Efficient Crawling Through URL Ordering

Junghoo Cho, Hector Garcia-Molina, Lawrence Page
Department of Computer Science
Stanford University

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 [Pinkerton 1994] 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 [1], InfoSeek [2], Excite [3], and Lycos [4]) 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 [5], WebSnake [6]).

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 [Koster 1995]. 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 [Kahle 1997] .

- 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 [Kahle 1997].

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 this paper we present several 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.

Of course, a crawler must also avoid overloading target sites, especially if they contain many important pages. In this paper we do not address this issue. That is, we assume that URLs selected for scanning may be delayed by a crawler component that monitors site loads, but we do not study here how this delay component works. Similarly, we do not consider in this paper rescanning pages. To simplify our evaluations, we assume that a crawler does not start revisiting pages until it has finished visiting pages. In practice one may of course wish to start rescanning important pages even before a crawl is completed, but this is beyond the scope of this paper.

2. Importance Metrics

Not all pages are of equal interest to the 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 poss ible. 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 [Salton 1989] and has been applied to WWW environment [Yuwono 1995]. 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 m-dimensional vector $<w_1, ..., w_n>$. The term w_i in this vector represents the i^{th} 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 i^{th} word appears in the document by the inverse document frequency (*idf*) of the i^{th} 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 terms can also take into account 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., *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)*, which can be computed 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 backlinks to it. Such a metric has been found to be very useful in ranking results of user queries [Page 1998.2]. 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 $T_1, ..., T_n$. 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

   $$IR(P) = (1-d) + d\,(IR(T_1)/c_1 + ... + IR(T_n)/c_n)$$

   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. PageRank is described in much greater detail in [Page 1998.2].

   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 [Pirolli 1996]. We could also define a weighted forward link metric, analogous to $IR(P)$, but we do not consider this 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 URL 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 [Marchiori 1997].)

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 Sections 6 and 7 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 $R_1, ..., R_K$, 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* $P_{CS}(C) = (M \cdot 100)/K$. The performance of the ideal crawler is of course 100%. A crawler that somehow manages to visit pages entirely at random, and may revisit pages, would have a performance of $(K \cdot 100)/T$, 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 $I(P) >= G$ 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 percentage of the H hot pages that have been visited when the crawler stops. If $K < H$, then an ideal crawler will have performance $(K \cdot 100)/H$. If $K >= H$, then the ideal crawler has 100% performance. A purely random crawler that revisits pages is expected to visit $(H/T) \cdot K$ hot pages when it stops. Thus, its performance is $(K \cdot 100)/T$. Only if the random crawler visits all T pages, is its performance expected to be 100%.

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 percentage of the B buffer pages that are hot gives us the performance $P_{BC}(C)$. We can define hot pages to be those with $I(P) >= G$, where G is a target importance, or those with $I(P) >= I(R_B)$, where R_B is the page with the B^{th} highest importance value. The performances 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 6 and 7) manageable, we define the entire Web to be the Stanford University pages, and we only evaluate performance in this context. That is, we assume that all pages outside Stanford have $I(P) = 0$, and that links to pages outside Stanford or from pages outside Stanford do not count in $I(P)$ computations. In Section 6.4 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 the 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 an $O(u) = IB'(P)$, where P is the page u points to. However, it might also make sense to consider an $O(u) = IR'(P)$, 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 $IL(P)$ importance metric, 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. Experimental Setup

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, a system designed to create and maintain large web repositories. It is capable of high indexing speeds (about 50 pages per second), and large data repositories (currently 150GB of HTML is stored).

After we built the repository, we ran our *virtual* crawlers on it to evaluate the 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 WebBase crawler operates, and how the particular database was obtained for our experiments.

5.1 WebBase Crawler

WebBase runs several processes at a time, which crawl web pages. These processes receive a list of URLs to be downloaded and simply return the full content of the HTML or any errors which happened trying to get the pages. The crawling processes open several hundred connections at a time, resulting in a crawling speed of about 25 pages/second for each process. Since servers can only handle a few tens of hits per second at most, and slowing down servers with a web crawler is a problem, we use two different kinds of load balancing in our system. First, our system splits all URLs which are going to be crawled into 500 queues based on a hash of their server name. This causes all URLs from a given server to go into the same queue. The crawlers then read one URL from each queue at a time, moving to a new queue for each URL. This makes sure a given server is hit only once for every 500 URLs that are crawled. Also, for servers that are slow at returning documents, only one connection is allowed from the crawler to a particular server at a time. As is mentioned in Section 1, these kinds of load balancing requirements affect the freedom a crawler has in deciding the crawl order.

The actual data the system is allowed to get is reduced 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 producing other undesirable side effects such as an unintended vote in an online election. Several other heuristics based on the *Location Metric* described above are used to weed out URLs which *look* undesirable. Another way the data set is reduced is through the robots exclusion protocol[7], which allows webmasters to define pages they do not want crawled by automatic systems.

5.2 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. 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, www.slac.stanford.edu [8], which has a program that generates infinite numbers 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 using roughly 2.5 GB of space for the raw pages.

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 Web Base 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.

6.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, and all crawls are started from the Stanford homepage [9]. 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 (including those of the following section).

Crawling algorithm (backward link based)

```
enqueue(url_queue, starting_url);
while (not empty(url_queue)) {
   url = dequeue(url_queue);
   page = crawl_page(url);
   enqueue(crawled_pages, (url, page));
   url_list = extract_urls(page);
   for each u in url_list
       enqueue(links, (url, u));
       if [u not in url_queue] and
          [(u,-) not in crawled_pages]
          enqueue(url_queue, u);
   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 Fig 2.

Figure 1. Basic crawling algorithm

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 makes the crawler visit pages in breadth-first order.

(1) breadth first

```
do nothing. (null operation)
```

(2) backlink count, *IB'(P)*

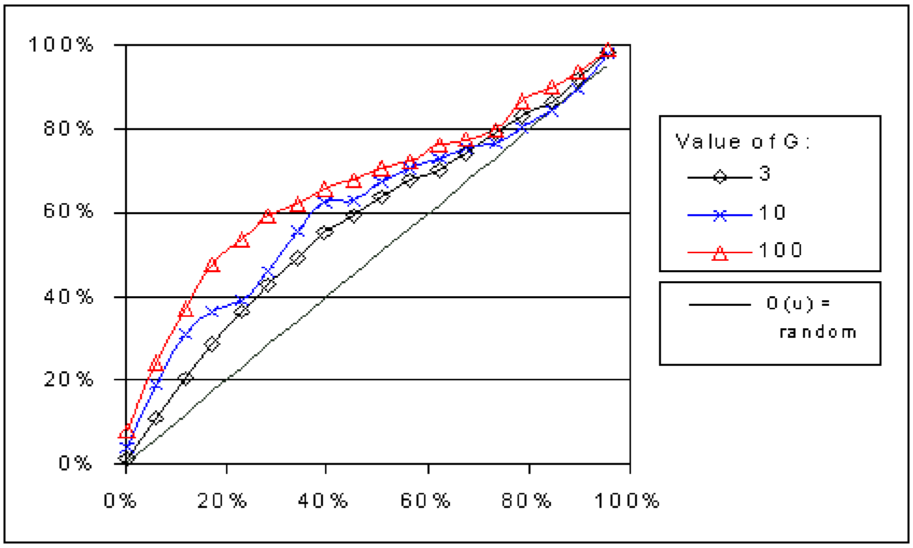
```
for each u in url_queue {
   backlink_count[u] = number of terms (-,u) in links;
}
sort url_queue by backlink_count[u];
```

(3) PageRank, *IR'(P)*

```
solve the following set of equations {
   for each u in url_queue:
      pagerank[u] = (1-0.9) + 0.9•sum_i(pagerank[v_i]/c_i)
         where (v_i, u) in links and
            c_i = number of links in the page v_i;
         ( sum_i(p_i) = p_1 + p_2 + ... + p_n )
}
sort url_queue by pagerank[u];
```

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

We start by showing in Graph 1 the crawler's performance with the backlink ordering metric. In this scenario, the importance metric is the number of backlinks ($I(P) = IB(P)$), and we consider a Crawl & Stop with Threshold model (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 valid web pages were considered hot, respectively. (As stated in Section 5, our dataset has 225,000 valid pages. However, out of these, 46,000 pages were unreachable from the starting point of the crawling, so the total number of pages for the experiment is 179,000.)



Graph 1. Percentage of Stanford Web crawled vs. P_{ST}

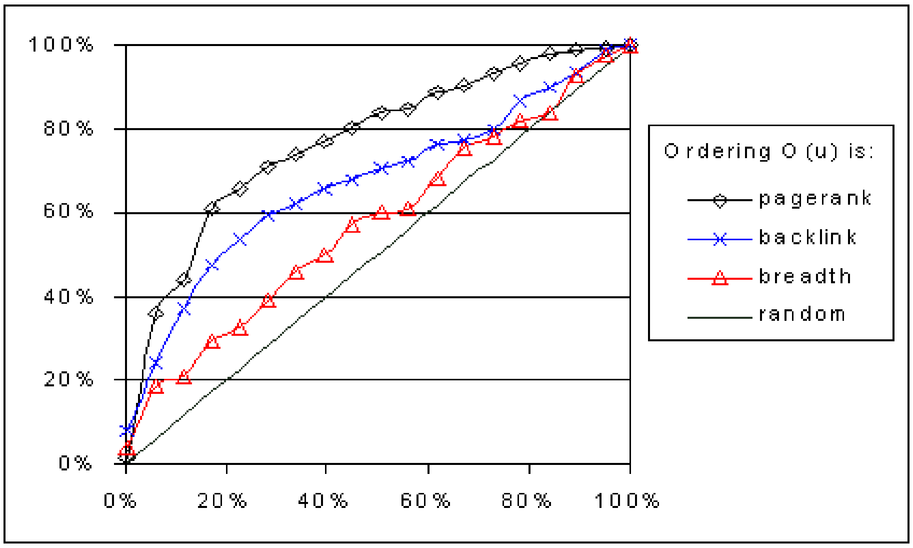
$I(P) = IB(P)$; $O(u) = IB'(P)$

In Graph 1, the horizontal axis is the percentage of the dataset that has been crawled over time. At the 100% mark, all 179,000 pages have been visited. For each visited fraction we report on the vertical axis P_{ST}, the percentage of the total hot pages that has been visited so far. Keep in mind that the crawler does not know which pages are hot for sure. Thus, Graph 1 can only be produced at the end of the complete crawl, once we know how many pages had more than *G* links to them.

Graph 1 also shows the performance P_{ST} of a random crawler. As discussed in Section 3, the performance of a random crawler is a straight diagonal line. An ideal crawler (not shown) would reach 100% performance when *H* pages have been crawled.

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 is to be expected, since pages with many backlinks will be seen quickly after the crawl starts. The graph also shows that even if *G* is large, finding the "last" hot pages is always difficult. That is, to the right of the 80% 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, breadth-first, backlink-count, and 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. Graph 2 shows the result of this experiment, together with those for a random crawler.



Graph 2. Percentage of Stanford Web crawled vs. P_{ST}
I(P) = IB(P); G = 100

The results are rather counter-intuitive. That is, intuitively one would expect that a crawler using an ordering metric *IB'(P)* that matches the importance metric *IB(P)* would perform the best. However, this is not the case, and the *IR'(P)* metric 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 consider the Web fragment of Figure 3.

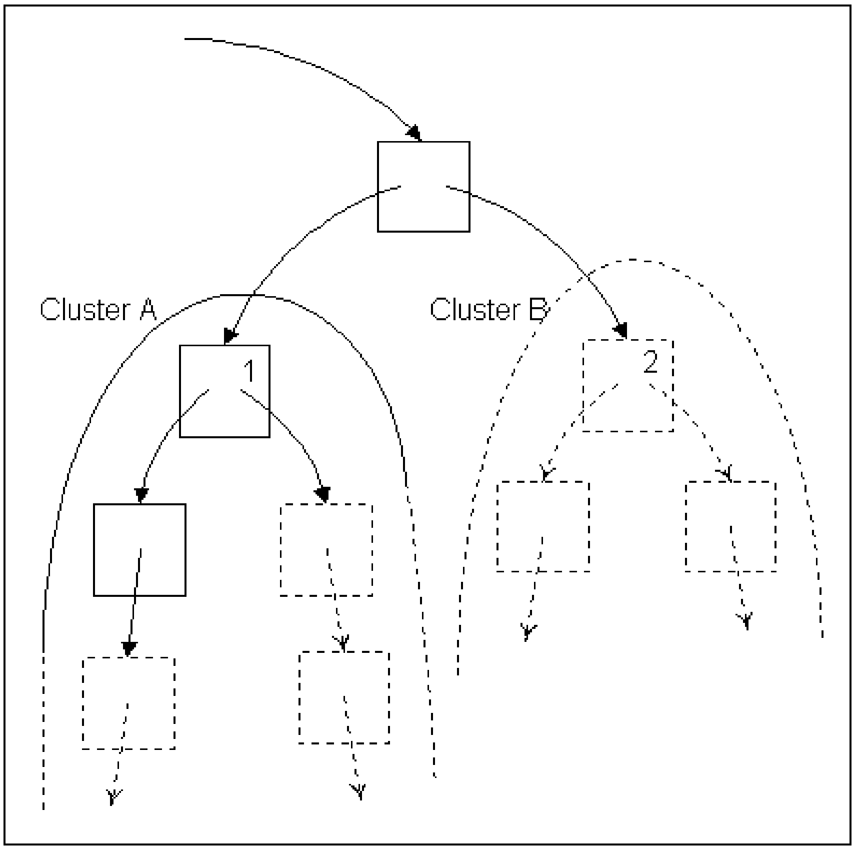
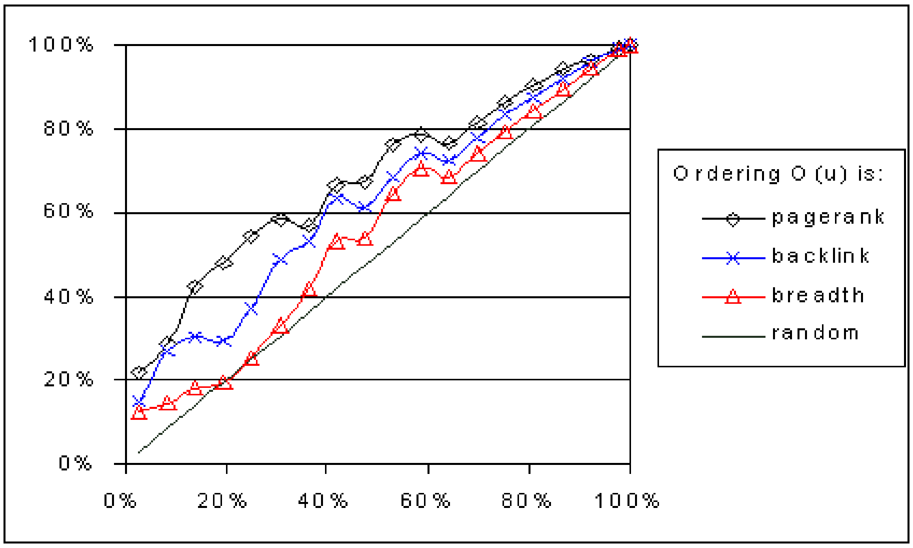


Figure 3. Crawling order

With *IB'(P)* ordering, the crawler visits a page like the one labeled 1 and quickly finds a cluster *A* of pages that point to each other. The *A* pages temporarily have more backlinks than page 2, so the visit of page 2 is delayed. However, since the whole Web has not been crawled, it may be that page 2 has more backlinks than the *A* pages. On the other hand, with *IR'(P)* ordering, page 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 2 is reached faster.

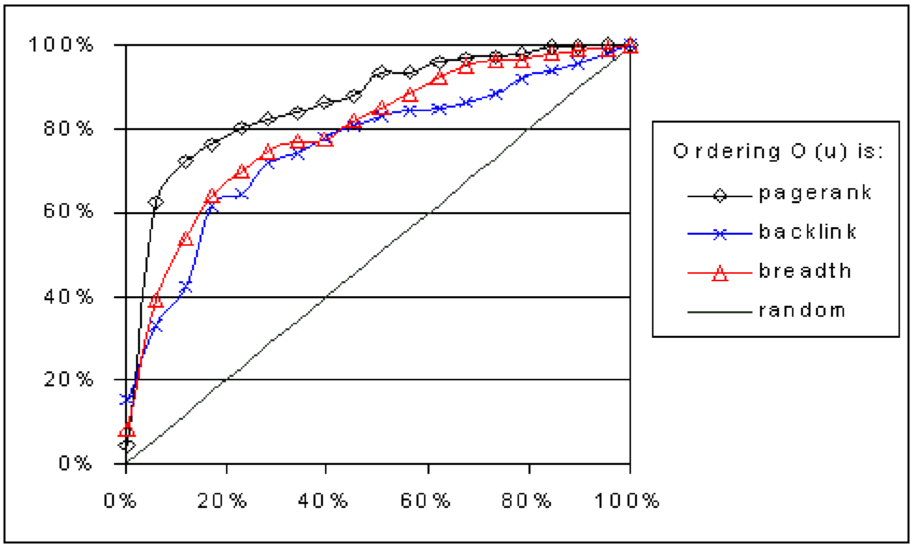
In summary, during the early stages of a crawl, the backlink information is biased by the starting point. If the crawler bases its decisions on this skewed information, it tries getting locally hot pages instead of globally hot pages, and this bias gets worse as the crawl proceeds. On the other hand, the *IR'(P)* PageRank crawler is not as biased towards locally hot pages, so it gives better results regardless of the starting point.



Graph 3. Percentage of Stanford Web crawled vs. P_{CS}
I(P) = IB(P)

Graph 3 shows the performance P_{CS} of the crawlers under the Crawl & Stop model. The setting is the same as for Graph 2. Keep in mind that an ideal crawler would now have 100% performance at all times. The results are analogous to those of the Crawl & Stop with Threshold model. The key different is that the P_{CS} results are not dependent on a G value. Thus, the metric P_{CS} may be appropriate if we do not have a predefined notion of what constitutes a hot page. If we do, then P_{ST} may be better suited, since it can tells us more accurately how the crawler performs in fetching the hot pages.

Returning to the Crawl & Stop with Threshold model, Graph 4 shows the results of using the *IR(P)* PageRank importance metric. The results are similar to those of Graph 2, except that the *IR'(P)* is even more effective now.

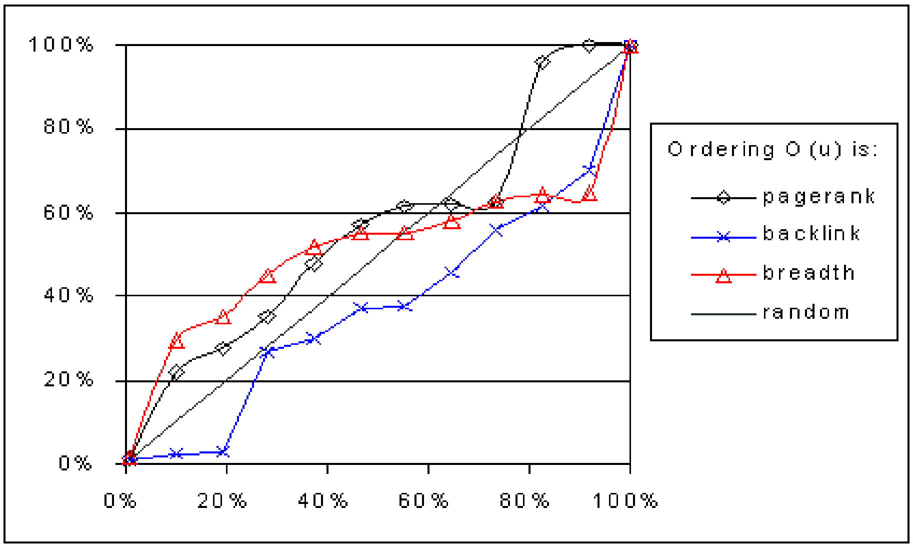


Graph 4. Percentage of Stanford Web crawled vs. P_{ST}

I(P) = IR(P); G = 13

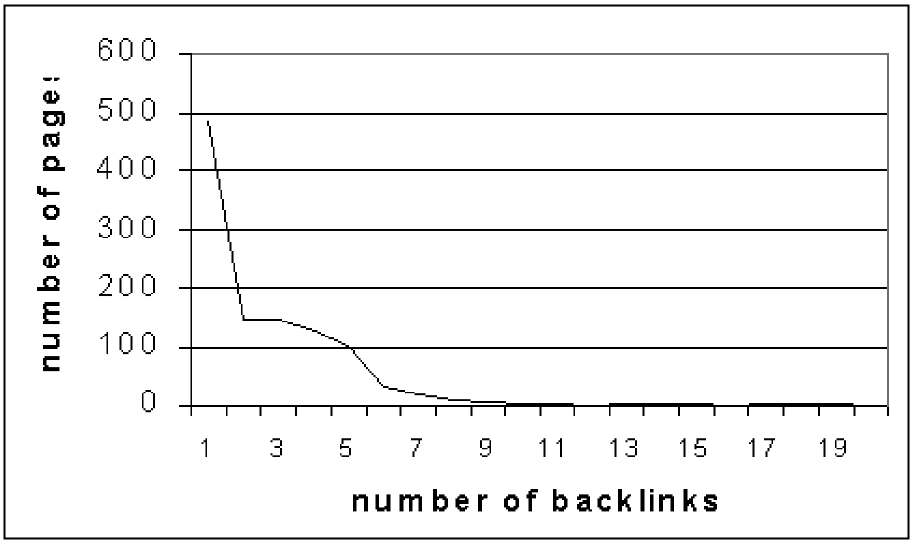
As discussed in Section 1, in some cases the crawler's client may only be interested in small portions of the Web. For instance, the client may be interested in a single site (to create a mirror, say). In our next experiment we evaluate the ordering metrics in such a scenario. The results will also let us study the impact of the differences in scale.

For this experiment we only crawled the pages of the Stanford Database Group (on server www-db.stanford.edu [10]). This subset of the Stanford pages consists of about 1,100 valid HTML pages. In general, crawling performance is not as good on the smaller subset. Graph 5 shows one representative result. In this case, we use the Crawl & Stop with Threshold model with $G = 5$. The importance metric is *IB(P)*. The graph shows that performance can be even worse than that of a random crawler at times, for all ordering metrics.



Graph 5. Percentage of DB group Web crawled vs. P_{ST}

I(P) = IB(P); G = 5



Graph 6. Histogram of baclink counts within DB group web.

The reason for this poor performance is that *IB(P)* is not a good importance metric for a small domain. To see this, Graph 6 shows the histogram for the number of backlinks, in our sample domain. The vertical axis shows the number of pages for each backlink count. From this histogram we can see that most pages have fewer than 10 backlinks. In this range, the rank of each page varies greatly according to the style used by the creator of the Web pages. For instance, it the creator generates many cross links between his pages, then his pages have a high *IB(P)* rank, otherwise they do not. If the high rank pages do not have many links from outside the cluster created by this person, it will be hard to find them. In any case, the rank is not a good measure of the global importance of the pages.

In Graph 5 we can see the impact of "locally dense" clusters. The performance of the backlink *IB'(P)* crawler is initially quite flat. This is because 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 there. On the other hand, the PageRank *IR'(P)* crawler found this large cluster earlier, so its performance is much better initially.

7. Similarity-Based Crawlers

In the experiments of Section 6, we compared three different backlink-based crawlers. In this section, 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 the *potential* of various ordering schemes in some sample scenarios. In particular, for our first three 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 th an 10 occurrences of *computer* in its body. In our fourth experiment we consider a different definition.

Crawling algorithm (similarity-based)

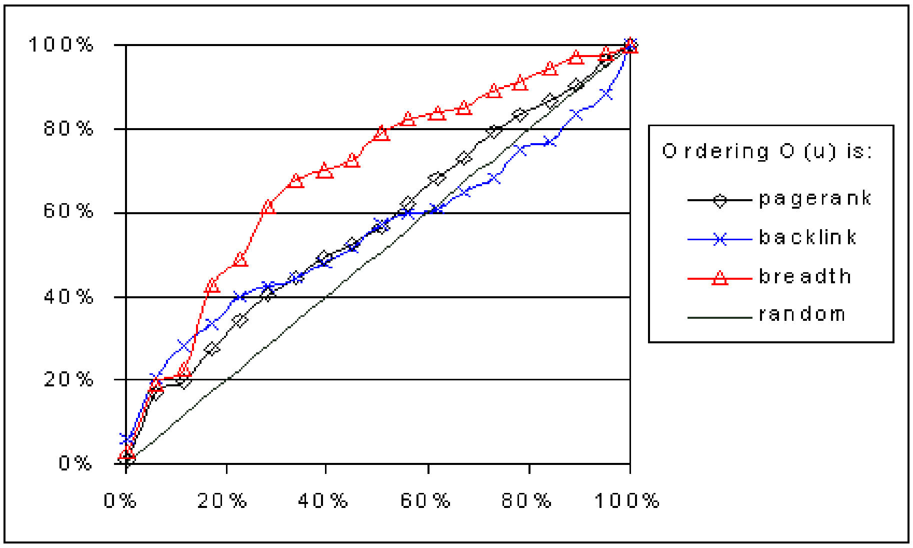
```
enqueue(url_queue, starting_url);
while (not empty(hot_queue) and not empty(url_queue)) {
    url = dequeue2(hot_queue, url_queue);
    page = crawl_page(url);
    enqueue(crawled_pages, (url, page));
    url_list = extract_urls(page);
    for each u in url_list
        enqueue(links, (url, u));
        if [u not in url_queue] and
           [u not in hot_queue] and
           [(u,-) not in crawled_pages]
            if [u contains computer in anchor or url]
                enqueue(hot_queue, u);
            else
                enqueue(url_queue, u);
    reorder_queue(url_queue);
    reorder_queue(hot_queue);
}
```

Function description

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

Figure 4. Similarity-based crawling algorithm

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 mentioning *computer*, we modified our crawler as shown in Figure 4. This crawler keeps two queues of URLs to visit: `hot_queue` stores the URLs that have been seen in an anchor mentioning the word *computer*, or that have the word *computer* within them. The second queue, `url_queue`, keeps the rest of the URLs. The crawler first takes URL to visit from `hot_queue`.



Graph 7. Basic similarity-based crawler
I(P) = IS(P); topic is *computer*

Graph 7 shows the P_{ST} results for this crawler, for the *IS(P)* importance metric defined above. 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 is a rather unexpected result. That is, all these crawlers differ only in their ordering metrics, which are neutral to the page content. All crawlers visited computer-related URLs immediately after their discovery. T herefore, 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 the 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 also 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 higher *IS(P)* value too. To take advantage of this property, we modified our crawler as shown in Figure 5. This crawler places in the `hot_queue` URLs that have the target keyword in their anchor or within, or that are within 3 links from a hot page.

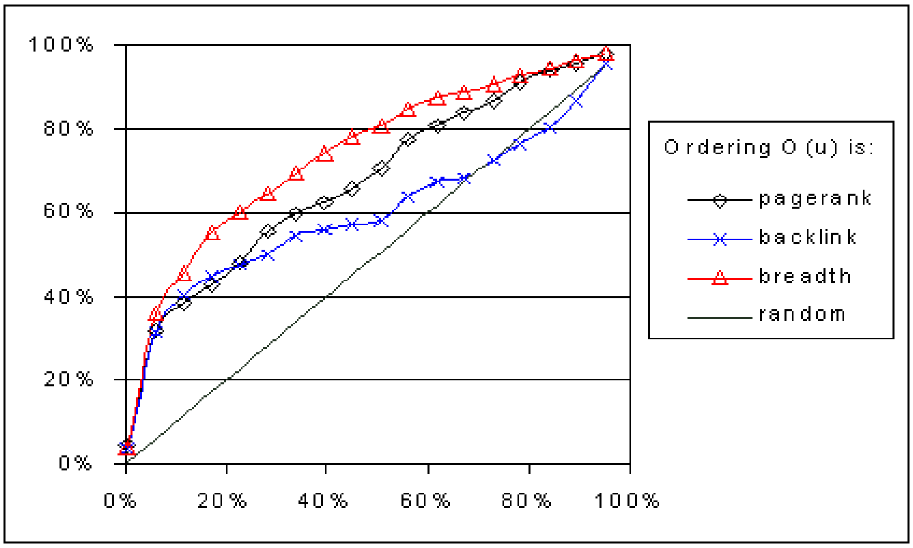
Crawling algorithm (modified similarity-based)

```
enqueue(url_queue, starting_url);
while (not empty(hot_queue) and not empty(url_queue)) {
   url = dequeue2(hot_queue, url_queue);
   page = crawl_page(url);
   if [page contains 10 or more computer in body
       or one computer in title]
       hot[url] = TRUE;
   enqueue(crawled_pages, (url, page));
   url_list = extract_urls(page);
   for each u in url_list
       enqueue(links, (url, u));
       if [u not in url_queue] and
          [u not in hot_queue] and
          [(u,-) not in crawled_pages]
          if [u contains computer in anchor or url]
              enqueue(hot_queue, u);
          else if [distance_from_hotpage(u) < 3]
              enqueue(hot_queue, u);
          else
              enqueue(url_queue, u);
   reorder_queue(url_queue);
   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) in links]
            for some v;
   return 2 if [hot[v] = TRUE] and
               [(v, w) in links] and [(w, u) in links]          for
   some v, w;
```

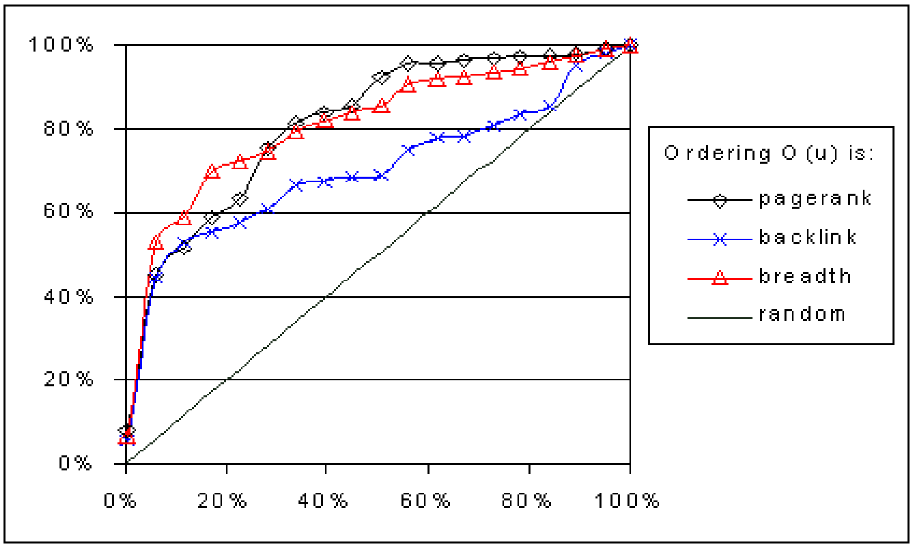
Figure 5. Modified similarity-based crawling algorithm



Graph 8. Modified similarity-based crawler
I(P) = IS(P); topic is *computer*

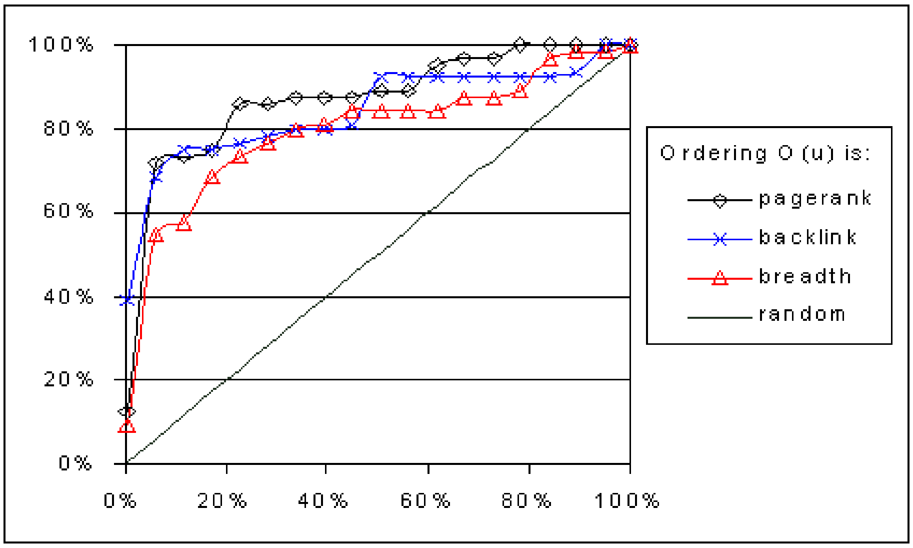
Graph 8 illustrates the result of this crawling strategy. All crawlers showed significant improvement and the difference between the breadth-first crawler and the others decreased. The main reason why the breadth-first strategy is still superior is that more than a third of the hot pages have only one backlink to them. If the anchors of these pages do not contain the word *computer* and if they are far from hot pages, their crawling is delayed. Of course, it is questionable if these pages are actually important, since nobody has pointed to them.

To address this problem, we now define a modified importance metric that takes into account similarity and backlink information. Under our new definition, a page is hot if it is on the *computer* topic (its title contains *computer* or its body has 10 or more occurrences of *computer*) and it has five or more backlinks. Graph 9 shows the P_{ST} results for this combined *IS(P)* and *IB(P)* importance metric. The results are very good now, especially for the crawlers that order based on PageRank *IR'(P)* and breadth-first order. For instance, after having seen only 40% of the pages, these crawlers have obtained over 80% of the hot pages. After visiting 60% of the pages, most of the hot pages have been gathered.



Graph 9. Modified similarity-based crawler
I(P) = combined IB(P), IS(P); topic is *computer*

In our final experiment, Graph 10, we repeat the last scenario with a different query topic. In this case, the word *admission* is considered of interest. (A page is hot is if the word appears in its title, or the word appear 10 or more times total in the body, and if the page has 5 backlinks to it.) The performance details vary from the previous case, but the overall conclusion is the same: When both similarity and the number of backlinks are important, it is effective to use a combined ordering metric that considers *IR'(P)*, the content of anchors, and the distance to pages known to be hot.



Graph 10. Modified similarity-based crawler
I(P) = combined IB(P), IS(P); topic is *admission*

8. Conclusion

In this paper we addressed the problem of ordering URLs for crawling. We listed 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, then it is also 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, it seems to be possible to build crawlers that can rather quickly obtain a significant portion of the hot pages. This can be extremely useful when we are trying to crawl large portions of the Web, when are resources are limited, or when we need to revisit pages often to detect changes.

9. References and URLs

9.1 References

[Kahle 1996] Brewster Kahle, Archiving the Internet. Extended version of the article "Preserving the Internet" that appeared in *Scientific American*, March 1997. Extended version available at: http://www.alexa.com/~brewster/essays/sciam_article.html(Published version at http://www.sciam.com/0397issue/0397kahle.html)

[Koster 1995] Martijn Koster, Robots in the Web: threat or treat? *ConneXions*, Volume 9, No. 4, April 1995. http://info.webcrawler.com/mak/projects/robots/threat-or-treat.html

[Marchiori 1997] Massimo Marchiori, The Quest for Correct Information on the Web: Hyper Search Engines. *The Sixth International WWW Conference* (WWW 97). Santa Clara, USA, April 7-11, 1997.

[Page 1998] Lawrence Page, Sergey Brin. The Anatomy of a Search Engine. *Submitted to the Seventh International WWW Conference* (WWW 98). Brisbane, Australia, April 14-18, 1998.

[Page 1998.2] Lawrence Page, Sergey Brin. PageRank, an Eigenvector based Ranking Approach for Hypertext. *Submitted to the 21st Annual ACM/SIGIR International Conference on Research and Development in Information Retrieval.* Melbourne, Australia, August 24 - 28, 1998.

[Pinkerton 1994] Brian Pinkerton, Finding What People Want: Experiences with the WebCrawler. *The Second International WWW Conference* Chicago, USA, October 17-20, 1994.

[Pirolli 1996] Peter Pirolli, James Pitkow, and Ramana Rao. Silk from a Sow's Ear: Extracting Useable Structures from the Web. *Conference on Human Factors in Computing Systems (CHI 96)* Vancouver, Canada, April 13-18, 1996.

[Salton 1989] G. Salton. *Automatic Text Processing*. Addison Wesley, Massachusetts, 1989.

[Yuwono 1995] Budi Yuwono, Savio L. Lam, Jerry H. Ying, Dik L. Lee. A World Wide Web Resource Discovery System. *The Fourth International WWW Conference* Boston, USA, December 11-14, 1995.

9.2 URLs

[1] Alta vista, http://altavista.digital.com/

[2] InfoSeek, http://www.infoseek.com/

[3] Excite, http://www.excite.com/

[4] Lycos, http://www.lycos.com/

[5] NetAttache, Tympani Development Inc., http://www.tympani.com/products/NAPro/NAPro.html

[6] WebSnake, Anawave Software Inc., http://www.anawave.com/websnake/

[7] Robots Exclusion Protocol, on line resource, http://info.webcrawler.com/mak/projects/robots/exclusion.html

[8] Stanford Linear Accelerator web site, http://www.slac.stanford.edu/

[9] Stanford University home page, http://www.stanford.edu/

[10] Stanford Database group web site, http://www-db.stanford.edu