Chapter 6

Scheduling Algorithms for Web Crawling

In the previous chapters, we have described the model, architecture and implementation of our Web crawler. In this chapter, we deal with the algorithms for scheduling the visits to the Web pages.

We started with a large sample of the Chilean Web, which 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 6.1 presents the problems of Web crawling scheduling and Section 6.2 introduces our experimental framework. Sections 6.3 and 6.4 compare different scheduling policies for long- and short-term scheduling. In Section 6.5 we test one of these policies using a real Web crawler. The last section presents our conclusions.

The results presented here were obtained in a joint work with Mauricio Marin, Andrea Rodriguez and Ricardo Baeza-Yates [CMRBY04].

6.1 The problem of crawler scheduling

We consider a Web crawler which 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:

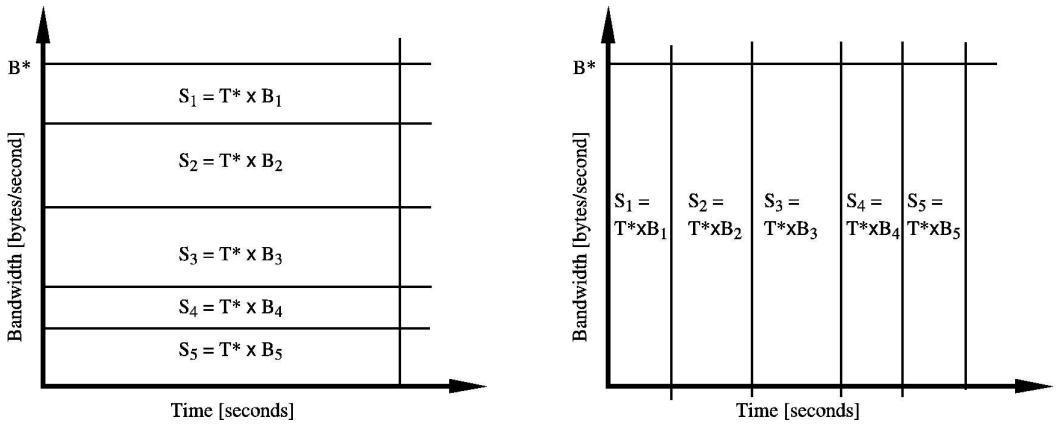
$$B_p = \frac{S_p}{T^*} \tag{6.1}$$

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

$$T^* = \frac{\sum S_p}{B} \tag{6.2}$$

This scenario is depicted in Figure 6.1a.

However, there are many restrictions that forbid this scenario. One restriction is that a scheduling policy must avoid overloading Web sites, enforcing a politeness policy as described in Section **??**: 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.



(a) Full parallelization

(b) Full serialization

Figure 6.1: 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* $\times$ t*ime*.

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 6.1b. 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 6.2. 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 (in the figure, pages $1-2$ and $4-5$ belong to the same site, and the crawler waits w seconds between them).

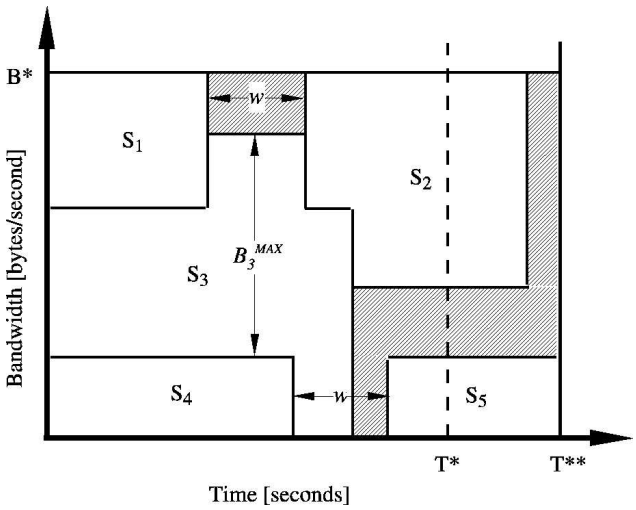


Figure 6.2: A more realistic download time line for Web crawlers. The hatched portion is wasted bandwidth due to the constraints in the scheduling policy. The optimal time T^* is not achieved.

To overcome the problems shown in Figure 6.2, 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 6.3, 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.

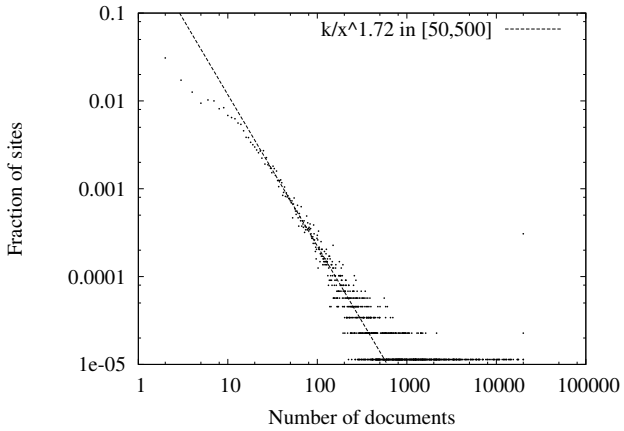


Figure 6.3: Distribution of site sizes.

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.

6.2 Experimental setup

We downloaded 3.5 million pages in April 2004 from over 50,000 Web sites using the WIRE crawler [BYC02]. Based on previous studies of the Chilean Web [BYP03], we estimate that this sample represents accurately a large fraction of the Chilean Web. We restricted the crawler to download only the first 25,000 pages from each Website using breadth-first order.

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 **??**.
- The page sizes, which were obtained during the crawl used to build the Web graph.

We considered a number of scheduling strategies, with a 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 6.4.

[1] The crawler simulator used for this experiment was developed by Mauricio Marin and Andrea Rodriguez, and it is not described here in detail, as it is not part of the work of this thesis.

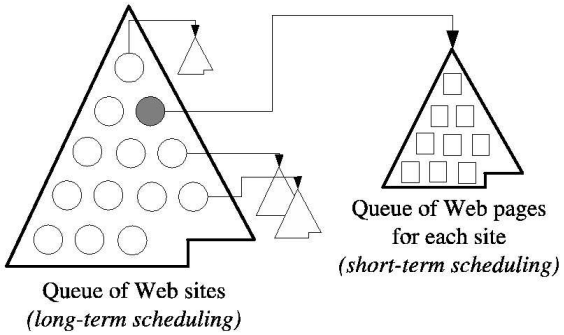


Figure 6.4: We tested the scheduling policies using a structure with two levels: one for Web sites and one for Web pages.

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

The parameters for our different scheduling policies are the following:

- 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 establish more than one connection to a Web site at a given time.

6.2.1 Interval between connections (w)

As noted in Section **??**, 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.

Note that the total time of a page download is in most cases under 10 seconds [Liu98], so this waiting time (plus the latency time) is usually larger than the actual transfer time. This makes the situation even more difficult than what was shown in Figure 6.2, as the time spent waiting cannot be amortized effectively.

6.2.2 Number of pages per connection (c)

Most Web crawlers download only one page per each connection, and don't 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 [FGM$^+$99]; the default 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.

6.2.3 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 which 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. This is specially 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 the 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.

6.3 Long-term scheduling

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

Retrieval real-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 moment 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 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 6.4. We named our strategies *Optimal*, *Depth*, *Length*, *Batch* and *Partial*. Their description is the following:

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. The next page to be retrieved is the one which is on top of the two heaps, namely the page that have the greatest oracle's Pagerank in the short-term Web pages heap, which in turn makes the corresponding site heap node to be the one with the greatest value in the long-term Web sites heap.

At a first glance, a way of simulating an "Optimal" strategy could be to use the Pagerank obtained on a previous crawl of the Web, and use this value as an estimator of the current Pagerank. However, this is not a good estimator: Cho and Adams [CA04] report that even if we consider only the pages whose Pagerank monotonically increase, 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".

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.

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.

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 $K = 100,000$ (equivalent to 1/350, 1/70 and 1/35 of the collection).

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}.

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 6.5. The figure shows that *Length* is more efficient than the strategies based on periodical Pagerank calculations and *Depth*. Notice that the implementation and processing requirements for *Length* are significantly simpler and lower than for all others. In particular, calculating periodical Pageranks takes a significant amount of resources in running time and space.

An important observation is that the *Optimal* strategy is too greedy if a high coverage 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. Except for the *Length* strategy, which specifically tries to keep its list of available Web sites as long as possible, all other strategies have the same performance with the last 20% of pages by Pagerank.

The effect of increasing the number of robots to $r = 64$ and $r = 256$ is shown in Figure 6.6. Observing the rate of growth of the cumulative Pagerank sum, the results show that *Length* is not affected by the number of robots; but *Depth* 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.

Finally, Table 6.1 shows the effects in retrieval time when we increase the number of robots for different bandwidths, using the *Length* strategy.

The results shows 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. Note that this result arises from a

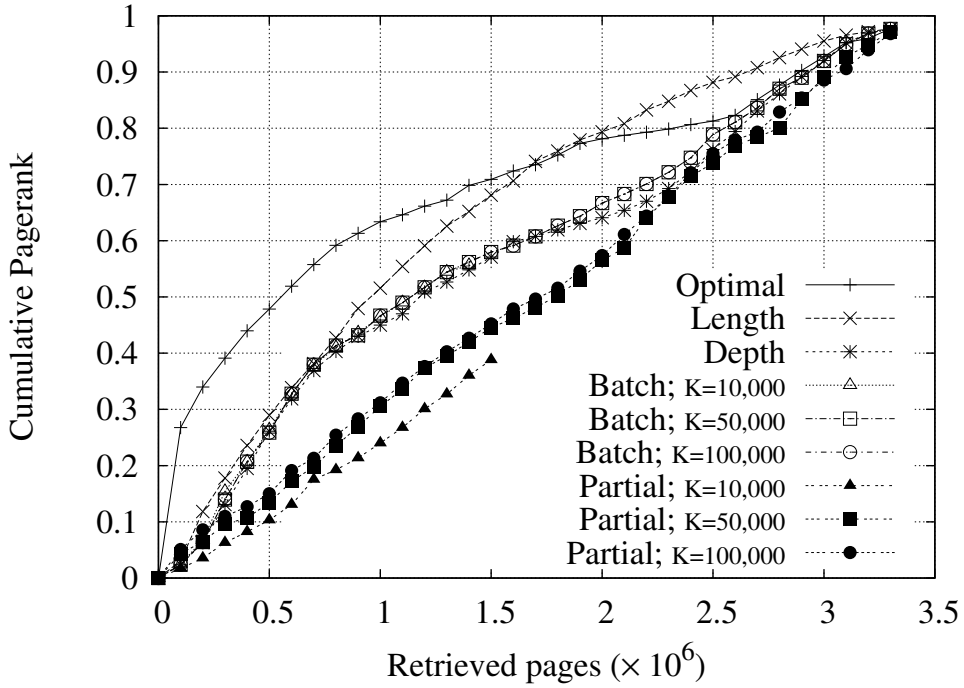


Figure 6.5: Cumulative sum of Pagerank values vs number of retrieved Web pages for the different strategies, in the case of $r = 1$, one robot.

simulation which does not consider CPU processing time, and adding more robots increases the performance monotonically. In a real 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 **??**: there are no context switches, and handling even a large amount of sockets is not very demanding in terms of processing power.

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

Table 6.1: Predicted speed-ups for parallelism in the crawling process.

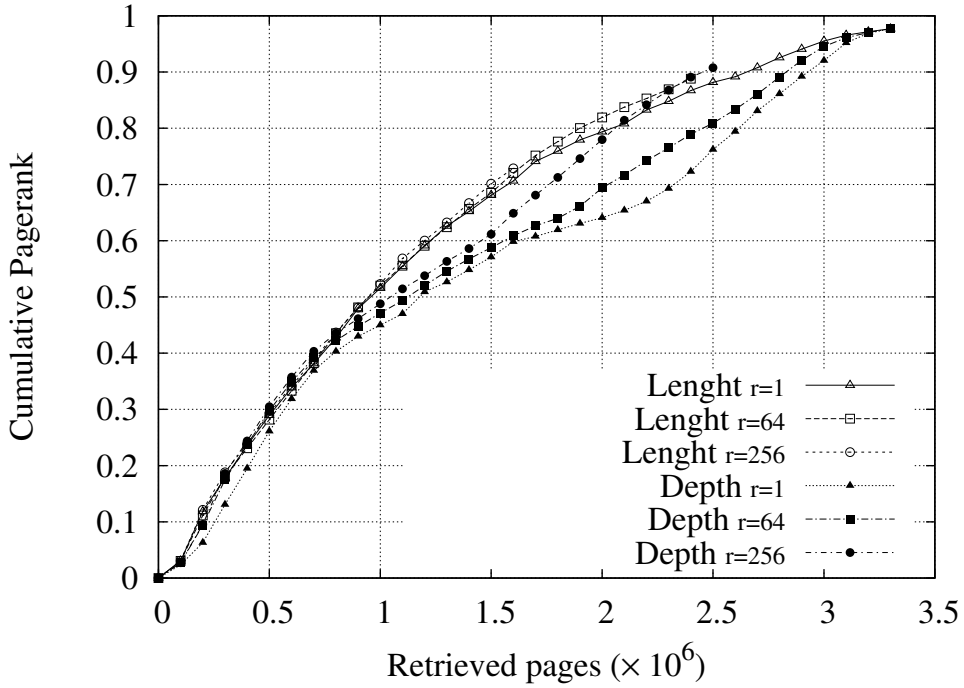


Figure 6.6: Cumulative sum of Pagerank values vs number of retrieved Web pages. Strategies *Length* and *Depth*, case for $r = 1$, $r = 64$ and $r = 256$ robots.

6.4 Short-term scheduling

When crawling, specially 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.

We have shown the distribution of pages on sites for the whole Web in Figure 6.3; on Figure 6.7 we show page distribution on sites for a typical batch, obtained at the middle of the crawl. The distribution is slightly different than for the entire Web, as Web sites with very few pages are completed early in the crawl, but it is nevertheless very skewed.

Because of the rule telling that the crawler cannot visit the same site before $w = 15$ seconds, this distribution of pages to sites is very bad in terms of crawler scalability. That is, it is not possible to use productively a large number r of robots and it is difficult to achieve good utilization of the network bandwidth.

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 6.8 shows the effective number of robots involved in the retrieval of a batch.

An approach to overcome this problem is to try to reduce waiting time. This can be done increasing c and letting robots get more than one page every time they connect to a Web server. In figure 6.9 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 results in savings in terms of the total time needed for downloading the pages, as more robots keep active for a longer part of the crawl.

Besides increasing c, the number of downloads per connection, we also use the heuristic of monitoring the

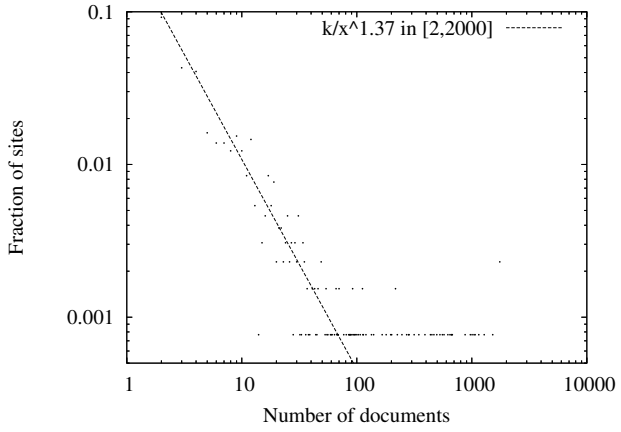


Figure 6.7: A typical batch: distribution of pages onto sites.

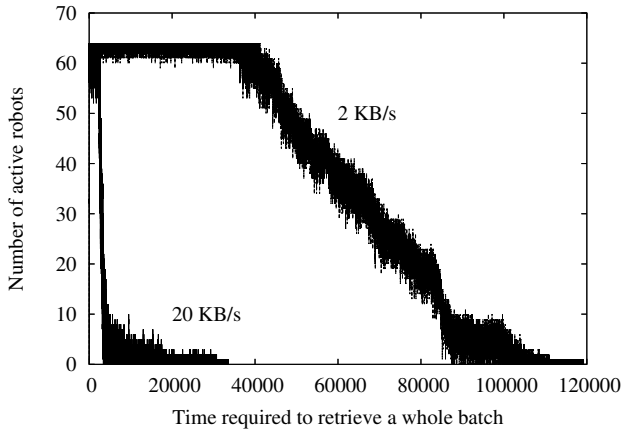


Figure 6.8: Number of active robots vs batch's total retrieval time. The two curves are for small (2 Kb/s) and large (20 Kb/s) bandwidth. In either case, most robots are idle most of the time.

number of threads being used while downloading pages, and stop the cycle if this number is too low. Pages that were not crawled are downloaded in the next batch. Also, we try to use large values for K, the size of the batch.

Another approach to short-term scheduling is to dynamically adjust the number of threads by predicting the bandwidth at which each Web site will transfer pages. 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.

6.5 Downloading the real Web

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 *Length* strategy. The characteristics of the downloaded pages are summarized in Table 6.2.

We ran the crawler in batches of up to 100,000 pages, using up to $r = 1000$ simultaneous network connec-

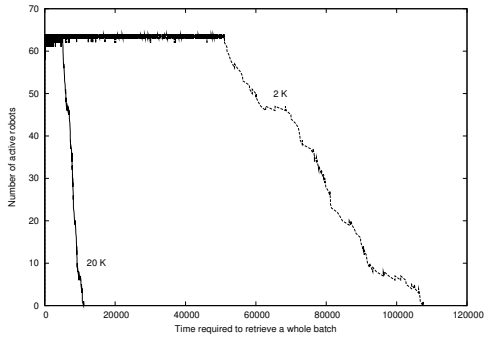


Figure 6.9: Number of active robots vs batch's total retrieval time. The two curves are for small (2 Kb/s) and large (20 Kb/s) bandwidth. In this case robots are allowed to request up to 100 pages with the same connection, which is the default maximum for the Apache Web server.

Pages visited	3 M
Successful downloads	(80%) 2.4 M
Bytes downloaded	57 GB
Web sites	53,196

Table 6.2: Downloaded pages

tions, 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 didn't 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 6.10.

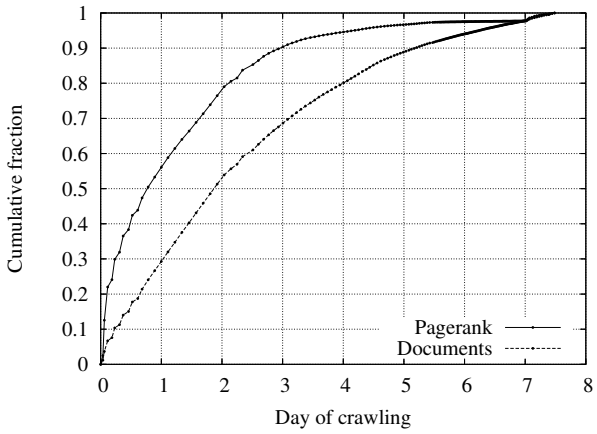


Figure 6.10: Cumulative sum of Pagerank values vs day of crawl, on an actual crawler using the *Length* strategy.

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, as shown in Figure 6.11, which is consistent with the findings of Najork and Wiener [NW01].

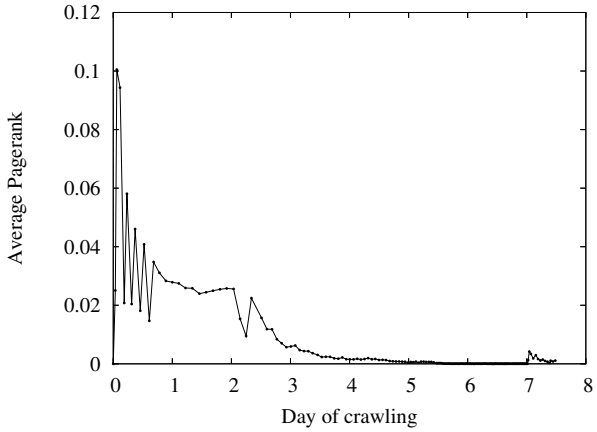


Figure 6.11: Average Pagerank per day of crawl.

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 6.12. Note 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.

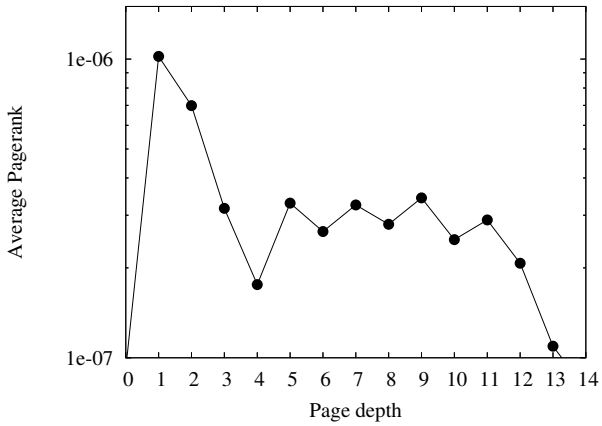


Figure 6.12: Average Pagerank versus page depth.

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 6.13. The results are similar, but the actual crawl performed slightly better than the simulated crawl.

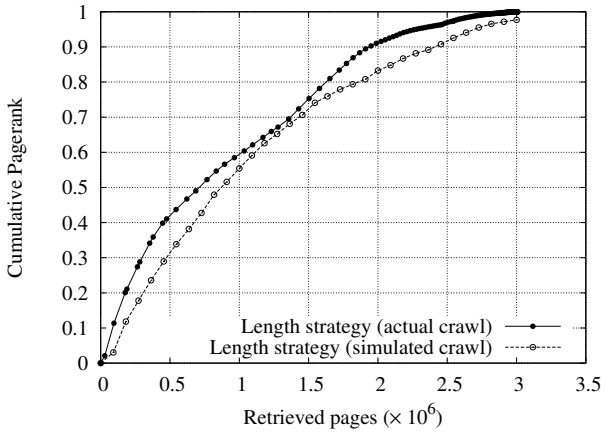


Figure 6.13: Cumulative sum of Pagerank values vs number of retrieved Web pages, on both actual and simulated Web crawls using *Length* strategy.

6.6 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 the one we called *Length*, is good enough to efficiently retrieve 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, to avoid exhausting Web sites too early. This is a practical alternative since it is fast, simple to implement, and it is able to retrieve the best ranked pages at a rate that is closer to the optimal than the other alternatives.

Our simulation results show that attempting to retrieve as many pages from a given site ($c >> 1$), 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, and is good for both the search engine and the Web site administrator.

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.

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.

Bibliography

[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. IOS Press, 2002.

[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.

[CA04] Junghoo Cho and Robert Adams. Page quality: In search of an unbiased Web ranking. Technical report, UCLA Computer Science, 2004.

[CMRBY04] Carlos Castillo, Mauricio Marin, Andrea Rodriguez, and Ricardo Baeza-Yates. Scheduling algorithms for Web crawling. In *Latin American Web Conference (WebMedia/LA-WEB)*, Riberao Preto, Brazil, 2004. IEEE Cs. Press. (To appear).

[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.

[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.

[Kos95] Martijn Koster. Robots in the web: threat or treat ? *ConneXions*, 9(4), April 1995.

[LF98] B. Liu and E. A. Fox. Web traffic latency: Characteristics and implications. *J.UCS: Journal of Universal Computer Science*, 4(9):763–??, 1998.

[Liu98] Binzhang Liu. Characterizing web response time. Master's thesis, Virginia State University, Blacksburg, Virginia, USA, April 1998.

[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.

[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.