Chapter 6

Content-Based Prediction

6.1 Introduction

The World Wide Web has become the ultimate hypertext interface to billions of individually accessible documents. Unfortunately, for most people, retrieval times for many of those documents are well above the threshold for perception, leading to impressions of the "World Wide Wait." From Chapter 2 we saw that Web caching is one approach that can effectively eliminate the retrieval time for many recently accessed documents, but it cannot help with documents that have never been visited in the past.

If future requests can be correctly predicted, those documents could potentially be pre-loaded into a local cache to provide fast access when requested by a user, even when they have not been accessed recently. This chapter proposes and evaluates such a predictive approach. While conventional predictive techniques look at past actions as a basis to predict the future, we consider an approach based on the hypertext characteristics of the Web.

In Chapter 5, we showed that Web pages are typically linked to pages with related textual content, and more specifically, that the anchor text was a reasonable descriptor for the page to which it pointed. Many applications already take advantage of this Web characteristic (e.g., for indexing terms not on a target page [Goo02, BP98] or to extract high-quality descriptions of a page [Ami98, AP00]). In this work we will examine in detail another application of this result.

Our motivating goal is to accurately predict the next request that an individual user is likely to make on the WWW. Therefore, this chapter examines the value of

using Web page content to make predictions for what will be requested next. Many researchers have considered complex models for history-based predictions for pre-loading (e.g., [PM96, Sar00]), but relatively few have considered using anything other than simplistic approaches to the use of Web page content. Unlike many techniques that do examine content, our approach does not noticeably interfere with the user experience at all — it does not ask for a statement of interest, nor does it modify the pages presented to the user. Instead, it can be used invisibly to improve user-perceived performance.

One naive content-based approach is to pre-load all links on a page. A slightly more intelligent approach is to pre-load as time permits the links of a page, in HTML source order from first to last (corresponding generally to links visible from top to bottom). In this work we compare those approaches with an information retrieval-based one that ranks the list of links using a measure of textual similarity to the set of pages recently accessed by the user. In summary, we find that textual similarity-based predictions outperform the simpler approaches.

6.2 Background

As we described in Chapter 1, by transparently reducing network latencies and bandwidth demands, Web resource caching [Dav01c] has become a significant part of the infrastructure of the Web. Unfortunately, caching can only help when the objects requested are already present in the cache. Typically, caches contain objects that have been accessed in the past. Prefetching [PM96, KLM97], however, can be used speculatively to put content into the cache in advance of an actual request. One difficulty, however, is in knowing what to prefetch. Typical approaches have used Markovian techniques (e.g., [Duc99, Sar00]) on the history of Web page references to recognize patterns of activity. Others prefetch bookmarked pages and often-requested objects (e.g., [Pea02]). Still others prefetch links from the currently requested page [Lie97, CY97, Cac02a, Web02, Ims02, Kle99, IX00, PP97, Dee02].

However, prefetching all of the links of the current page is not a viable option, given that the number of links per page can be quite large, and that a prefetching

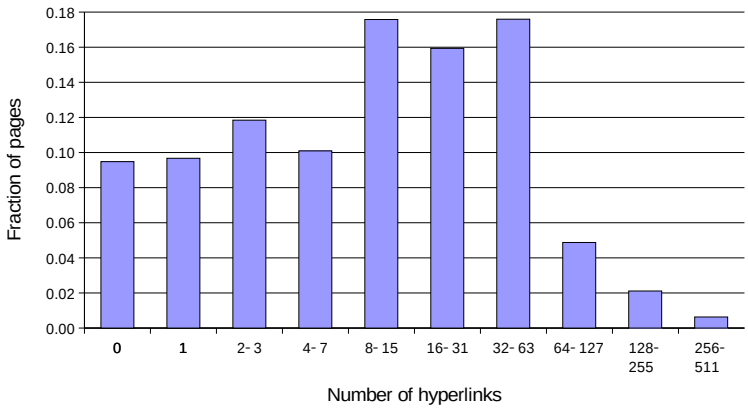


Figure 6.1: The distribution of the number of hypertext (non-embedded resource) links per retrieved HTML page.

system typically has only a limited amount of time to prefetch before the user makes a new selection. While likely heavy-tailed [CP95, CBC95], this "thinking-time" is typically less than one minute [KR01]. Likewise, prefetching content that is never used exacts other costs as additional resources such as bandwidth are consumed. We do not consider the thinking time or prefetching costs further in this chapter (see [Bes96, JK98, AZN99, ZAN99] for various utility theoretic approaches to balancing cost and benefits).

The difficulty of selecting a choice in content-based analysis is illustrated in Figure 6.1. It shows the distribution of the number of unique hypertext (non-embedded resource) links per retrieved HTML page for the dataset used in this chapter. The median number of links per page is 11, while the mean is 26, indicating the presence of some pages with a large number of links. (We describe this dataset further in Section 6.5.1.) Note that this histogram reflects the distribution of pages requested by users (which includes repeated retrievals), versus the more or less static distribution of pages on the Web (such as described by Bray [Bra96]).

The prefetching of content may cause problems, as not all content is cacheable (so prefetching it only wastes resources), and prefetching even cacheable content can abuse server and network resources, worsening the situation [Dav01a, CB98]. An alternative is to do everything but prefetch [CK00] — that is, to resolve the DNS in advance, connect

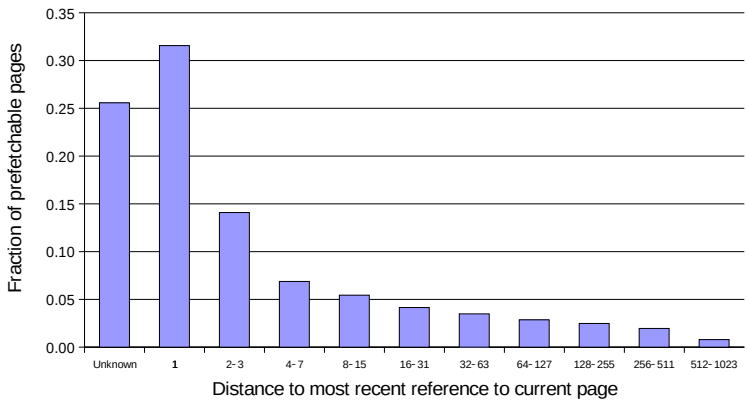


Figure 6.2: The distribution of the minimum distance, in terms of the number of requests back that the current page could have been prefetched.

in advance to the Web server, and even warm up the Web server with a dummy request.

Prefetching systems are difficult to evaluate, especially content-based prefetching ones (although we will consider one approach in Chapters 10 and 11) because even state-of-the-art proxy evaluation techniques (e.g., Polygraph [Rou02] as used in caching competitions [RWW01]) use artificial data sets without actual content. Even when using captured logs as the workload, real content will need to be retrieved, and will no longer be the same as the content seen when the logs were generated.

A more complex analysis is shown in Figure 6.2. It shows (by summing the fractions for columns 1 and 2-3) that approximately 46% of prefetchable pages (those that are considered likely to be cacheable) can be reached by examining the links of the current page and its two immediate predecessors in the current user's request stream. (The bar marked unknown represents those pages that were never linked from any pages seen during our data collection.)

For comparison, the per-client average recurrence rate (that is, the rate at which requests are repeated [Gre93]) for clients making more than 500 prefetchable requests was 49.2%, and for all clients was 22.5% (since some clients made very few requests). This means that if each client utilized an infinite cache that passively recorded cacheable responses, the per client average hit rate on prefetchable content would be less than

25% (corresponding to the average recurrence rate). However, if the clients additionally prefetched all perceived prefetchable links, the hit rates on prefetchable content would increase to almost 75%, because the prefetchable pages that would not be in the cache (i.e., the misses) correspond at most to the pages represented by a distance of unknown in Figure 6.2. Note that these are the results of per-user analysis on `text/html` responses not satisfied by the browser cache. Additionally, a single shared cache (that stores all user requests) would have a recurrence rate of 50.9% for cacheable resources (47.3% for all resources). From this analysis we conclude that there is significant potential for content-based prediction of future Web page requests when caching is considered.

6.3 Related Work

Chan [Cha99] outlines a non-invasive learning approach to construct Web user profiles that incorporates content, linkage, as well as other factors. It uses frequency of visitation, whether bookmarked, time spent on page, and the percentage of child links have been visited to estimate user interest in a page. Predictions of user interest in a page are made by building a classifier with word phrases as features and labeled with the estimation of page interest. Thus, new pages can be classified as potentially being of interest from the examination of their contents. Chan's goal is similar but not the same as ours. He is interested in modeling what the user might like to see, while we are concerned with what the user will do. (See Haffner *et al.* [HRI+00] for a theoretical discussion of the similarities of the two goals.) Unlike the approaches considered here, his method requires that anything that is to be considered for recommendation must already be prefetched (for analysis). Additionally, the methods he proposed were not incremental, nor did they change over time.

Chinen and Yamaguchi [CY97] describe and evaluate the performance of their Wcol prefetching proxy cache. Their system simply prefetches up to the first n embedded images and m linked documents. In a trace with a large number of clients, they found that if all targets were prefetched, a hit rate of 69% was achievable. Prefetching ten targets resulted in approximately 45%, and five targets corresponded to about 20% hit

rate. Here we use a variant of their approach (termed original order) as one of the prediction methods that we will use for evaluation.

More sophisticated link ranking approaches are possible. Klemm's WebCompanion [Kle99] is a prefetching agent that parses pages retrieved, tracks costs to retrieve from each host, and prefetches the pages with the highest estimated round trip time. Average access speedups of over 50% with a network byte overhead (i.e., wasted bandwidth) of 150% are claimed from tests using an automated browser.

Ibrahim and Xu [IX00] describe a neural net approach to prefetching, using clicked links to update weights on anchor text keywords for future predictions. Thus they rank the links of a page by a score computed by an artificial neuron. System performance of approximately 60% page hit ratio is claimed based on an artificial news-reading workload.

A number of others have considered the similar problem of Web page recommendation and Web page similarity. However, they (e.g., [DH99]) tend to use information that can only be acquired by the retrieval of additional pages, which would likely overwhelm the link bandwidth and negate the purpose of intelligent prefetching. Our approach, in contrast, uses only the content that the client has previously requested to base decisions on what to prefetch next.

Recently Yang *et al.* [YSG02] considered various approaches to hypertext classification. Their results are mixed, finding that identification of hypertext regularities and appropriate representations are crucial to categorization performance. They note, however, that "algorithms focusing on automated discovery of the relevant parts of the hypertext neighborhood should have an edge over more naive approaches." While their study examines a related question, unlike our work it treats all links (both incoming and outgoing) equally, and does not consider the use of anchor or surrounding text.

One of the difficulties in comparing work in the area of content-based prediction and prefetching is the inability to share data. Some, like Chinen and Yamaguchi, evaluate their work by using their system on live data. Even if the trace of resources requested were available, such traces deteriorate in value quickly because the references contained within them change or disappear. Others, such as Klemm as well as Ibrahim and Xu

test their systems with relatively contrived methods, which make it difficult to ascertain general applicability. Our initial approach here has been to capture a full-content Web trace so that analysis can be performed offline. However, because of privacy concerns, we (like other researchers) are unable to provide such data to others. In Chapter 10 we will propose a mechanism capable of comparing multiple prefetching proxy caches simultaneously. We will develop a prototype system in Chapter 11, but it is only applicable to fully realized prefetching systems.

6.4 Content-Based Prediction

In many user interfaces, multiple user actions may appear atomic and thus may be inaccessible for analysis. For those environments, prediction methods are often limited to those that can recognize and extrapolate from patterns found in a history log (e.g., UNIX shell commands as we described in Chapter 3, or telecommunication switch alarms [Wei01]). In contrast, the Web provides easy access to the content being provided to the user. Moreover, this content prescribes boundaries within which the user is likely to act. As described in Chapter 5, studies [CP95, TG97, Dav99c] have shown that the user is likely to select a link on the current page between 80 and 90% of the time. Therefore, the set of links shown to the user is a significant guide to what the user is likely to request next.

In addition, the content provides possible next steps that historically-based prediction approaches cannot. A user model based on past experience cannot predict an action previously never taken. Thus, a content-based approach can be a source of predictions for new situations with which history cannot help.

While a user does provide some thinking time between requests in which prefetching may occur, it is limited, and so there may be insufficient time to prefetch all links. Moreover, even if there were sufficient time, the user and network provider may find the extra bandwidth usage to be undesirable. Thus, the approach is typically not to predict all links, but instead to assign some weight to each link and use only the highest scoring links for prediction (and subsequent prefetching).

A prefetching system, whether integrated into a browser, or operating at a proxy, has access to the content of the pages served to the user. Given a particular page, the external links that it contains can be extracted. The anchor and surrounding text can be used as a description of the page (as shown in [Dav00c] and used in [Ami98, AP00]). Given a set of links and their descriptions, the problem is then how to rank them.

Ideally, we would have a model of a user's interest and the current context in which the user is found (e.g., as in [BJ01]). One approach is to explicitly ask the user about their current interest, as done in WebWatcher [AFJM95, JFM97] and in AntWorld [KBM+00] or to rate pages (as in Syskill and Webert [PMB96, PB97] and AntWorld). We could also consider modifying the content seen by the user to give hints or suggestions of recommended or preloaded content (as in Letizia [Lie95, Lie97], WebWatcher, QuIC [EBHDC01], the prefetcher implemented by Jiang and Kleinrock [JK97], and common uses of WBI [BMK97]).

However, our preference is to build a user model as unobtrusively as possible (as in [Cha99]) to perform prefetching behind the scenes. For the purposes of this study, this means we cannot ask the user explicitly about his or her interest, nor change the content, but it does enable us to work offline with logs of standard usage. Therefore, we will not consider obtrusive approaches further here.

Instead we will use the textual contents of recently requested pages as a guide to the current interests of the user. This method allows us to model a user's changing interests without mind-reading or explicit questions, and is intended to combine with other sources of predictions to generate an adaptive Web prefetching approach [Dav99a]. How well it performs is the subject of the next section.

6.5 Experimental Method

As mentioned earlier, experimental evaluation of a content-based prefetching system is difficult. Rather than deploying and testing a complete prefetching system, we have elected to test the content-based prediction methods described above in an offline manner. For comparison, one baseline we will use is the accuracy of a random ranking.

6.5.1 Workload collection

We implemented a custom Java-based non-caching HTTP/1.0-compliant proxy to monitor and record the full HTML content and all headers of user requests and responses. We captured approximately 135,000 requests, of which just over a third generated HTML responses. (The rest correspond to embedded resources such as images and non-data responses such as errors or content moved.) Users were predominantly computer science students and staff from Rutgers University. More than fifty distinct clients used the service from August 1998 to March 1999.

While covering close to seven months, this trace is rather small, and predominantly reflects the traffic of university users. However, with concerns over privacy, the choice was made to recruit explicit volunteers instead of surreptitiously recording the activity of all users of an existing cache or packet snooping on a network link (e.g., as in [FCD+99, Fel98]).

This log does not distinguish users — it only distinguishes clients based on IP address, and so if multiple users of this proxy operated browsers on the same machine (a possibility under UNIX) or behind the same proxy, those users' requests could be interleaved. Likewise, users with dynamic IP addresses are seen as distinct users on each session and multiple users (if assigned the same IP address at different times) may seem like the same user returning over multiple sessions. We believe the likelihood of multiple users per IP (whether by proxy, or the re-use of dynamic addresses) happening is small for this data. More importantly, though, the log does not reflect the requests made by users that are satisfied by browser caches. Users of this proxy were directed to use the proxy, and not told to disable browser caching.

The data was recorded in several segments, generally corresponding to separate runs of the proxy. The first segment, representing about 5% of the log, was used for exploratory analysis and was manually examined. As we describe specific adaptations made to the data, it is from the measurement and analysis of this initial segment. Overall results shown in graphs will reflect measurements over the entire dataset. As a result, any effects of fitting the initial data is limited to just 5% of the dataset.

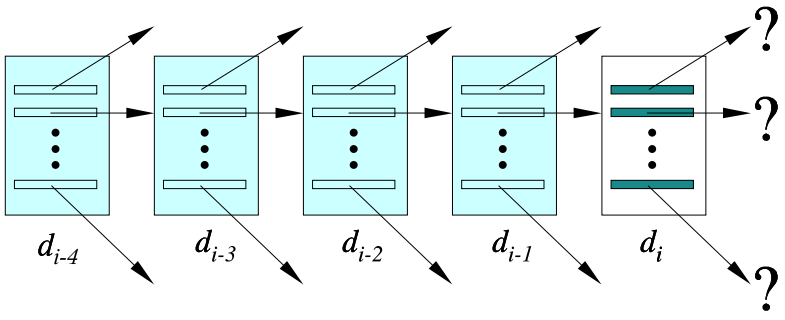


Figure 6.3: Sequence of HTML documents requested. The content of recently requested pages are used as a model of the user's interest to rank the links of the current page to determine what to prefetch.

6.5.2 Data preparation

Our custom proxy attempted to record all HTTP [FGM+99] headers of requests and responses, and the contents of all HTML responses. These were written in chronological order, thus interleaving requests when multiple users were active. Since we are interested in predictions from sequences of requests for individual users, we extracted the HTML records and sorted by client and timestamp. From these sorted access requests, we built sets of up to five HTML responses. (While a larger number of responses might have provided more data, we wanted to model the current user interest which might be diluted with data from further in the past.) Thus, for example, responses d_1, d_2, d_3, d_4, d_5 would be the first set, from which we would attempt to predict d_6. Likewise, d_2, d_3, d_4, d_5, d_6 corresponded to the second set, from which we would attempt to predict d_7. However, occasionally error messages (e.g., HTTP response codes 403, 404), redirections (e.g., 301, 302), and content unchanged (code 304) ended up being included, as Web servers sometimes described these as of type `text/html` as well.

Since we had stored the content, whenever we encountered a 304 content unchanged response for content that we had previously recorded, we used our copy of the earlier content. Requests for URLs generating the remaining 304 responses could be predicted, but not used as content (since that content must have been sent before starting our measurements we couldn't have it) for use in future predictions.

We discarded those responses with HTTP error codes (i.e., response codes ≥ 400), but retained redirections (e.g., response codes 301, 302) since links were made to one

URL, which would then redirect elsewhere. In this case, the original log might have included retrieval events $d_1, d_2, d_3, d_4, d_5, r_6, d_6, d_7$ (where r_6 corresponds to a redirection to d_6) but we would instead generate the two cases d_1, d_2, d_3, d_4, d_5 predicting r_6 and d_2, d_3, d_4, d_5, d_6 predicting d_7. We use r_6 as a target of prediction since any link followed (if present at all) would have been made to r_6, but there is no usable content in r_6 so we don't use it for text analysis. Unfortunately, as mentioned earlier, we only wanted HTML responses, but occasionally we would see redirections for non-HTML content, such as banner advertising that would use one or more redirections. We manually filtered URLs for some of the more popular advertising hosting services to minimize this effect.

The pages were parsed with the Perl HTML::TreeBuilder library. Text extraction from the HTML pages was performed using custom code that down-cased all terms and replaced all punctuation with whitespace so that all terms are made strictly of alphanumerics. Content text of the page includes title but not META tags nor alt text for images. URLs were parsed and extracted using the Perl URI::URL library plus custom code to standardize the URL format (down-casing host, dropping #, etc.) to maximize matching of equivalent URLs. The basic representation of each textual item was bag-of-words with term frequency [SM83].

6.5.3 Prediction methods

We present a total of four methods to rank the URLs of the current page for prediction. Two are simple methods: the baseline *random* ordering, and *original rank ordering* (i.e., first-occurrence order in page). The second two rank each link based on the similarity of the link text with the combined non-HTML text of the preceding pages (as illustrated in Figure 6.3). To measure the similarity between two text documents (D_1, D_2), we use a very simple metric[1]:

$$\text{TF}(w, D_i) = \text{number of times term } w \text{ appears in } D_i$$

[1]Other variations of this similarity metric (e.g., to include factors for document length) were tested on the first data segment but they performed similarly or worse. We make no claims as to the optimality of this similarity metric for this task.

$$\text{Text-Sim}(D_1, D_2) = \sum_{all\ w} \text{TF}(w, D_1) * \text{TF}(w, D_2)$$

The first method (termed *similarity* in our tests) uses the highest similarity score (in the case of multiple instances of the same link to a URL) of the anchor and surrounding text to the preceding pages. The second (termed *cumulative*) is identical, except that it sums the similarity scores when there are multiple links to the same target (giving extra weight to such a URL).

Links recognized include the typical `a href`, but also `area href`, and `frame src`. We did not parse embedded JavaScript to find additional links, although in reviewing the initial data segment we did find JavaScript with such links.

Since links are typically presented within some context, in addition to anchor text we tested the addition of terms before and after the anchor. For example, if the HTML content of a page were:

```
The <a href="http://www.acm.org/acm1/">ACM</a> Conference will
take you beyond cyberspace.
```

then 0 additional terms would provide just {ACM}. If we permit 5 additional terms on each side, then we would get the set {The, ACM, Conference, will, take, you, beyond} to be used for comparison against the text of the preceding pages. Note that text around an anchor may be used by more than one link, in the case that the window around an anchor overlapped that of another anchor. In experiments on the first data segment, we tested the use of up to 20, 10, 5, and 0 additional terms and found that in most cases, when more terms were used, performance was slightly better. Thus we use as many as twenty additional terms before and after the anchor text. However, the text around a URL was never permitted to extend into another URL's anchor text.

Furthermore, we noticed that when exploring pages within a site, certain elements of that site were often repeated on every page. Examples would include a copyright notice with a link to a use license page. In this case, such repeated elements would get undue weight over non-repeated text. As a result, a link containing the repeated text (the link to a license, in the case above) would be ranked highly, even if irrelevant to the true content of the pages. To combat this effect, we tested a variation that no

longer used all text. Instead we used a Perl version of htmldiff [DB96] to compare sequentially requested pages and generate only the text that was present on the second (more recent) page. Thus, we used the first page, plus the textual differences to the second page, plus the differences between pages 2 and 3, etc. In this way we hoped to eliminate significant emphasis on repeated structure in the pages.

We used the well-known Porter [Por97] stemming (since it helped slightly), but did not use stop word elimination (since it performed either slightly worse or no change). The non-HTML text (including title, keywords, and meta description) of the preceding pages was combined to serve as a "document" representing the current user interest against which we would measure similarity. However, since parts of the current page are also represented in the anchor texts pointing to the target documents, we tested the use of the current page in the model for user interest, and found that it also decreased performance on the initial segment slightly, and so we do not include it. Thus in summary, we use the non-repeating text of the preceding four pages as our model of current user interest against which we measure similarity, as illustrated in Figure 6.3.

6.5.4 Evaluation

Given the ranking methods described above, we will evaluate their predictive accuracy (the fraction of times that they include the correct next request in their prediction) over the entire dataset. In most cases, however, we will scale the results to show the fraction predicted correctly out of those possible to get correct (defined below). We will consider variations in the number of predictions by evaluating algorithms that use their best 1, 3, or 5 predictions. Additionally, we will consider the case in which the clients have an infinite cache into which predictions are prefetched. With such a cache, success is measured not by whether the immediate prediction was correct, but whether the requested object is present in the cache (a test arguably closer to real world considerations).

The ultimate goal of this effort is to use the best prediction method to provide suggestions for prefetching. It is generally not possible to prefetch the results of form entry, and is potentially undesirable to prefetch dynamically generated pages, so we

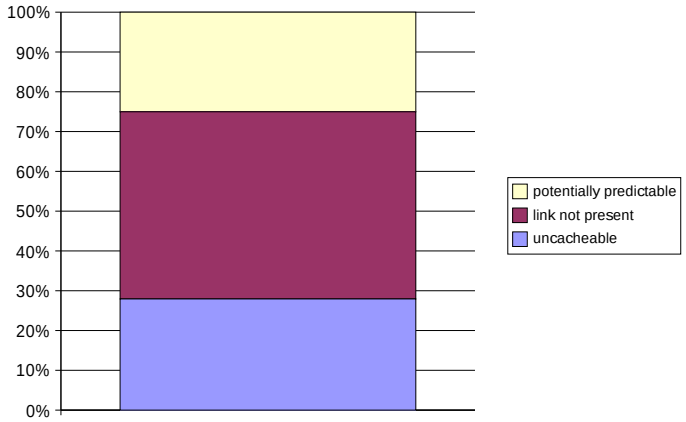


Figure 6.4: Fraction of pages that are uncacheable, found as a link from previously requested page (potentially predictable), or not found as a link from previous page.

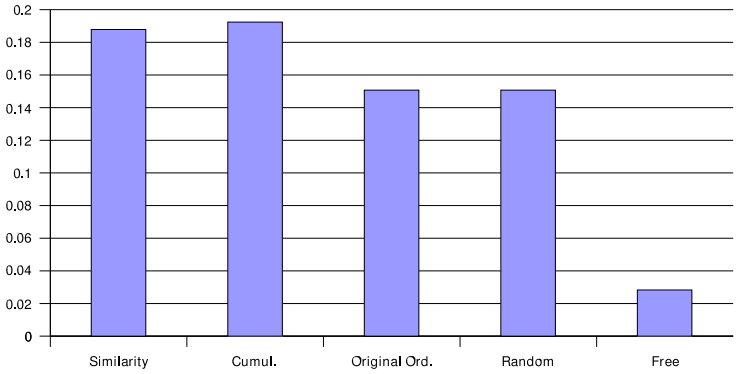
would like to not allow predictions for such pages (i.e., all responses generated by POSTs or with URLs containing "cgi" or "?"). Those responses represent 28% of all HTML collected. Additionally, we can recognize those cases where the next cacheable URL requested was not present in the preceding page, making a correct prediction an impossibility in another 47%. The remaining 25% have the potential to be predicted correctly by choosing from the links of the current page (see Figure 6.4). Approximately 2.8% of those pages (.7% of the total) had only one URL, and so will be predicted correctly under all prediction mechanisms.

Typically a prefetching system has time to prefetch more than one page, and so we consider the case in which we can select the top three and five most-likely URLs and mark the set as "correct" if one of them is the requested page. Under this arrangement, the set of potentially predictable pages stays the same (25%), but the number of default or "free" wins grows since there will be more cases in which the number of distinct links available is within the size of the predicted set.

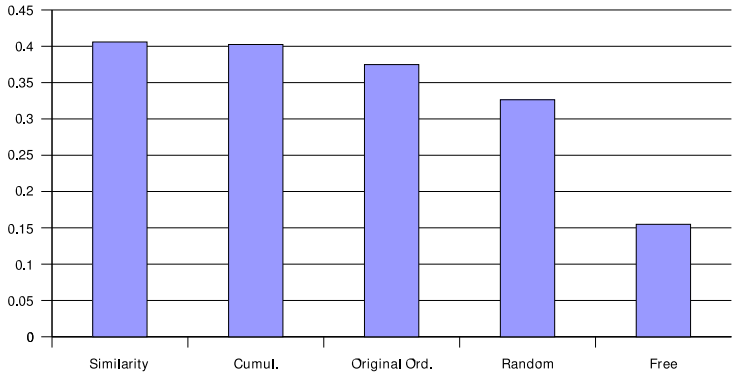
6.6 Experimental Results

6.6.1 Overall results

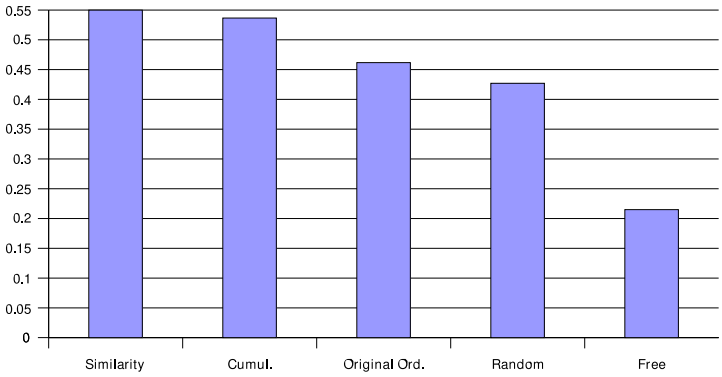
Figure 6.5 shows the overall accuracy for each of the content prediction algorithms, as a fraction of predictions that are possible to get correct from the last page of links.



(a) Top one prediction



(b) Top three predictions



(c) Top five predictions

Figure 6.5: Overall predictive accuracy (within potentially predictable set) of content prediction methods when considering links from the most recently requested page.

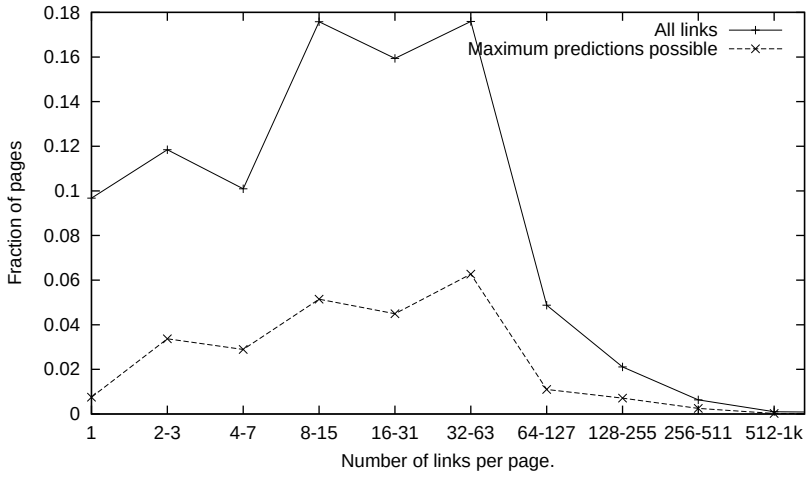


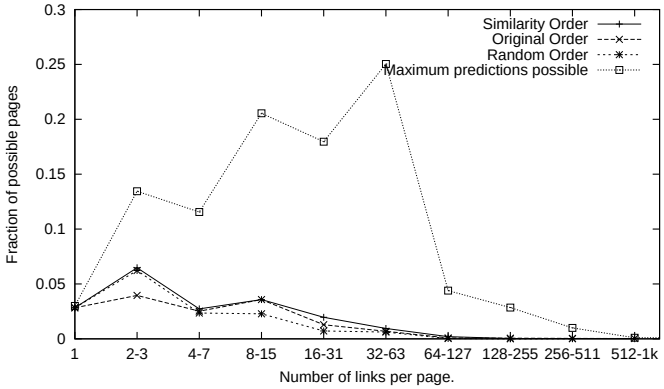
Figure 6.6: Potential for prediction when considering links from the most recently requested page.

It depicts performance for the cases in which only the highest-scoring prediction is evaluated as well as when the top-3- and top-5-highest-scoring predictions are used (note the different y-axis scales). In addition to being included within the performance for each algorithm, the "free" cases are also plotted to show a lower bound (recall that the free cases are pages in which the number of links in the page is no larger than the number of predictions permitted, thus automatically correct).

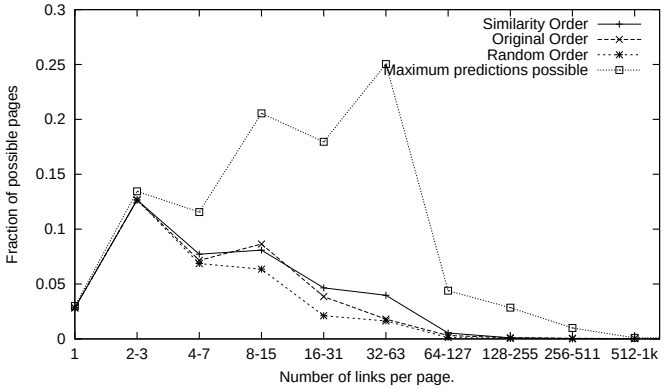
It can be seen that the similarity and cumulative ranking methods outperform the original rank ordering, and all three outperform the baseline random order. This suggests that the similarity-based approaches are beneficial for this link-ranking task. It also suggests that either by custom or by design, users tend to select links earlier in the page. The cumulative approach edges out the similarity approach for top-one, but similarity is best for top-three and top-five. This is likely because top-three and top-five are more likely to include entries that might appear more than once (thus benefiting from the cumulative emphasis).

6.6.2 Distributions of performance

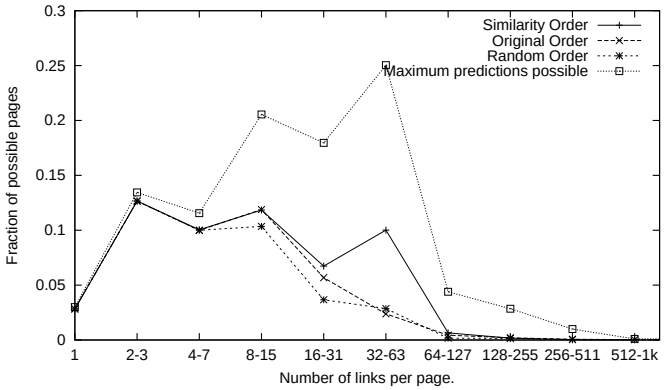
This chapter examines the predictability of Web page requests based on the links of the current page. In this section we consider the distribution of successful predictions with



(a) Top one prediction



(b) Top three predictions



(c) Top five predictions

Figure 6.7: Performance distribution for content prediction algorithms when allowing guesses from the most recently requested page.

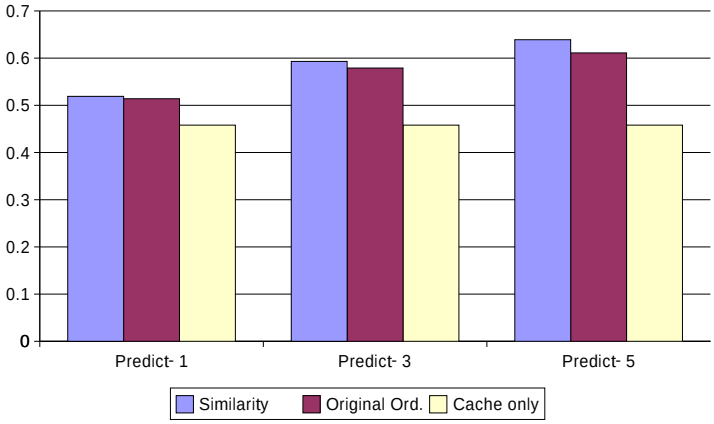


Figure 6.8: Predictive accuracy (as a fraction of all pages) of content prediction methods with an infinite cache compared to no prediction.

respect to the number of links on the current page. Figure 6.6 compares the overall distribution of the number of links per page with the distribution of cases in which the predicted page is found in the links of the last page. This figure demonstrates the limits of strict prediction from the links of the current page — there are many requests that do not appear within that set.

Focusing on those links that do appear within the current page, Figure 6.7 shows the distribution of the number of correct predictions per page size for each method plus the distribution of cases that can possibly be predicted correctly. Note that for clarity we have omitted the cumulative approach from these and further graphs, as its performance is closely tied with that of the similarity method. For most points of Figure 6.7(a), in which a single prediction is permitted, the performances of the various approaches are almost indistinguishable, but with more allowed predictions, more variation becomes visible.

6.6.3 Results with infinite cache

Since the system prefetching Web pages would be putting them into a cache, we consider briefly here the performance of prediction approaches in such a context. Figure 6.8 shows a summary of the predictive accuracy of two ranking methods assuming the use of an infinite cache, and compares them to an infinite cache without prefetching. While

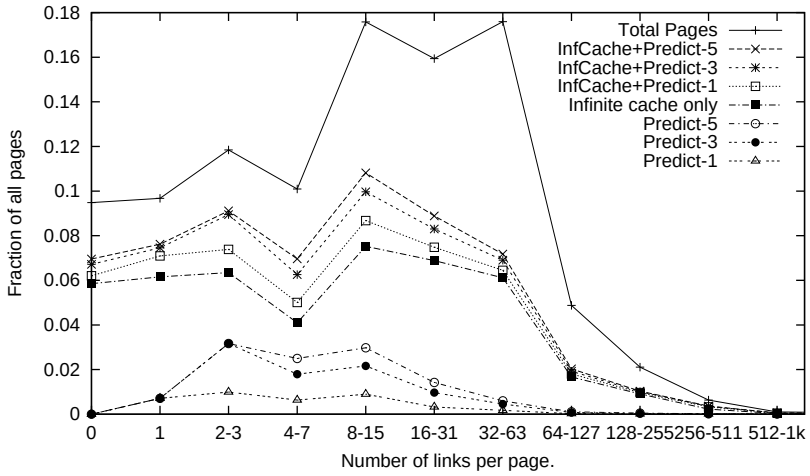


Figure 6.9: Performance distribution for original order with and without an infinite cache.

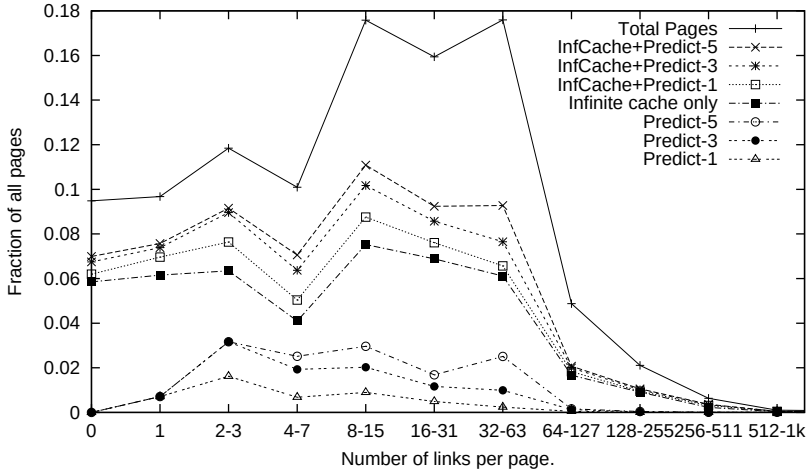


Figure 6.10: Performance distribution for similarity ranking with and without an infinite cache.

caching under these circumstances is obviously a significant contributor to performance, the predictive systems are able to improve on the caching-only performance, particularly when more than one prediction is permitted.

Figures 6.9 and 6.10 provide performance distributions of the original rank orderings and similarity orderings respectively. In these figures, note that the top curve represents the total page distribution, since we are no longer limited to the pages potentially

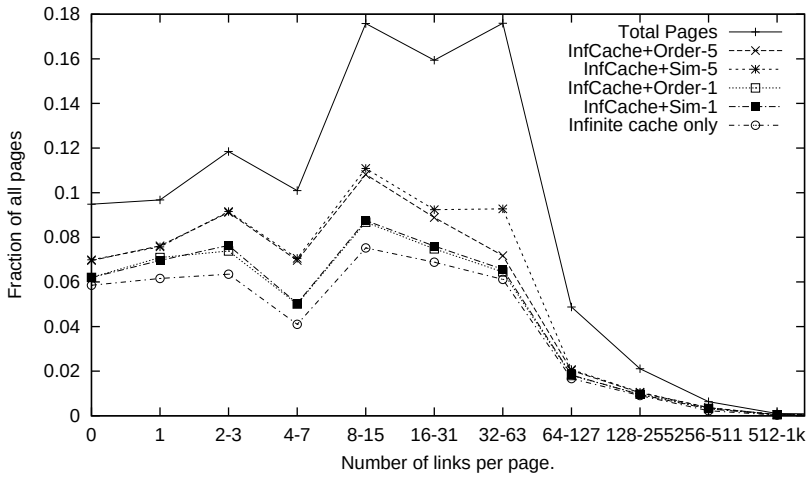


Figure 6.11: Comparison of performance distributions for original order and similarity with an infinite cache.

reached from the last ones retrieved. The original (non-cached) performance is shown (at the bottom), while the performance using an infinite cache is much higher. However, the cache does not do all the work, as the performance of an infinite cache alone is visibly below the cache+prefetching lines.

Figure 6.11 compares the cached performance of both the original rank ordering and similarity based ordering (we omit the top-three curves for clarity). The performance of cached original ranking and similarity ranking are nearly identical in most points; only for one point (at 32-63 links per page) does similarity rise much above the original ordering (when five predictions are allowed).

6.7 Discussion

Recall that the non-cached results presented are in terms of potentially predictable pages. This is, in fact, a relatively small fraction of all requests generated. For example, the best predictive method was shown to be the similarity ranking when predictions of the top 5 links are permitted, achieving an accuracy of approximately 55% (a relative improvement of close to 30% over random link selection). However, this represents success in only 14% of all cases, since only a quarter of all pages are possible to predict

in this manner. In the case of an infinite cache we are able to provide hits for 64% of all requests (a 40% improvement over a system without prefetching). Actual performance is likely to lie in between, as caches are finite and prefetched content can expire before it is needed.

Perhaps more importantly, this analysis is of a relatively small trace, and may not be representative of users in non-academic environments. However, more extensive analysis would require capturing content from a larger, more general population (such as that of an ISP), and is likely to raise significant privacy concerns.

The prefetching examined here is only for HTML resources. However, HTML resources represent only a fraction of all requests made by users. Most are embedded resources (such as images, sounds, or Java applets or the result of JavaScript that are automatically retrieved by the browser). We made the choice to ignore such resources as most of them are easy to predict by examining the HTML of the page in which they are embedded. This choice has a drawback, though, as users do indeed make requests for non-HTML resources, such as PDF and PostScript files (which can often be quite large), plain text resources, and the downloading of programs, albeit much less frequently.

In practice, a prefetching system will have time to fetch a variable number of resources. We have examined only three points on the range — allowing one, three, and five predictions. We believe that the typical number will fall within this range, however, as it will likely be useful to prefetch the embedded resources of prefetched pages, and some resources and pages will be cached from previous retrievals.

While extensive, these tests are not comprehensive — we have not attempted to disprove the possibility of other algorithms (text-based or otherwise) outperforming those presented here. In particular, we believe that systems with a stronger model of the user's interest (e.g., AntWorld or WebWatcher) could provide for better prediction, but alternately may lose when the user's interest shifts (as is often the case when surfing the Web). For comparison, recall that WebWatcher [AFJM95, JFM97] compares the user's stated goal, the given page, and each link within it to other pages requested by

previous users, their stated goals, and the links they selected. Using assessed similarity on this information in a restricted domain, it picked three most likely links, from which the selected link was present 48.9% of the time. In a second, smaller trial, WebWatcher scored just 42.9% (as compared to humans at 47.5% on the same task). Our most comparable experiments have shown a lower accuracy (approximately 40%), but additionally they have been used on a general task in an unlimited domain, with no user goals nor past users for guidance. Thus, these experiments provide evidence of the applicability of content-based methods for predicting future Web pages access.

6.8 Summary

This chapter has examined the performance of Web request prediction on the basis of the content of the recently requested Web pages. We have compared text-similarity-based ranking methods to simple original link ordering and a baseline random ordering and found that similarity-based rankings performed 29% better than random link selection for prediction, and 40% better than no prefetching in a system with an infinite cache. In general, textual similarity-based rankings outperformed the simpler methods examined in terms of accuracy, but in the context of an infinite cache, the predictive performance of similarity and original rankings are fairly similar.

Most proposed Web prefetching techniques make predictions based on historical data (either from an individual's past or from the activity of many users as hints from servers or proxies). History-based prefetching systems can do well, when they have a model for the pages that a user is visiting. However, since users often request unseen (e.g., new) resources (perhaps 40% of the time [TG97]), history may not always be able to provide suggestions, or may have insufficient evidence for a strong prediction. In such cases, content-based approaches, such as those presented here, are ideal alternatives, since they can make predictions of actions that have never been taken by the user and potentially make predictions that are more likely to reflect current user interests.