Crawling the Web with Limited Memory

Carlos Castillo
Università di Roma
"La Sapienza"
Rome, Italy
castillo@dis.uniroma1.it

Alberto Nelli*
Università di Roma
"La Sapienza"
Rome, Italy
anelli@tim.it

Alessandro Panconesi
Università di Roma
"La Sapienza"
Rome, Italy
ale@dsi.uniroma1.it

ABSTRACT

Search engines rely on Web crawlers to create a Web index, by exploring the Web graph, downloading pages, and finding links to new pages to be explored. At any given moment, there are a number of pages waiting to be downloaded in the crawler queue. We study the growth of the queue of pending pages during a crawl of a large subset of the Web. In a normal breadth-first crawler, the queue of pending pages quickly grows very large.

We present a strategy for managing the pending queue that reduces its maximum size by 50% while preserving the coverage and quality of the pages visited. This can be applied in general Web search as well as topic-specific crawling, peer-to-peer search, on-demand Web crawling, and other environments in which memory usage has to be kept to a minimum.

1. INTRODUCTION

Web crawlers are used by Web search engines to visit Web pages automatically, by recursively following links until a certain stopping criteria is met. Large-scale Web search engines index more than 11.5 billion pages [8], and this large size is explained mostly by the existence of a large number of dynamic pages. Dynamic pages are pages that are generated at the time they are requested by querying a data source and formatting the query output in HTML format.

When crawling the Web, a queue of pages to be visited is initialized with a set of starting pages that are downloaded and parsed to extract links. Newly discovered pages are added to this queue, downloaded, and so on. Because the Web has an effectively infinite number of pages, by the time the crawler is stopped in a large-scale search engine, the number of pages remaining in the queue is often larger than the number of pages that were really crawled.

The typical scheduling policy for Web crawling is breadth-first search [15], because it produces a collection with high quality, by finding "good" pages early in the crawl. Unfortunately, the size of the queue of pending pages grows very quickly. To visualize this phenomenon, we used the WebBase [10] repository of 118 million pages and simulated a breadth-first (henceforth BFS) and depth-first (henceforth DFS) visit of its nodes starting from an arbitrary, but fixed, set of nodes. The size of the pending queue for both scheduling policies is shown in Figure 1, which shows the essence of the problem that we study in this paper.

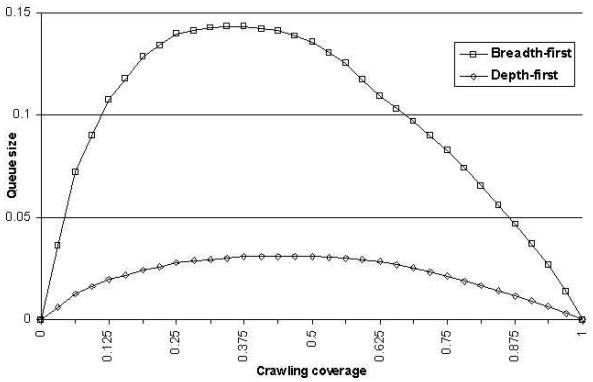

Figure 1: Queue size, as a fraction of the size of the collection, in breadth-first search and depth-first search (WebBase collection).

In Figure 1, the y-axis reports the queue size as the fraction of nodes of the web sample. The x-axis loosely correspond to time, but also to coverage: the fraction of the web that is visited. A value of 1 corresponds to what we define to be full coverage: the portion of the web that can be reached starting from an arbitrary, but fixed, set of nodes. Naturally, in all our experiments the different visiting strategies start from the same initial set of nodes.

As we can see in the figure, in breadth-first search the queue size grows very fast, up to a maximum of roughly 14% of the total number of pages (which can be in the order of billions of pages in the case of the full Web).

On the other hand, in depth-first search the queue is always much smaller. However, depth-first search has a number of drawbacks that prevent its use in practice. First, it takes a long time in finding pages with high quality [2, 5, 15]; second, it focuses the crawler on a few sites, potentially violating typical "politeness" properties toward Web servers [11]; and it is prone to get stuck in artificially crafted Web page loops, also known as "crawler traps".

*Partially supported by a grant of Yahoo! Research Barcelona.

This is a pre-print of a work in progress.
Last updated: June 16, 2006

The goal of our research is to find a strategy that matches the good properties of breadth-first search in terms of quality and politeness with Web servers, while using a queue size comparable to that of depth-first search.

Specifically, the main contributions of this paper are:

- We describe a simple crawling strategy that we dubbed the **Sydney Strategy**, for reducing the size of the crawling queue by up to 50% without losing coverage.
- We describe a variation of the Sydney Strategy for reducing the size of the crawling queue even more, at the expense of very limited loss of coverage.
- We show that both strategies behave very well in terms of their ability to find high-quality pages, and that moreover these are collected early in the crawl. In the fixed-budget case, when some coverage is lost, we show that the pages that are not downloaded are not likely to be of high-quality.

The next section presents our test collections. Section 3 presents the crawling strategies and analyzes them in terms of queue size and coverage. Section 4 compares the strategies in terms of quality of the resulting set. The last section discusses applications and future work.

2. DATA SETS

For our experiments, we used three data sets representing large samples from the World Wide Web. The largest collection we used was obtained from the Stanford WebBase project [10]. This is a graph with 118 million pages and 1,019 million links, crawled in 2001.

We also used other two collections crawled by the Laboratory of Web Algorithmics, Dipartimento di Scienze dell' Informazione, Universitá degli studi di Milano. These collections are available on-line at `http://law.dsi.unimi.it/`. The specific data sets we used were a sample of 18 million pages from the `.UK` domain downloaded in 2002, and a sample of 41 million pages from the `.IT` domain downloaded in 2004.

We tested all the algorithms in the three datasets and the results were basically the same. In many of the graphs in this paper, we show the results from one sample for conciseness.

3. THE SYDNEY STRATEGY

We present a strategy that can reduce the queue size while at the same time preserving coverage, quality of the retrieved pages and politeness toward Web servers.

The basic scheme is as follows. We use two queues, referred to as the **primary queue**, denoted by P, and the **secondary queue**, denoted by S. We also keep a list of visited pages, denoted by V and initially empty.

The algorithm is initialized by inserting a set of starting nodes in the primary queue P and with an empty secondary queue S.

As long as P is not empty, nodes are extracted from it and downloaded, one node at a time. Let v be the node just extracted from P, and let $N^+(v)$ be the set of **new** pages pointed to by v by means of hyperlinks, i.e. we do not consider links pointing to already seen pages (in P, S or V).

The algorithm selects a random subset of (up to) t nodes from $N^+(v)$, where t is a parameter of the algorithm. Let R denote this set. Now, if $N^+(v) - R = \emptyset$, that is, if we have visited all out-neighbours of v, then node v is discarded. Otherwise, if there are yet unvisited out-neighbours of v, we insert v in the secondary queue S, to explore its remaining neighbours later.

When P becomes empty, the contents of S are emptied into P. The visit terminates when both queues become empty. Figure 2 presents the algorithm in pseudo code.

Require: Starting URLS, t: number of links to sample
1: $P \leftarrow$ starting URLs (primary queue)
2: $S \leftarrow \emptyset$ (secondary queue)
3: $V \leftarrow \emptyset$ (visited pages)
4: **while** $P \neq \emptyset$ **do**
5: Pick a page v from P and download it
6: $V \leftarrow V \cup \{v\}$ (mark as visited)
7: $N^+(v) \leftarrow v$'s out-links pointing to new pages ("new" means not in P, S or V)
8: **if** $|N^+(v)| > t$ **then**
9: $R \leftarrow$ a random sample of t nodes from $N^+(v)$
10: $S \leftarrow S \cup \{v\}$
11: **end if**
12: $P \leftarrow P \cup R$
13: **if** $P = \phi$ **then**
14: $P \leftarrow S$
15: $S \leftarrow \emptyset$
16: **end if**
17: **end while**

Figure 2: Algorithm of the Sydney Strategy with sampling parameter t.

We call this strategy the **Sydney strategy** with parameter t, for the peculiar shape of the curve describing memory utilization, reminiscent of the famous Sydney Opera House. The typical trend is shown in Figure 3 where we compare breadth-first search and depth-first search against the Sydney strategy (with parameter $t = 8$ and $t = 16$).

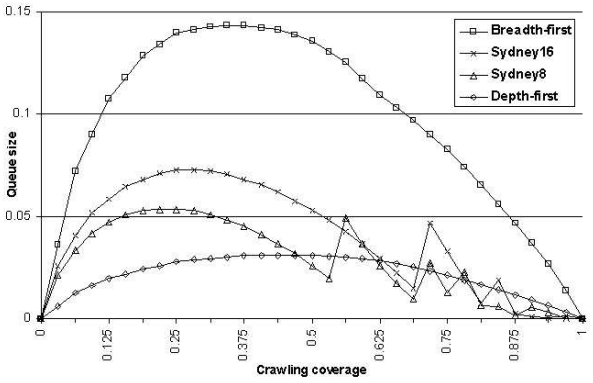

Figure 3: Memory usage of the Sydney Strategy. In this case, we take at most $t = 8$ or $t = 16$ outlinks from each page. Coverage is preserved, and the maximum queue size can be reduced to half the size of breadth-first search.

We see from the figure that the Sydney Strategy attains full coverage of the collection. This is because the two visits only differ in their ordering, but the set of links that they

traverse is the same (albeit at different times). The same holds for DFS.

The figure reports the cumulative memory usage of the Sydney strategy, i.e. $|P| + |S|$. The saving is apparent and can be quantified as being half of BFS. That the Sydney strategy saves substantially in terms of memory w.r.t. BFS is a robust conclusion. Identical results are obtained with the `.uk` and .it datasets: full coverage, and a memory consumption which is roughly half of that of BFS. This is obtained by Sydney with parameter 8.

This data raises a few interesting questions:

1. Why is this happening?
2. How should the parameter t be set for best results?
3. Can the strategy be improved?
4. What is the performance of the Sydney strategy in terms of other relevant parameters: in particular, how does it perform in terms of quality of the retrieved pages and politeness?

Section 4 is devoted to the last question. As we shall see, the strategy performs remarkably well in terms of quality. The rest of this section deals with the other questions.

3.1 Why the Sydney strategy performs well

One plausible explanation at the outset is that the nodes with more than t out-links are a very small fraction of the total, so they do not affect the results. However, this is not true. Figure 4 shows the out-degree distribution of the WebBase data set.

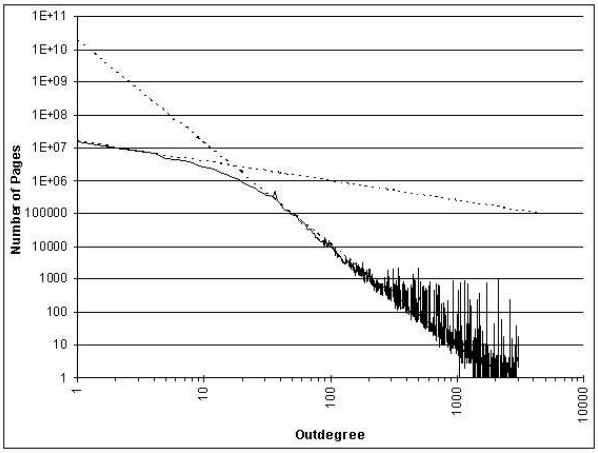

Figure 4: Out-degree distribution in the WebBase data set.

The figure displays the typical distribution on a log-log scale that has been observed by several authors (see for instance [3, 14]). The distribution of the out-degree is well approximated by a double-pareto or by a log-normal distribution. Roughly, the distribution says that even for very large values of the out-degree, there is a small, but non-negligible fraction of nodes with that out-degree.

Figure 5 reports the fraction of nodes with out-degree greater than 8 and 16 respectively, in the three datasets. The WebBase dataset is older than the others and thus has a lower out-degree, as expected given that the Web graph has been growing more dense in the past years [12]. However, this difference does not seems to have a noticeable effect in the behavior of the crawling strategies we discuss in this paper.

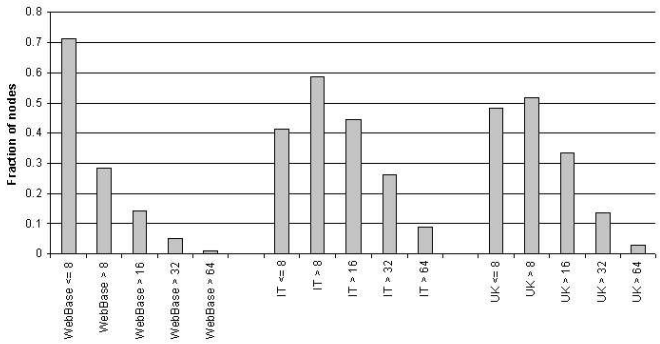

Figure 5: Out-degree distribution in all three datasets. The fraction of nodes with more than 8 and 16 out-links is significant.

As it can be seen, in principle the fraction of nodes that should be transferred to the secondary queue S is more than half of the nodes! Why does it never grow to more than 3% of the Web in our datasets?

If most of the Web pages were reachable from a small set of pages, that would make the secondary queue unnecessary. However, it is not the case. The secondary queue is necessary to have high coverage of the collection.

Figure 6 shows the portion of the Web that is covered only by the nodes of the primary queue P. Their coverage is compared to that of BFS and DFS, i.e. what we defined to be full coverage. The two strategies are referred to in the figure as Random 8 and Random 16, since the links of the extracted node that are followed are chosen at random.

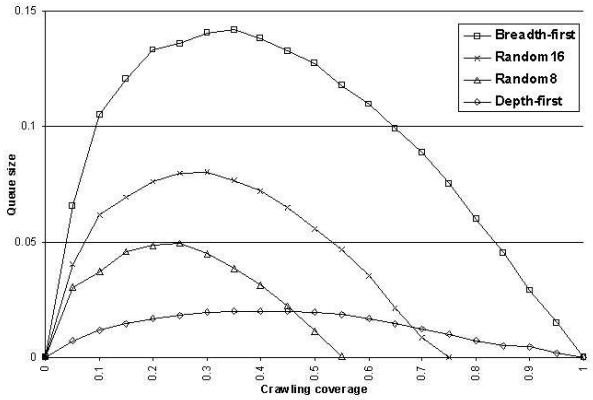

Figure 6: Ignoring the secondary queue, and taking at random 8 and 16 out-links per page, we cannot attain full coverage of the graph (dataset `.IT`)

Under these strategies, some coverage is lost. When $t = 8$ the coverage of nodes in primary queue is 55%. This means that nodes with out-degree greater than 8 cover about 45% of the Web. Likewise, nodes with out-degree greater than 16 cover roughly 25% of the Web.

A related phenomenon was observed recently in a large collection of 200 million Chinese pages by Meng and Tan [13]. They ran a breadth-first crawler that discarded 2% of the out-links of every page, and observed a loss of 50% of the coverage. This is partially explained by the existence of "peninsulas" on the Web; that is, pages that are connected to the rest of the Web by a single page. According to the authors of [13], roughly 4% of the pages on the Web connect to a peninsula having more than 10 pages.

Even if it loses coverage, this "truncated" strategy is interesting in itself. In the next section we will show that it performs well in terms of quality. Therefore it might provide a quick-and-dirty solution to collect what is perhaps the most significant portion of the Web. One could envisage frequent periodic crawls of this most significant part of the Web, interleaved by less frequent and more expensive, but more thorough, explorations of a larger portion.

Figure 7 shows the separate growth of the two queues, primary and secondary, as well as their combination and the growth of BFS for the `.it` dataset. The parameter t of the Sydney strategy is $t = 8$. The secondary queues grows to a maximum of roughly 3% of the Web.

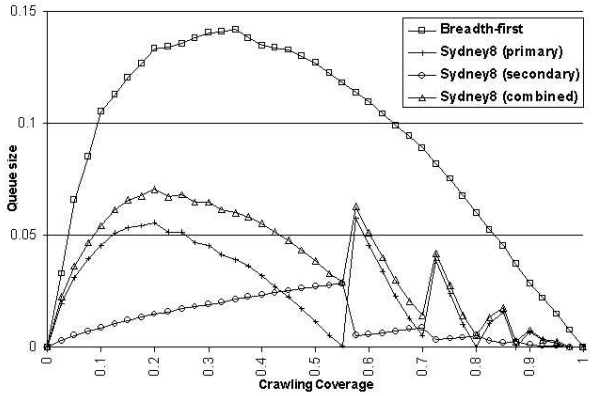

Figure 7: Separate growth of both queues (dataset `.IT`)

To summarize, we know that low-degree nodes (i.e. nodes with degree less than the parameter t of the Sydney strategy) are roughly half of the Web and cover roughly half of the Web. The secondary queue however, where the remaining half of the Web consisting of high degree nodes should eventually go, only grows to a tiny fraction of the Web. Why?

The explanation is related to the phenomenon exploited in [6] to very efficiently compress the web graph. If we sort the Web lexicographically with respect to the URLs, we see that pages having a large prefix of the URL in common, i.e. pages of the same host, or under the same directory, will have many common out-neighbors.

As an example suppose we have two vertices u and v each of out-degree 10 that have the same out-neighbourhood, and we set our sampling parameter $t = 8$. If u is extracted from the primary queue first, t out-neighbours will be marked as visited and inserted in the primary queue. Therefore when v is extracted, it will point at most to two unexplored nodes and will never enter the secondary queue. As observed in [6], this structural observation about links on the Web is robust.

3.2 The choice of the sampling parameter

Thus we have a satisfactory explanation of why the Sydney strategy is so efficient in terms of memory. We now turn to the other questions. Concerning the best value of t, naturally setting $t = \infty$ yields a pure breadth-first search with a very large primary queue. On the other end, setting a t that is too small implies that the secondary queue will grow too large.

The problem with the secondary queue is that we need to visit those nodes again to extract their out-links. These re-visits impose extra costs in terms of network connectivity, but fortunately, only a few nodes have to be visited more than once. In Table 1 we show the maximum queue size, and the fraction of nodes that have to be re-visited in the WebBase collection, for different values of t. There is an interesting trade-off between the number of maximum out-links taken per page and the maximum queue size. For instance, when we increase t from 8 to 16, the maximum queue size grows from 50% to 64% of that of BFS, but only 1% of the pages have to be visited more than once.

Table 1: Maximum total queue size (primary and secondary queue) and fraction of re-visits for the Sydney Strategy with parameter t.

Max. outlinks (t)	*All*	64	32	16	8
Max. queue size	0.14	0.12	0.11	0.09	0.07
(w.r.t. BFS)	100%	86%	79%	64%	50%
Re-visited nodes	0%	0.1%	0.4%	1.0%	2.2%

If we are operating in a less restrictive environment, doing a second network request for the nodes in the secondary queue may not be necessary. The URLs contained in the pages with many out-links can be stored on disk sequentially as (compressed) text, and then added to the crawler's queue later, when necessary. In fact, it should be done in that way in large collections, as the Web is a very dynamic environment and the nodes in the secondary queue may change.

3.3 A fixed-budget Sydney Strategy

In the experiments we have been discussing, the memory at our disposal was potentially unlimited. In some situations there might be a fixed, pre-defined amount of memory available to use. With a fixed amount of memory, at some point the algorithm will have to discard some links, so some pages will not be downloaded. Therefore we ran some experiments to see how this strategy performs in terms of coverage.

Figure 8 shows the coverage when the (combined) queue size Sydney strategy with parameter $t = 8$ is limited to, 3%, 5% and 7% of the portion of the Web that is explored respectively. Once the limit is reached, the crawler does not add new links to any of the two queues until more space becomes available.

Note that coverage is full when the memory budget is equal to 7%. In general, it seems that a smaller t gives better coverage. Figure 8 (bottom) is the same as Figure 8 (top) with respect to the Sydney strategy with parameter $t = 16$. With the same memory budget coverage is inferior.

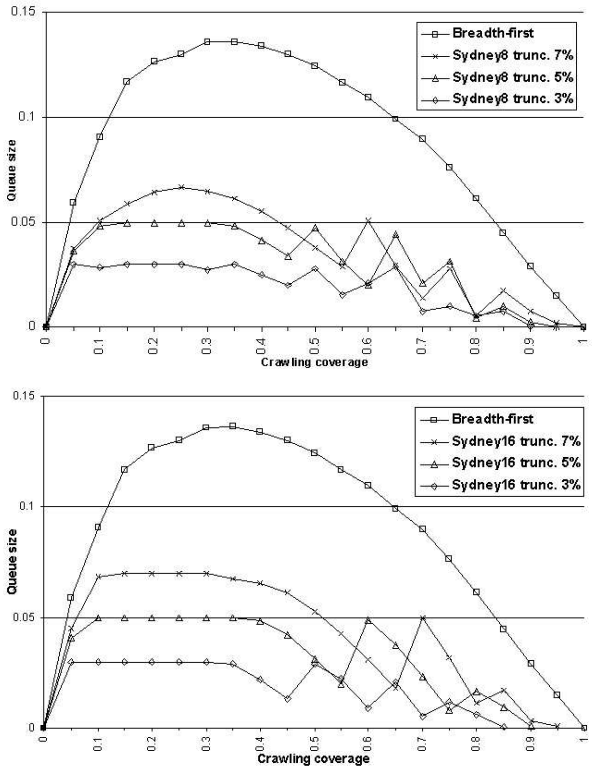

Figure 8: Memory usage of the fixed-budget Sydney strategy. Here the combined size of the queues is limited to 3%, 5% and 7% of the total exploration space. Top: $t = 8$, bottom: $t = 16$ (dataset .IT).

Table 2 shows the coverage of the different fixed-budget Sydney strategies. Interestingly, allowing more out-links to be sampled ($t = 16$) actually **worsens** the coverage compared to a more restrictive approach ($t = 8$). The explanation is that under a fixed-budget scenario, it is important to limit the growth of the primary queue to attain better coverage. In this regard, a more conservative sampling ($t = 8$) achieves better results.

Table 2: Coverage of the fixed-budget Sydney strategy. Sampling more out-links seems to have a detrimental effect on the coverage, mostly due to the fast growth of the primary queue.

	Queue limit		
Sampling parameter	3%	5%	7%
$t = 8$ out-links max.	93.8%	97.5%	99.9%
$t = 16$ out-links max.	89.0%	94.8%	98.1%

4. QUALITY FIRST

We now investigate the quality of the retrieved pages under these strategies. We used three objective criteria that may provide reasonable proxies for quality: (a) Cumulative PageRank (used in basically all crawling scheduling studies such as [2, 5, 7, 9]); (b) Cumulative in-degree, and (c) Cumulative number of home pages that are collected as the visit proceeds.

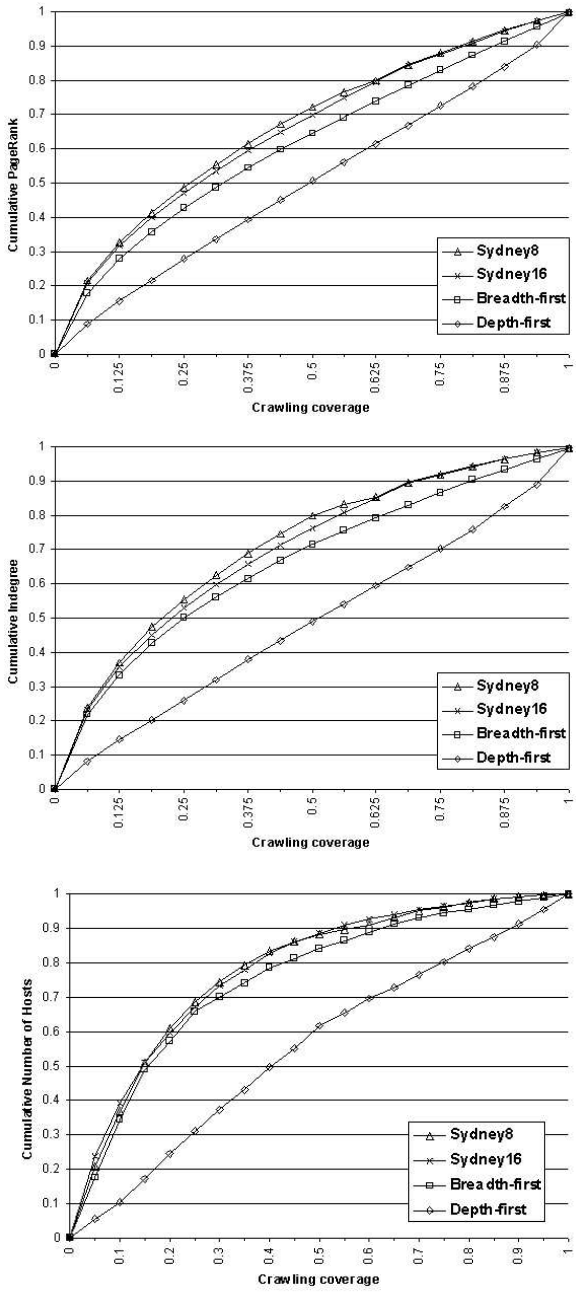

Figure 9: Measures of quality of the obtained collections with the different crawling strategies. Top: cumulative PageRank, center: cumulative in-degree, bottom: cumulative fraction of home pages (WebBase data set).

According to all of these metrics the Sydney strategy performs very well. In particular, it is comparable to, and usually better than pure BFS. Figure 9 (top) shows the cumulative PageRank of various strategies for the WebBase collection. The same trend is observed in the other data sets we tried. The Sydney strategy is somewhat better than BFS for both values of t that we tried ($t = 8, 16$). We do not have a convincing explanation of this fact.

Figure 9 (center) shows the cumulative in-degree of the same set of strategies, again for the WebBase collection. The same trend is exhibited by the other data sets we tried. The Sydney strategy is somewhat better than BFS.

Figure 9 (bottom) shows the cumulative number of home pages that are visited by the strategies as the visit proceeds, and the difference is small among the strategies, with a slight advantage for the Sydney strategy over BFS for both values of t ($t = 8, 16$).

As for the truncated strategy that selects a random sample of the out-links and does not use a secondary queue (shown in Figure 6), the behavior for the quality metrics is the same, except that the curves are interrupted because the truncated strategies do not attain full coverage. The truncated strategy with $t = 8$ that covers roughly 45% of the Web captures about 81% of the total PageRank, while the strategy with $t = 16$ that covers 75% of the Web captures 89% of the total PageRank. This supports the idea that the loss of coverage of the truncated strategies does not include many high-quality nodes.

Finally, for the fixed-budget strategy (shown in Figure 8), the performance in terms of quality is also better than breadth-first search. The loss of coverage in the worst case we observed was about 11%, but the maximum loss of cumulative PageRank is 5% in our experiments.

5. CONCLUSION

The conclusion is robust: the Sydney strategy reduces the queue size, without reducing the quality of the obtained collection and in fact often improving it. The same holds for the Sydney strategy with fixed budget. Next we discuss applications for this result and future work.

5.1 Applications

Why is it important to keep the queue size small? A simple, but valid, answer is that saving resources such as memory, where the queue is usually to be stored, is always important. This is especially true for Web applications such as crawling, for which the typical size is of data is huge.

More specifically, the list of URLs in the crawling frontier can be stored as a priority queue, in which the next URL, or set of URLs, has to be chosen carefully; part of this priority queue has to be manipulated in main memory or kept as a data structure on disk. As we have seen, this may become very large compared to the size of the collection. In the strategies we have presented here, up to 50% of the size of this data structure can be saved, and this can be done at the expense of a few re-visits to some nodes, or by saving lists of links in secondary memory in sequential files that do not need to be searched nor sorted and that are read back to main memory in batch mode.

In our case we are dealing with a simulated crawl over a known subset of the Web; in the case of a large-scale search engine, the techniques we have presented are even more useful as instead of growing and then shrinking, the queue of pending pages continues to grow without bounds. For crawlers that operate at Web scale, it is typical that when the search engine operator stops the crawler, there are more pages in the crawling queue than the pages that were actually downloaded. It is sensible, then, to avoid putting in the crawling queue too many pages, especially if many of them will never be actually visited.

The issue of saving space in the crawling queue is even more relevant for crawlers that do not operate over the entire Web graph. An example of this are information agents that must compute aggregate queries over a set of Web pages, for instance, to find the minimum price for a good or service. These information agents do "on demand" crawling on behalf of one or several users, and may even operate in the background on desktop PCs, where resources are at a premium. On the Semantic Web [4], it may become increasingly common that some information needs are not answered by a search engine by inspecting a set of pre-indexed pages, but have to be fulfilled by recursively querying a set of data sources and aggregating the results.

5.2 Future work

We have also studied other techniques involving selective visits of pages. A promising approach we have tested is to exploit the directory structure of some sites, picking only a fraction of the out-links of home pages and pages in the first few levels of a Web site, but selecting all of the out-links of deeper pages. The reason is that pages deeper inside a Web site are more difficult to find as they have lower in-degrees. This approach also leads to savings in the queue size without having a large impact on the coverage. However, the savings are comparable to those obtained with the simpler strategy we presented on this paper.

The ideas we have presented here can be combined with quality estimators of Web pages (for instance, with on-line page importance computations [1]) so a page with a large out-degree, but a low quality estimator should be added to the to the secondary queue. We are currently studying these quality-aware strategies.

6. REFERENCES

[1] S. Abiteboul, M. Preda, and G. Cobena. Adaptive on-line page importance computation. In *Proceedings of the twelfth international conference on World Wide Web*, pages 280–290, Budapest, Hungary, 2003. ACM Press.

[2] R. Baeza-Yates, C. Castillo, M. Marín, and A. Rodríguez. Crawling a country: Better strategies than breadth-first for web page ordering. In *Proceedings of the 14th international conference on World Wide Web / Industrial and Practical Experience Track*, pages 864–872, Chiba, Japan, 2005. ACM Press.

[3] A. L. Barabási and R. Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, October 1999.

[4] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific American*, May 2001.

[5] P. Boldi, M. Santini, and S. 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.

[6] P. Boldi and S. Vigna. The webgraph framework I: compression techniques. In *WWW '04: Proceedings of the 13th international conference on World Wide Web*, pages 595–602, New York, NY, USA, 2004. ACM Press.

[7] J. Cho, H. Garcia-Molina, and L. Page. Efficient crawling through url ordering. In *Proceedings of the seventh conference on World Wide Web*, Brisbane, Australia, 1998. Elsevier Science.

[8] A. Gulli and A. Signorini. The indexable Web is more than 11.5 billion pages. In *Poster proceedings of the 14th international conference on World Wide Web*, pages 902–903, Chiba, Japan, 2005. ACM Press.

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

[10] J. Hirai, S. Raghavan, H. Garcia-Molina, and A. Paepcke. Webbase: a repository of web pages. *Computer Networks (Amsterdam, Netherlands: 1999)*, 33(1–6):277–293, 2000.

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

[12] J. Leskovec, J. Kleinberg, and C. Faloutsos. Graphs over time: densification laws, shrinking diameters and possible explanations. In *KDD '05: Proceeding of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 177–187, New York, NY, USA, 2005. ACM Press.

[13] T. Meng and H.-F. Yan. On the peninsula phenomenon in web graph and its implications on web search. *Computer Networks*, In Press, 2006.

[14] M. Mitzenmacher. Dynamic models for file sizes and double pareto distributions. *Internet Mathematics*, 1(3):305–333, 2003.

[15] 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 Science.