Efficient Crawling Through URL Ordering

Junghoo Cho Hector Garcia-Molina Lawrence Page
Department of Computer Science
Stanford, CA 94305
{cho, hector, page}@cs.stanford.edu

Abstract

In this paper we study in what order a crawler should visit the URLs it has seen, in order to obtain more "important" pages first. Obtaining important pages rapidly can be very useful when a crawler cannot visit the entire Web in a reasonable amount of time. We define several importance metrics, ordering schemes, and performance evaluation measures for this problem. We also experimentally evaluate the ordering schemes on the Stanford University Web. Our results show that a crawler with a good ordering scheme can obtain important pages significantly faster than one without.

1 Introduction

A *crawler* is a program that retrieves Web pages, commonly for use by a search engine [Pin94] or a Web cache. Roughly, a crawler starts off with the URL for an initial page p_0. It retrieves p_0, extracts any URLs in it, and adds them to a queue of URLs to be scanned. Then the crawler gets URLs from the queue (in some order), and repeats the process. Every page that is scanned is given to a *client* that saves the pages, creates an index for the pages, or summarizes or analyzes the content of the pages.

Crawlers are widely used today. Crawlers for the major search engines (e.g., AltaVista, InfoSeek, Excite, and Lycos) attempt to visit most text Web pages, in order to build content indexes. Other crawlers may also visit many pages, but may look only for certain types of information (e.g., email addresses). At the other end of the spectrum, we have personal crawlers that scan for pages of interest to a particular user, in order to build a fast access cache (e.g. NetAttche).

The design of a good crawler presents many challenges. Externally, the crawler must avoid overloading Web sites or network links as it goes about its business [Kos95]. Internally, the crawler must deal with huge volumes of data. Unless it has unlimited computing resources and unlimited time, it must carefully decide what URLs to scan and in what order. The crawler must also decide how frequently to revisit pages it has already seen, in order to keep its client informed of changes on the Web. Inspite of all these challenges, and the importance of crawlers on the Internet, very little research has been done on crawlers.

In this paper we address one of these important challenges: How should a crawler select URLs to scan from its queue of known URLs? If a crawler intends to perform a single scan of the entire Web, and the load placed on target sites is not an issue, then any URL order will suffice. That is, eventually every single known URL will be visited, so the order is not critical. However, most crawlers will not be able to visit every possible page for two main reasons:

- Their client may have limited storage capacity, and may be unable to index or analyze all pages. Currently the Web contains about 1.5TB and is growing rapidly, so it is reasonable to expect that most clients will not want or will not be able to cope with all that data [Kah97].
- Crawling takes time, so at some point the crawler may need to start revisiting previously scanned pages, to check for changes. This means that it may never get to some pages. It is currently estimated that over 600GB of the Web changes every month [Kah97].

In either case, it is important for the crawler to visit "important" pages first, so that the fraction of the Web that is visited (and kept up to date) is more meaningful. In the following sections, we present several different useful definitions of importance, and develop crawling priorities so that important pages have a higher probability of being visited first. We also present experimental results from crawling the Stanford University Web pages that show how effective the different crawling strategies are.

2 Importance metrics

Not all pages are necessarily of equal interest to a crawler's client. For instance, if the client is building a specialized database on a particular topic, then pages that refer to that topic are more important, and should be visited as early as possible. Similarly, a search engine may use the number of Web URLs that point to a page, the so-called *backlink count*, to rank user query results. If the crawler cannot visit all pages, then it is better to visit those with a high backlink count, since this will give the end-user higher ranking results.

Given a Web page p, we can define the importance of the page, $I(p)$, in one of the following ways. (These metrics can be combined, as will be discussed later.)

1. *Similarity to a Driving Query* Q*:* A query Q drives the crawling process, and $I(p)$ is defined to be the textual similarity between p and Q. Similarity has been well studied in the Information Retrieval (IR) community [Sal83] and has been applied to the Web environment [YLYL95]. We use $IS(p)$ to refer to the importance metric in this case. We also use $IS(p, Q)$ when we wish to make the query explicit.

   To compute similarities, we can view each document (p or Q) as an n-dimensional vector $\langle w_1, \ldots, w_n \rangle$. The term w_i in this vector represents the ith word in the vocabulary. If w_i does not appear in the document, then w_i is zero. If it does appear, w_i is set to represent the significance of the word. One common way to compute the significance w_i is to multiply the number of times the ith word appears in the document by the inverse document frequency (*idf*) of the ith word. The *idf* factor is one divided by the number of times the word appears in the entire "collection," which in this case would be the entire Web. The *idf* factor corresponds to the content discriminating power of a word: A term that appears rarely in documents (e.g., "queue") has a high *idf*, while a term that occurs in many documents (e.g., "the") has a low *idf*. (The w_i terms can also take into account where in a page the word appears. For instance, words appearing in the title of an HTML page may be given a higher weight than other words in the body.) The similarity between p and Q can then be defined as the inner product of the p and Q vectors. Another option is to use the cosine similarity measure, which is the inner product of the normalized vectors.

Note that if we do not use *idf* terms in our similarity computation, the importance of a page, $IS(p)$, can be computed with "local" information, i.e., just p and Q. However, if we use *idf* terms, then we need global information. During the crawling process we have not seen the entire collection, so we have to estimate the *idf* factors from the pages that have been crawled, or from some reference *idf* terms computed at some other time. We use $IS'(p)$ to refer to the estimated importance of page p, which is different from the actual importance $IS(p)$ that is computable only after the entire Web has been crawled. If *idf* factors are not used, then $IS'(p) = IS(p)$.

2. *Backlink Count:* The value of $I(p)$ is the number of links to p that appear over the entire Web. We use $IB(p)$ to refer to this importance metric. Intuitively, a page p that is linked to by many pages is more important than one that is seldom referenced. This type of "citation count" has been used extensively to evaluate the impact of published papers. On the Web, $IB(p)$ is useful for ranking query results, giving end-users pages that are more likely to be of general interest.

   Note that evaluating $IB(p)$ requires counting backlinks over the entire Web. A crawler may estimate this value with $IB'(p)$, the number of links to p that have been seen so far.

3. *PageRank:* The $IB(p)$ metric treats all links equally. Thus, a link from the Yahoo home page counts the same as a link from some individual's home page. However, since the Yahoo home page is more important (it has a much higher IB count), it would make sense to value that link more highly. The PageRank backlink metric, $IR(p)$, recursively defines the importance of a page to be the weighted sum of the importance of the pages that have backlinks to p. Such a metric has been found to be very useful in ranking results of user queries [PB98, Goo]. We use $IR'(p)$ for the estimated value of $IR(p)$ when we have only a subset of pages available.

   More formally, if a page has no outgoing link, we assume that it has outgoing links to every single Web page. Next, consider a page p that is pointed at by pages $t_1, \dots, t_n$. Let c_i be the number of links going out of page t_i. Also, let d be a damping factor (whose intuition is given below). Then, the weighted backlink count of page p is given by

$$IR(p) = (1 - d) + d\,[IR(t_1)/c_1 + \cdots + IR(t_n)/c_n]$$

   This leads to one equation per Web page, with an equal number of unknowns. The equations can be solved for the IR values. They can be solved iteratively, starting with all IR values equal to 1. At each step, the new $IR(p)$ value is computed from the old $IR(t_i)$ values (using the equation above), until the values converge. This calculation corresponds to computing the principal eigenvector of the link matrices.

   One intuitive model for PageRank is that we can think of a user "surfing" the Web, starting from any page, and randomly selecting from that page a link to follow. When the user reaches a page with no outlinks, he jumps to a random page. Also, when the user is on a page, there is some probability, d, that the next visited page will be completely random. This damping factor d makes sense because users will only continue clicking on one task for a finite amount of time before they go on to something unrelated. The $IR(p)$ values we computed above give us the probability that our random surfer is at p at any given time.

4. *Forward Link Count:* For completeness we may want to consider a metric $IF(p)$ that counts the number of links that emanate from p. Under this metric, a page with many outgoing links is very valuable, since it may be a Web directory. This metric can be computed directly from p, so $IF'(p) = IF(p)$. This kind of metric has been used in conjunction with other factors to reasonably identify index pages [PPR96]. We could also define a weighted forward link metric, analogous to $IR(p)$, but we do not consider it here.

5. *Location Metric:* The $IL(p)$ importance of page p is a function of its location, not of its contents. If URL u leads to p, then $IL(p)$ is a function of u. For example, URLs ending with ".com" may be deemed more useful than URLs with other endings, or URLs containing the string "home" may be more of interest than other URLs. Another location metric that is sometimes used considers URLs with fewer slashes more useful than those with more slashes. All these examples are local metrics since they can be evaluated simply by looking at the URL u.

As stated earlier, our importance metrics can be combined in various ways. For example, we may define a metric $IC(p) = k_1 \cdot IS(p, Q) + k_2 \cdot IB(p)$, for some constants k_1, k_2. This combines the similarity metric (under some given query Q) and the backlink metric. Pages that have relevant content *and* many backlinks would be the highest ranked. (Note that a similar approach was used to improve the effectiveness of a search engine [Mar97].)

3 Problem definition

Our goal is to design a crawler that if possible visits high $I(p)$ pages before lower ranked ones, for some definition of $I(p)$. Of course, the crawler will only have available $I'(p)$ values, so based on these it will have to guess what are the high $I(p)$ pages to fetch next.

Our general goal can be stated more precisely in three ways, depending on how we expect the crawler to operate. (In our evaluations of Section 5 we use the second model in most cases, but we do compare it against the first model in one experiment. Nevertheless, we believe it is useful to discuss all three models to understand the options.)

- **Crawl & Stop:** Under this model, the crawler C starts at its initial page p_0 and stops after visiting K pages. At this point a perfect crawler would have visited pages $r_1, \ldots, r_K$, where r_1 is the page with the highest importance value, r_2 is the next highest, and so on. We call pages r_1 through r_K the "hot" pages. The K pages visited by our real crawler will contain only M pages with rank higher than or equal to $I(r_K)$. We define the performance of the crawler C to be $P_{CS}(C) = M/K$. The performance of the ideal crawler is of course 1. A crawler that somehow manages to visit pages entirely at random, and may revisit pages, would have a performance of K/T, where T is the total number of pages in the Web. (Each page visited is a hot page with probability K/T. Thus, the expected number of desired pages when the crawler stops is K^2/T.)

- **Crawl & Stop with Threshold:** We again assume that the crawler visits K pages. However, we are now given an importance target G, and any page with $I(p) \geq G$ is considered hot. Let us assume that the total number of hot pages is H. The performance of the crawler, $P_{ST}(C)$, is the fraction of the H hot pages that have been visited when the crawler stops. If

$K < H$, then an ideal crawler will have performance K/H. If $K \geq H$, then the ideal crawler has the perfect performance 1. A purely random crawler that revisits pages is expected to visit $(H/T) \cdot K$ hot pages when it stops. Thus, its performance is K/T. Only when the random crawler visits all T pages its performance is expected to be 1.

- **Limited Buffer Crawl:** In this model we consider the impact of limited storage on the crawling process. We assume that the crawler can only keep B pages in its buffer. Thus, after the buffer fills up, the crawler must decide what pages to flush to make room for new pages. An ideal crawler could simply drop the pages with lowest $I(p)$ value, but a real crawler must guess which of the pages in its buffer will eventually have low $I(p)$ values. We allow the crawler to visit a total of T pages, equal to the total number of Web pages. At the end of this process, the fraction of the B buffer pages that are hot gives us the performance $P_{BC}(C)$. We can define hot pages to be those with $I(p) \geq G$, where G is a target importance, or those with $I(p) \geq I(r_B)$, where r_B is the page with the Bth highest importance value. The performance of an ideal and a random crawler are analogous to those in the previous cases.

Note that to evaluate a crawler under any of these metrics, we need to compute the actual $I(p)$ values of pages, and this involves crawling the "entire" Web. To keep our experiments (Section 5) manageable, we imagines that the Stanford University pages form the entire Web, and we only evaluate performance in this context. That is, we assume that all pages outside of Stanford have $I(p) = 0$, and that links to pages outside of Stanford or links from pages outside of Stanford do not count in $I(p)$ computations. In Section 5.2 we study the implications of this assumption by also analyzing a smaller Web within the Stanford domain, and seeing how Web size impacts performance.

4 Ordering metrics

A crawler keeps a queue of URLs it has seen during a crawl, and must select from this queue the next URL to visit. The ordering metric O is used by the crawler for this selection, i.e., it selects the URL u such that $O(u)$ has the highest value among all URLs in the queue. The O metric can only use information seen (and remembered if space is limited) by the crawler.

The O metric should be designed with an importance metric in mind. For instance, if we are searching for high $IB(p)$ pages, it makes sense to use $O(u) = IB'(p)$, where p is the page u points to. However, it might also make sense to use $O(u) = IR'(p)$, even if our importance metric is not weighted. In our experiments, we explore the types of ordering metrics that are best suited for either $IB(p)$ or $IR(p)$.

For a location importance metric $IL(p)$, we can use that metric directly for ordering since the URL of p directly gives the $IL(p)$ value. However, for forward link $IF(p)$ and similarity $IS(p)$ metrics, it is much harder to devise an ordering metric since we have not seen p yet. As we will see, for similarity, we may be able to use the text that anchors the URL u as a predictor of the text that p might contain. Thus, one possible ordering metric $O(u)$ is $IS(A, Q)$, where A is the anchor text of the URL u, and Q is the driving query.

5 Experiments

To avoid network congestion and heavy loads on the servers, we did our experimental evaluation in two steps. In the first step, we physically crawled all Stanford Web pages and built a local repository of the pages. This was done with the Stanford WebBase [BP98], a system designed to create and maintain large Web repositories.

After we built the repository, we ran our *virtual* crawlers on it to evaluate different crawling schemes. Note that even though we had the complete image of the Stanford domain in the repository, our virtual crawler based its crawling decisions only on the pages it saw for itself. In this section we briefly discuss how the particular database was obtained for our experiments.

5.1 Description of dataset

To download an image of the Stanford Web pages, we started WebBase with an initial list of "`stanford.edu`" URLs. These 89,119 URLs were obtained from an earlier crawl. During the crawl, non-Stanford URLs were ignored. Also, we limited the actual data that we collected for two reasons. The first is that many heuristics are needed to avoid automatically generated, and potentially infinite, sets of pages. For example, any URLs containing "`/cgi-bin/`" are not crawled, because they are likely to contain programs which generate infinite sets of pages, or produce other undesirable side effects such as an unintended vote in an online election. Several other heuristics similar to aspects discussed in the location metric described earlier are used to eliminate URLs which *look* undesirable. Another way the data set is reduced is through the *robots exclusion protocol* [Rob], which allows Webmasters to define pages they do not want crawled by automatic systems.

At the end of the process, we had 784,592 known URLs to Stanford pages. It should be noted that 352,944 of the known URLs were on one server, `http://www.slac.stanford.edu`, which has a program that could generate an unlimited number of Web pages. The crawl was stopped before it was complete, but most of the uncrawled URLs were on only a few servers, so we believe the dataset we used to be a reasonable representation of the `stanford.edu` Web. This dataset consisted of about 225,000 crawled "valid" HTML pages, [1] which consumed roughly 2.5GB of disk space. However, out of these 225,000 pages, 46,000 pages were unreachable from the starting point of the crawl, so the total number of pages for our experiments was 179,000.

We should stress that the virtual crawlers that will be discussed next do not use WebBase directly. As stated earlier, they use the dataset collected by the WebBase crawler, and do their own crawling on it. The virtual crawlers are simpler than the WebBase crawler. For instance, they can detect if a URL is invalid simply by seeing if it is in the dataset. Similarly, they do not need to distribute the load to visited sites. These simplifications are fine, since the virtual crawlers are only used to evaluate ordering schemes, and not to do real crawling.

5.2 Backlink-based crawlers

In this section we study the effectiveness of various ordering metrics, for the scenario where importance is measured through backlinks (i.e., either the $IB(p)$ or $IR(p)$ metrics). We start by describing the structure of the virtual crawler, and then consider the different ordering metrics. Unless otherwise noted, we use the Stanford dataset described in Section 5.1, and all crawls are

[1] We considered a page *valid* when its Web server responded with the HTTP header "`200 OK`."

```
Algorithm 5.1         Crawling algorithm (backlink based)
Input: starting_url: seed URL
Procedure:
  [1] enqueue(url_queue, starting_url)
  [2] while (not empty(url_queue))
  [3]   url = dequeue(url_queue)
  [4]   page = crawl_page(url)
  [5]   enqueue(crawled_pages, (url, page))
  [6]   url_list = extract_urls(page)
  [7]   foreach u in url_list
  [8]     enqueue(links, (url, u))
  [9]     if (u∉url_queue and (u,-)∉crawled_pages)
  [10]      enqueue(url_queue, u)
  [11]  reorder_queue(url_queue)

Function description:
  enqueue(queue, element): append element at the end of queue
  dequeue(queue)         : remove the element at the beginning
                             of queue and return it
  reorder_queue(queue)   : reorder queue using information in
                             links (refer to Figure 2)
```

Figure 1: Basic crawling algorithm

started from the Stanford homepage. For the PageRank metric we use a damping factor d of 0.9 (for both $IR(p)$ and $IR'(p)$) for all of our experiments.

Figure 1 shows our basic virtual crawler. The crawler manages three main data structures. Queue `url_queue` contains the URLs that have been seen and need to be visited. Once a page is visited, it is stored (with its URL) in `crawled_pages`. Finally, `links` holds pairs of the form (u_1, u_2), where URL u_2 was seen in the visited page with URL u_1. The crawler's ordering metric is implemented by the function `reorder_queue()`, shown in Figure 2. We used three ordering metrics: (1) breadth-first (2) backlink count $IB'(p)$, and (3) PageRank $IR'(p)$. The breadth-first metric places URLs in the queue in the order in which they are discovered, and this policy makes the crawler visit pages in breadth-first order.

We start by showing in Figure 3 the crawler's performance with the backlink ordering metric. In this scenario, the importance metric is the number of backlinks to a page ($I(p) = IB(p)$) and we consider a *Crawl & Stop with Threshold* model in Section 3 with G either 3, 10, or 100. Recall that a page with G or more backlinks is considered important, i.e., hot. Under these hot page definitions, about H = 85,000 (47%), 17,500 (10%) and 1,400 (0.8%) pages out of 179,000 total Web pages were considered hot, respectively.

In Figure 3, the horizontal axis is the fraction of the Stanford Web pages that has been crawled over time. At the right end of the horizontal axis, all 179,000 pages have been visited. The vertical axis represents P_{ST}, the fraction of the total hot pages that has been crawled at a given point. The

```
(1) breadth first
      do nothing (null operation)

(2) backlink count, $IB'(p)$
      foreach $u$ in url_queue
         backlink_count[$u$] = number of terms (-,$u$) in links
      sort url_queue by backlink_count[$u$]

(3) PageRank $IR'(p)$
      solve the following set of equations:
         $IR[u] = (1 - 0.9) + 0.9 \sum_i \frac{IR[v_i]}{c_i}$, where
         $(v_i, u) \in$ links and $c_i$ is the number of links in the page $v_i$
      sort url_queue by $IR(u)$
```

Figure 2: Description of `reorder_queue()` of each ordering metric

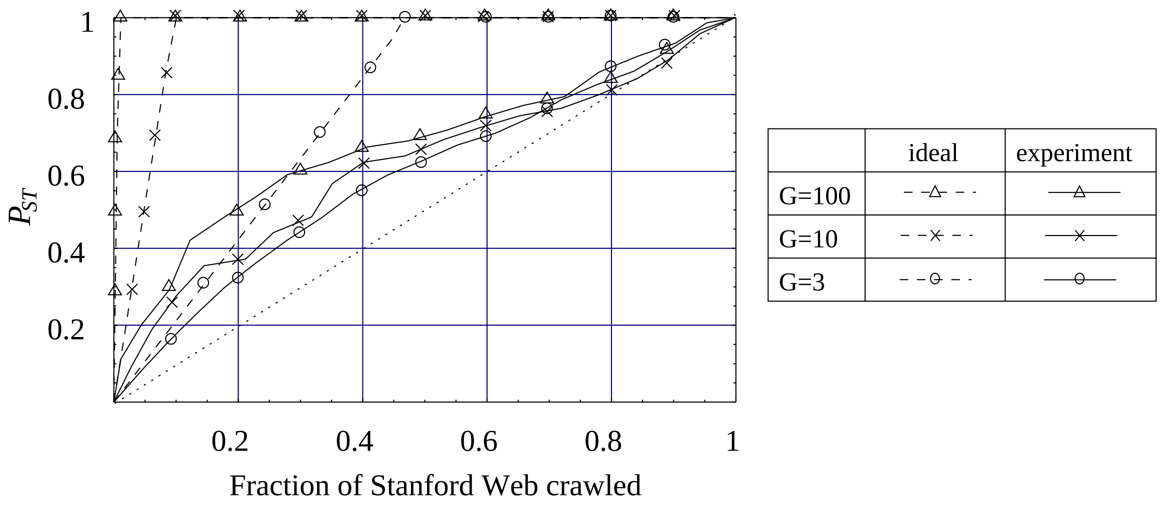

Figure 3: Fraction of Stanford Web crawled vs. P_{ST}. $I(p) = IB(p)$; $O(u) = IB'(p)$.

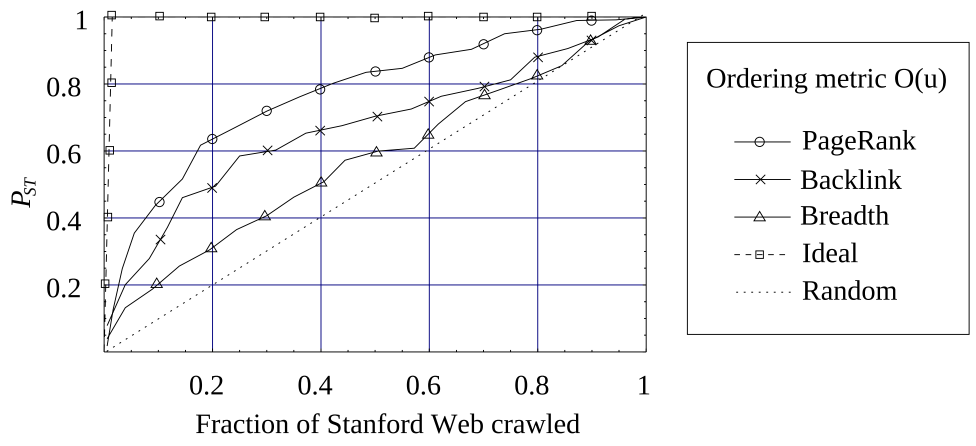

Figure 4: Fraction of Stanford Web crawled vs. P_{ST}. $I(p) = IB(p)$; $G = 100$.

solid lines in the figure show the results from our experiments. For example, when the crawler in our experiment visited 0.2 (20%) of the Stanford pages, it crawled 0.5 (50%) of the total hot pages for $G = 100$. The dashed lines in the graph show the expected performance of ideal crawlers. An ideal crawler reaches performance 1 when H pages have been crawled. The dotted line represents the performance of a random crawler, and it increases linearly over time.

The graph shows that as our definition of a hot page becomes more stringent (larger G), the faster the crawler can locate the hot pages. This result is to be expected, since pages with many backlinks will be seen quickly after the crawl starts. Figure 3 also shows that even if G is large, finding the "last" group of hot pages is always difficult. That is, to the right of the 0.8 point on the horizontal axis, the crawler finds hot pages at roughly the same rate as a random crawler.

In our next experiment we compare three different ordering metrics: 1) breadth-first 2) backlink-count and 3) PageRank (corresponding to the three functions of Figure 2). We continue to use the *Crawl & Stop with Threshold* model, with $G = 100$, and a $IB(p)$ importance metric. Figure 4 shows the results of this experiment. The results are rather counterintuitive. Intuitively one would expect that a crawler using the backlink ordering metric $IB'(p)$ that matches the importance metric $IB(p)$ would perform the best. However, this is not the case, and the PageRank metric $IR'(p)$ outperforms the $IB'(p)$ one. To understand why, we manually traced the crawler's operation. We noticed that often the $IB'(p)$ crawler behaved like a depth-first one, frequently visiting pages in one "cluster" before moving on to the next. On the other hand, the $IR'(p)$ crawler combined breadth and depth in a better way.

To illustrate, let us consider the Web fragment of Figure 5. With $IB'(p)$ ordering, the crawler visits a page like the one labeled p_1 and quickly finds a cluster A of pages that point to each other. The A pages temporarily have more backlinks than page p_2, so the visit of page p_2 is delayed even if page p_2 actually has more backlinks than the pages in cluster A. On the other hand, with $IR'(p)$ ordering, page p_2 may have higher rank (because its link comes from a high ranking page) than the pages in cluster A (that only have pointers from low ranking pages within the cluster). Therefore, page p_2 is reached faster.

In summary, during the early stages of a crawl, the backlink information is biased by the starting point. If the crawler bases its decisions on this skewed information, it tries getting locally hot pages instead of globally hot pages, and this bias gets worse as the crawl proceeds. On the other hand,

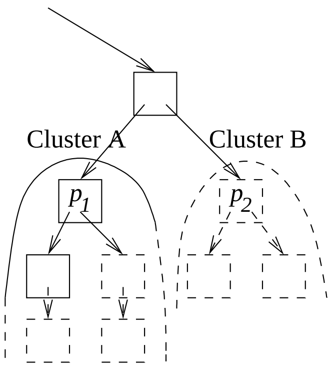

Figure 5: Crawling order

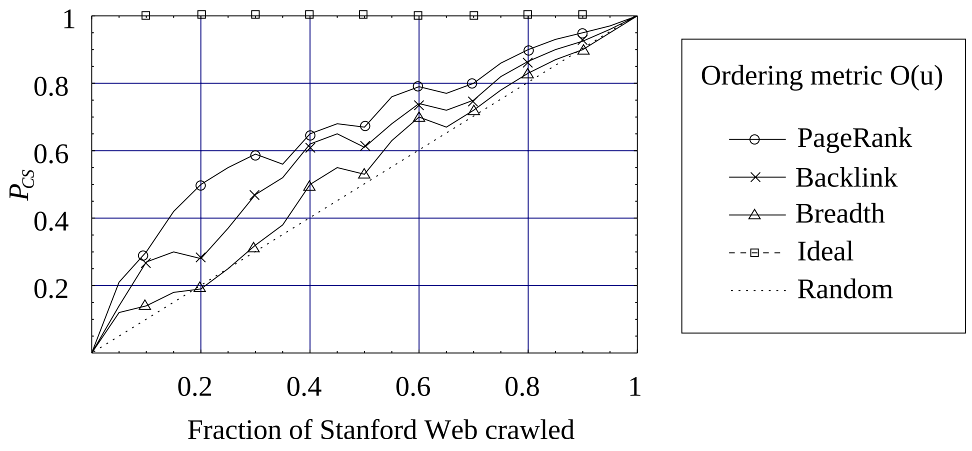

Figure 6: Fraction of Stanford Web crawled vs. P_{CS}. $I(p) = IB(p)$.

the $IR'(p)$ PageRank crawler is not as biased towards locally hot pages, so it gives better results regardless of the starting point.

Figure 6 shows that this conclusion is not limited to the *Crawl & Stop with Threshold* model. In the figure we show the performance of the crawlers under the *Crawl & Stop* model (Section 3). Remember that under the *Crawl & Stop* model, the definition of hot pages changes over time. That is, the crawler does not have a predefined notion of hot pages, and instead, when the crawler has visited, say, 30% of the entire Web, it considers the top 30% pages as hot pages. Therefore, an ideal crawler would have performance 1 at all times because it would download pages in the order of their importance. Figure 6 compares 1) breadth-first 2) backlink and 3) PageRank ordering metrics for the $IB(p)$ importance metric under this model. The vertical axis represents P_{CS}, the crawled fraction of hot pages at each point under the varying definition of hot pages. The figure shows that the results of the *Crawl & Stop* model are analogous to those of the *Crawl & Stop with Threshold* model: The PageRank ordering metric shows the best performance.

Returning to the *Crawl & Stop with Threshold* model, Figure 7 shows the results when we use the $IR(p)$ PageRank importance metric with $G = 13$.[2] Again, the PageRank ordering metric shows

[2]When $G = 13$ for the $IR(p)$ metric, the number of hot pages was about 1,700 (1%), which is close to the 1,400

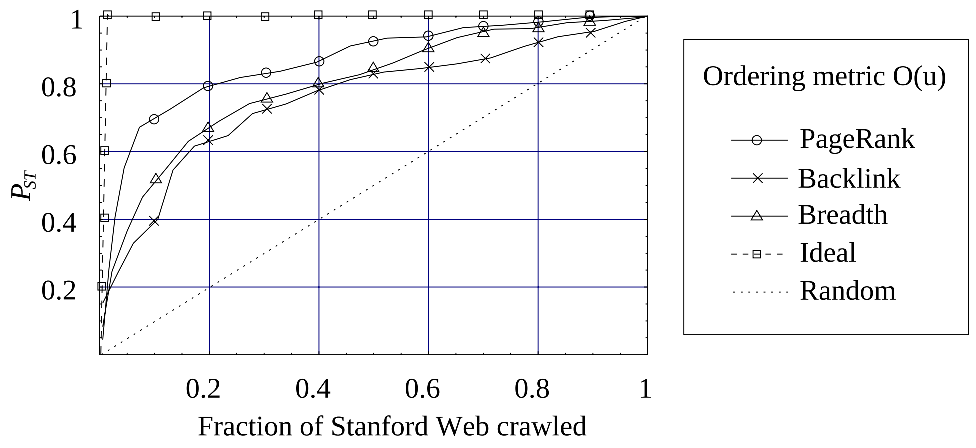

Figure 7: Fraction of Stanford Web crawled vs. P_{ST}. $I(p) = IR(p)$; $G = 13$.

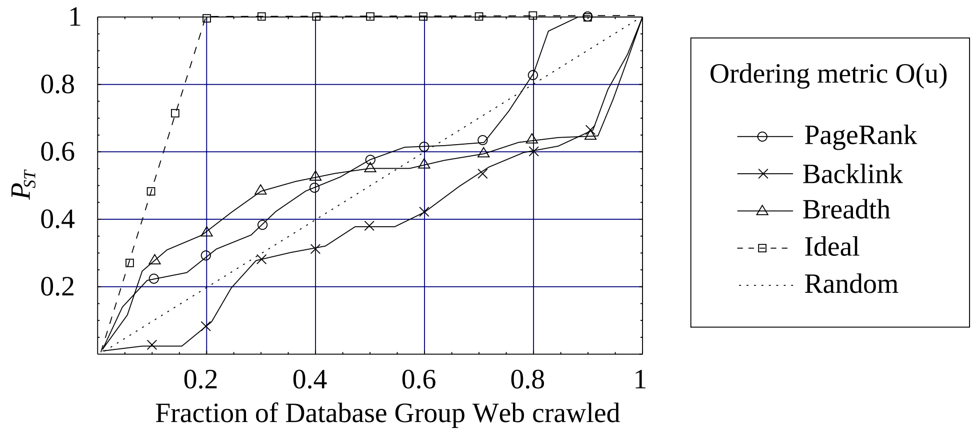

Figure 8: Fraction of DB group Web crawled vs. P_{ST}. $I(p) = IB(p)$; $G = 5$.

the best performance. The backlink and the breadth-first metrics show similar performance. Based on these results, we recommend using the PageRank ordering metric for both the $IB(p)$ and the $IR(p)$ importance metrics.

5.3 Small-scale crawl

In some cases the crawler's client may only be interested in small portions of the Web. For instance, the client may be interested in a single site (to create a mirror, say). In this subsection we evaluate the ordering metrics in such a scenario.

To study the impact of scale on the performance of a crawler we ran experiments similar to those in Section 5.2 only on the pages of the Stanford Database Group (on the server `http://www-db.stanford.edu`). This subset of the Stanford pages consists of about 1,100 HTML pages, which is much smaller than the entire Stanford domain. In most of our experiments for the Database Group domain, crawling performance was not as good as for the Stanford domain. Figure 8 shows

of Figure 4.

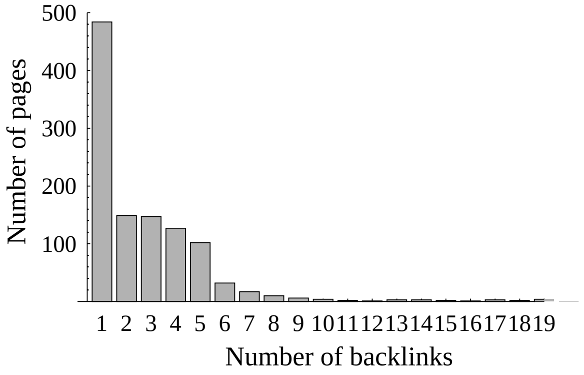

Figure 9: Histogram of backlink counts within DB group Web

one of the results. In this case, we use the *Crawl & Stop with Threshold* model with $G = 5$ with the importance metric $IB(p)$. The graph shows that performance can be even worse than that of a random crawler at times, for all ordering metrics.

This poor performance is mainly because an importance metric based on backlinks is not a good measure of importance for a small domain. In a small domain, most pages have only a small number of backlinks and the number of backlinks therefore is very sensitive to a page creator's style. For example, Figure 9 shows the histogram for the number of backlinks in the Database Group domain. The vertical axis shows the number of pages with a given backlink count. We can see that most pages have fewer than 5 backlinks. In this range, the rank of each page varies greatly according to the style used by the creator of the page. If the creator generates many cross-links between his pages, then his pages have a high $IB(p)$ rank, otherwise they do not. Therefore, the rank is very sensitive and is not a good measure of the importance of the pages.

In Figure 8 we can see the impact of "locally dense" clusters that have many "locally popular" but "globally unpopular" pages. The performance of the backlink $IB'(p)$ crawler is initially quite flat: It initially does a depth-first crawl for the first cluster it found. After visiting about 20% of the pages, the crawler suddenly discovers a large cluster, and this accounts for the jump in the graph. On the other hand, the PageRank $IR'(p)$ crawler found this large cluster earlier, so its performance is much better initially.

In Figure 10 we show the results on the Database Group Web when the importance metric is $IR(p)$ PageRank metric with $G = 3$.[3] All three ordering metrics show better performance under the $IR(p)$ metric than under the $IB(p)$ metric, but still performance is not as good as that of the larger Stanford domain. Again, the $IR'(p)$ ordering metric shows the best performance.

5.4 Similarity-based crawlers

In the experiments of two previous subsections, we compared three different backlink-based crawlers. In this subsection, we present the results of our experiments on similarity-based crawlers. The similarity-based importance metric, $IS(p)$, measures the relevance of each page to a topic or a query that the user has in mind. There are clearly many possible $IS(p)$ metrics to consider, so our experiments here are not intended to be comprehensive. Instead, our goal is to briefly explore

[3]When $G = 3$, the number of hot pages is similar to that of Figure 8.

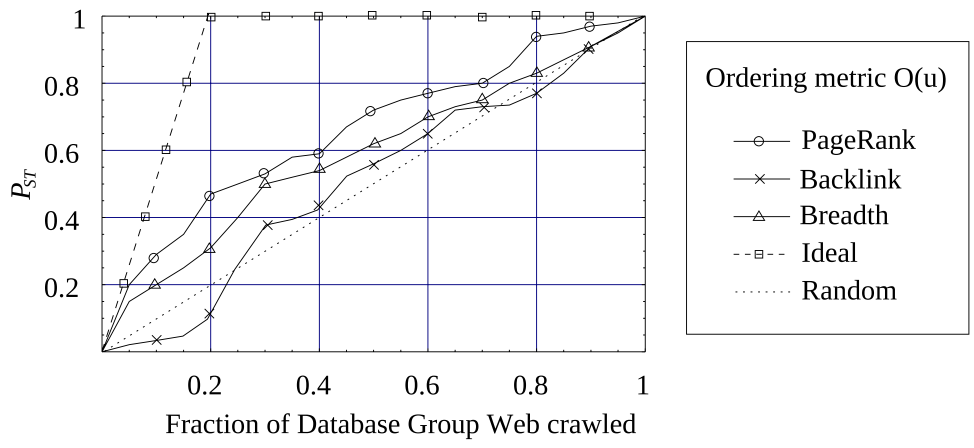

Figure 10: Percentage of DB group Web crawled vs. P_{ST}. $I(p) = IR(p)$; $G = 3$.

the *potential* of various ordering schemes in some sample scenarios. In particular, for our first two experiments we consider the following $IS(p)$ definition: A page is considered hot if it contains the word *computer* in its title or if it has more than 10 occurrences of *computer* in its body.[4]

For similarity-based crawling, the crawler of Figure 1 is not appropriate, since it does not take the content of the page into account. To give priority to the pages related to the topic of interest, we modified our crawler as shown in Figure 11. This crawler keeps two queues of URLs to visit: `hot_queue` stores the URLs with the topic word *computer* in their anchors or in the URLs themselves. The second queue, `url_queue`, keeps the rest of the URLs. The crawler always prefers to take URLs to visit from `hot_queue`.

Figure 12 shows the P_{ST} results for this crawler for the $IS(p)$ importance metric defined above. The horizontal axis represents the fraction of the Stanford Web pages that has been crawled and the vertical axis shows the crawled fraction of the total hot pages. The results show that the backlink-count and the PageRank crawler behaved no better than a random crawler. Only the breadth-first crawler gave a reasonable result. This result is rather unexpected: All three crawlers differ only in their ordering metrics, which are neutral to the page content. All crawlers visited computer-related URLs immediately after their discovery. Therefore, all the schemes are theoretically equivalent and should give comparable results.

The observed unexpected performance difference arises from the breadth-first crawler's FIFO nature. The breadth-first crawler fetches pages in the order they are found. If a computer-related page is crawled earlier, then the crawler discovers and visits its child pages earlier as well. These pages have a tendency to be computer related, so performance is better.

Thus the observed property is that if a page has a high $IS(p)$ value, then its children are likely to have a high $IS(p)$ value too. To take advantage of this property, we modified our crawler as shown in Figure 13. This crawler places in the `hot_queue` URLs that have the target keyword in their anchor or URL, or that are within 3 links from a hot page.

Figure 14 illustrates the result of this crawling strategy. All crawlers showed significant improvement and the difference between the breadth-first crawler and the others decreased. While the breadth-first crawler is still superior to the other two, we believe that this difference is mainly

[4]In our third experiment we consider a different topic, *admission*, and show the results.

Algorithm 5.2 Crawling algorithm (modified similarity based)
Input: `starting_url`: seed URL
Procedure:

```
[1] enqueue(url_queue, starting_url)
[2] while (not empty(hot_queue) and not empty(url_queue))
[3]   url = dequeue2(hot_queue, url_queue)
[4]   page = crawl_page(url)
[5]   enqueue(crawled_pages, (url, page))
[6]   url_list = extract_urls(page)
[7]   foreach u in url_list
[8]     enqueue(links, (url, u))
[9]     if (u ∉ url_queue and u ∉ hot_queue and (u,-) ∉ crawled_pages)
[10]      if (u contains computer in anchor or url)
[11]        enqueue(hot_queue, u)
[12]      else
[13]        enqueue(url_queue, u)
[14]  reorder_queue(url_queue)
[15]  reorder_queue(hot_queue)
```

Function description:

```
dequeue2(queue1, queue2): if (not empty(queue1)) dequeue(queue1)
                          else dequeue(queue2)
```

Figure 11: Similarity-based crawling algorithm

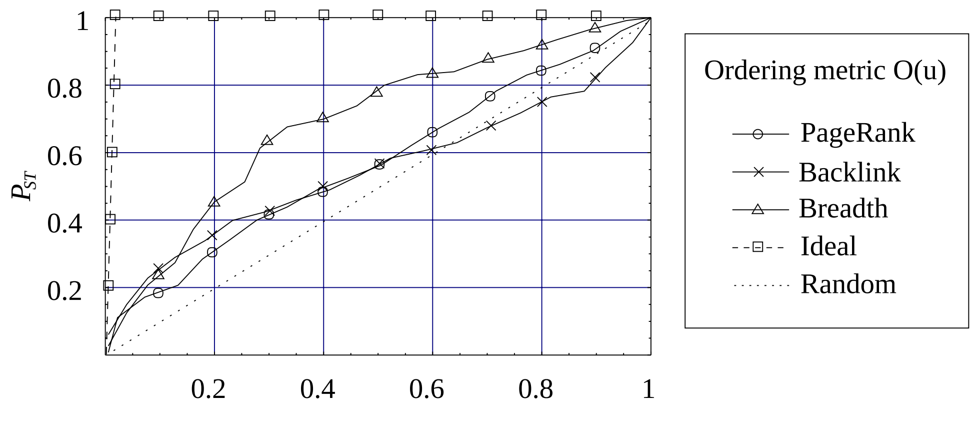

Figure 12: Basic similarity-based crawler. $I(p) = IS(p)$; topic is *computer*.

```
Algorithm 5.3          Crawling algorithm (similarity based)
Input: starting_url: seed URL
Procedure:
  [1] enqueue(url_queue, starting_url)
  [2] while (not empty(hot_queue) and not empty(url_queue))
  [3]   url = dequeue2(hot_queue, url_queue)
  [4]   page = crawl_page(url)
  [5]   if (page contains 10 or more computer in body
              or one computer in title)
  [6]     hot[url] = True
  [7]   enqueue(crawled_pages, (url, page))
  [8]   url_list = extract_urls(page)
  [9]   foreach u in url_list
  [10]    enqueue(links, (url, u))
  [11]    if (u ∉ url_queue and u ∉ hot_queue and (u,-) ∉ crawled_pages)
  [12]      if (u contains computer in anchor or url)
  [13]        enqueue(hot_queue, u)
  [14]      else if (distance_from_hotpage(u) < 3)
  [15]        enqueue(hot_queue, u)
  [16]      else
  [17]        enqueue(url_queue, u)
  [18]  reorder_queue(url_queue)
  [19]  reorder_queue(hot_queue)

Function description:
  distance_from_hotpage(u):
    return 0 if (hot[u] = True)
    return 1 if (hot[v] = True and (v, u) ∈ links for some v)
    ...
```

Figure 13: Modified similarity-based crawling algorithm

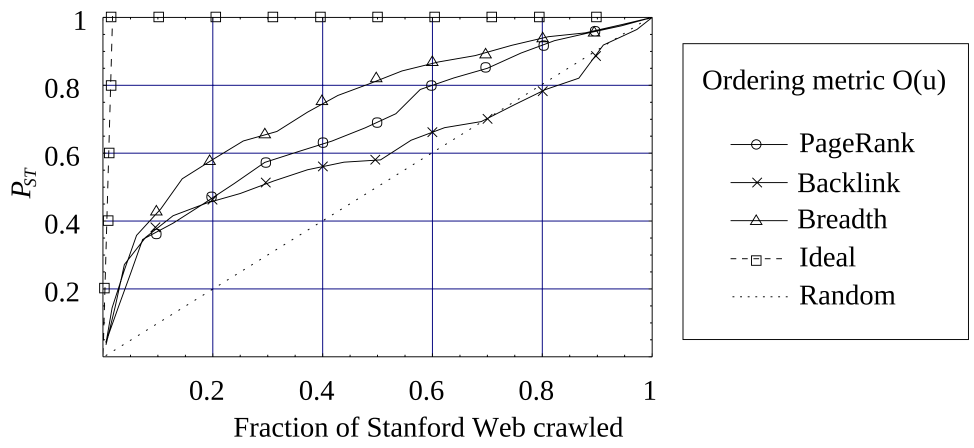

Figure 14: Modified similarity-based crawler. $I(p) = IS(p)$; topic is *computer*.

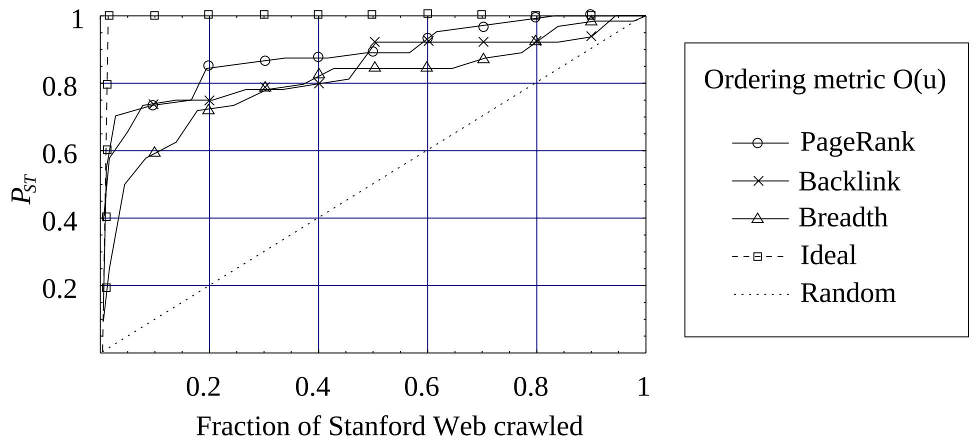

Figure 15: Modified similarity-based crawler. Topic is *admission*.

due to statistical variation. In our other experiments, including the next one, the PageRank crawler shows similar or sometimes better performance than the breadth-first crawler.

In our final experiment, with results shown in Figure 15, we repeat the scenario reported in Figure 14 with a different query topic. In this case, the word *admission* is considered of interest. Details are not identical to the previous case, but the overall conclusion is the same: When similarity is important, it is effective to use an ordering metric that considers 1) the content of anchors and URLs and 2) the distance to the hot pages that have been discovered.

6 Conclusion

In this paper we addressed the problem of ordering URLs for crawling. We defined several different kinds of importance metrics, and built three models to evaluate crawlers. We experimentally evaluated several combinations of importance and ordering metrics, using the Stanford Web pages.

In general, our results show that PageRank, $IR'(p)$, is an excellent ordering metric when either pages with many backlinks or with high PageRank are sought. In addition, if the similarity to a driving query is important, it is useful to visit earlier URLs that:

- Have anchor text that is similar to the driving query;
- Have some of the query terms within the URL itself; or
- Have a short link distance to a page that is known to be hot.

With a good ordering strategy, we can build crawlers that can obtain a significant portion of the hot pages relatively early. This property can be extremely useful when we are trying to crawl a fraction of the Web, when our resources are limited, or when we need to revisit pages often to detect changes.

One limitation of our experiments is that they were run only over the Stanford Web pages. We believe that the Stanford pages are a reasonable sample: For example, they are managed by many different people who structure their pages in a variety of ways. They include many individual home pages, and also many clusters that are carefully managed by organizations. Nevertheless, it will be interesting to investigate in the future non-Stanford Web pages to analyze structural differences and their implication for crawling.

7 Acknowledgement

This material is based upon work supported by the National Science Foundation under Cooperative Agreement IRI-9411306. Funding for this cooperative agreement is also provided by DARPA, NASA, and the industrial partners of the Stanford Digital Libraries Project.

References

[BP98] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. In *Proceedings of the Seventh International World-Wide Web Conference*, Brisbane, Australia, April 1998.

[Goo] Google Inc. `http://www.google.com`.

[Kah97] Brewster Kahle. Archiving the internet. *Scientific American*, March 1997.

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

[Mar97] Massimo Marchiori. The quest for correct information on the web: Hyper search engines. In *Proceedings of the Sixth International World-Wide Web Conference*, pages 265–276, Santa Clara, California, April 1997.

[PB98] Lawrence Page and Sergey Brin. The anatomy of a large-scale hypertextual web search engine. In *Proceedings of the Seventh International World-Wide Web Conference*, Brisbane, Australia, April 1998.

[Pin94] Brian Pinkerton. Finding what people want: Experiences with the web crawler. In *Proceedings of the Second World-Wide Web Conference*, Chicago, Illinois, October 1994.

[PPR96] Peter Pirolli, James Pitkow, and Ramana Rao. Silk from a sow's ear: Extracting usable structures from the web. In *Proceedings of the Conference on Human Factors in Computing Systems CHI'96*, pages 118–125, Vancouver, British Columbia, Canada, April 1996.

[Rob] Robots exclusion protocol. `http://info.webcrawler.com/mak/projects/robots/exclusion.html`.

[Sal83] Gerard Salton. *Introduction to modern information retrieval*. McGraw-Hill, first edition, 1983.

[YLYL95] Budi Yuwono, Savio L. Lam, Jerry H. Ying, and Dik L. Lee. A world wide web resource discovery system. In *Proceedings of the Fourth International World-Wide Web Conference*, Darmstadt, Germany, April 1995.