Scheduling Algorithms for Web Crawling

Carlos Castillo
Center for Web Research
Universidad de Chile
ccastill@dcc.uchile.cl

Mauricio Marin
Center for Web Research
Universidad de Magallanes
mauricio.marin@umag.cl

Andrea Rodriguez
Center for Web Research
Universidad de Concepción
andrea@udec.cl

Ricardo Baeza-Yates
Center for Web Research
Universidad de Chile
rbaeza@dcc.uchile.cl

Abstract

This article presents a comparative study of strategies for Web crawling. We show that a combination of breadth-first ordering with the largest sites first is a practical alternative since it is fast, simple to implement, and able to retrieve the best ranked pages at a rate that is closer to the optimal than other alternatives. Our study was performed on a large sample of the Chilean Web which was crawled by using simulators, so that all strategies were compared under the same conditions, and actual crawls to validate our conclusions. We also explored the effects of large scale parallelism in the page retrieval task and multiple-page requests in a single connection for effective amortization of latency times.

1. Introduction

Web search is currently generating more than 13% of the traffic to Web sites [42]. The main problem search engines have to deal with is the size of the Web, which currently is in the order of thousands of millions of pages. This large size induces a low coverage, with no search engine indexing more than one third of the publicly available Web [31].

We would like to develop a scheduling policy for downloading pages from the Web which guarantees that, even if we do not download all of the known pages, we still download the most "valuable" ones. As the number of pages grows, it will be increasingly important to download the "better" ones first, as it will be impossible to download them all.

The main contributions of this paper are:

- We propose several strategies for Web page ordering during a Web crawl, and compare them using a Web crawler simulator.
- We choose a strategy which is very competitive and efficient.
- We test this strategy using a real Web crawler and show that it achieves the objective of downloading important pages early in the crawl.

Next section presents previous work on Web crawling, and the rest of this paper is organized as follows: section 3 outlines the problems of Web crawling, section 4 presents our experimental framework, sections 5 and 6 compare different scheduling policies. In section 7 we test one of these policies using a real Web crawler. The last section presents our conclusions.

2. Previous work

Web crawlers are a central part of search engines, and details on their crawling algorithms are kept as business secrets. When algorithms are published, there is often an important lack of details that prevents other from reproduce the work. There are also emerging concerns about "search engine spamming", which prevent major search engines from publishing their ranking algorithms.

However, there are some exceptions: some descriptions of crawlers (in chronological order) are: RBSE [26], the WebCrawler [39], the World Wide Web Worm [35], the crawler of the Internet Archive [11], the personal search agent SPHINK [36], an early version of the Google crawler [8], the CobWeb [20], Mercator, a modular crawler [28], Salticus [10], the WebFountain incremental crawler [25], and the WIRE crawler [5] used for this research. Descriptions of crawler architectures include a parallel crawler architecture by Cho [16] and a general crawler architecture described by Chakrabarti [12]. Some crawlers released under the GNU public license include Larbin [3], WebBase [21] and HT://Dig [2].

Besides architectural issues, studies about Web crawling have focused on parallelism [16, 41], discovery and control of crawlers for Web site administrators [44, 43, 1, 29], accessing content behind forms (the "hidden" web) [40], detecting mirrors [18], keeping the freshness of the search engine copy high [15, 19], long-term scheduling [37, 24, 17] and focused crawling [22]. There have been also studies on characteristics of the Web, which are relevant to the crawler performance, such as detecting communities [30], characterizing server response time [33], studying the distribution of web page changes [13, 23, 7, 32], studying the macroscopic web structure [9, 4], and proposing protocols for web servers to cooperate with crawlers [6].

3. The problem of Web crawling

A Web crawler works by starting with a set of "seed" pages, downloading those pages and extracting links from them, and recursively following those links. Even if we disregard the technical difficulties of transferring, processing and storing a very large amount of data, there is still an important research problem, namely, the problem of scheduling the visits to the un-visited pages (we are not considering in this article the case of re-visits).

At a first glance, it might seem that this scheduling problem has a trivial solution. If a crawler needs to download a series of pages whose file sizes in bytes are P_i, and has B bytes per second of available bandwidth for doing it, then it should download all the pages simultaneously at a speed proportional to the size of each page:

$$B_i = \frac{P_i}{T*} \quad ; \quad T* = \frac{\sum Pi}{B} \tag{1}$$

T^* is the optimal time to use all the available bandwidth. This optimal scenario is depicted in Figure 1.

However, there are many restrictions that forbid this optimal scenario. The main restriction of any scheduling policy is that it must avoid overloading Web sites: a Web crawler can impose a substantial amount of work on a Web server, specially if it opens many simultaneous connections for downloading [29]. It is customary that a Web crawler does not download more than one page from the same Web site at a time, and that it waits between 30 and 60 seconds between accesses. This, together with the fact that Web sites have usually a bandwidth B_i^{MAX} that is lower than the crawler bandwidth B, originate download time-lines similar to the one shown in Figure 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 we must wait between accesses to a Web site (in the figure, pages $1-2$ and $4-5$ belong to the same site).

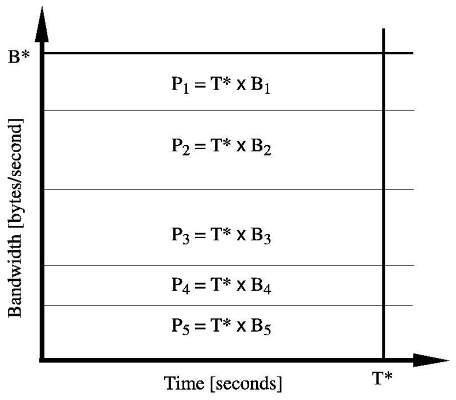

Figure 1. Optimal scenario for downloads.

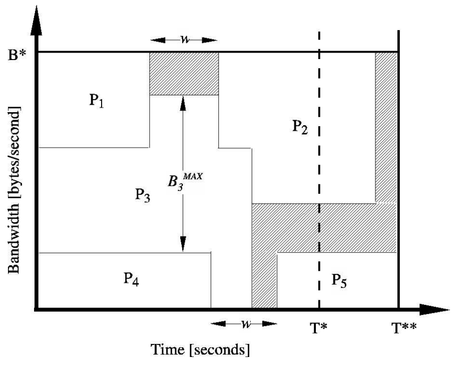

Figure 2. A more realistic scenario; the hatched portion is wasted bandwidth.

To overcome the problems shown in Figure 2, it is clear that we should try to download 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, 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 good use of the network bandwidth.

There is another serious practical restriction: the HTTP request has latency, and the latency time can be over 25% of the total time of the request [34]. 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.

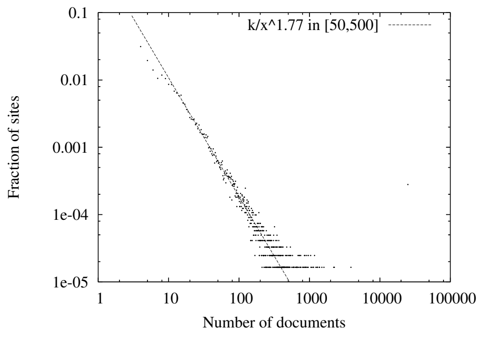

Figure 3. Distribution of site sizes.

4. Experimental setup

We crawled 3.5 million pages in April 2004 from over 50,000 Web sites using the WIRE crawler [5]. We estimate that this is more than 90% of the publicly available Chilean Web pages. We restricted the crawler to download at most 25,000 pages from each Web site.

Using this data, we created a Web graph, and ran a simulator on this graph using using different scheduling policies. This allowed us to compare different strategies under exactly the same conditions. This simulator models page transfers and bandwidth saturation of the Internet link when too many connections are in process.

We considered a number of scheduling strategies whose design is based on a heap priority queue with nodes representing sites. For each site-node we have another heap representing the pages in the Web site, as depicted in Figure 4.

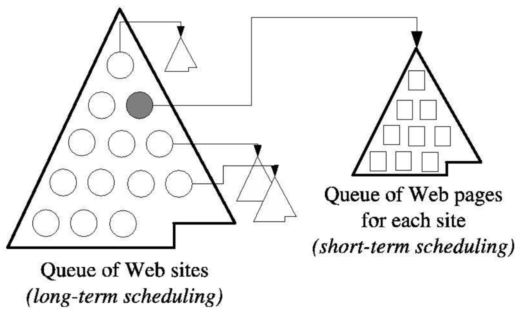

Figure 4. Queues used for the scheduling.

The scheduling is divided in two parts: the policy for ordering the queue of Web sites (long-term scheduling) and the policy for ordering the queues of Web pages (short-term scheduling).

At each simulation step, the scheduler chooses the topmost Website from the queue of Web sites and sends this site's information to a module that will simulate downloading pages from the Website. The parameters for our different scheduling policies are the following:

- The interval w in seconds between connections to a single Web site.
- The number of pages k downloaded for each connection when re-using HTTP connections.
- The number r of simultaneous connections to different Web sites, i.e.: the degree of parallelization. Although we used a large degree of parallelization, we restricted the robots to never establish more than 1 connection to a Web site at a given time.

4.1. Interval between connections (w)

The first proposal for the interval w between connections was given by Koster [29]: $w = 60$ seconds. 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 the entire Web site; also, we would be permanently using one of the processes or threads from that Web server. This does not seems acceptable.

Today, it is common to consider less than $w = 15$ seconds impolite. Anecdotal evidence from access logs shows that access intervals from known crawlers vary between 20 seconds and 3–4 minutes. We use $w = 15$ seconds in our simulations and real-life experiments.

4.2. Number of pages per connection (k)

Most Web crawlers 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 in the quality of the scheduling.

The protocol for keeping the connection open was introduced as the `Keep-alive` header in HTTP/1.1 [27]; the default configuration of the Apache web server enables this feature by default, and allows for a maximum of 100 objects downloaded per request, with a timeout of 15 seconds between requests, so when using $k > 1$ in practice, we should also set $w \leq 15$ to prevent the server from closing the connection. This is another reason to set $w = 15$ during the experiments.

4.3. 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 from the same computer 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 Web sites that have pages which have not yet been visited. If this 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. This happens at the end of a large crawl, when we have covered a substantial fraction of Web pages, or by the end of a batch of pages, when downloading pages grouped by batches. In the later case, the batch of pages must be carefully built to include pages from as many Web sites as possible; this should be a primary concern when parallelism is considered.

5. Long-term scheduling

The complete crawl on the real Chilean Web takes about 8 days, so, apart from comparing strategies under exactly the same scenario, it is much more efficient to test many strategies using the crawling simulator.

Retrieval real-time for Web pages is simulated by considering the real-crawl observed latency, transfer rate, page size for every downloaded page, and (possible) saturation of bandwidth in accordance with the number of active connections at a given moment of the simulation. We also validated our results with measures from the real 8-days crawl.

As mentioned, for evaluating the different strategies, we calculated before hand 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 those values "oracle" ones since in practice they are not known until the complete crawl is finished. The strategies which are able to reach faster values close to the target total value 1.0 are considered the most efficient ones.

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 which are able to reach values close to the target total value 1.0 faster are considered the most efficient ones.

All of the considered strategies are based on the two-level scheduling shown in Figure 4. We named our strategies *Optimal*, *Depth*, *Length*, *Batch* and *Partial*:

Optimal Under this strategy, the crawler visits the pages in Pagerank order. To do that, it asks for the Pagerank to an "oracle" which knows the final value of the Pagerank for each page. Note that during the crawl, a crawler does not have access to this information, as it only knows a portion of the Web graph and therefore can only estimate the final Pagerank.

At a first glance, a way of approximating this strategy in practice could be to use the Pagerank obtained on a previous crawl of the Web; however, this is not a good estimator: Cho and Adams [14] report that the average relative error for estimating the Pagerank four months ahead is about 78%. Also, a study by Ntoulas *et. al* [38] 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".

Depth Under this strategy, the crawler visits the pages in breadth-first ordering. Web page heaps are kept in such a way that the pages with the smallest depth in the Web graph are the ones with the greatest heap's priorities. This is the same strategy described by Najork and Wiener [37], which in their experiments showed to capture high-quality pages first.

Length This strategy sorts the pages on each Web site according to depth, but Web sites are ordered by considering the number of pages in the respective queue as the priority for each Web site. Nodes in the sites heap are re-arranged dynamically to follow changes in their priorities as new pages are found.

Batch The crawler downloads a batch of K pages and once all of those pages are retrieved, the Pagerank algorithm is run on the subset of known Web pages (i.e.: no oracle Pagerank values are used). The next batch is formed with K pages sorted by the Pagerank value at that moment. This is like the strategy described by Cho *et al.* [17], except that in their case $K = 1$ and Pagerank is re-calculated every time a new URL is downloaded. This can be very slow in practice.

Partial The crawler executes the Pagerank algorithm every time K pages are retrieved, but between re-calculations, new pages are given a "temporary" Pagerank equivalent to the sum of the normalized rankings of the pages that point to them. A newly discovered page could be crawled as soon as it is found, as the page to be downloaded is always the page with the highest partial Pagerank.

Initial (home-pages) are assumed to have the same Pagerank value at the start. In the case of batches, or partial re-calculations, we performed experiments for $K = 10,000$, $K = 50,000$ and $K = 100,000$ pages. 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 band-

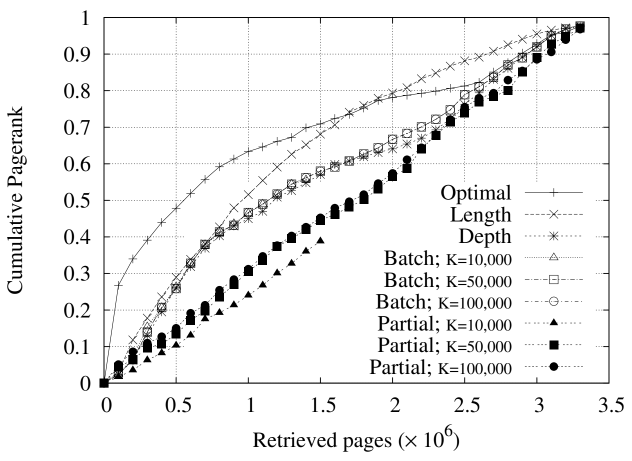

Figure 5. Cumulative Pagerank vs retrieved pages, case for $r = 1$, one robot.

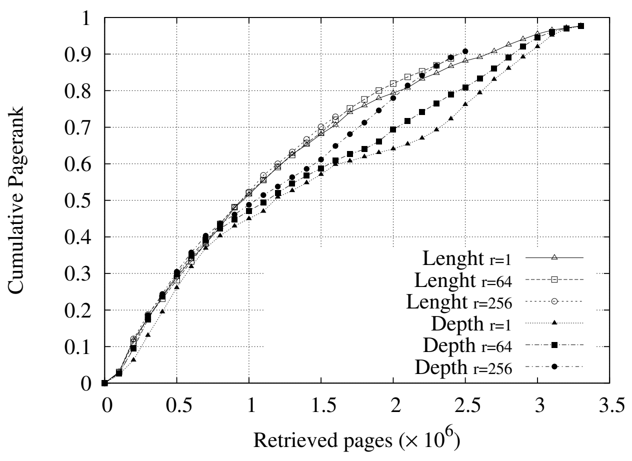

Figure 6. Cumulative Pagerank vs retrieved pages. Case for 1, 64 and 256 robots.

width of the Web crawler B is larger than any of the maximum bandwidths of the Web servers B_i^{MAX}.

The results for the cumulative sum of the Pagerank values considering one robot ($r = 1$, a single network connection at a time; we show the effect of parallelizing the downloads with more robots in the following figures) are shown in Figure 5. Note that a random policy, not shown in the Figure, would produce a straight line.

Regarding the results, the *Optimal* strategy is too greedy if a high coverage (more than 75%) is expected, as it downloads all the pages with good Pagerank very early, but later it has only a few Web sites to choose from, so it is then surpassed by other strategies due to the restrictions of Web crawling.

The strategies *Depth* and *Batch* are similar, with a small difference in favor of *Batch*. This is consistent with the findings of Cho *et al.* [17]. Finally, the *Partial* strategy performs poorly and close to random. This means that the suggested approximation of Pagerank is not a good approximation.

The *Length* strategy is better than the strategies based on periodical Pagerank calculations and *Depth*. Notice that the implementation for the *Length* policy, and its processing requirements are significantly lower than for the strategies which calculate link-based ranking. In particular, calculating periodical Pageranks takes a significant amount of resources in running time and space.

Figure 6 shows the effect of increasing the number of robots to $r = 64$ and $r = 256$. Observing the rate of growth of the cumulative Pagerank sum, the results show that *Length* is more stable than *Depth*, though it improves as the number of robots increases.

Table 1 shows the effects in retrieval time when we increase the number of robots for different bandwidths. The results shows how bandwidth saturation makes pointless the use of several robots.

Bandwidth	r=1	r=64	r=256
20000000	1.0	54.6	220.1
2000000	1.0	54.3	204.3
200000	1.0	43.0	114.1
20000	1.0	27.0	83.3
2000	0.7	3.8	16.0
200	0.2	1.6	3.0

Table 1. Predicted speed-ups for parallelism.

6. Short-term scheduling

While crawling, specially in distributed crawling architectures, it is typical to work by downloading groups of pages. During that period of time, no care is taken about Pagerank values. The problem is that on a typical batch, most of the pages are located in a reduced number of sites: the distribution is similar that for the whole Web (Figure 3).

Even if a batch involves a large number of Web sites, if a large fraction of the Web sites has very few pages, then quickly many of the robots will be idle, as the number of connections r is always smaller than the number of different available Web sites. Figure 7 illustrates this problem, showing the effective number of robots involved in the retrieval of a typical batch in the middle of the crawl for scenarios with large (20 KB/s) and small (2 KB/s) bandwidth. A solution is to increase k and let robots get more than one page every they visit a site, in the Figure, $k = 100$.

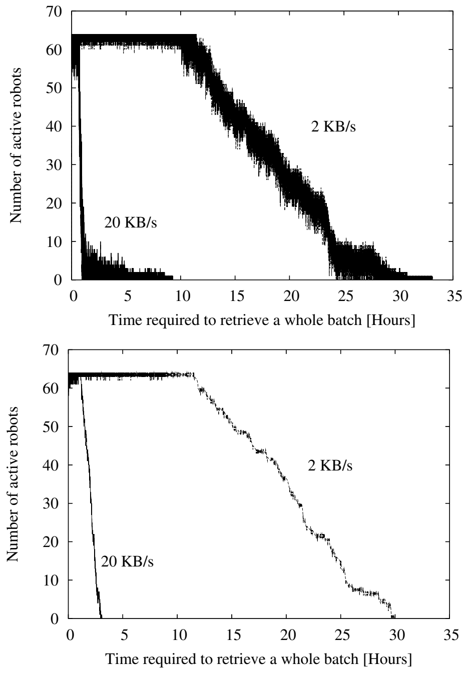

Figure 7. Number of active robots vs time: one page per connection (top) and 100 pages per connection (bottom)

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.

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 which are downloaded in the same connection is the same as 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.

7. Application: downloading the real Web

We started with a list of Web sites registered with the Chilean Network Information Center, and ran the WIRE crawler during 8 days with the *Length* strategy. We visited over 50,000 Web sites with 3 million pages and were able to download successfully 2.4 million of them (80%). In total, 57 Gb of data were downloaded.

We ran the crawler in batches of up to 100,000 pages, using up to $r = 1000$ simultaneous network connections, with $w = 15$ seconds between accesses to the same Web site and $k = 1$ download per connection. The crawler used both the `robots.txt` file and meta-tags in Web pages according to the robot exclusion protocol [29].

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 8.

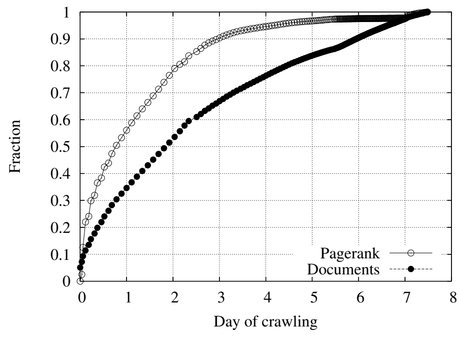

Figure 8. Cumulative fraction of pages downloaded and cumulative Pagerank of the downloaded portion of the Web.

We can see that by the end of day 2, 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 4, 80% of the pages were downloaded, and more than 95% of the Pagerank, so in general this approach leads to "good" pages early in the crawl. In fact, the average Pagerank decreased dramatically after a few days (Figure 9), which is consistent with the findings of Najork and Wiener [37].

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 10. Note

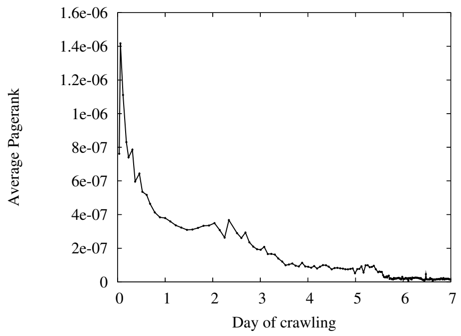

Figure 9. Average Pagerank vs day of crawl.

that home pages have, *in average*, a low Pagerank as there are many of them with very few or no in-links: we were able to found them only by their registration under the `.cl` top-level domain database.

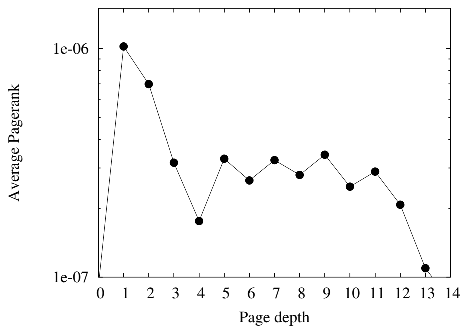

Figure 10. Average Pagerank vs page depth.

8. Conclusions

The restrictions involved in Web crawling make this problem a very interesting one. In particular, a strategy which uses an "oracle" to detect pages with high Pagerank early, is not very good because as the crawl advances, few Web sites are available and inefficiencies arise.

For long-term scheduling, our results show that a really simple crawling strategy, such as the one we called *Length*, is good enough for efficiently retrieving a large portion of the Chilean Web. As the idea is to try to keep as many Web sites active as possible, this strategy prioritizes Web sites based on the number of pages available from them, such that is avoids exhausting Web sites too early.

Also our simulation results show that attempting to retrieve as many pages from a given site ($k >> 1$), allows one 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.

Experiments with a real crawl using the *Length* 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.

References

[1] Robotcop. www.robotcop.org, 2002.

[2] HT://Dig. http://www.htdig.org/, 2004. GPL software.

[3] S. Ailleret. Larbin. http://larbin.sourceforge.net/index-eng.html, 2004. GPL software.

[4] R. Baeza-Yates and C. 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.

[5] R. Baeza-Yates and C. Castillo. Balancing volume, quality and freshness in web crawling. In *Soft Computing Systems - Design, Management and Applications*, pages 565–572. IOS Press, 2002.

[6] O. Brandman, J. Cho, H. Garcia-Molina, and N. Shivakumar. Crawler-friendly web servers. In *Proceedings of the Workshop on Performance and Architecture of Web Servers (PAWS)*, Santa Clara, California, USA, June 2000.

[7] B. Brewington, G. Cybenko, R. Stata, K. Bharat, and F. Maghoul. How dynamic is the web? In *Proceedings of the Ninth Conference on World Wide Web*, pages 257 – 276, Amsterdam, Netherlands, May 2000.

[8] S. Brin and L. Page. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 30(1–7):107–117, April 1998.

[9] A. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and J. 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.

[10] R. D. Burke. Salticus: guided crawling for personal digital libraries. In *Proceedings of the first ACM/IEEE-CS joint conference on Digital Libraries*, pages 88–89, Roanoke, Virginia, June 2001.

[11] M. Burner. Crawling towards eternity - building an archive of the world wide web. *Web Techniques*, 2(5), May 1997.

[12] S. Chakrabarti. *Mining the Web*. Morgan Kaufmann, 2003.

[13] J. 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.

[14] J. Cho and R. Adams. Page quality: In search of an unbiased Web ranking. Technical report, UCLA Computer Science, 2004.

[15] J. Cho and H. 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.

[16] J. Cho and H. 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.

[17] J. Cho, H. García-Molina, and L. Page. Efficient crawling through URL ordering. In *Proceedings of the seventh conference on World Wide Web*, Brisbane, Australia, April 1998.

[18] J. Cho, N. Shivakumar, and H. Garcia-Molina. Finding replicated web collections. In *ACM SIGMOD*, pages 355–366, 1999.

[19] A. Czumaj, I. Finch, L. Gasieniec, A. Gibbons, P. Leng, W. Rytter, and M. Zito. Efficient Web searching using temporal factors. *Theoretical Computer Science*, 262(1–2):569–582, 2001.

[20] A. S. da Silva, E. A. Veloso, P. B. Golgher, B. A. Ribeiro-Neto, A. H. F. Laender, and N. 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.

[21] L. Dacharay. WebBase. http://freesoftware.fsf.org/webbase/, 2002. GPL Software.

[22] M. Diligenti, F. Coetzee, S. Lawrence, C. L. Giles, and M. Gori. Focused crawling using context graphs. In *Proceedings of 26th International Conference on Very Large Databases (VLDB)*, pages 527–534, Cairo, Egypt, September 2000.

[23] F. Douglis, A. Feldmann, B. Krishnamurthy, and J. 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.

[24] R. W. Edward G. Coffman, Z. Liu. Optimal robot scheduling for web search engines. *Journal of Scheduling*, 1(1):15–29, 1998.

[25] J. Edwards, K. S. McCurley, and J. 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.

[26] D. Eichmann. The RBSE spider: balancing effective search against web load. In *Proceedings of the first World Wide Web Conference*, Geneva, Switzerland, May 1994.

[27] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. RFC 2616 - HTTP/1.1, the hypertext transfer protocol. http://w3.org/Protocols/rfc2616/rfc2616.html, 1999.

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

[29] M. Koster. Robots in the web: threat or treat ? *ConneXions*, 9(4), April 1995.

[30] R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins. Trawling the Web for emerging cyber-communities. *Computer Networks*, 31(11–16):1481–1493, 1999.

[31] S. Lawrence and C. L. Giles. Searching the World Wide Web. *Science*, 280(5360):98–100, 1998.

[32] L. Lim, M. Wang, S. Padmanabhan, J. S. Vitter, and R. 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.

[33] B. Liu. Characterizing web response time. Master's thesis, Virginia State University, Blacksburg, Virginia, USA, April 1998.

[34] B. Liu and E. A. Fox. Web traffic latency: Characteristics and implications. *J.UCS: Journal of Universal Computer Science*, 4(9):763–778, 1998.

[35] O. A. McBryan. GENVL and WWWW: Tools for taming the web. In *Proceedings of the first World Wide Web Conference*, Geneva, Switzerland, May 1994.

[36] R. Miller and K. 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.

[37] M. Najork and J. 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.

[38] A. Ntoulas, J. Cho, and C. 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.

[39] B. Pinkerton. Finding what people want: Experiences with the WebCrawler. In *Proceedings of the first World Wide Web Conference*, Geneva, Switzerland, May 1994.

[40] S. Raghavan and H. 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.

[41] V. Shkapenyuk and T. 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.

[42] StatMarket. Search engine referrals nearly double worldwide. http://websidestory.com/pressroom/-pressreleases.html?id=181, 2003.

[43] J. Talim, Z. Liu, P. Nain, and E. G. C. 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.

[44] P.-N. Tan and V. Kumar. Discovery of web robots session based on their navigational patterns. *Data Mining and Knowledge discovery*, 6(1):9–35, 2002.