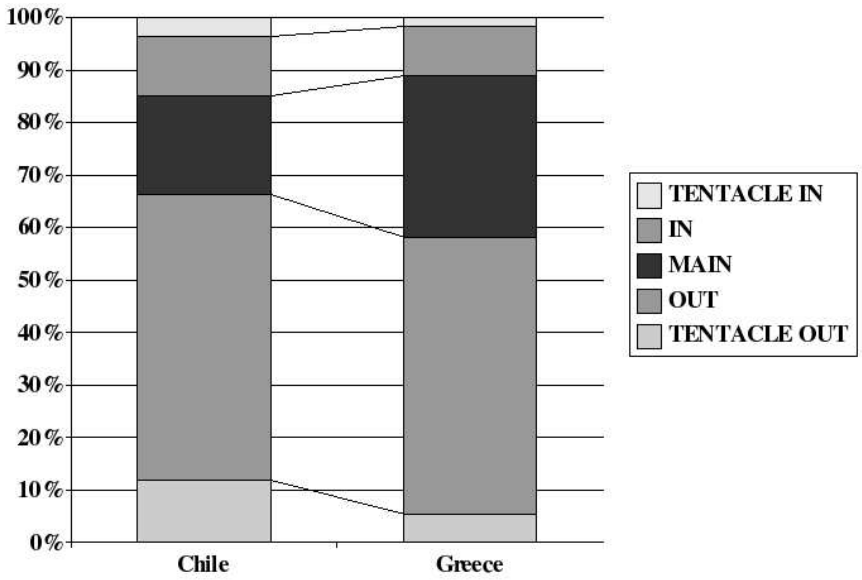


Figure 8.27: Comparison of the relative size of graph components.

We also studied the relationship of the collections with other top level domains reflecting cultural and economic relationships. The most linked domains (COM, ORG, NET, etc.) are equally important in both collections, but there are differences which are presented in Table 8.7. While the top linked Web sites for Greece are in Europe and Asia, for Chile they are mostly in America.

8.8 Conclusions

In this chapter, we have analyzed several characteristics of a large sample of the Web, and most of those characteristics suggest a distribution of quality that is very skewed. This is good for Web search, because only a few of the Web pages have some relevance, but this is also bad for Web crawling, because it is necessary to download large amounts of pages that are probably irrelevant.

World Wide Web users have a certain perception of how the World Wide Web is. This perception is based on what they see while interacting with the Web with the usual tool: a Web browser. The behavior of different users involves different parts of the Web, but in most cases it is limited to a few highly important Web sites with topics such as news, shopping or Web-based e-mail.

Most users do not go too deep inside Web sites. This means that there are thousands or millions of pages that are visited very rarely, or that are not visited at all. When characterizing the Web, we must forget what

Greece		Chile	
COM	49.2%	COM	58.6%
ORG	17.9%	ORG	15.4%
NET	8.5%	NET	6.4%
Germany	3.7%	Germany	2.6%
United Kingdom	2.6%	United Kingdom	1.4%
EDU	2.6%	EDU	1.3%
TV	1.3%	Mexico	1.2%
Russian Federation	1.3%	Brazil	1.1%
Taiwan	1.1%	Argentina	0.9%
Netherlands	0.9%	Spain	0.9%
Italy	0.8%	Japan	0.6%
GOV	0.6%	France	0.6%
Norway	0.6%	Netherlands	0.6%
France	0.5%	Italy	0.6%
Canada	0.5%	Australia	0.6%

Table 8.7: Comparison of the most referenced external top-level domains.

we have seen while browsing Web pages, because what we see through a Web browser is just the surface of something much deeper. For instance, there are very large and very small pages, pages with thousands of in-links and pages with only one, and so on.

Our results also show a dominance of standard formats such as PDF or plain text, and open source tools such as PHP and GZIP, which is quite natural given the open nature of the Web.

Chapter 9

Conclusions

During this thesis, we studied Web crawling at many different levels. Our main objectives were to develop a model for Web crawling, to study crawling strategies and to build a Web crawler implementing them. This section points out what we have done and what could be done as future work.

The next section summarizes our contributions, Section 9.2 describes some guidelines for future work, and Section 9.3 discusses open problems for Web crawling.

9.1 Summary of our contributions

This thesis dealt with Web crawling from a practical point of view. From this point of view, we worked in a series of problems that appeared during the design and implementation of a Web crawler.

We started by describing Web crawling in the context of information retrieval and summarizing relevant literature on the topic. Although there are many studies about Web search, Web crawler designs and algorithms are mostly kept as business secret, with some exceptions (Chapter 2).

Our model divides the problem of Web crawling into short-term and long-term scheduling. We explained why this separation is possible and how we can exploit it for efficient Web crawling, by generating batches of pages using a long-term scheduling policy, and then re-ordering these batches to apply a short-term scheduling policy. The long-term scheduling is related to page quality, while the short-term scheduling is related to network efficiency (Chapter 3).

We also proposed a better integration between the Web crawler and the rest of the search engine, and a framework for measuring the quality of a crawl. Within this framework, we classified several existing Web crawlers according to the relevance they give to different parameters of our model (Chapter 3).

Regarding page quality, we compared several scheduling algorithms for long-term scheduling, and we found a very simple and fast strategy for downloading important pages early in the crawl. We ran experiments

in both a simulated and a real Web environment for proving that this strategy is successful. For short-term scheduling –network efficiency– we showed the effect of downloading several pages using the same connection and explained why a long waiting time between accesses to pages in the same server decreases crawling efficiency (Chapter 4).

It become clear very quickly that if the crawler downloads dynamic pages then the space of URLs to download from is practically infinite. In this context, it is important to know when to stop a crawl. Instead of setting and arbitrary crawling depth, we studied this problem by estimating the estimate the value of the pages that were not crawled. To do this, user behavior was studied and modeled using a Markov chain created from data from actual Web sites. The conclusion was that users explorations are in general very shallow, and we were able to set an appropriate crawling depth (Chapter 5).

To improve freshness in the search engine, there are a number of actions that Web server administrators can take. We proposed and compared several cooperation schemes for Web servers and Web crawlers, including polling and interrupt versions for them. These cooperation schemes can help a Web crawler in detecting changes in Web sites, and lower the network usage of the crawling process, which is beneficial for both the search engine and the Web site (Chapter 6).

We implemented a high-performance Web crawler in C/C++. This crawler implements the scheduling algorithms proposed in this thesis, and serves as a proof of concept for the cooperation schemes. Our implementation is publicly available as it is released under the GNU public license. This Web crawler can efficiently download several million pages per day, and can be used for Web search and Web characterization (Chapter 7).

We downloaded and analyzed the Chilean Web using our crawler, extracting statistics on HTML pages, multimedia files, Web sites and link structure. We also used the crawler for downloading the Greek Web and compared both datasets, finding that despite large differences in the context of both countries –for instance, the GDP per capita of Greece doubles the one of Chile, as well as the number of pages– both Web collections are very similar. (Chapter 8).

During the crawling processes that we carried out, we discovered and studied several practical issues that arise only after large crawls. Those issues are mostly anomalies in the implementations of the underlying protocols that form the Web, and must be taken into account when developing a Web crawler (Appendix A).

9.2 Future work

As most research, this thesis opens more problems than it solves. Reviewers of our work and ourselves found several avenues for future work. We consider the following as the main ones:

Validating the collection The Chilean Web with its 3.5 million pages represents 1/1000 of the World Wide Web. We have been studying this collection since 2000, and the distribution of several variables seems

very similar to the data published for other samples of the Web, but this should be tested with other collections.

A related question would be if the Web is a representative collection of text. The answer given by Kilgariff and Grefenstette [KG04] is very pragmatical:

> "The Web is not representative of anything else. But neither are other corpora, in any well-understood sense."

Considering the contents of Web pages Besides checking for duplicates, we largely ignore the text of pages. This is done on purpose: we wanted to study only links to use them as independent information, which can be combined later with other evidence.

In particular, we think that the methods of focused crawling [CvD99], in which a crawler is directed to pages on *specific* topics based on properties of the text, can be adapted to direct the crawler to interesting pages on *all* topics.

Developing complex models of user behavior The models of user behavior we presented in Chapter 5 were chosen for their simplicity. A more complex model would be a model with memory, in which the probability of users following a link depends on their previous actions. We have observed that this is indeed the case, because users that go deeper into the Web site have a higher probability of continuing to follow links. We also pointed out that user behavior in Blogs is different than on other sites.

Testing short-term scheduling strategies We have focused mostly on long-term scheduling strategies, although we made experiments on short-term scheduling and described them in Chapter 4. The most important issue would be to implement persistent connections in the Web crawler, to download several pages from the Web site without re-connecting. In theory, this has a significant positive impact on Web crawl, but further testing is required.

Continuous crawling We focused mostly in a "crawl-and-stop" type of crawl, in which we only crawl each page once. Our crawler can also continuously check pages for updates, using the value function to decide when to check for a page for changes, and when to download new pages. In this context, the measure of importance is not the time it takes to complete the crawl, but the average freshness and/or the average quality when a stationary state is reached. This is also related to measure how good are the change prediction in a real Web environment.

Benchmarking Measuring the efficiency of a Web crawler –considering only short-term scheduling –is not an easy task. We need a framework for comparing crawlers that accounts for the network usage, the processing power required, and the memory usage. It would be good to have a measure that allows us to compare, e.g., 30 pages per second in a 1GHz processor with 1Gb RAM with 20 pages per second in a 800MHz processor with 640Mb RAM. A more important problem is that network conditions vary so for testing different Web crawlers we must consider the time of the day, network capacity, etc.

Finding combined strategies Specifically the best parameters for the manager program, in terms of the importance that should be given to intrinsic quality, representational quality and freshness. We also consider that the scheduling policies should adapt to different Web sites, and our model provides a framework for that kind of adaptability.

Exploiting query logs for Web crawling The usage of user query logs in a search engine for guiding a Web crawling is an important step in integrating the collection and search processes. The query logs can be used to refresh frequently returned pages faster, so the crawler refreshes the active set of the search engine more often. Moreover, the query terms used in Web search can be given priority, so the Web crawler scheduling could be biased toward pages that contains the query terms that are being queried more frequently by the search engine's users.

9.3 Open problems

The importance of a good crawler strategy depends on the balance between these quantities:

1. The total amount of information available on the Web.
2. The amount of bandwidth available for the Web crawler.
3. The amount of good information about specific topics on the Web.

Web search is difficult today because the Web is very large, the bandwidth available at most locations is relatively small, and good pages are also relatively scarce. The three quantities are very dynamic, and their evolution is very difficult to predict. For instance, there is no clear equivalent of a "Moore's law" for network connectivity. Thus, we do not know if the problem of keeping the repository of a search engine is going to get easier or harder.

It is likely that Web crawling continues to be a difficult problem, at least during the next years, and we expect several challenges. Multimedia information such as digital photographies and recordings could account for a larger proportion of the Web content, and the number of Web posting in Blogs will be larger than the number of Web pages, further reducing the signal-to-noise ratio of the Web. Finally, pages with semantic markup could become a significant fraction of Web pages, radically changing the problem of Web search.

Appendix A

Practical Web Crawling Issues

When we tested our implementation, which is described in Chapter 7, we found that there were several problems of Web crawling that did not become evident until a large crawl was executed. Our experiences arise from several crawls of the Greek, Spanish and Chilean Web carried out during the this thesis.

We are interested in documenting these problems for two reasons:

- To help other crawler designers, because most of the problems we found are related to the characteristics of the Web, independent of the Web crawler architecture chosen.
- To encourage Web application developers to check their software and configurations for compliance to standards, as this can improve their visibility on search engine's results and attract more traffic to their Web sites.

The rest of this chapter is organized as follows: Section A.1 deals with network problems in general. Section A.2 deals with more specific problems with massive DNS resolving. Section A.3 presents the problems of dealing with wrong implementations of HTTP. Regarding the server-side, Section A.4 deals with bad HTML coding, Section A.5 with problems in the contents of the pages, and Section A.6 with difficulties arising from the programming logic of some Web sites.

A.1 Networking in general

An estimation for the cost of an entire crawl of the World Wide Web is about US $1.5 Million [CCHM04], considering just the network bandwidth necessary to download the pages, so it is very important to use the network resources efficiently to maximize the crawler throughput and avoid wasting the allocated bandwidth.

A.1.1 Variable quality of service

One of the most challenging aspects of Web crawling is how to download pages from multiple sources in a stream of data that is as uniform as possible, considering that Web server response times are very variable.

Web server up-time cannot be taken for granted, and it is usual to find Web servers that are down for a long time, even days or weeks, and re-appear later. This is why sometimes "dead pages" are called "comatose pages" [Koe04]. If the Web crawler aims for a comprehensive coverage of the Web, it should consider that some of the pages which are not available now, could become available in the future.

Recommendation: the crawler should re-try each Web page a number of times if the page is down; the interval should be several hours. We used 12 hours as the default).

A.1.2 Web server administrators concerns

Web crawlers prompted suspicion from Web site administrators when they first appeared, mostly because of concerns about bandwidth usage and security, and some of those concerns are still in place today. In our experience, repeated access to a Web page can trigger some alarms on the Web server, and complaints from its administrator.

We consider that the two most important guidelines given by Koster [Kos93] are:

- A crawler must identify itself, including an e-mail address for contact, or some Web site administrators will send complaints to the listed owner of the entire originating network segment.
- A crawler must wait between repeated accesses to the same Web site.

These guidelines are even more important if we consider that many host names point to the same IP, usually belonging to a Web hosting provider, and in general several Web sites are hosted by a few physical servers. Being unpolite with a Web site can result in being banned from all the Web sites hosted by the same ISP.

Recommendation: the crawler should avoid overloading Web sites, and it must provide an e-mail address in the `From` HTTP header, and/or a Web site address as a comment in the `User-Agent` HTTP header.

A.1.3 Inconsistent firewall configurations

Some Web servers are behind a firewall, and we have found firewall configurations that we did not expect. We detected cases when the `connect()` call succeeds, i.e. a TCP connection is established with port `80` of the Web server, then the `write()` call succeeds, but there is no answer from the Web server.

This appears to be a problem with data packets to port `80` being dropped, but connections accepted, which is not a consistent configuration. This caused some threads of the harvester to hang up indefinitely in one of our early versions.

Recommendation: all network operations should have a timeout. The crawler must be prepared, because at any point of the download operation it could stop receiving data.

A.2 Massive DNS resolving

A.2.1 Crashing your local DNS servers

We found that some of our local DNS servers crash under heavy loads, instead of just queuing or denying connections. From the Web crawler's point of view, a DNS failure of the local servers is a critical situation because, if after repeated attempts it cannot get an IP address for connecting, it has to assume the Web site does not exist, and if all DNS lookups are failing, this can make an entire crawl useless.

Recommendation: local DNS servers should be tested for their response to high work loads. The Web crawler should detect a condition in which, e.g.: 90% of DNS lookups failed during one cycle, and stop under this condition. The Web crawler also could avoid resolving more than a fixed number of domain names at the same time and with the same DNS server.

A.2.2 Temporary DNS failures

This is related to the quality of service of Web servers themselves, as for small organizations typically the Web server and the DNS server are both under the same administration and even in the same physical computer. A DNS failure (e.g.: a DNS server crash) is very likely to go unnoticed, because of the default caching policy: one week. People who visit the Web site often will not notice that something is wrong until several days have passed.

Recommendation: if high coverage is desired, at least one attempt to resolve a DNS record should be done one week after a DNS failure. However, it can also be argued that a Web site with DNS problems has a lower quality than other Web sites and should not be added to the collection: in our model, Web server response quality can be used as a component of the intrinsic quality of a Web page.

A.2.3 Malformed DNS records

DNS report [Per04] is a tool for analyzing DNS records. Its author reports that a significant fraction of DNS records present some problems, ranging from inconsistencies in the serial numbers to misspelling errors or malformed responses.

Recommendation: DNS resolvers should be tolerant to errors in DNS records, and try to retrieve the IP address for a host name even if other portions of the record seems malformed.

A.2.4 Wrong DNS records

Consider the scenario depicted in Figure A.1:

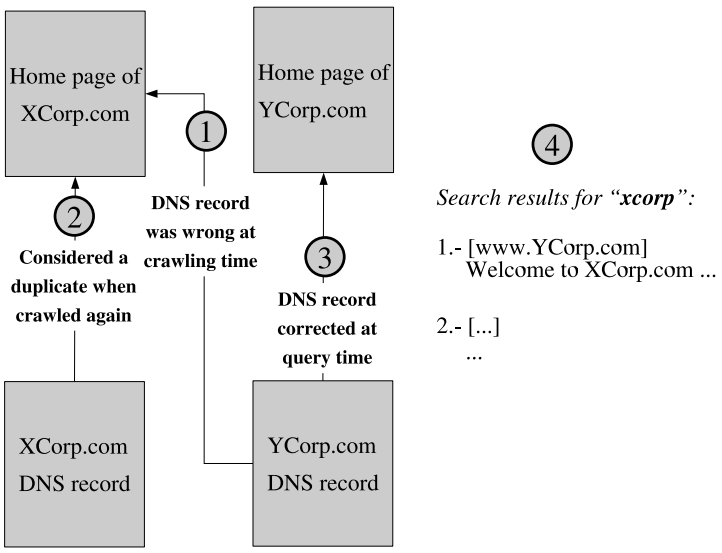


Figure A.1: A misconfiguration in the DNS record for "Ycorp.com" resulted in the wrong contents being assigned to its URL.

1. At crawling time, the DNS record for `Ycorp.com` pointed to the Website of `Xcorp.com`, so the contents of the later were indexed as if their URL was `Ycorp.com`. This DNS misconfiguration can be accidental, or malicious.

2. When the home page of `Xcorp.com` was downloaded, its contents were found to be a duplicate of `Ycorp.com`, so the pages of `Xcorp.com` were not downloaded again.

3. The wrong DNS record of `Ycorp.com` was fixed later.

4. In the search results, when users search for "Xcorp", they can be mistakenly redirected to the Web site of "Ycorp".

Recommendation: it is possible for Web site administrators to avoid these kind of problems by a careful configuration of virtual hosts. Any access to the IP address of the Web server that does not contain a known `Host` field in the request, should be redirected to the default virtual host, referencing the later by its canonical name.

A.2.5 Use of the "www" prefix

Due to the usage of the `www` prefix for the host part of the URLs, in most Websites both "www.example.com" and "example.com" names resolve to the same IP address, and have the same contents. Indeed, we have found that for many Web site administrators, this is the expected behavior, as some users do not type the full address when browsing.

However, if the Web site is built using some application that includes small changes in the pages (e.g.: the current date and time, or a poll question, or advertising, etc.), the Web crawler may not be able to detect that both Web sites are duplicates, and crawl the same contents twice.

Recommendation: we considered that `http://www.example.com/` and `http://example.com/` are the same URL.

A.3 HTTP implementations

A.3.1 Accept headers not honored

In some cases, it is impossible to tell the type of a file just by looking at its URL. Some URLs have no extensions, and some URL have extensions that are ambiguous, e.g.: links to files ending in `.exe` could be either links to dynamically generated HTML pages in the server side, or links to programs that should be downloaded.

A user agent, such as a Web browser or a Web crawler, can have limited capabilities and only be able to handle some data types. If it cannot handle a file (e.g.: an image), then it should not download it. For instance, a Web crawler searching only for plain text and HTML pages, should issue a request of the form:

```
GET /page.html HTTP/1.1
Accept: text/plain, text/html
...
```

This indicates that the Web crawler can only handle plain text or HTML documents. According to the HTTP specification, "the server SHOULD send a 406 (not acceptable) response code" [FGM+99] when a valid object of the desired type is not present at the given URL.

Several Web browsers simply issue a header of the form `Accept: */*`, so some Web server implementations do not check the "accept" header at all. It has somehow lost relevance, and today a Web server can send a response with almost any data type.

A related concern is that some Web sites return a header indicating content-type HTML, but the information returned is a large binary file (such as a ZIP archive, etc.). The crawler can waste bandwidth downloading such a file.

Recommendation: the returned `Content-type` header should always be checked in the downloaded pages, as it might not be a data type that the Web crawler can handle. A download limit is necessary because potentially any file type can be returned by the Web server, even when it is indicating HTML content type.

A.3.2 Range errors

To ensure a good coverage of the Web, we must limit the amount of data that is downloaded from every Web server. This can be done by limiting both the maximum page size, and the number of Web pages that are downloaded from a single Web site.

We limit page size usualy to a default of 300-400 Kb per Web page. We consider that this should capture enough keywords to index each document. To inform the Web servers of the download limit, we use the HTTP `Range` header:

```
GET /page.html HTTP/1.1
Range: 0-400000
...
```

However, a few Web sites return a response code 416 (range error). We have found that these responses correspond to files that are smaller than the desired size. This is not correct, because the HTTP specification indicates that "if the [second] value is greater than or equal to the current length of the entity-body, last-byte-pos is taken to be equal to one less than the current length of the entity- body in bytes" [FGM+99].

Recommendation: in the case of range errors, a second attempt for download could be made without the `Range` header. In all cases, the Web server may ignore the range, so the Web crawler must be prepared to disconnect from the Web server, or discard part of the contents, if the server sends a long document.

A.3.3 Response lacking headers

We have found that most Web browsers are very tolerant to strange behavior from the Web servers. For instance, we have tested Opera and Internet Explorer against a "dummy" Web server that only sends the contents of the Web page requested, with *no status line and no headers*. Both browsers displayed the downloaded pages. The browser Mozilla shows an error message.

Some real Web sites exhibit the same misbehavior as our dummy Web servers, probably because of misconfigured software, or misconfigured firewalls. The Web crawler should be prepared to receive content without headers.

A related problem is that of imcomplete headers. We have found, for instance, responses indicating a redirection, but lacking the destination URL.

Recommendation: from the point of view of Web crawlers or other automated user agents, pages from Web sites that fail to comply with basic standards should be considered of lower quality, and consistently, should not be downloaded. We consider a lack of response headers a protocol error and we close the connection.

A.3.4 Found where you mean error

It is hard to build a Web site without internal broken links, and the message shown by Web servers when a page is not found, i.e.: when the Web server returns a 404 (not found) response, is considered by many Web site administrators as too annoying for users.

Indeed, the default message looses the context of the Web site, so the Web site administrators of some Web sites prefer to build error pages that maintain visual and navigational consistency with the rest of their Web sites.

The problem is that in many cases the response for a page that does not exists is just a normal redirect to a custom-built error page, without the response header signaling the error condition. Bar-Yosef *et al.* [BYBKT04], refer to these error pages as "soft-404", and observe that about 29% of dead links point to them.

The indexing process could consider a redirect to a "soft-404" error page as a link between the URL in which the page was not found and the error page, and this can increase the score of the later.

Recommendation: Web site administrators should configure their Web servers in such a way that the error messages have the correct response codes signaling the error conditions. Servers can be tested by Web crawlers issuing a request for a known non-existent page (e.g.: an existent URL concatenated with a random string [BYBKT04]) and checking the result code.

A.3.5 Wrong dates in headers

A significant fraction of computers are configured to a wrong date, wrong time, or wrong time zone. These configurations are sometimes done on purpose, e.g.: to extend the trial period of shareware software.

During Web crawling, we found that 17% of Web servers returned no last-modification data, dates in the future, or a date prior to the invention of the Web, as shown in Section 8.3.3 (page 128). These wrong dates affect the `Last-Modified` field in the Web server response, which in these cases cannot be used for estimating freshness.

However, not all wrong dates should be discarded: if a Web server replies with a time stamp in the future, but just a few minutes or a few hours, we can consider that it is likely that the Web server clock is just skewed with respect to ours (e.g.: it has the wrong time zone, or it is wrongly set).

Recommendation: we consider that a last modification date for a Web page older than the year 1993 is wrong and should be ignored. For dates in the future, we consider that up to 24 hours can be considered just a small problem, so those time stamps are changed into the current date. If the date is more than 24 hours ahead it is ignored. This is depicted in Figure A.2.

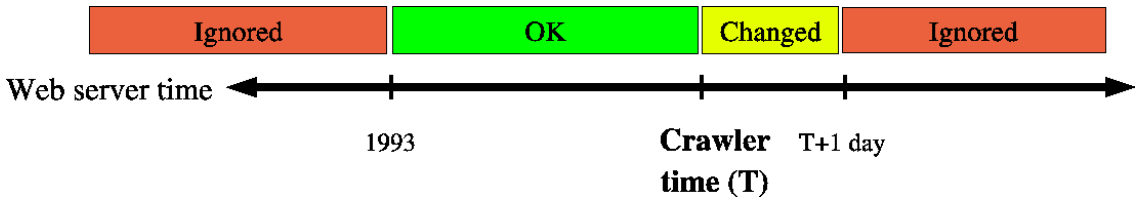


Figure A.2: Diagram showing how we deal with last-modification dates in the responses.

A.4 HTML coding

A.4.1 Malformed markup

HTML coding, when done by hand, tends to be syntactically very relaxed. Most HTML coders only check if the page can be seen in their browsers, without further checking for compliance. We have found several errors in HTML coding, and we have adapted our parser accordingly. These errors include:

- Mixing single quotes, double quotes, and no quotes in attributes, e.g.: `<IMG ALT="This is a photo" SRC='photo.jpg' border=1>`.
- Mixing empty tags in HTML form (such as `<BR>`) and in XHTML form (such as `<BR/>`).
- Unbalanced tags, e.g.: `<SMALL><B>...</SMALL></B>`.
- Mixed case in tags and attributes, e.g.: `<IMG src="...">`. For HTML, the tags should be written in uppercase, and for XHTML, in lowercase.
- Unterminated strings, e.g.: `<IMG ALT="...>`. This can be very problematic, because it will cause a buffer overflow if the parser is not properly written. These unterminated or long strings can also appear in HTTP response codes.

Recommendation: as described in Section 7.2.3 (page 117), we use an events-oriented parser for the HTML pages, as in many cases it is very difficult to map the Web page to a tree. For Web site administrators, the usage of a tool for cleaning markup such as "HTML tidy" [Rag04] is encouraged.

A.4.2 Physical over logical content representation

The search engine must build an index of the Web pages. This index may consider only the text, or it may also consider the HTML tags by, e.g.: assigning a higher weight to terms appearing in section headers.

However, HTML markup is usually oriented to the visual characteristics of the documents; consider this HTML fragment:

```
<div align="center"><font size="+1" color="red">Important facts</font></div>
<p>Read this ...</p>
```

The visual characteristics of the phrase “Important facts” are: larger font size, red color, aligned to the center of the page. These visual aspects indicate an important block of text, but they are not visible by most search engines.

Recommendation: the following markup should be preferred:

```
<style>
h1 {
font-size: larger;
color: red;
text-align: center
}
</style>
<h1>Important facts</h1>
<p>Read this ...</p>
```

This markup separates content from representation, and visually produces the same results. For improved maintainability, the style rules can be provided in a separate file.

A.5 Web content characteristics

A.5.1 Blogging, mailing lists, forums

Blogs, Web forums and mailing list archives are large repositories of information, comprised of many small postings by individual users. They can be a useful source of information when the topic is not covered somewhere else; typical examples are technical support messages, usually describing solutions to problems with very specific software or hardware configurations.

However, sometimes individual postings are not as valuable as other pages, as they are very short, or lack clarity or factual information. Also, there is a problem with the granularity of the data, i.e.: a single posting contains little information, but the complete conversation can be valuable.

There is no easy solution for this problem, but as Web sites archiving user discussions can have hundreds of thousands of pages, they make even more important the use of a good scheduling order to try to download important pages first.

A.5.2 Duplicate detection

The prevalence of mirrored content on the web is very high. For exact duplicates, it is estimated in over 30% [CSGM99].

We calculate a hash function of the contents of the pages to avoid storing the same content twice. To account for minor variations in the Web pages, this hash function is calculated *after the page have been parsed*, so two pages with identical content but different colors or formatting will still be detected as duplicates.

Note that this method only avoids storing the duplicate Web pages, it does not prevent downloading the page, and duplicate content can generate a waste of valuable network resources.

Recommendation: in our case, the crawler does not follow links from a Web page if the Web page is found to be a duplicate; applying this heuristic, we downloaded just about 6% of duplicates.

A.6 Server application programming

A.6.1 Embedded session ids

As a way of tracking users, some Web sites embed identifiers in the URLs (e.g. `/dir/page.html;-jsessid=09A89732`). These identifiers are later used for detecting logical sessions in log analysis. From the point of view of the Web crawler, these session ids are an important source of duplicates, because the crawler cannot accuratelly tell when two pages have semantically the same content.

A Web crawler must consider session-ids. “Unless prior knowledge of his fact is given to a crawler, it can find an essentialy unbounded number of URLs to crawl at this one site alone” [EMT04] .

Typical variable names for storing session ids in the URLs include e.g.: `CFID`, `CFTOKEN`, `PHPSESSID`, `jsessionid`, etc. These variables are widely used in Web sites, and two pages that differ only in the session id are very likely to hold the same contents.

Recommendation: the crawler has a manually-built list of known session-id variables, and whenever it detects one, it changes the variable to a null value. We found that the Google crawler does not seem to download any page with a not-null value in the `PHPSESSID` variable (verified in June 2004), this can be

checked by issuing a `allinurl:phpsessid` query.

A.6.2 Repeated path components

A common mistake when encoding links is to forget to include the root directory, e.g.: referencing `a/b/c` when we want to reference `/a/b/c`. This problem can accumulate, and it is common to find URLs with path components repeated several times, such as `a/b/c/c/c/c/`; this is due to dynamic pages in which the author has mistakenly created a relative link when it should be an absolute link.

These repeated path components usually refer to the same page, and the crawler downloads repeatedly the same information.

Recommendation: some crawler implementations, such as CobWeb [dSVG+99] discard repeated components in paths, as an heuristic to avoid this problem. Our heuristic of not following links from duplicate Web pages helps to avoid this problem, so we do not check explicitly for duplicate path components.

A.6.3 Slower or erroneous pages

Dynamically generated pages are, in general, slower to transfer than static pages, some times by a factor of 10 or 100, depending on the implementation and of caching issues. In some cases this is because building the page requires querying different sources of information, and in other cases this can be due to a programming error. A slow page can waste crawler resources by forcing it to keep a connection open for a long time.

Recommendation: besides a timeout, a lower speed limit should be enforced by the crawler.

A.7 Conclusions

The practical problems of Web crawling are mostly related to bad implementations of some Web servers and Web applications. These issues are not visible until a substantial amount of pages have been downloaded from the Web, and can affect the design of the Web crawler.

Implementing a Web crawler is, in a certain way, like building a vehicle for exploring the surface of mars: you need to build the vehicle to explore the terrain, but once you have tested it, you know more about the terrain and you have to modify your vehicle's design accordingly. In that sense, the wrong implementations we have presented in this chapter are just constraints that the Web crawler designer must consider: the Web crawler must accommodate to bad coding in the same way as Web browsers do.

However, the lack of good implementations imposes costs on the design of all kinds of applications. The idea of standards is to be able to interoperate. If the standars are not respected, then the only alternatives are either design for the smallest common denominator, or design for a proprietary, fixed platform. Both alternatives are detrimental to the quality of the Web.

Bibliography

[ACGM+01] Arvind Arasu, Junghoo Cho, Hector Garcia-Molina, Andreas Paepcke, and Sriram Raghavan. Searching the Web. *ACM Transactions on Internet Technology (TOIT)*, 1(1):2–43, August 2001.

[AH00] Eytan Adar and Bernardo A. Huberman. The economics of web surfing. In *Poster Proceedings of the Ninth Conference on World Wide Web*, Amsterdam, Netherlands, May 2000.

[Ail04] Sebastien Ailleret. Larbin. http://larbin.sourceforge.net/index-eng.html, 2004. GPL software.

[AM99] J. Allen and M. Mealling. RFC 2651: The architecture of the Common Indexing Protocol. http://www.ietf.org/rfc/rfc2651.txt, 1999.

[AOP03] G. Amati, I. Ounis, and V. Plachouras. The dynamic absorbing model for the web. Technical Report TR-2003-137, Department of Computing Science, University of Glasgow, April 2003.

[APC03] Serge Abiteboul, Mihai Preda, and Gregory Cobena. Adaptive on-line page importance computation. In *Proceedings of the twelfth international conference on World Wide Web*, pages 280–290. ACM Press, 2003.

[arc04] Internet archive project. http://www.archive.org/, 2004.

[asp04] Microsoft developer network - asp resources. http://msdn.microsoft.com/asp, 2004.

[ava04] AvantGO. http://www.avantgo.com/, 2004.

[BA99] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, October 1999.

[Bar01] Albert-László Barabási. The physics of the web. *PhysicsWeb.ORG, online journal*, July 2001.

[Bar02] Albert-László Barabási. *Linked: the new science of networks*. Perseus Publishing, 2002.

[Bar04] Bradford L. Barrett. WebAlizer: log file analysis program. http://www.mrunix.net/webalizer/, 2004.

[BBDH00] Krishna Bharat, Andrei Z. Broder, Jeffrey Dean, and Monika Rauch Henzinger. A comparison of techniques to find mirrored hosts on the WWW. *Journal of the American Society of Information Science*, 51(12):1114–1122, 2000.

[BCF+03] András A. Benczúr, Károly Csalogány, Dániel Fogaras, Eszter Friedman, Tamás Sarlós, Máté Uher, and Eszter Windhager. Searching a small national domain – a preliminary report. In *Poster Proceedings of Conference on World Wide Web*, Budapest, Hungary, May 2003.

[BCGMS00] Onn Brandman, Junghoo Cho, Hector Garcia-Molina, and Narayanan Shivakumar. Crawler-friendly web servers. In *Proceedings of the Workshop on Performance and Architecture of Web Servers (PAWS)*, Santa Clara, California, USA, June 2000.

[BCS+00] Brian Brewington, George Cybenko, Raymie Stata, Krishna Bharat, and Farzin Maghoul. How dynamic is the web? In *Proceedings of the Ninth Conference on World Wide Web*, pages 257 – 276, Amsterdam, Netherlands, May 2000.

[BCSV04] Paolo Boldi, Bruno Codenotti, Massimo Santini, and Sebastiano Vigna. UbiCrawler: a scalable fully distributed Web crawler. *Software, Practice and Experience*, 34(8):711–726, 2004.

[Ber01] Michael K. Bergman. The deep web: Surfacing hidden value. *Journal of Electronic Publishing*, 7(1), 2001.

[BH98] Krishna Bharat and Monika R. Henzinger. Improved algorithms for topic distillation in a hyperlinked environment. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 104–111, Melbourne, Australia, August 1998. ACM Press, New York.

[BK97] S. Brandt and A. Kristensen. Web push as an Internet Notification Service. In *W3C workshop on push technology*, Boston, MA, USA, 1997.

[BKM+00] Andrei Broder, Ravi Kumar, Farzin Maghoul, Prabhakar Raghavan, Sridhar Rajagopalan, Raymie Stata, Andrew Tomkins, and Janet Wiener. Graph structure in the web: Experiments and models. In *Proceedings of the Ninth Conference on World Wide Web*, pages 309–320, Amsterdam, Netherlands, May 2000.

[BMN03] Sourav Bhowmick, Sanjay Kumar Madria, and Wee Keong Ng. Detecting and representing relevant web deltas in WHOWEDA. *IEEE Transactions on Knowledge and Data Engineering*, (2):423–441, 2003.

[bot04] Botspot. http://www.botspot.com/, 2004.

[Bou04] Thomas Boutell. WUsage: Web log analysis software. http://www.boutell.com/wusage/, 2004.

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

[Bro03] Terrence A. Brooks. Web search: how the Web has changed information retrieval. *Information Research*, 8(3):(paper no. 154), April 2003.

[BS00] Bettina Berendt and Myra Spiliopoulou. Analysis of navigation behaviour in web sites integrating multiple information systems. *The VLDB journal*, (9):56–75, 2000.

[BSV04] Paolo Boldi, Massimo Santini, and Sebastiano Vigna. Do your worst to make the best: Paradoxical effects in pagerank incremental computations. In *Proceedings of the third Workshop on Web Graphs (WAW)*, volume 3243 of *Lecture Notes in Computer Science*, pages 168–180, Rome, Italy, October 2004. Springer.

[Bur97] Mike Burner. Crawling towards eternity - building an archive of the world wide web. *Web Techniques*, 2(5), May 1997.

[Buz03] Marina Buzzi. Cooperative crawling. In *Proceedings of Latin American Conference on World Wide Web (LA-WEB)*, pages 209–211. IEEE CS Press, 2003.

[BV04] Paolo Boldi and Sebastiano Vigna. The webgraph framework: Compression techniques. In *Proceedings of the 13th conference on World Wide Web*, pages 595 – 602, New York, NY, USA, May 2004. ACM Press.

[BY03] Ricardo Baeza-Yates. The Web of Spain. *UPGRADE*, 3(3):82–84, 2003.

[BY04] Ricardo Baeza-Yates. Challenges in the interaction of information retrieval and natural language processing. In *Proceedings of 5th international conference on Computational Linguistics and Intelligent Text Processing (CICLing)*, volume 2945 of *Lecture Notes in Computer Science*, pages 445–456. Springer, February 2004.

[BYBKT04] Ziv Bar-Yossef, Andrei Z. Broder, Ravi Kumar, and Andrew Tomkins. Sic transit gloria telae: towards an understanding of the web's decay. In *Proceedings of the 13th conference on World Wide Web*, pages 328 – 337, New York, NY, USA, May 2004.

[BYC01] Ricardo Baeza-Yates and Carlos Castillo. Relating Web characteristics with link based Web page ranking. In *Proceedings of String Processing and Information Retrieval*, pages 21–32, Laguna San Rafael, Chile, November 2001. IEEE CS Press.

[BYC02] Ricardo Baeza-Yates and Carlos Castillo. Balancing volume, quality and freshness in web crawling. In *Soft Computing Systems - Design, Management and Applications*, pages 565–572, Santiago, Chile, 2002. IOS Press Amsterdam.

[BYC04] Ricardo Baeza-Yates and Carlos Castillo. Crawling the infinite Web: five levels are enough. In *Proceedings of the third Workshop on Web Graphs (WAW)*, volume 3243 of *Lecture Notes in Computer Science*, pages 156–167, Rome, Italy, October 2004. Springer.

[BYCSJ04] Ricardo Baeza-Yates, Carlos Castillo, and Felipe Saint-Jean. *Web Dynamics*, chapter Web Dynamics, Structure and Page Quality, pages 93–109. Springer, 2004.

[BYD04] Ricardo Baeza-Yates and Emilio Davis. Web page ranking using link attributes. In *Alternate track papers & posters of the 13th international conference on World Wide Web*, pages 328–329. ACM Press, 2004.

[BYdSV+04] Ricardo A. Baeza-Yates, Javier Ruiz del Solar, Rodrigo Verschae, Carlos Castillo, and Carlos A. Hurtado. Content-based image retrieval and characterization on specific Web collections. In *Third international conference on image and video retrieval (CIVR)*, pages 189–198, Dublin, Ireland, July 2004. Springer LNCS.

[BYP03] Ricardo Baeza-Yates and Bárbara Poblete. Evolution of the Chilean Web structure composition. In *Proceedings of Latin American Web Conference*, pages 11–13, Santiago, Chile, 2003. IEEE CS Press.

[BYP04] Ricardo Baeza-Yates and Bárbara Poblete. Dynamics of the Chilean Web structure. In *Proceedings of the 3rd International Workshop on Web Dynamics*, pages 96 – 105, New York, USA, May 2004.

[BYRN99] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval*. ACM Press / Addison-Wesley, 1999.

[BYSJC02] Ricardo Baeza-Yates, Felipe Saint-Jean, and Carlos Castillo. Web structure, dynamics and page quality. In *Proceedings of String Processing and Information Retrieval (SPIRE)*, pages 117 – 132, Lisbon, Portugal, 2002. Springer LNCS.

[CA04] Junghoo Cho and Robert Adams. Page quality: In search of an unbiased Web ranking. Technical report, UCLA Computer Science, 2004.

[Cas03] Carlos Castillo. Cooperation schemes between a web server and a web search engine. In *Proceedings of Latin American Conference on World Wide Web (LA-WEB)*, pages 212–213, Santiago, Chile, 2003. IEEE CS Press.

[CBY02] Carlos Castillo and Ricardo Baeza-Yates. A new crawling model. In *Poster proceedings of the eleventh conference on World Wide Web*, Honolulu, Hawaii, USA, May 2002. (Extended Poster).

[CCHM04] Nick Craswell, Francis Crimmins, David Hawking, and Alistair Moffat. Performance and cost tradeoffs in web search. In *Proceedings of the 15th Australasian Database Conference*, pages 161–169, Dunedin, New Zealand, January 2004.

[CCMW01] Erik Christensen, Francisco Curbera, Greg Meredith, and Sanjiva Weerawarana. WSDL: Web services description language. http://www.w3.org/TR/wsdl, 2001.

[CDR+98] Soumen Chakrabarti, Byron Dom, Prabhakar Raghavan, Sridhar Rajagopalan, David Gibson, and Jon Kleinberg. Automatic resource compilation by analyzing hyperlink structure and associated text. In *World Wide Web Conference*, pages 65–74, Brisbane, Australia, 1998. Elsevier Science Publishers B. V.

[CGM00] Junghoo Cho and Hector Garcia-Molina. Synchronizing a database to improve freshness. In *Proceedings of ACM International Conference on Management of Data (SIGMOD)*, pages 117–128, Dallas, Texas, USA, May 2000.

[CGM02] Junghoo Cho and Hector Garcia-Molina. Parallel crawlers. In *Proceedings of the eleventh international conference on World Wide Web*, pages 124–135, Honolulu, Hawaii, USA, May 2002. ACM Press.

[CGM03a] Junghoo Cho and Hector Garcia-Molina. Effective page refresh policies for web crawlers. *ACM Transactions on Database Systems*, 28(4), December 2003.

[CGM03b] Junghoo Cho and Hector Garcia-Molina. Estimating frequency of change. *ACM Transactions on Internet Technology*, 3(3), August 2003.

[CGMP98] Junghoo Cho, Hector García-Molina, and Lawrence Page. Efficient crawling through URL ordering. In *Proceedings of the seventh conference on World Wide Web*, Brisbane, Australia, April 1998.

[Cha02] Ben Charny. CNET news: Wireless Web embraces “push”. http://news.com/2100-1033-958522.html, 2002.

[Cha03] Soumen Chakrabarti. *Mining the Web*. Morgan Kaufmann Publishers, 2003.

[Cho00] Junghoo Cho. The evolution of the web and implications for an incremental crawler. In *Proceedings of 26th International Conference on Very Large Databases (VLDB)*, pages 527–534, Cairo, Egypt, September 2000. Morgan Kaufmann Publishers.

[cit04] Cite seer. http://citeseer.nj.nec.com/, 2004.

[CK97] S. Jeromy Carriére and Rick Kazman. Webquery: searching and visualizing the web through connectivity. *Computer Networks and ISDN Systems*, 29(8-13):1257–1267, September 1997.

[CMRBY04] Carlos Castillo, Mauricio Marin, Andrea Rodríguez, and Ricardo Baeza-Yates. Scheduling algorithms for Web crawling. In *Latin American Web Conference (WebMedia/LA-WEB)*, pages 10–17, Riberao Preto, Brazil, October 2004. IEEE CS Press.

[CMS99] Robert Cooley, Bamshad Mobasher, and Jaideep Srivastava. Data preparation for mining world wide web browsing patterns. *Knowledge and Information Systems*, 1(1):5–32, 1999.

[CP95] L. Catledge and J. Pitkow. Characterizing browsing behaviors on the world wide web. *Computer Networks and ISDN Systems*, 6(27), 1995.

[CSGM99] J. Cho, N. Shivakumar, and H. Garcia-Molina. Finding replicated web collections. In *ACM SIGMOD*, pages 355–366, 1999.

[CvD99] Soumen Chakrabarti, Martin van den Berg, and Byron Dom. Focused crawling: a new approach to topic-specific web resource discovery. *Computer Networks*, 31(11–16):1623–1640, 1999.

[Dac02] Lois Dacharay. WebBase. http://freesoftware.fsf.org/webbase/, 2002. GPL Software.

[Dav00] Brian D. Davison. Topical locality in the web. In *Proceedings of the 23rd annual international ACM SIGIR conference on research and development in information retrieval*, pages 272–279. ACM Press, 2000.

[Dav03] Emilio Davis. Módulo de búsqueda en texto completo para la web con un nuevo ranking estático, October 2003. Honors Thesis.

[dc04] Dublin Core Metadata Initiative. http://dublincore.org/, 2004.

[DCL+00] Michelangelo Diligenti, Frans Coetzee, Steve Lawrence, C. Lee Giles, and Marco Gori. Focused crawling using context graphs. In *Proceedings of 26th International Conference on Very Large Databases (VLDB)*, pages 527–534, Cairo, Egypt, September 2000.

[Der04] Renaud Deraison. Nessus: remote security scanner. http://www.nessus.org/, 2004.

[DFKM97] Fred Douglis, Anja Feldmann, Balachander Krishnamurthy, and Jeffrey C. Mogul. Rate of change and other metrics: a live study of the world wide web. In *USENIX Symposium on Internet Technologies and Systems*, pages 147–158, Monterey, California, USA, December 1997.

[DGM04] Michelangelo Diligenti, Marco Gori, and Marco Maggini. A unified probabilistis framework for Web page scoring systems. *IEEE Transactions on Knowledge and Data Engineering*, 16(1):4–16, 2004.

[DKM+02] Stephen Dill, Ravi Kumar, Kevin S. Mccurley, Sridhar Rajagopalan, D. Sivakumar, and Andrew Tomkins. Self-similarity in the web. *ACM Trans. Inter. Tech.*, 2(3):205–223, 2002.

[DMPS04] Michelangelo Diligenti, Marco Maggini, Filippo Maria Pucci, and Franco Scarselli. Design of a crawler with bounded bandwidth. In *Alternate track papers & posters of the 13th international conference on World Wide Web*, pages 292–293. ACM Press, 2004.

[dSVG+99] Altigran Soares da Silva, Eveline A. Veloso, Paulo Braz Golgher, Berthier A. Ribeiro-Neto, Alberto H. F. Laender, and Nivio Ziviani. Cobweb - a crawler for the brazilian web. In *Proceedings of String Processing and Information Retrieval (SPIRE)*, pages 184–191, Cancun, México, September 1999. IEEE CS Press.

[Eco02] The Economist. What does the internet look like? *The Economist*, October 2002.

[Eco03] Umberto Eco. Vegetal and mineral memory: The future of books. http://weekly.ahram.org.eg/2003/665/bo3.htm, 2003.

[EGC98] R. Weber Edward G. Coffman, Z. Liu. Optimal robot scheduling for web search engines. *Journal of Scheduling*, 1(1):15–29, 1998.

[Eic94] D. Eichmann. The RBSE spider: balancing effective search against web load. In *Proceedings of the first World Wide Web Conference*, Geneva, Switzerland, May 1994.

[EMT01] Jenny Edwards, Kevin S. McCurley, and John A. Tomlin. An adaptive model for optimizing performance of an incremental web crawler. In *Proceedings of the Tenth Conference on World Wide Web*, pages 106–113, Hong Kong, May 2001. Elsevier Science.

[EMT04] Nadav Eiron, Kevin S. McCurley, and John A. Tomlin. Ranking the web frontier. In *Proceedings of the 13th international conference on World Wide Web*, pages 309–318. ACM Press, 2004.

[ER60] Paul Erdõs and Alfred Rényi. Random graphs. *Publication of the Mathematical Institute of the Hungarian Acadamy of Science*, 5:17 – 61, 1960.

[Fal97] Sean Falton. Linux threads frequently asked questions. http://www.tldp.org/FAQ/Threads-FAQ/, January 1997.

[FGM+99] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and Tim Berners-Lee. RFC 2616 - HTTP/1.1, the hypertext transfer protocol. http://w3.org/Protocols/rfc2616/-rfc2616.html, 1999.

[FMNW03] Dennis Fetterly, Mark Manasse, Marc Najork, and Janet L. Wiener. A large-scale study of the evolution of web pages. In *Proceedings of the Twelfth Conference on World Wide Web*, pages 669 – 678, Budapest, Hungary, May 2003. ACM Press.

[GA04] Thanaa M. Ghanem and Walid G. Aref. Databases deepen the web. *Computer*, 37(1):116 – 117, 2004.

[GC01] Vijay Gupta and Roy H. Campbell. Internet search engine freshness by web server help. In *Proceedings of the Symposium on Internet Applications (SAINT)*, pages 113–119, San Diego, California, USA, 2001.

[GCGMP97] Luis Gravano, Kevin Chen-Chuan Chang, Hector Garcia-Molina, and Andreas Paepcke. STARTS: Stanford proposal for internet meta-searching. In Joan Peckham, editor, *Proceedings of International Conference on Management of Data (SIGMOD)*, pages 207–218. ACM Press, 1997.

[goo04] Google search engine. http://www.google.com/, 2004.

[gru04] Grub, a distributed crawling project. http://www.grub.org, 2004. GPL software.

[GS96] James Gwertzman and Margo Seltzer. World-wide web cache consistency. In *Proceedings of the 1996 Usenix Technical Conference*, San Diego, California, USA, January 1996.

[GS03] Daniel Gomes and Mrio J. Silva. A characterization of the portuguese web. In *Proceedings of 3rd ECDL Workshop on Web Archives*, Trondheim, Norway, August 2003.

[HA99] Bernardo A. Huberman and Lada A. Adamic. Evolutionary dynamics of the World Wide Web. *Condensed Matter*, January 1999. (paper 9901071).

[Hav02] Taher H. Haveliwala. Topic-sensitive pagerank. In *Proceedings of the Eleventh World Wide Web Conference*, pages 517–526, Honolulu, Hawaii, USA, May 2002. ACM Press.

[Hen01] Monika Henzinger. Hyperlink analysis for the web. *IEEE Internet Computing*, 5(1):45–50, 2001.

[HHMN99] Monika R. Henzinger, Allan Heydon, Michael Mitzenmacher, and Marc Najork. Measuring index quality using random walks on the Web. *Computer Networks*, 31(11–16):1291–1303, 1999.

[HHMN00] Monika Henzinger, Allan Heydon, Michael Mitzenmacher, and Marc Najork. On near–uniform url sampling. In *Proceedings of the Ninth Conference on World Wide Web*, pages 295–308, Amsterdam, Netherlands, May 2000. Elsevier Science.

[HM98] Susan Haigh and Janette Megarity. Measuring web site usage: Log file analysis. *Network Notes*, 57, 1998.

[HN99] Allan Heydon and Marc Najork. Mercator: A scalable, extensible web crawler. *World Wide Web Conference*, 2(4):219–229, April 1999.

[HPPL98] Bernardo A. Huberman, Peter L. T. Pirolli, James E. Pitkow, and Rajan M. Lukose. Strong regularities in world wide web surfing. *Science*, 280(5360):95–97, April 1998.

[htd04] HT://Dig. http://www.htdig.org/, 2004. GPL software.

[Jac02] Ian Jackson. ADNS. http://www.chiark.greenend.org.uk/∼ian/adns/, 2002.

[Jae04] Andreas Jaeger. Large file support in linux. http://www.suse.de/∼aj/linux_lfs.html, June 2004.

[JdSV+03] A. Jaimes, J. Ruiz del Solar, R. Verschae, D. Yaksic, R. Baeza-Yates, E. Davis, and C. Castillo. On the image content of the Chilean Web. In *Proceedings of Latin American Conference on World Wide Web (LA-WEB)*, pages 72–83, Santiago, Chile, 2003. IEEE CS Press.

[Kam03] Poul-Henning Kamp. OpenBSD CTM. http://www.openbsd.org/ctm.html, 2003.

[Ken70] Maurice G. Kendall. *Rank Correlation Methods*. Griffin, London, England, 1970.

[KG04] Adam Kilgarriff and Gregory Grefenstette. Introduction to the special issue on the Web as corpus. *Computational Linguistics*, 29(3):333–348, 2004.

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

[KNL98] Tuula Kapyla, Isto Niemi, and Aarno Lehtola. Towards an accessible web by applying push technology. In *Fourth ERCIM Workshop on "User Interfaces for All"*, Stockholm, Sweden, 1998.

[Koe04] Wallace Koehler. A longitudinal study of Web pages continued: a consideration of document persistence. *Information Research*, 9(2):(paper 174), January 2004.

[Kos93] Martijn Koster. Guidelines for robots writers. http://www.robotstxt.org/wc/guidelines.html, 1993.

[Kos95] Martijn Koster. Robots in the web: threat or treat ? *ConneXions*, 9(4), April 1995.

[Kos96] Martijn Koster. A standard for robot exclusion. http://www.robotstxt.org/wc/exclusion.html, 1996.

[KRR+00] R. Kumar, P. Raghavan, S. Rajagopalan, D. Sivakumar, A. Tomkins, and E. Upfal. Stochastic models for the web graph. In *Proceedings of the 41st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 57–65. IEEE CS Press, 2000.

[LBL01] Mark Levene, Jose Borges, and George Loizou. Zipf's law for web surfers. *Knowledge and Information Systems*, 3(1):120–129, 2001.

[LF98] B. Liu and E. A. Fox. Web traffic latency: Characteristics and implications. *J.UCS: Journal of Universal Computer Science*, 4(9):763–778, 1998.

[LG98] Steve Lawrence and C. Lee Giles. Searching the World Wide Web. *Science*, 280(5360):98–100, 1998.

[LG99] Steve Lawrence and C. Lee Giles. Accessibility of information on the web. *Nature*, 400(6740):107–109, 1999.

[LG00] Steve Lawrence and C. Lee Giles. Accessibility of information on the web. *Intelligence*, 11(1):32–39, 2000.

[LH98] Rajan M. Lukose and Bernardo A. Huberman. Surfing as a real option. In *Proceedings of the first international conference on Information and computation economies*, pages 45–51. ACM Press, 1998.

[Li98] Yanhong Li. Toward a qualitative search engine. *IEEE Internet Computing*, pages 24 – 29, July 1998.

[Lib00] Dan Libby. History of RSS. http://groups.yahoo.com/group/syndication/message/586, 2000.

[lib02] Libxml - the xml c library for gnome. http://www.xmlsoft.org/, 2002.

[Liu98] Binzhang Liu. Characterizing web response time. Master's thesis, Virginia State University, Blacksburg, Virginia, USA, April 1998.

[LO99] Danny B. Lange and Mitsuru Oshima. Seven good reasons for mobile agents. *Communications of the ACM*, 42(3):88–89, 1999.

[LS99] Ora Lassila and Ralph Swick. World Wide Web Consortium - RDF. http://www.w3.org/TR/REC-rdf-syntax, 1999.

[LV03] Peter Lyman and Hal R. Varian. How much information. http://www.sims.berkeley.edu/how-much-info-2003, 2003.

[LWP+01] Lipyeow Lim, Min Wang, Sriram Padmanabhan, Jeffrey Scott Vitter, and Ramesh Agarwal. Characterizing Web document change. In *Proceedings of the Second International Conference on Advances in Web-Age Information Management*, volume 2118 of *Lecture Notes in Computer Science*, pages 133–144, London, UK, July 2001. Springer-Verlag.

[LZY04] Jiming Liu, Shiwu Zhang, and Jie Yang. Characterizing web usage regularities with information foraging agents. *IEEE Transactions on Knowledge and Data Engineering*, 16(5):566 – 584, 2004.

[MAR+00] D. Menasce, V. Almeida, R. Riedi, F. Pelegrinelli, R. Fonseca, and W. Meira Jr. In search of invariants for e-business workloads. In *Proceedings of the second ACM Conference on Electronic Commerce*, Minneapolis, October 2000.

[MB98] Robert Miller and Krishna Bharat. Sphinx: A framework for creating personal, site-specific web crawlers. In *Proceedings of the seventh conference on World Wide Web*, Brisbane, Australia, April 1998.

[MB03] John Markwell and David W. Brooks. Link-rot limits the usefulness of Web-based educational materials in biochemistry and molecular biology. *Biochem. Mol. Biol. Educ.*, 31:69–72, 2003.

[McB94] Oliver A. McBryan. GENVL and WWWW: Tools for taming the web. In *Proceedings of the first World Wide Web Conference*, Geneva, Switzerland, May 1994.

[McL02] Gregory Louis McLearn. Autonomous cooperating web crawlers, 2002.

[MDFK97] Jeffrey C. Mogul, Fred Douglis, Anja Feldmann, and Balachander Krishnamurthy. Potential benefits of delta encoding and data compression for HTTP. In *Proceedings of ACM conference of Applications, Technologies, Architectures and Protocols for Computer Communication (SIGCOMM)*, pages 181–194, Cannes, France, 1997.

[Meg04] David Megginson. Simple API for XML (SAX 2.0). http://sax.sourceforge.net/, 2004.

[Mil04] Rob Miller. Websphinx, a personal, customizable web crawler. http://www-2.cs.cmu.edu/ rcm/websphinx, 2004. Apache-style licensed, open source software.

[MT93] S. P. Meyn and R. L. Tweedie. *Markov Chains and Stochastic Stability*. Springer, London, 1993.

[NCO04] Alexandros Ntoulas, Junghoo Cho, and Christopher Olston. What's new on the web?: the evolution of the web from a search engine perspective. In *Proceedings of the 13th conference on World Wide Web*, pages 1 – 12, New York, NY, USA, May 2004. ACM Press.

[nic04] Network Information Center, NIC Chile. http://www.nic.cl/, 2004.

[Nie03] Jakob Nielsen. Statistics for traffic referred by search engines and navigation directories to useit. http://www.useit.com/about/searchreferrals.html, 2003.

[NW01] Marc Najork and Janet L. Wiener. Breadth-first crawling yields high-quality pages. In *Proceedings of the Tenth Conference on World Wide Web*, pages 114–118, Hong Kong, May 2001. Elsevier Science.

[Pat00] Nick Paton. Information overload. *The Guardian*, 2000. (Gallup/Institute for the Future study).

[Pat04] Anna Patterson. Why writing your own search engine is hard. *ACM Queue*, pages 49 – 53, April 2004.

[PBMW98] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The Pagerank citation algorithm: bringing order to the web. Technical report, Stanford Digital Library Technologies Project, 1998.

[Per04] R. Scott Perry. DNS report. http://www.dnsreport.com/, 2004.

[PFL+02] David M. Pennock, Gary W. Flake, Steve Lawrence, Eric J. Glover, and C. Lee Giles. Winners don't take all: Characterizing the competition for links on the web. *Proceedings of the National Academy of Sciences*, 99(8):5207–5211, April 2002.

[php04] Php - the hypertext preprocessor. http://www.php.net/, 2004.

[Pin94] Brian Pinkerton. Finding what people want: Experiences with the WebCrawler. In *Proceedings of the first World Wide Web Conference*, Geneva, Switzerland, May 1994.

[PlaBC] Plato. *Phaedrus*. 360 BC.

[POA03] V. Plachouras, I. Ounis, and G. Amati. A Utility-oriented Hyperlink Analysis Model for the Web. In *Proceedings of the First Latin Web Conference*, pages 123–131. IEEE Press, 2003.

[Pro04] ProChile. Estadísticas de exportaciones (statistics of exports). http://www.prochile.cl/-servicios/estadisticas/exportacion.php, 2004.

[Rag04] Dave Raggett. HTML tidy. http://tidy.sourceforge.net/, 2004. GPL software.

[RAW+02] Andreas Rauber, Andreas Aschenbrenner, Oliver Witvoet, Robert M. Bruckner, and Max Kaiser. Uncovering information hidden in web archives. *D-Lib Magazine*, 8(12), 2002.

[RGM01] Sriram Raghavan and Hector Garcia-Molina. Crawling the hidden web. In *Proceedings of the Twenty-seventh International Conference on Very Large Databases (VLDB)*, pages 129–138, Rome, Italy, 2001. Morgan Kaufmann.

[RM02] Knut Magne Risvik and Rolf Michelsen. Search engines and web dynamics. *Computer Networks*, 39(3), June 2002.

[RRDB02] J.F. Reschke, S. Reddy, J. Davis, and A. Babich. DASL - DAV searching and locating protocol. http://www.webdav.org/dasl/, 2002.

[Sal71] Gerard Salton. *The SMART retrieval system - experiments in automatic document processing*. Prentice-Hall, 1971.

[SB88] Gerard Salton and Christopher Buckley. Term-weighting approaches in automatic text retrieval. *Information Processing and Management: an International Journal*, 24(5):513–523, 1988.

[SM02] Torsten Suel and Nasir Memon. *Lossless compression handbook*, chapter Algorithms for delta compression and remote file synchronization. Academic Press, 2002.

[Sob03] Markus Sobek. Google dance – the index update of the Google search engine. http://dance.efactory.de/, 2003.

[Spi03] Diomidis Spinellis. The decay and failures of web references. *Communications of the ACM*, 46(1):71–77, January 2003.

[SS02] Vladislav Shkapenyuk and Torsten Suel. Design and implementation of a high-performance distributed web crawler. In *Proceedings of the 18th International Conference on Data Engineering (ICDE)*, pages 357 – 368, San Jose, California, February 2002. IEEE CS Press.

[SS03] Anubhav Savant and Torsten Suel. Server-friendly delta compression for efficient web access. In *Proceedings of the Eighth International Workshop on Web Content Caching and Distribution (WCW)*, September 2003.

[SS04] Danny Sullivan and Chris Sherman. Search Engine Watch reports. http://www.searchenginewatch.com/reports/, 2004.

[Sta03] StatMarket. Search engine referrals nearly double worldwide. http://websidestory.com/pressroom/pressreleases.html?id=181, 2003.

[SY01] Torsten Suel and Jun Yuan. Compressing the graph structure of the Web. In *Proceedings of the Data Compression Conference DCC*, pages 213 – 222. IEEE CS Press, 2001.

[TG97] Linda Tauscher and Saul Greenberg. Revisitation patterns in world wide web navigation. In *Proceedings of the Conference on Human Factors in Computing Systems CHI'97*, 1997.

[TGM93] Anthony Tomasic and Hector Garcia-Molina. Performance of inverted indices in shared-nothing distributed text document informatioon retrieval systems. In *Proceedings of the second international conference on Parallel and distributed information systems*, pages 8–17. IEEE Computer Society Press, 1993.

[The02] The Economist. Country Profiles, 2002.

[TK02] Pang-Ning Tan and Vipin Kumar. Discovery of web robots session based on their navigational patterns. *Data Mining and Knowledge discovery*, 6(1):9–35, 2002.

[TLNJ01] Jerome Talim, Zhen Liu, Philippe Nain, and Edward G. Coffman Jr. Controlling the robots of web search engines. In *Proceedings of ACM Joint International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS/Performance)*, pages 236–244, Cambridge, Massachusetts, USA, June 2001.

[Tom03] John A. Tomlin. A new paradigm for ranking pages on the world wide web. In *Proceedings of the Twelfth Conference on World Wide Web*, pages 350–355, Budapest, Hungary, May 2003. ACM Press.

[TP03] Andrew Tridgell and Martin Pool. RSYNC: fast incremental file transfer. http://samba.anu.edu.au/rsync/, 2003.

[TR99] W. Theilmann and K. Rothermel. Maintaining specialized search engines through mobile filter agents. In M. Klusch, O. Shehory, and G. Weiß, editors, *Proc. 3rd International Workshop on Cooperative Information Agents (CIA'99)*, pages 197–208, Uppsala, Sweden, 1999. Springer-Verlag: Heidelberg, Germany.

[TT04] Doru Tanasa and Brigitte Trousse. Advanced data preprocessing for intersites Web usage mining. *IEEE Intelligent Systems*, 19(2):59–65, 2004.

[Tur04] Stephen Turner. Analog: WWW log file analysis. http://www.analog.cx/, 2004.

[Uni02] United Nations. Population Division, 2002.

[Uni03] United Nations. Human Development Reports, 2003.

[VdMG+00] Eveline A. Veloso, Edleno de Moura, P. Golgher, A. da Silva, R. Almeida, A. Laender, B. Ribeiro-Neto, and Nivio Ziviani. Um retrato da web brasileira. In *Proceedings of Simposio Brasileiro de Computacao*, Curitiba, Brasil, July 2000.

[vHGH+97] Arthur van Hoff, John Giannandrea, Mark Hapner, Steve Carter, and Milo Medin. DRP - distribution and replication protocol. http://www.w3.org/TR/NOTE-drp, 1997.

[web04a] WebDAV resources. http://www.webdav.org/, 2004.

[web04b] WebTrends corporation. http://www.webtrends.com/, 2004.

[WEDM00] Larry Wall, Paul Eggert, Wayne Davison, and David MacKenzie. GNU patch. http://www.gnu.org/software/patch/patch.html, 2000.

[wik04] Library of Alexandria. http://en.wikipedia.org/wiki/Library_of_Alexandria, 2004. (Article on the Wikipedia).

[WMB99] Ian H. Witten, Alistair Moffat, and Timothy C. Bell. *Managing Gigabytes*. Morgan Kaufmann Publishing, 1999.

[YL96] Budi Yuwono and Dik Lun Lee. Search and ranking algorithms for locating resources on the world wide web. In *Proceedings of the twelfth International Conference on Data Engineering (ICDE)*, pages 164–171, Washington, DC, USA, February 1996. IEEE CS Press.

[ZYD02] Demetrios Zeinalipour-Yazti and Marios D. Dikaiakos. Design and implementation of a distributed crawler and filtering processor. In *Proceedings of the fifth Next Generation Information Technologies and Systems (NGITS)*, volume 2382 of *Lecture Notes in Computer Science*, pages 58–74, Caesarea, Israel, June 2002. Springer.