# 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 **??** (page **??**). 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 **??** (page **??**).
- 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.

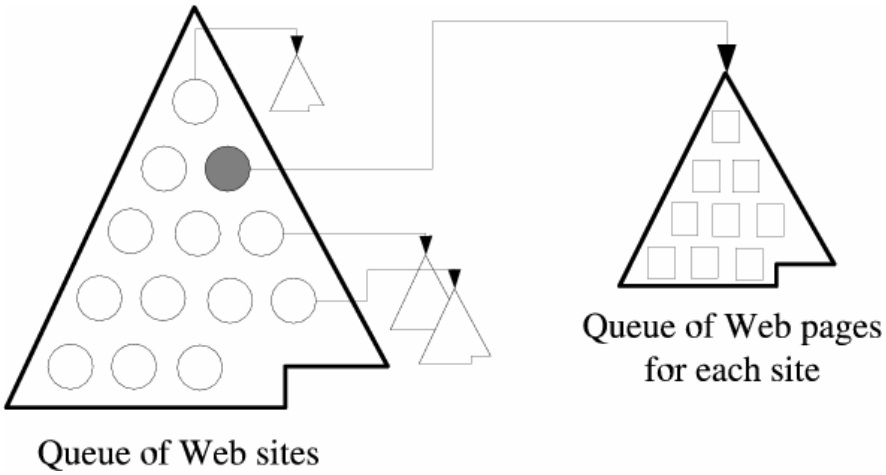


**Figure 4.1:** Queues used in the crawling simulator. We tested the scheduling policies using a structure with two levels: one queue for Web sites and one queue for the Web pages of each Web site.

- 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 **??**, 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 **??**, as the time spent waiting cannot be amortized effectively.

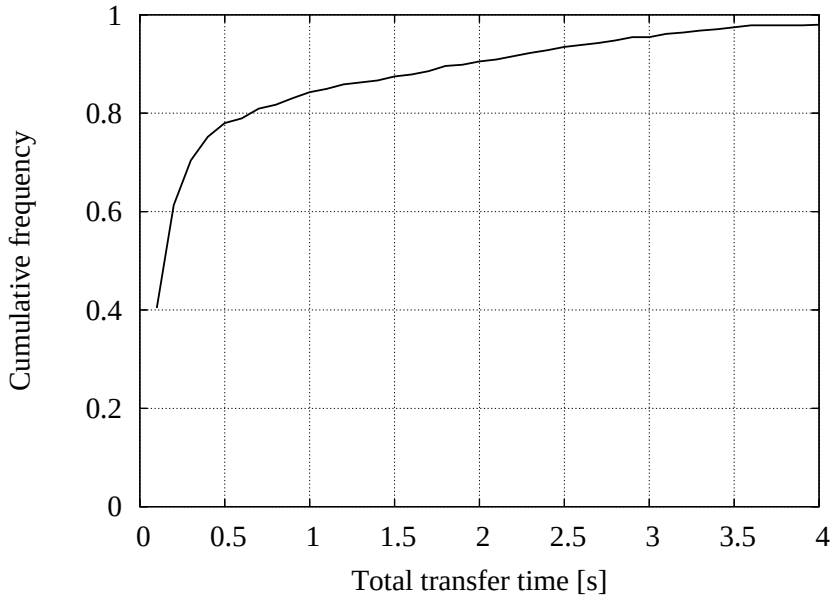


**Figure 4.2:** Total download time for sequential transfer of Web pages; this data provides from experiments in the Chilean Web and was used as an input for the Web crawler simulator.

### 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 $\tau$.

**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.

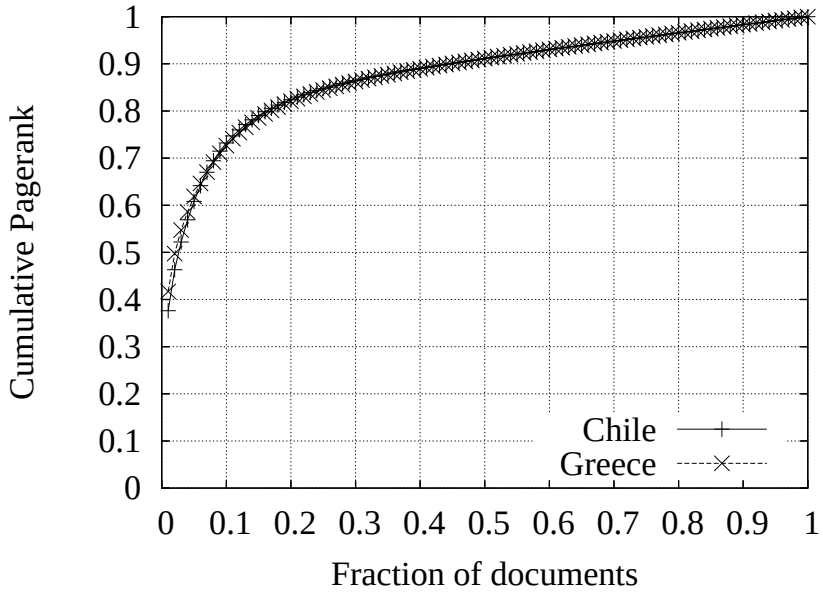


**Figure 4.3:** Cumulative Pagerank in the `.CL` and `.GR` domain, showing almost exactly the same distribution; these curves represents an upper bound on the cumulative Pagerank of any crawling strategy.

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.

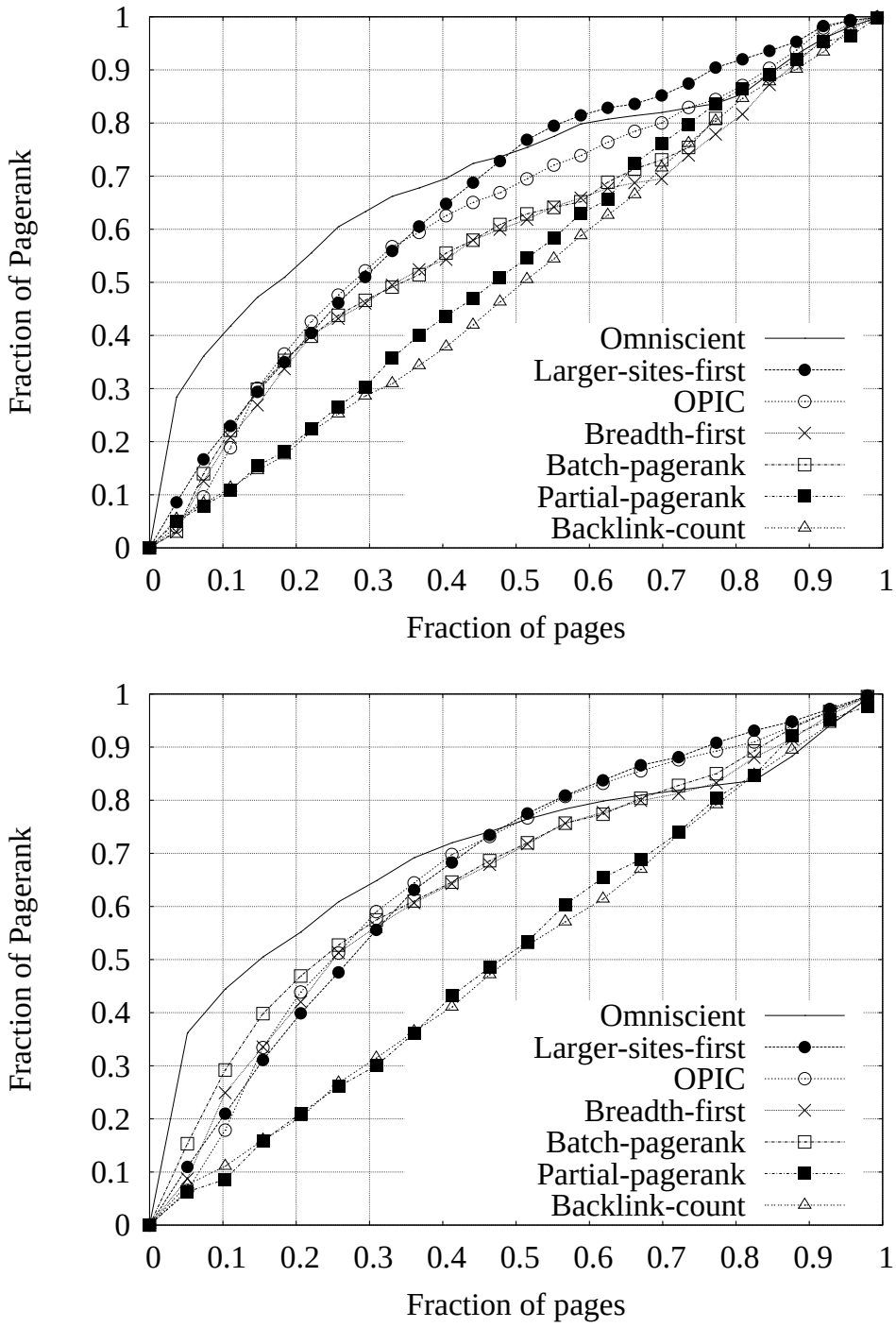


**Figure 4.4:** Comparison of cumulative Pagerank vs retrieved pages with the different strategies, excluding the historical strategies, in the Chilean sample during April and May 2005.

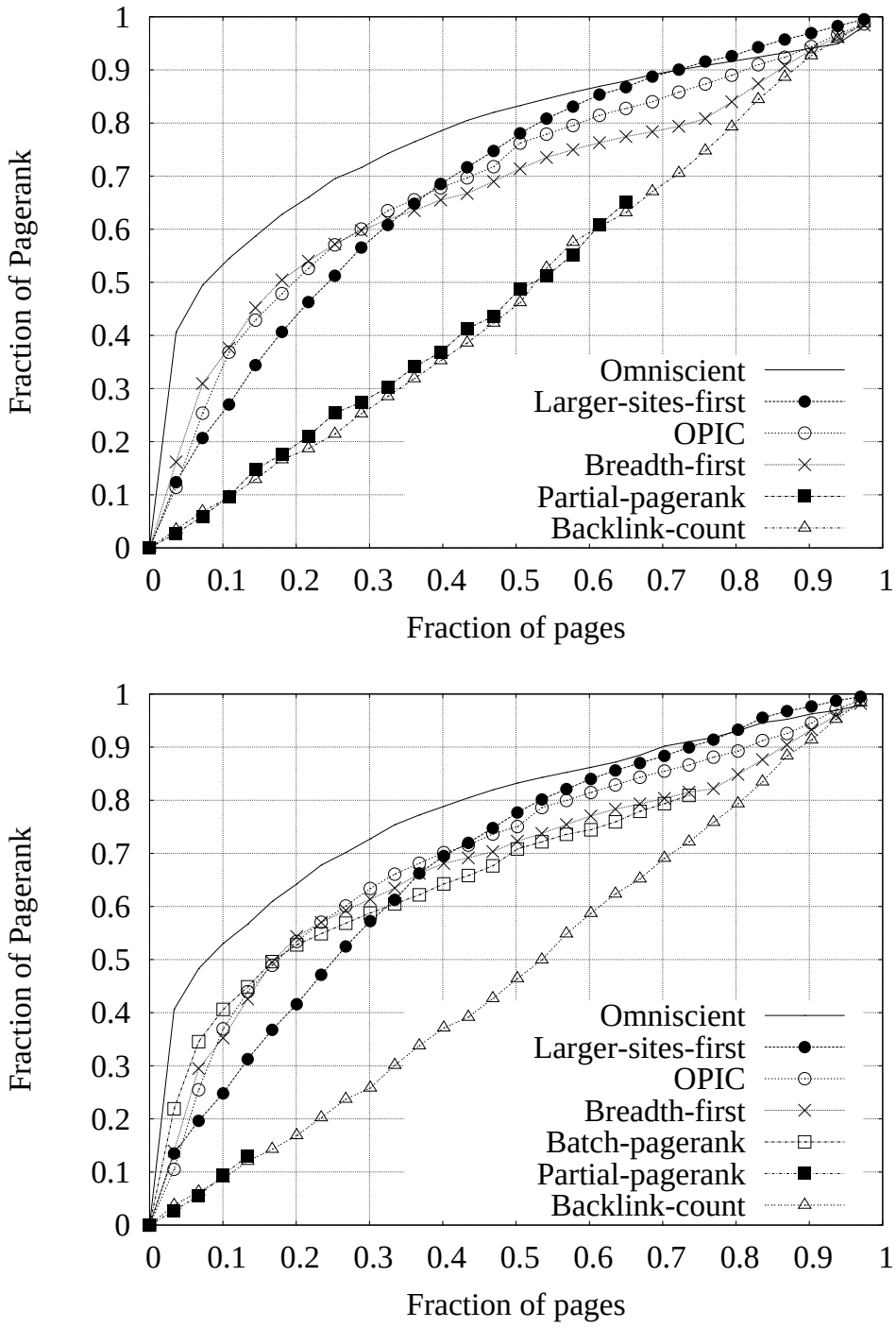


**Figure 4.5:** Comparison of cumulative Pagerank vs retrieved pages with the different strategies, excluding the historical strategies, in the Greek sample during May and September 2005.

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.

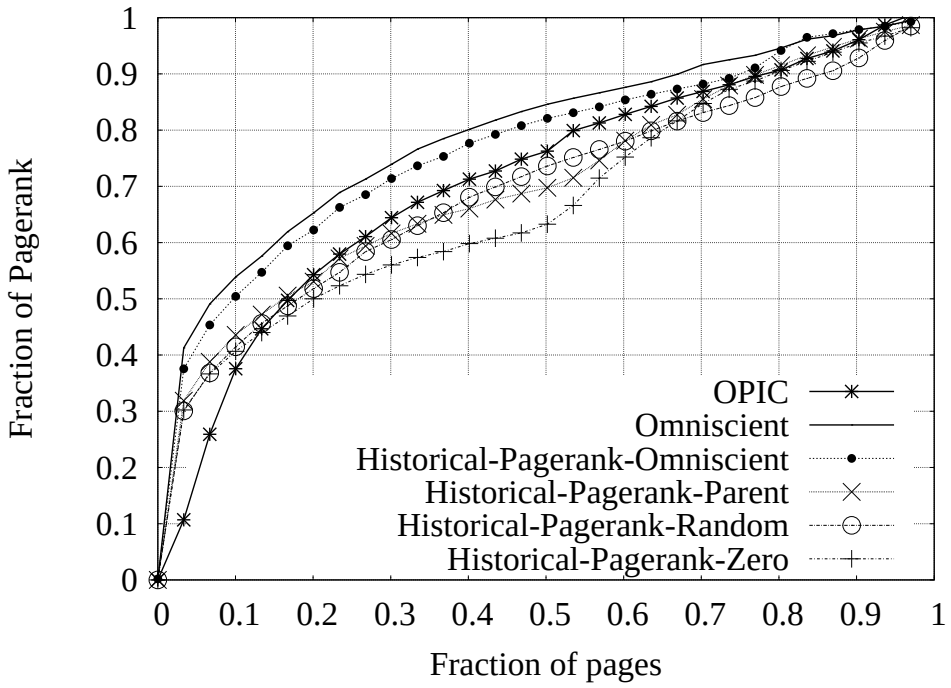


**Figure 4.6:** Comparison of cumulative Pagerank using the historical strategies against the *omniscient* and *OPIC* strategies, for a crawl of the Greek Web in September 2004, using Pagerank information from May 2004.

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 4.2, 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 4.2.

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

| Strategy | Avg. Pagerank | $\tau$ |
|---|---|---|
| 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.

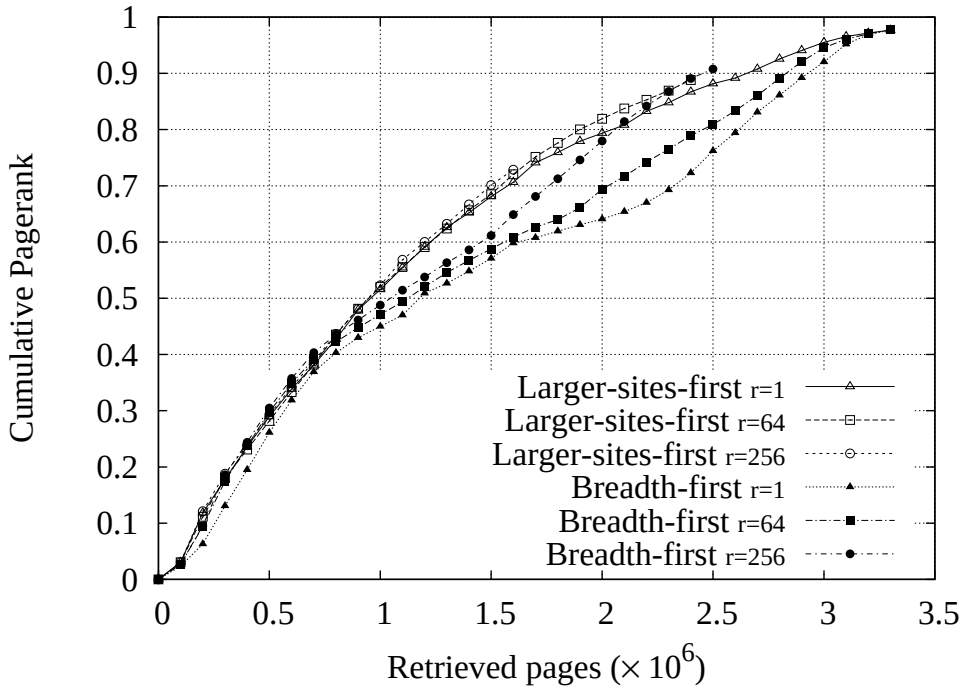


**Figure 4.7:** Cumulative sum of Pagerank values vs number of retrieved Web pages. Strategies *larger-sites-first* and *breadth-first*, case for $r = 1$, $r = 64$ and $r = 256$ robots.

Finally, Table 4.3 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 **??**: 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 |

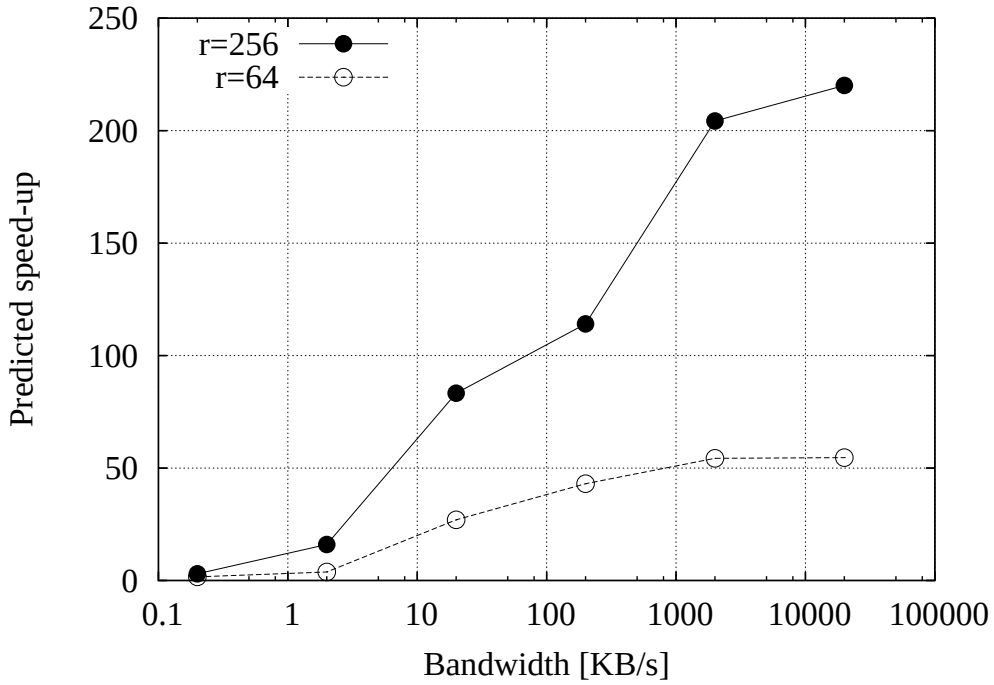


**Figure 4.8:** Predicted speed-up at different bandwidths, showing sub-linear growth and saturation. Note that the scale for the bandwidth is logarithmic.

## 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 **??**; 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.

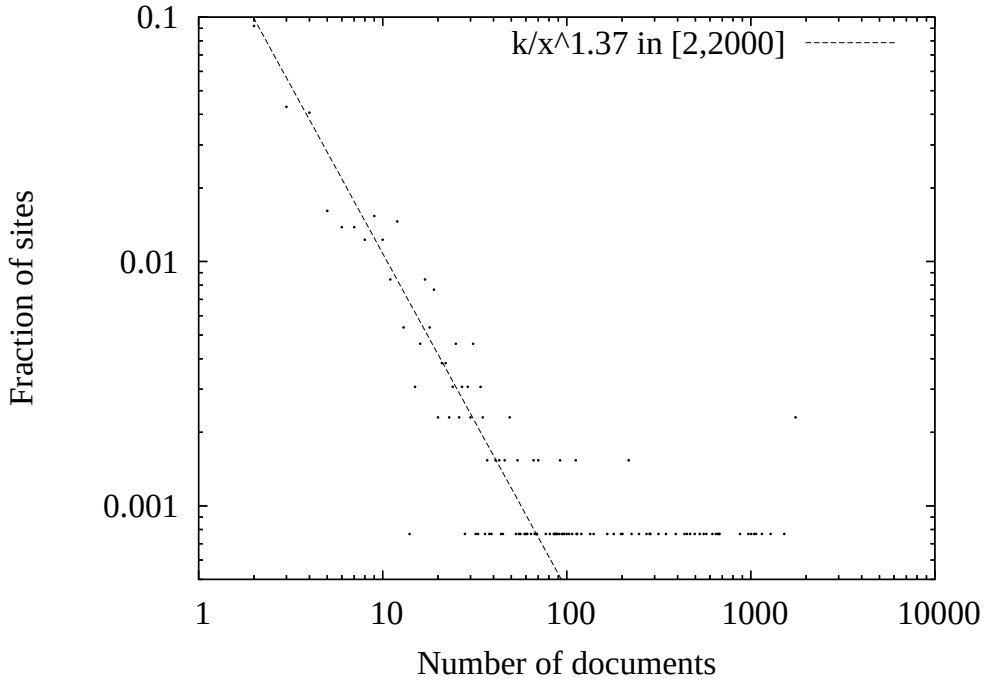


**Figure 4.9:** Distribution of Web pages to Web sites in a typical batch in the middle of the crawl using breadth-first crawling.

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.

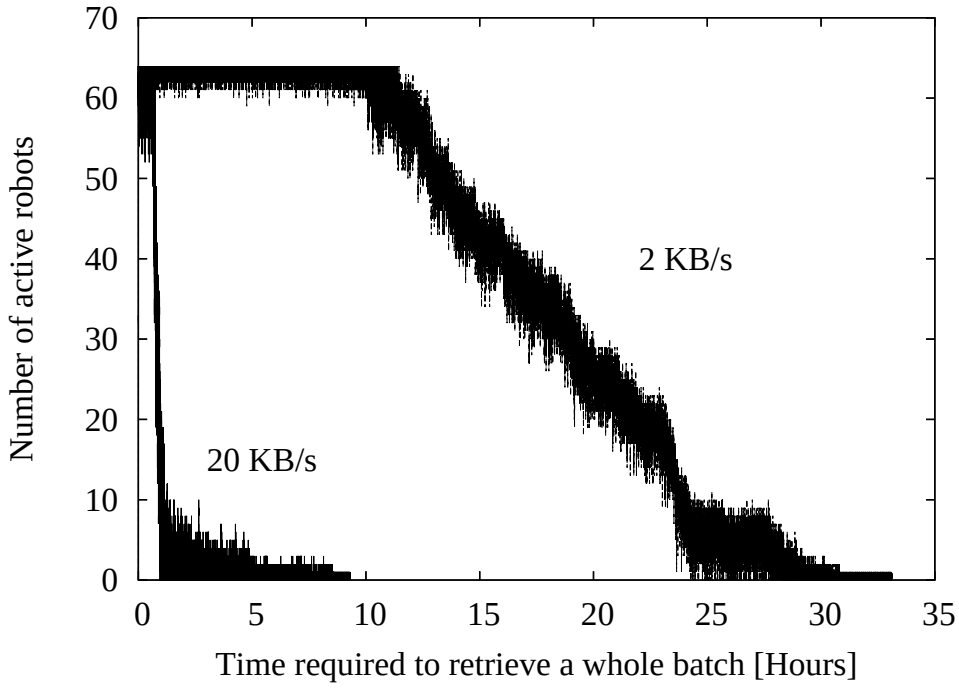


**Figure 4.10:** Number of active robots vs batch's total retrieval time. The two curves are for small (2 Kb/s) and large (20 Kb/s) bandwidth. In either case, most robots are idle most of the time, and the number of active robots varies as robots get activated and deactivated very often during the crawl.

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.

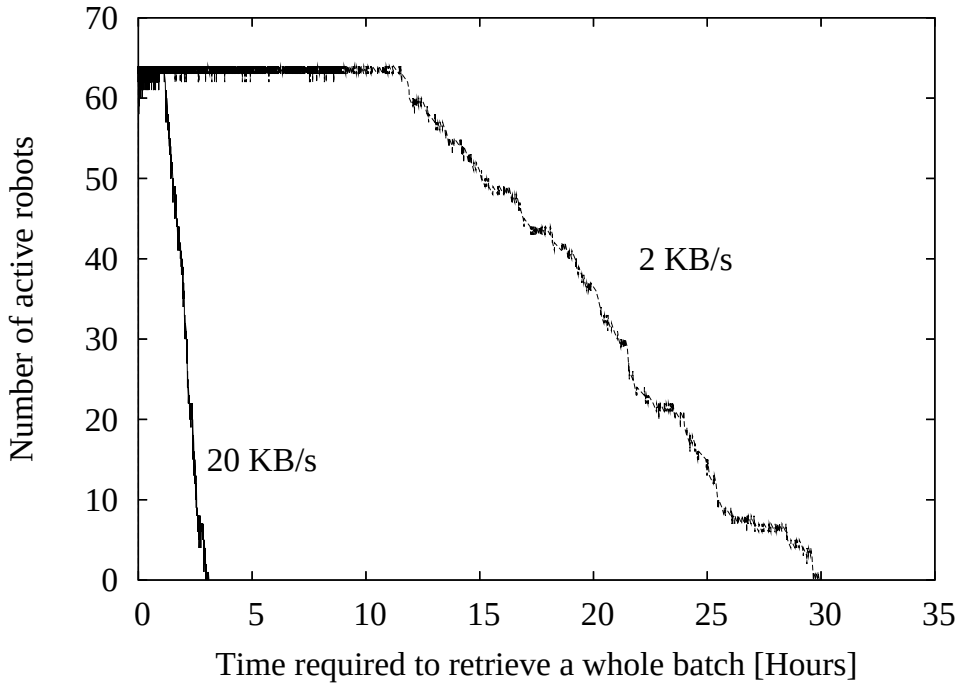


**Figure 4.11:** Number of active robots vs batch's total retrieval time. The two curves are for small (2 Kb/s) and large (20 Kb/s) bandwidth. In this case robots are allowed to request up to 100 pages with the same connection, that is the default maximum for the Apache Web server. In this case there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

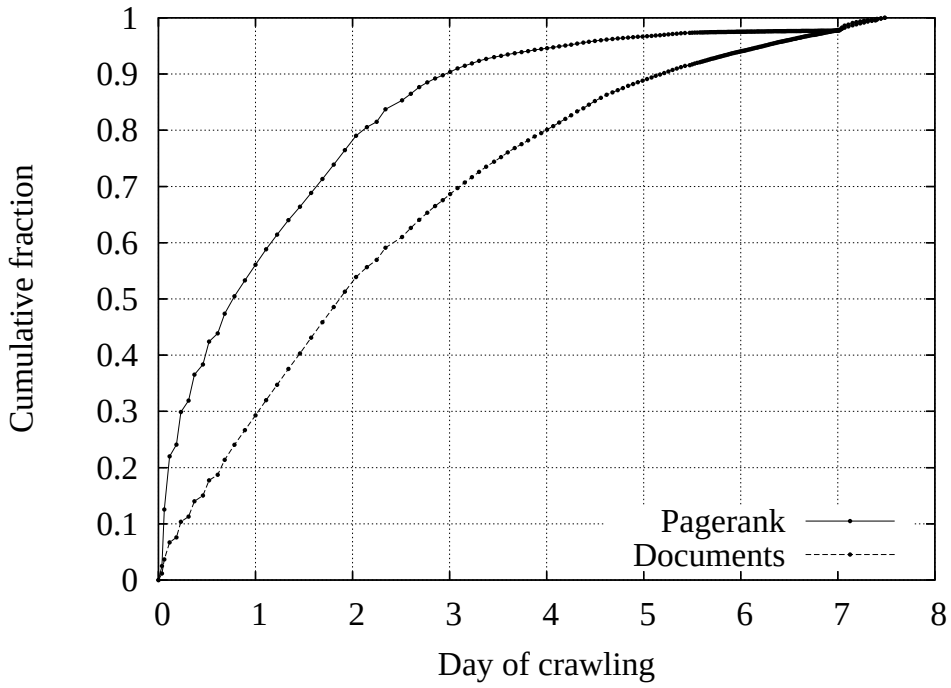


**Figure 4.12:** Cumulative sum of Pagerank values vs day of crawl, on an actual crawler using the *larger-sites-first* strategy. The fraction of retrieved Pagerank is larger than the fraction of retrieved documents during the entire crawl.

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.

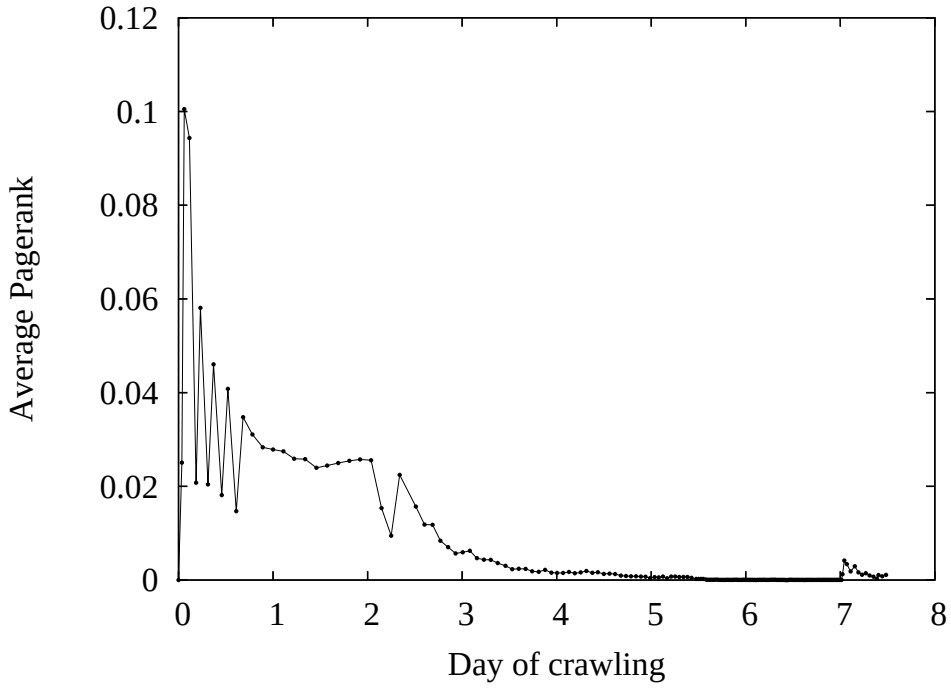


**Figure 4.13:** Average Pagerank per day of crawl using the *larger-sites-first* strategy.

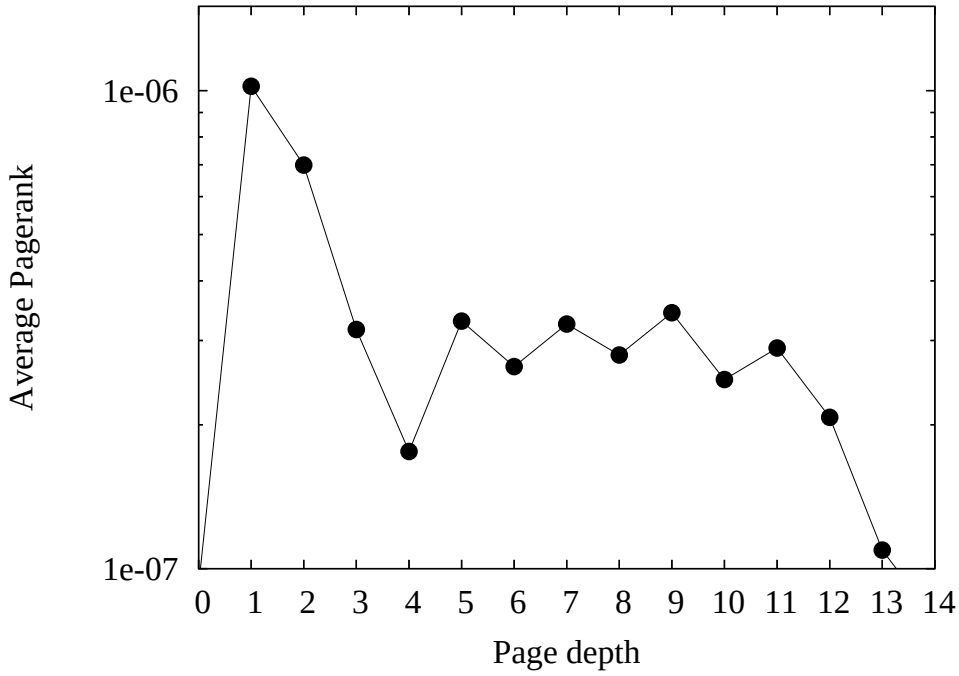


**Figure 4.14:** Average Pagerank versus page depth, showing that there is a correlation only in the first few levels.

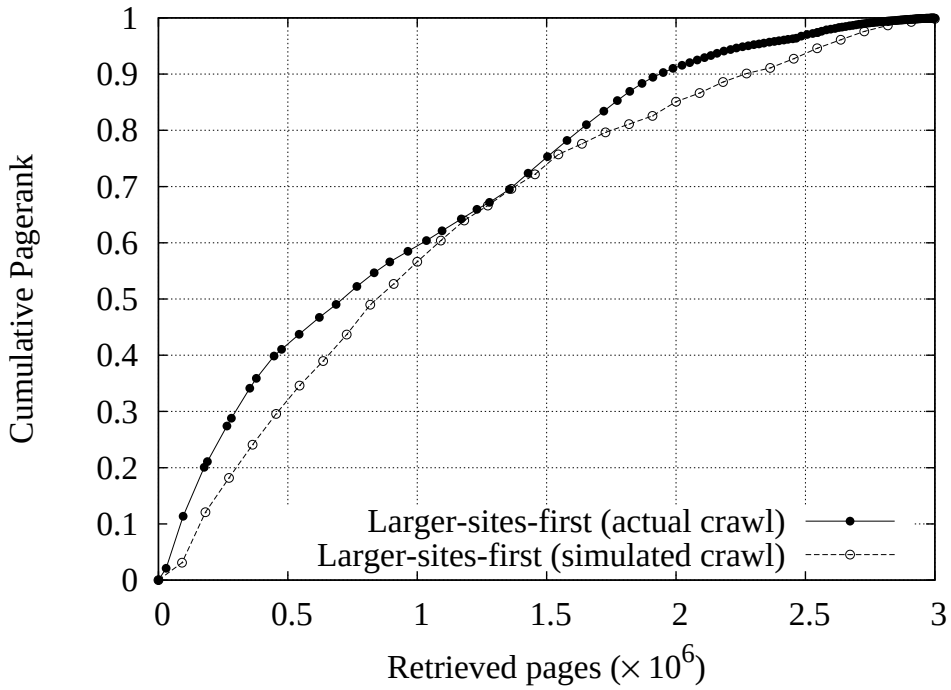


**Figure 4.15:** Cumulative sum of Pagerank values vs number of retrieved Web pages, on both actual and simulated Web crawls using *larger-sites-first* strategy.

### 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.

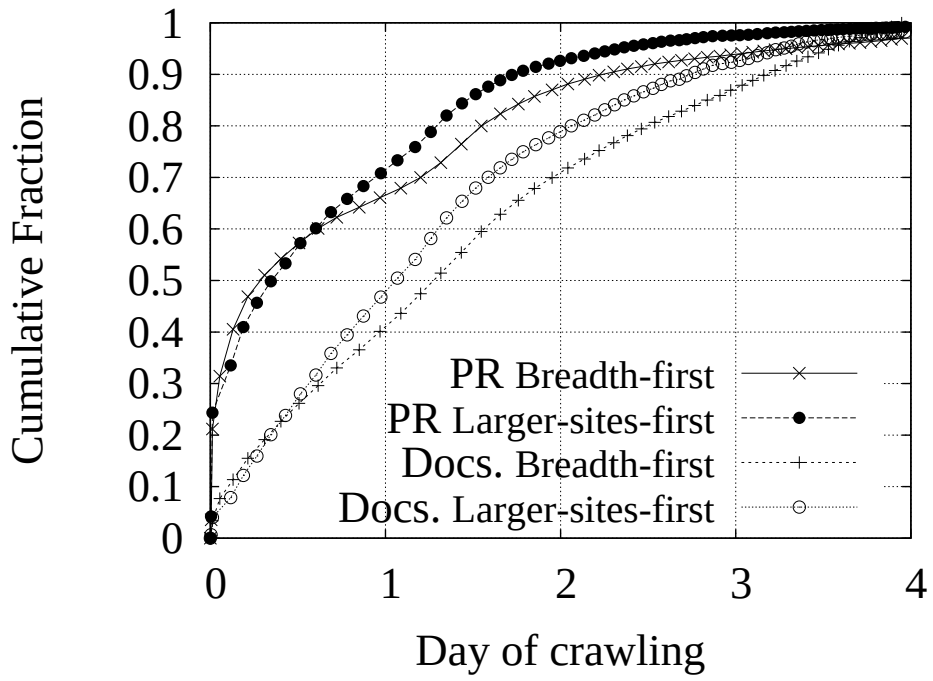


**Figure 4.16:** Cumulative Pagerank (PR) and cumulative fraction of documents (Docs.) of an actual crawl of the *.GR* domain using two strategies: *Breadth-first* and *Larger-sites-first*.

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.

# Bibliography


[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.

[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.

[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.

[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.

[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.

[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.

[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.

[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.

[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.

[Ken70] Maurice G. Kendall. *Rank Correlation Methods*. Griffin, London, England, 1970.

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

[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 Science.

[The02] The Economist. Country Profiles, 2002.

[Uni02] United Nations. Population Division, 2002.

[Uni03] United Nations. Human Development Reports, 2003.