A Cooperative Approach to Web Crawler URL Ordering

A. Chandramouli[1], S. Gauch[2], and J. Eno[2]

[1] Department of Computer Science, University of Kansas, Lawrence, KS
aravindc@ku.edu
[2] Department of Computer Science, University of Arkansas, Fayetteville, AR
{sgauch,jeno}@uark.edu

Abstract. Uniform Resource Locator (URL) ordering algorithms are used by Web crawlers to determine the order in which to download pages from the Web. The current approaches for URL ordering based on link structure are expensive and/or miss many good pages, particularly in social network environments. In this paper, we present a novel URL ordering system that relies on a cooperative approach between crawlers and web servers based on file system and Web log information. In particular, we develop algorithms based on file timestamps and Web log internal and external counts. By using this change and popularity information for URL ordering, we are able to retrieve high quality pages earlier in the crawl while avoiding requests for pages that are unchanged or no longer available. We perform our experiments on two data sets using the Web logs from university and CiteSeer websites. On these data sets, we achieve a statistically significant improvement in the ordering of the high quality pages (as indicated by Google's PageRank) of 57.2% and 65.7% over that of a breadth-first search crawl while increasing the number of unique pages gathered by skipping unchanged or deleted pages.

1 Introduction

Search engines use crawlers to collect Web pages from Web servers distributed across the Internet. Crawlers are programs that automatically collect Web pages by starting with a Uniform Resource Locator, URL, downloading the Web page at that location, and recursively retrieving all the pages pointed to by the hyperlinks on the page. In contrast to traditional crawlers based on link structure, recent efforts [Brandman et al. 2000] have focused on providing support for crawlers by exploiting Web server information like Web logs and file system to provide the list of URLs on the website. Along similar lines, Google has developed site maps (http://www.sitemaps.org) to allow web sites to provide hints to the Google crawler.

In contrast to providing the list of all URLs on a website, we proposed a cooperative architecture that uses Web log and file system timestamps to provide a

Z.S. Hippe et al. (Eds.): Human - Computer Systems Interaction, AISC 98, Part I, pp. 343–357.
springerlink.com

ranked list of new, modified, and deleted pages since the last visit from a crawler. In our Web services-based co-operative approach, the individual websites make use of their own Web logs and file systems to gather this information. This information can be used to increase the number of pages collected compared to traditional crawling and achieve significant bandwidth savings for the same set of pages collected over multiple crawls. Additionally, hidden web content accessible only through forms can be added to the index based on the URL encoded arguments from a GET form request.

Even with these improvements, the size of the Web combined with the necessarily limited resources available to a crawler and the limited bandwidth on the websites, crawlers do not collect all pages from the Web. As the size of the Web keeps increasing, crawlers typically try to download the "important" pages for indexing by search engines. To determine the important pages, crawlers make use of URL ordering algorithms. The connectivity-based document quality ordering [Cho et al. 1998] and breadth-first search crawling [Nojork and Wiener 2001] are two such well-known URL ordering algorithms. However, both techniques have drawbacks. A connectivity-based metric penalizes new pages and is expensive to compute. On the other hand, breadth-first ordering is relatively inexpensive to implement, but this technique misses good pages deeper in the hierarchy of the site.

Social web content presents challenges for either connectivity-based or breadth-first URL ordering strategies. Social media tends to be fast-moving and may be out of date by the time it develops the robust link structure that would bring it to prominence in a connectivity-based algorithms. Another problem is that many of the links of interest will be shared on private social network profiles, limiting the ability to use either connectivity or breadth-first crawls to identify resources.

As part of the co-operative file system approach, we have developed an URL ordering algorithm based on popularity information extracted from Web logs. This algorithm is inexpensive to compute because it distributes the URL ordering calculation overhead among the participating websites, and identifies pages that users of the website find important enough to view. Because it does not rely on identifying external links to resources to discover content pages, it can identify pages before they are widely linked in the broader Web as well as within social networks. In this paper, we present both timestamp-based and access count algorithms, discuss the advantages and drawbacks of these approaches, and empirically compare them with the computationally similar breadth-first search crawl using Google's PageRank as the metric to indicate each page's importance.

2 Related Works

Researchers have investigated several ways to provide improved support for search engine crawlers. Most of these focus on exploiting Web server information and processing power. [Brandman et al. 2000] suggest the creation of a file on the

Web server that provides a list of all the URLs and their meta-data. In their approach, the crawler would download this file to identify modified pages. They make use of the file system to create their meta-data file. Although the Web server is a partner in the crawling process by providing digested information, the bulk of the processing to detect new, modified, and deleted pages is still left to the crawler. In contrast to the pull strategy employed by crawlers today, [Gupta and Campbell 2001] describe an algorithm that would push updates on popular pages from Web servers to search engines. [Castillo 2004] discusses both pull and push architectures to support cooperation between a Web server and a crawler. They also implemented a cooperation scheme that created an XML file storing update information based on the file system, and demonstrated that a crawler making use of this information would experience 40% bandwidth savings when compared to traditional crawling. [Buzzi 2003] describe a similar approach to [Castillo 2004] and propose the creation of a text file that has information about Web pages such as the last update time, file size, local request frequency, and local update frequency. The paper discussed the type of information that should be provided, but they do not discuss how this information would be gathered by the Web site, nor how it would be used by the crawler. More recently, Google introduced Google sitemaps, which is essentially an extension of the approach proposed in [Brandman et al. 2000]. Webmasters can install a program on their website that creates a text or XML file containing the URLs on the website, called a sitemap. Google sitemaps make use of both file system and Web logs to create the list of URLs.

The earliest work on URL ordering algorithms was by [Cho et al. 1998]. They used connectivity-based metrics to identify the "important" pages to download, and their experiments showed that using the PageRank metric downloaded important pages earlier than the other algorithms. [Najork and Wiener 2001] extended the work of [Cho et al. 1998] and demonstrated that it was possible to discover the important pages early in the crawl by using a breadth-first ordering. However, they do not compare breadth-first crawl with other techniques. More recent work on URL ordering by [Castillo et al. 2004] compared different ordering techniques, namely Optimal, Depth, Length, Batch, and Partial, for long term and short term scheduling for crawling on the Web.

As discussed above, the breadth-first search and the PageRank are the two of the most popular URL ordering techniques reported in literature. However, using PageRank to compute the URL ordering can be very expensive. In fact, [Najork and Weiner 2001] observe that performing the PageRank computations for all the Web pages in real time is not feasible. [Cho and Schonfeld 2007] demonstrate a more efficient method of computing a partial PageRank by relying on a vector of trusted pages to compute a lower bound on the true rank of queued pages. For a crawl of 80 million pages, the technique took only three times as long as a breadth-first ordering. However, new pages and pages outside the trusted set and pages without PageRank are still penalized by this metric. Recent work has

focused on modifying PageRank to use last-modified date or more complex web graph modeling to improve performance for new pages [Cho et al. 2005], but these algorithms are still computationally expensive. On the other hand, a major drawback with breadth-first ordering is that important pages deeper in the hierarchy of the websites will not be collected.

As an alternative to the above two techniques, we propose a URL ordering of pages on individual websites calculated using popularity information extracted from web logs. A major advantage of such an algorithm is that it is relatively inexpensive to compute when compared to PageRank and, since the ordered list of URLs is produced by the individual websites, the workload for the search engines is reduced. Because the websites can process their own file systems and Web logs efficiently, and the results of this effort can be shared with multiple search engine crawlers, the burden on the individual websites is acceptable. This upfront work also decreases the amount of effort the websites must spend serving pages to crawlers.

Another factor that may affect the optimal ordering of URLs is the probability that a new page will improve the existing index. [Pandey and Olston 2008] developed a model that uses sample queries and the existing index state, combined with content clues such as URL text and anchor text words to order URLs for crawling. Pages that have a high likelihood of being highly ranked and relevant to a topic that is sparsely populated in the index are given greater priority in the crawl order. Although this algorithm helps to alleviate some of the problems of other connectivity-based ordering algorithms, such as focusing too much on popular topics to the detriment of niche topics, it still relies on existing links. In some ways, the existing link problem is exacerbated, since the existing links are used for both reference counts and topic discovery. Such an algorithm might struggle with the common practice of using a link shortening service such as tinyurl or bit.ly in social network links.

3 Approach and Implementation

The goal of any URL ordering algorithm is to produce an ordering of URLs so that the Web crawler can collect the most important pages first. Our approach improves this ordering by providing a web service that generates an XML document including only new or modified URLs and ranking them based on popularity as measured by access counts. During a crawl, a cooperative crawler will request a list of all new, updated, or deleted URLs since the last request. The server will generate an XML document with an entry for each qualifying URL, marking the URL as new, modified, or deleted. URLs that have not been modified will not need to be requested by the crawler, so they are not included in the XML response.

For the popularity-based URL ordering algorithms, we exploit the popularity information present in the Web logs on a website and look at a variety of ways to produce this URL ordering. We classify these approaches broadly as non-learning

algorithms that use a predetermined ordering function and learning algorithms that order URLs adaptively based on a training set of URLs with quality information.

3.1 URL List Generation

The list of URLs for the web service may be generated based on the file system, Web logs, or a combination of both. The file system approach has the advantage of providing a comprehensive list of all accessible documents along with accurate modification times. However, it will fail to discover or update dynamic pages, multiple pages specified by URL arguments, or content that may have been modified in a database or content management system (CMS) without modifying the base page. It also cannot flag deleted URLs, since they no longer exist on the file system.

The Web log approach can mitigate these omissions in three ways. First, it can recognize some dynamic pages based on URL-encoded arguments to the web server. Second, by examining the number of bytes returned by a request, it can recognize when the size of a page has changed, indicating a change even when the file system timestamp is unchanged. Finally, it can discover deleted pages based on 404 (Not Found) errors in the Web log. However, it can only recognize pages that have been accessed within the time covered by the log file. In practice, a hybrid approach that gathers data from the file system and Web logs while maintaining a history of file system and Web log activity provides a means to get the most accurate information from all possible sources without relying on long log histories.

Fig. 1 shows the architecture for our system. Periodically, the Web Log Harvester harvests the Web logs for processing. Currently, the Web server used archives its Web logs weekly, so the Web Log Harvester gathers data weekly using the Web log file name provided in a configuration file. Similarly, the File System Harvester uses a text file that contains the list of directory paths on the website and the corresponding base URL for that directory. Every week, the File System Harvester recursively retrieves the filenames. The harvesters pass their information along to the Data Parsing Module. From the Web logs, the module extracts: IP address, access time, URL, number of bytes, and status code. From the file system, the module extracts: path, filename, date of last modification and maps the filenames to its corresponding URLs. The information directly extracted is stored in a URL Database that has the following entries: URL, created date, modified date, deleted date, byte count, and the source. The first time the harvesting is performed the modified date and the created date are set to be the same, while the deleted date is empty for all the URLs collected. However, during the subsequent weeks, the Data Parsing Module uses the data gathered by the harvesters and the information present in the database to infer the modified date and the deleted date using the techniques discussed in section 3.

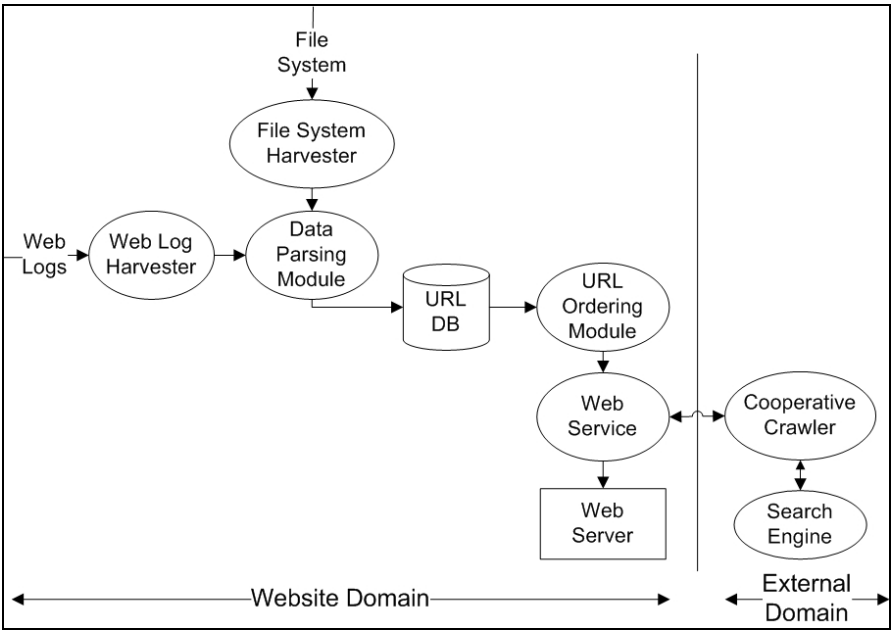

Fig. 1 System Architecture

The information stored in the database is shared with enhanced crawlers via a Web service implemented using the REST protocol. The crawler queries the Web service for information about the Web site contents. The crawler can query the Web service to find URLs that match based on the following criteria:

FileType - Text, Video, Audio, All types.
FromDate – This is the date from which it requires information.
ChangeType – Modified, Deleted, Created, All types.
Changes/Url – Give the changes to the file only or the URL of the entire file.

The ToDate is implicitly set to the current date. Currently, our system only gives the URL for the Web page, but, in future, we will explore a mechanism to provide only the changes made to a page. The crawler then parses the supplied XML file in order to identify the list of URLs that need to be collected. Finally, the Web pages are collected and passed along to the search engine for indexing.

During the experiments, the list was generated using three algorithms: file system, Web log, and file system hybrid. The file system and Web log methods used data exclusively from the file system or Web logs, respectively. However, the Web log data tended to be less reliable both because users sometimes requested pages that no longer existed and some pages were not requested at all, making them unavailable to the Web log-based system. The file system hybrid approach started with the file system list, then supplemented it with Web log data to discover hidden web content. This led to a larger list that still had high reliability.

3.2 Non-learning Algorithms for URL Ordering

The Web logs on the website register the access made to every Web page on the site. Thus, from the Web logs, the total access count for each Web page can be calculated. Then, the Web pages are sorted based on their Total Access Count (TAC). However, pages with high total access count need not necessarily indicate highly popular pages. For example, a Web page might be accessed often by its owner thus inflating the access count value. It may be possible to more accurately identify important pages by incorporating information about the number of different IP addresses from which a page is accessed.

In addition to unique IP address metrics, the hierarchical nature of IP addresses enables us to differentiate between internal and external page accesses. Hence, in order to explore a variety of URL ordering algorithms, we extract four different types of access information from the Web logs, namely, the Total External Count (TEC), the Unique External Count (UEC), the Total Internal Count (TIC), and the Unique Internal Count (UIC) where the external count refers to the requests made to a URL on the website from outside the local network and the internal count refers to the local requests made to a URL. Different URL orderings are then produced by ordering the Web pages based on different combinations of these factors. One limitation to this approach is its inability to differentiate multiple accesses originating from behind a single proxy server. However, obtaining session-level data would require more knowledge than can be derived from a typical Web access log an is beyond the scope of this paper.

The non-learning algorithms discussed so far order the URLs based on four different factors. However, they do not take into account the relative importance of each factor. Are all the parameters equally important or should they be weighed differently? To address these issues, a simple approach would be to calculate the accuracy of the different parameters to predict high quality pages and then use these accuracy values to assign different weights for the parameters. This approach, the Weighted Access Count (WAC) algorithm, has the advantage that the parameters are weighed differently based on their ability to predict high quality pages. The weighted score for each URL is calculated as shown in Equation 1 and an URL ordering is produced by sorting the URLs based on this weighted score.

$$WS = \alpha * \frac{TECacc}{Totalacc} + \beta * \frac{UECacc}{Totalacc} + \gamma * \frac{TICacc}{Totalacc} + \delta * \frac{UICacc}{Totalacc} \quad (1)$$

where
WS = Weighted Score
TECacc = TEC algorithm accuracy
UECacc = UEC algorithm accuracy
TICacc = TIC algorithm accuracy
UICacc = UIC algorithm accuracy
Totalacc = TECacc + UECacc + TICacc + UICacc
and α, β, γ and δ = raw external, unique external, internal, and unique internal counts for the URL.

3.3 Learning Algorithms for URL Ordering

The non-learning algorithms either make use of four different factors or a combination of these factors. In order to learn the best combination of factors, and to develop an adaptive algorithm that would work on any website, we implemented two learning algorithms, Total Access Count-Learning (TAC-L) and Split Access Count-Learning (SAC-L). The TAC-L algorithm takes the total access count for each URL as the attribute while the SAC-L algorithm takes the four different parameters discussed in Section 3.2 as the attributes for the data and they predict the PageRank categories for new URLs based on their attribute(s). Both algorithms have a training and a testing phase. In the training phase, a set of URLs with their access counts and quality information are given as input to a learning algorithm like decision trees or k-Nearest Neighbor algorithm and a model is learned. The quality information is determined using PageRank. PageRank has been used for URL ordering algorithms [Cho et al. 1998] to measure the quality of a page, that is, higher the PageRank, higher the quality of the page. In addition, by relying on the global Google PageRank value as an indicator of the 'true' importance of a page, we are able to verify our results against a much broader metric even though we only use local information to compute our ranking.

Although true PageRank values are floating-point values that provide a total ordering of URLS, we are somewhat restricted in our access to Google's PageRank values. In order to determine the true PageRank for a URL, we use the freeware Parameter tool that determines the PageRank of a URL on a 1 to 10 scale. Since learning algorithms typically predict a category, we make use of the integer PageRank values as the categories for classification. During the testing phase, a set of URLs with their access counts are given as input and the learned model is then used to place each URL in the best matching category. The confidence factor for these assignments are used to rank order the URLs within each category, producing total ordering of the URLs.

4 Evaluation Method

In this section, we describe our experimental evaluation method used to compare popularity-based URL ordering algorithms to a breadth-first search crawl, using a PageRank ordering as a benchmark.

4.1 Data Collection

We make use of two data sets for our experiments. Data set 1 (DS1) contains the Web logs of the ITTC website over 5 weeks (http://www.ittc.ku.edu), to which we had access. Data set 2 (DS2) contains the Web logs of the CiteSeer website (http://citeseer.ist.psu.edu/) whose Web logs for a five week period was shared with us. For DS1, using the home page as a start page, we produced an URL

ordering based on the breadth-first search crawl. In contrast, since CiteSeer does not have an exposed tree hierarchy, a breadth-first crawl is essentially a random crawl. Hence, the random crawl is used as a baseline for DS2. For our popularity-based URL ordering algorithms, access count information from Web logs covering a five week period were extracted. Some of the URLs collected using a breadth-first search crawl for DS1 did not have popularity information (i.e., were not accessed during this five week period) and, similarly, breadth-first search crawl was unable to collect the hidden Web pages that were accessed but were not linked explicitly on the site. Thus, our experiments used a total of 5,480 URLs for DS1 and 102,360 URLs for DS2 that could be collected by the breadth-first search/random crawl for which we also had access information from the Web logs. It is useful to note that one of the benefits of the proposed approach is the ability to find pages that are accessible by means other than links, which is not possible with a link-graph or breadth-first approach.

4.2 Metrics

As discussed briefly in Section 3.2, PageRank has been used in literature to measure the quality of a page. PageRanks for a page are calculated recursively based on the link structure on the Web, with pages linked from many highly ranked pages receiving the highest scores. In order to enable categorization, PageRanks are assigned from a scale of 0-10, with 10 being the most important page. To find the PageRank, as outlined in Section 3.3, we make use of a freeware Parameter version 1.2.

One problem with using PageRank categories as an evaluation metric is that a URL ordering algorithm produces a total ordering on the list of URLs whereas a discretized PageRank does not. That is, although URLs with different PageRank categories can be ordered, URLs with the same PageRank is essentially an unordered set. Hence, for our evaluation, the pages with higher PageRank should be ranked higher than pages with lower PageRank but the order among the URLs with the same PageRank does not matter.

$$Accuracy = \frac{\sum_{i=1}^{n} Match_i}{n} * 100 \tag{2}$$

Equation 2 is the evaluation metric we use, where
n = total number or URLs,
$Match_i$ = 1 if $PPR_i = APR_i$, and 0 otherwise,
PPR_i = PageRank category of URL i produced by the ordering algorithm,
APR_i = Actual PageRank category for i.

In order to illustrate this metric, consider the following URLs (denoted A-E) with their associated PageRank categories: A-6, B-6, C-5, D-5 and E-5. Since PageRank categories do not distinguish among elements in the two sets {A, B} and {C, D, E}, A and B can be in any of the first 2 slots while C, D and E can be in any of

the next 3 slots to produce an accuracy of 100%. Hence, a rank order of ABCDE will be given an accuracy of 100% while a rank order of ACDBE will be given an accuracy of 60% since B should be in one of the first 2 slots while C should be in any of the final 3 slots.

5 Results

Results are presented for both URL list generation and URL ordering systems. The URL lists are evaluated in terms of request and bandwidth savings over time, while the URL orderings are compared with PageRank rankings to determine ordering accuracy.

5.1 Evaluating the Effectiveness of URL List Generation

The list of URLs may be effective in two ways. First, it can identify additional pages that are part of the hidden web or have not been linked yet by other pages. Second, it can reduce the amount of bandwidth used to gather unmodified content. Fig. 2 shows the effect of the URL list generator in terms of additional pages collected. The results show that while the number of pages available through links was fairly constant and occasionally dropped from week to week, the file system, Web log, and file system hybrid approaches gathered significantly more pages, and increased the size of the collection from week to week. Because the pure Web log approach could only discover pages that had already been requested, it had lower performance than the approaches which included file system information. However, the combined approach was able to discover hidden web pages that were not visible to a pure file system approach.

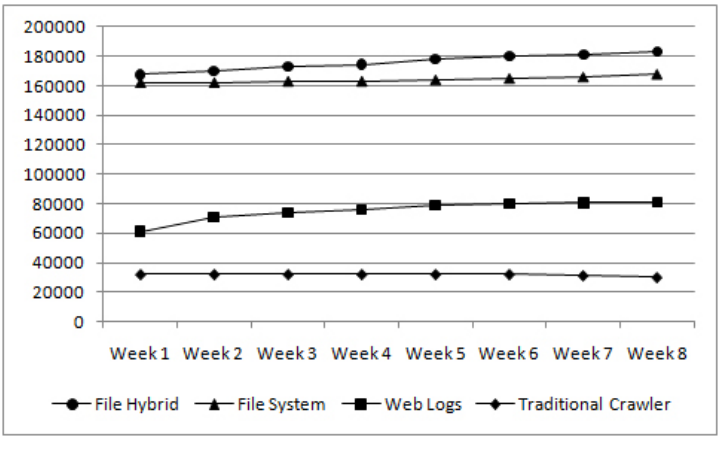

Fig. 2 Number of URLs Collected Over Time

In terms of bandwidth savings, the set of pages that were collected by the traditional crawl were largely static throughout the eight week experiment. As a result, after the first week, the cooperative approach required less than 0.2% as much bandwidth to collect the modified pages from the set of pages gathered by the traditional crawler. Because the hybrid approach discovered more pages, it gathered 159 MB compared to the 13.5 MB collected by the traditional crawler. In the process, it collected six times as many pages overall.

5.2 Evaluating the Accuracy of URL Ordering Algorithms

We evaluate the performance of our URL ordering algorithms by using a five-fold cross validation on DS1. First, the 5,480 URLs were randomly divided into 5 sets. Next, the training and testing was carried out 5 times, each time using 4 sets for training and the remaining set for testing. The numbers reported in this section are the averages obtained over the 5 trials.

We first establish the baseline with the breadth-first search crawl using the testing sets. We found that the breadth-first crawl URL ordering had a 38.8% match with the PageRank category ordering (as calculated using formula 2). Also, a random ordering of the URLs produced an accuracy of 32.9% using the same metric. Next, we used the same test sets for the popularity-based URL ordering algorithms described in Section 3. The results in Fig. 3 show that, even after 5 weeks, the total internal count (TIC) (28.7%) algorithm performs poorly when compared to the baseline. The unique internal count had similarly poor results of 28.5% accuracy. However, the total external count (TEC) (45.7%) performs better. Similarly, the total access count (TAC) algorithm also perform better than the baseline, producing an accuracy of 44.7%. The weighted access count did not improve the TAC algorithm, and neither outperformed the exclusively external count algorithm.

Fig. 3 also shows the accuracy values obtained using the total access count learning algorithms (TAC-L) and the split access count learning algorithms (SAC-L). As discussed in Section 3.3, we make use of decision trees (TAC-L_DT and SAC-L_DT) and k-Nearest Neighbors (TAC-L_kNN and SAC-L_kNN) as our learning algorithms. At the end of 5 weeks, both the TAC and SAC k-Nearest Neighbor algorithms performed slightly worse than the decision tree algorithms shown in fig. 3. The TAC-L_kNN algorithm produces an accuracy of 57.4%, compared to the TAC-L_DT algorithm produces an accuracy of 58.2%. The highest accuracy is produced by the split access count learning (SAC-L_DT) algorithm, at 64.3%. The SAC-L_kNN algorithm was also good, producing an accuracy of 63.3%. We performed a two-tailed t-test with $\alpha = 0.05$ for the SAC-L_DT algorithm and found a statistically significant improvement ($p = 5.40E-12$) of 63.1% over that of a breadth-first crawl.

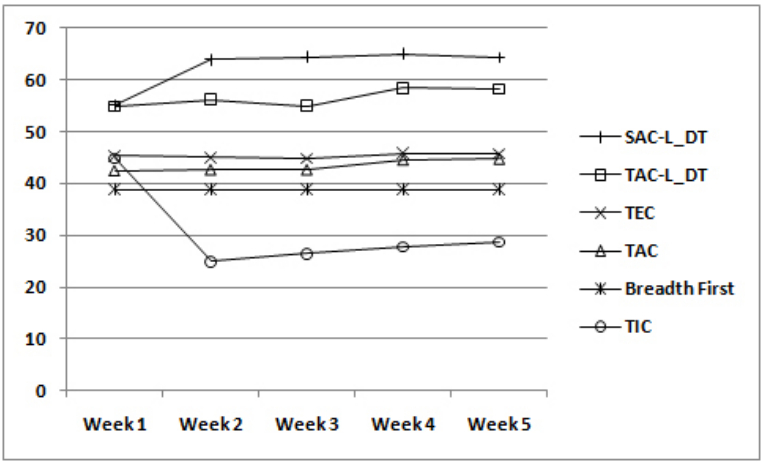

Fig. 3 URL Ordering Accuracy for DS1

5.3 Analysis of Results Obtained Using DS1

Table 1 gives the number and the average access counts of URLs/PageRank. From Table 1, we can see that the total number of URLs with PageRank 5 and 6 are much lower when compared to the number of URLs with other PageRanks. In addition, the average access counts do not increase linearly with the PageRank values. Table 1 also provides the average accuracy values/PageRank for all the popularity based URL ordering algorithms after Week 5.

Table 1 Accuracy, URL Count, and Hits per PageRank

PageRank	Avg. Accuracy	URL Count	Avg. External Hits	Avg. Unique Ext. Hits	Avg. Internal Hits	Avg. Unique Int. Hits
0	68.7	2672	20.8	14.8	3.6	2.2
1	0.6	240	8.9	8.4	1.1	1.1
2	29.5	993	22.7	19.5	2.4	1.9
3	38.7	1183	86.9	31.7	2.0	1.1
4	17.9	345	83.8	56.3	9.5	7.4
5	3.2	38	97.6	83.0	12.6	9.7
6	14.8	9	652.8	510.1	524.8	270.1

One observation from Table 1 is that all the algorithms seem to do well or badly on the same PageRank. For example, all the algorithms seem to perform better on

URLs with PageRank 0, 2 and 3 than they do on pages with higher PageRank because there are so few pages with high PageRank (only 2808 of the 5480 URLs have PageRank higher than 0 and only 392 URLs have a PageRank of 4 or higher). On the pages with moderate PageRank values, the popularity-based learning algorithms outperform the other algorithms, leading to their high overall accuracy. It is worth noting that all these PageRanks have a high number of URLs. In contrast, none of the algorithms do well for URLs with PageRanks 1, 5 and 6. For URLs with PageRank 1, we see that their average access count is much lower than the access count for URLs with PageRank 0 and 2. Although the average counts for URLs with PageRank 5 and 6 is higher than the access counts for URLs with lower PageRank, the poor performance of the popularity-based techniques may be due to the low number of URLs in this category. This is not surprising for techniques using learning algorithms since it is consistent with the axiom that the accuracy of a learning algorithm increases with more number of examples per category.

5.4 Discussion

From the results obtained in Sections 5.2 and 5.3, we conclude that, in general, the popularity-based learning algorithms order important URLs higher than breadth-first crawlers, a statistically significant result. Furthermore, our experiments show that page accesses from external domains are more important for URL ordering than page accesses from internal domains. Moreover, among the popularity-based URL ordering techniques, the methods that used learning algorithms outperformed the methods that used raw access counts. This shows that although the access count is correlated to the importance of the pages, they are not directly proportional as shown by the improved accuracy obtained by our learning algorithms (highest accuracy of 64.3%) when compared to techniques that make use of raw access counts (highest accuracy of 45.7%). In addition, this correlation may be different on different websites and learning algorithms may be able to identify this correlation and hence, find "important" pages better than non-learning algorithms.

In order to evaluate the effect of a different website, we used the best performing non-learning (TEC) and learning (SAC-L_DT) algorithms on a larger data set, the CiteSeer data set (DS2) with 102,360 URLs. Similar to DS1, we perform a five-fold cross validation and report the averages obtained. Recall that the CiteSeer data set does not have a hierarchical linking system, so we must use a random crawl to obtain a baseline. Using a random crawl baseline, we obtained an accuracy of 28.1%.

Fig. 4 provides the results obtained for TEC algorithm and the SAC-L_DT algorithm. From Fig. 4, we see that the accuracy of SAC-L-DT is 44.2% versus 28.1% for random crawl. We performed a two-tailed t-test with $\alpha = 0.05$. We achieve a statistically significant improvement ($p = 6.4E\text{-}17$) of 57.2% in our URL ordering algorithm over that of a random crawl on a large data set. This demonstrates that, once again, popularity-based URL ordering techniques outperforms a baseline (random) crawl.

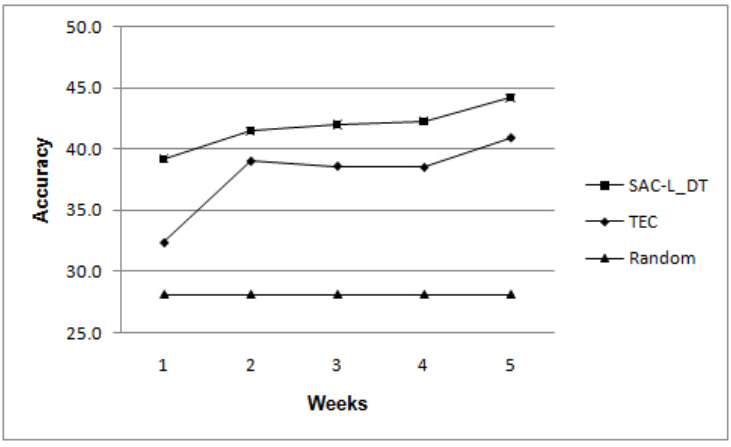

Fig. 4 URL Ordering Accuracy on DS2

6 Conclusions and Future Work

In this paper, we propose a new class of URL list creation and ordering algorithms based on file modification and popularity information from file systems and Web logs. Specifically, we present different strategies to discover new or modified pages and perform URL ordering using popularity information and compare our approaches to a breadth-first search crawls. Our evaluations show improved performance of these algorithms when compared to breadth-first search crawl.

This approach seems well-suited to social media settings where URLs may be shared in a variety of forms beyond links embedded in other web pages. In particular, the approach mitigates some of the difficulties in determining page popularity when the URL is shared in RSS feeds, text messages, or through URL shortening services. The combination of URL list generation and ordering in the system could keep up with quickly evolving social network media much more efficiently than traditional connectivity-based or breadth-first approaches.

One drawback with our approach is that new pages that have not been accessed are penalized. Adding last modified dates as a factor along with popularity information may address this issue. Another improvement to the experiment would be to rely on a larger collection where we could compute our own PageRank values, rather than relying on the discrete categories provided by Google. This would allow us to compare two full ordering algorithms to further discover how well the two methods' rankings match.

One concern with an approach that relies on web server logs for popularity ranking is the danger of manipulation to boost rankings. In the context of URL ordering, this is less of a concern, since the ordering on a site will primarily be used to modify the order of a crawl within a site rather than to request more pages from the server. In a result ranking context, one approach to mitigate the manipulation

problem would be to establish a similar ranking budget for each site, so that boosting one page only cannibalizes other resources on the site. Another option is to develop tools that can digitally sign logs and log harvesting tools to prevent external manipulation.

A final area of future work is to expand the scope of the information used for the URL list and ordering system. The URL ordering algorithms proposed in this paper are for ordering pages on a single website based only on file system and web log information. Understanding how to combine the popularity information from different logs to order pages from various websites could be another interesting problem to explore. Another possible source of information is directly from content management systems on large websites. This would allow dynamic or hidden web pages to be discovered before they appeared in Web logs.

Acknowledgment

This work was partially supported by NSF ITR 0225676 (SEEK).

References

[Brandman et al. 2000] Brandman, O., Cho, J., Garcia-Molina, H., Shivakumar, N.: Crawler friendly Web servers. In: Proc Workshop on Performance and Architecture of Web Servers (PAWS), Santa Clara, California (2000)

[Buzzi 2003] Buzzi, M.: Cooperative crawling. In: Proc. Latin American Conference on World Wide Web (LA-Web), Santiago, Chile, pp. 209–211 (2003)

[Castillo 2004] Castillo, C.: Effective Web crawling PhD Thesis, University of Chile, Chile (2004)

[Castillo et al. 2004] Castillo, C., Marin, M., Rodriguez, A., Baeza-Yates, R.: Scheduling algorithms for web crawling. In: Proc. Latin American Web Conference, Brazil, pp. 10–17 (2004)

[Cho et al. 1998] Cho, J., Garcia-Molina, H., Page, L.: Efficient crawling through URL ordering. In: Proc. 7th World Wide Web Conference, Brisbane, Australia, pp. 161–172 (1998)

[Cho et al. 2005] Cho, J., Roy, S., Adams, R.E.: Page quality: In search of an unbiased web ranking. In: Proc. 2005 ACM SIGMOD International Conference on Management of Data, Baltimore, Maryland, pp. 551–562 (2005)

[Cho and Schonfeld 2007] Cho, J., Schonfeld, U.: RankMass crawler: A crawler with high PageRank coverage guarantee. In: Proc. 33rd International Conference on Very Large Data Bases, Vienna, Austria, pp. 375–396 (2007)

[Najork and Wiener 2001] Najork, M., Wiener, J.L.: Breadth-first search crawling yields high-quality. In: Proc. 10th International World Wide Web Conference, Hong Kong, pp. 114–118 (2001)

[Pandey and Olston 2008] Pandey, S., Olston, C.: Crawl ordering by search impact. In: Proc. of the International Conference on Web Search and Data Mining, Palo Alto, California, pp. 3–14 (2008)