Efficiently Detecting Webpage Updates Using Samples

Qingzhao Tan[1], Ziming Zhuang[2], Prasenjit Mitra[1,2], and C. Lee Giles[1,2]

[1] Computer Science and Engineering
[2] Information Sciences and Technology
The Pennsylvania State University, University Park, PA 16802, USA
qtan@cse.psu.edu, {zzhuang, pmitra, giles}@ist.psu.edu

Abstract. Due to resource constraints, Web archiving systems and search engines usually have difficulties keeping the local repository completely synchronized with the Web. To address this problem, sampling-based techniques periodically poll a subset of webpages in the local repository to detect changes on the Web, and update the local copies accordingly. The goal of such an approach is to discover as many changed webpages as possible within the boundary of the available resources. In this paper we advance the state-of-art of the sampling-based techniques by answering a challenging question: *Given a sampled webpage that has been updated, which other webpages are also likely to have changed?* We propose a set of sampling policies with various downloading granularities, taking into account the link structure, the directory structure, and the content-based features. We also investigate the update history and the popularity of the webpages to adaptively model the download probability. We ran extensive experiments on a real web data set of about 300,000 distinct URLs distributed among 210 websites. The results showed that our sampling-based algorithm can detect about three times as many changed webpages as the baseline algorithm. It also showed that the changed webpages are most likely to be found in the same directory and the upper directories of the changed sample. By applying clustering algorithm on all the webpages, pages with similar change pattern are grouped together so that updated webpages can be found in the same cluster as the changed sample. Moreover, our adaptive downloading strategies significantly outperform the static ones in detecting changes for the popular webpages.

1 Introduction

Search engine relies on crawlers to harvest webpages and the downloaded webpages are stored in the local repositories. These local copies of webpages are later retrieved to answer relevant user queries. Due to the fact that webpages change independently and that the crawling and indexing process takes time to complete, searching the Web could be somewhat similar to searching for the stars through a telescope: *what is seen is the past*. Although ideally the archiving systems and search engines may synchronize the local copies of webpages with their online counterparts, due to resource constraints it is not feasible for crawlers to constantly

L. Baresi, P. Fraternali, and G.-J. Houben (Eds.): ICWE 2007, LNCS 4607, pp. 285–300, 2007.
© Springer-Verlag Berlin Heidelberg 2007

monitor and download every webpage in the local repositories. In this case, a round-robin polling strategy will waste resources in downloading unchanged webpages. To be more efficient, crawlers can periodically re-visit a portion of the webpages that are more likely to have changed since previous visit.

We investigate a typical scenario in which due to resource constraints, a crawler is only allowed to periodically download a fixed number of webpages and update the corresponding local copies when a change has been found. We define this fixed number of pages as the *download resources* and the periodical interval as the *download cycle*. Thus, the crawler's goal is to *maximize* the number of updated webpages detected in each *download cycle* with the fixed amount of *download resources*. Because a crawler can verify whether a webpage has changed only *after* it has been downloaded, the challenge therefore becomes how to predict, as accurately as possible, the change probabilities for the webpages in the local repository. Those webpages with higher change probabilities will be assigned a higher priority for download.

Cho and Ntoulas proposed a sampling-based algorithm [7] to address the above challenge. In their approach, a small number of webpages are sampled from the search engine's local repository. Their counterparts on the Web are then downloaded and examined whether they have changed. Based on the results, the crawler can determine which webpages in the local repository are more likely to change, and those webpages are also downloaded. Cho et al. theoretically and empirically proposed the optimal sample size and downloading policy based on the sampling results. While their method has been proved effective in predicting and downloading updated webpages, it is carried out in the unit of a website, i.e., pages are sampled from each website and the sampling results are used to decide whether to download other pages within the same website. However, webpages' update behavior could be carried out very differently throughout a website [14]. Within one website, some webpages are more dynamic than others.

In this paper, we investigate and improve the sampling-based techniques to detect webpage changes by making the following contributions:

- We propose a new sampling algorithm to detect webpage updates. In our algorithm, sampling is done at the webpage level and each webpage is equally likely to be selected as a sample. In addition, each sample has a tunable downloading granularity.
- We propose three downloading policies for change detection. Specifically, given a sampled webpage, we exploit its linkages (*link-based downloading policy*), directory structure (*directory-based downloading policy*), or content-based features (*cluster-based downloading policy*) to detect updates on other webpages. We also show empirically that a website may not be a good granularity for change detection.
- We propose two biased sampling algorithms based on two metrics, the change history [8] and PageRank [15]. We predict webpages updates based on their change history using the first metric, and based on their popularity using the second one.

The rest of the paper is organized as follows. In Section 2 we describe the context of our work. We propose our sampling algorithm in Section 3. In Section 4 and 5 we investigate details of the parameter space in the proposed algorithm and further propose techniques to adaptively optimize the parameter. In Section 6 we present and discuss the empirical results. We outline our conclusion in Section 7.

2 Related Work

Existing studies have investigated the dynamics of the Web pages by showing that webpages have a broad spectrum of change intervals varying from a few hours, a few weeks, to a year [12,3,9,16,6]. As a result, search engines frequently update their indices and the search results could be quite unstable [17]. In order for search engines to keep up with the dynamic Web, server-side approaches require that the Web servers keep a file with a list of URLs and their respective modification dates [2]. Before visiting a site, a crawler downloads the URL list, identifies the URLs that were modified since its last visit, and retrieves only the modified pages. This approach is very efficient and avoids waste of Web server bandwidth and crawler resources, but requires modifications to the server-side implementation.

On the other hand, client-side techniques are independent of server-side implementation. First, probabilistic models have been proposed to approximate the observed update history of a webpage and to predict its change in the future [10,13]. Most of these change-frequency–based refresh policies assume that the webpages are modified by *Poisson processes* [18]. Besides page's change frequency, several metrics have been proposed to guide the crawler how to choose webpages to re-download, such as *freshness* and *age* [10], *"embarrassment"* metric [23] and user-centric metric [22]. However, a limitation common to all these approaches is that they need to gather enough accurate historical information of each webpage.

To address this problem, sampling-based approaches are proposed to detect webpage changes by analyzing the update patterns. Their sampling method samples and downloads data in the unit of a website. Specifically, they proposed a greedy downloading policy, in which a number of pages are first sampled from each website. After getting the change status of the samples, the crawler re-visits all the webpages in the website with the largest number of changed samples, and then the website with the second largest number of changed samples, and so on, until the download resources are used up. Different from their approaches, we analyze sampling and downloading in different granularities, which is based on webpages linkages, directory structure, and content-based features.

Recently, a classification-based algorithm [1] takes into account both the history and the content of pages to predict their change behavior. Their experiment results on real web data indicate that their solution had better performance than the change-frequency–based policy [9]. The key difference between their algorithm and ours is that they assume the crawler knows the change history of

a set of webpages. And these webpages are used as training data to learn the change pattern of other new pages. While in our approach, we do not assume the crawler have any existing historical information. Therefore, our approach is more practical than the classification-based method.

3 Sampling-Based Update Detection

We present a sampling-based update detection algorithm based on the assumption that *webpages that are relevant (discussed later) to an updated webpage are also likely to be updated as well*. The goal of the algorithm is to *maximize* the number of updated webpages downloaded in each *download cycle*, with a fixed amount of *download resources*. Please note that our sampling-based algorithm can detect an *"update"* including the creation, change, or removal of a webpage. In this section, we describe in detail the sampling and downloading process of our change detection algorithm. This process constitutes the main part of the algorithm.

Let L_0 denote the initial set of webpages in the local repository during the initiation of the algorithm. We also denote the *download resources* for each *download cycle* as R, and subsequently updated versions of L_0 in the following n download cycles as $L_i (i = 1, ..., n)$. Algorithms 1 and 2 show the sampling and downloading process during each download cycle, respectively. There are two tunable parameters, the *download probability* φ and the *download granularity* d, which are explained in details in Section 4 and 5. The algorithm proceeds as follows. The sampling process randomly chooses a set of pages as *samples* from the local repository. For each sampled page $p_s \in L_{i-1}$, its current version on the Web is downloaded and compared with the local version p_s. Based on this comparison, the process triggers the downloading process with the probability φ. During the execution of the downloading process, p_s is used as a *seed* page and its neighbors within d are downloaded. Please note that in the downloading process p will be downloaded in either of the following cases. In the first case, p is downloaded as a **sample**. In the second case, p is a **neighbor within distance** d **of a changed sample**, and is downloaded with a probability φ.

4 Tuning Download Granularity

The download granularity d determines that, given a changed sample, which additional webpages and how many of them the crawler should choose to download. We propose three definitions of d and investigate which definition gives us the best performance, in terms of maximizing the probability of discovering and downloading webpages that have been updated.

4.1 Link-Based Downloading Policy

Consider the directed graph created from the World-Wide Web where the nodes are webpages and a directed edge between two nodes p_1 and p_2 corresponds to

Algorithm 1. Sampling Process in the i^{th} Download Cycle

Input: L_{i-1}, download probability φ, download granularity d, download resources R
Procedure:
1: **repeat**
2: Randomly sample a webpage p_s from L_{i-1};
3: Download p_s and check its change status;
4: With probability φ, $L_i \leftarrow \text{Download}(p_s, d)$;
5: **until** R is used up
6: **if** $|L_i| < |L_{i-1}|$ **then**
7: $L_i \leftarrow (L_{i-1} - L_i)$;
8: **end if**
Output: L_i

Algorithm 2. Download(p_s, d)

Input: Seed webpage p_s and download granularity d
Procedure:
1: download a set of webpages P^d within d from p_s;
Output: P^d

a link from the webpage represented by p_1 to the webpage represented by p_2. In this policy, when the algorithm identifies a sampled webpage, say p_s has been updated, it crawls all webpages whose nodes are within a distance d of the node representing p_s. We refer to this approach as the *link-based downloading policy* (LB). The intuition behind this policy is the topical locality in the Web [11].

The depth, d_{LB}, between two webpages p_s and p can be formally defined as follows. Let DG denote the directed graph generated based on the topology of a website, by considering the webpages as nodes and the hyperlinks as the directed edges between the nodes. Let $\overline{DG}$ denote the directed graph with the same set of nodes as DG but all the directed edges reversed. We then define d_{LB} as:

$$d_{LB} = \begin{cases} h & \text{if } p_s \rightsquigarrow p \text{ in } DG, \\ -h & \text{if } p_s \rightsquigarrow p \text{ in } \overline{DG}, \\ 0 & \text{if } p_s \text{ and } p \text{ have a common parent,} \\ \infty & \text{otherwise.} \end{cases} \quad (1)$$

where $p_s \rightsquigarrow p$ indicates that a path exists from p_s to p (i.e. p is reachable from p_s), and $h > 0$ denotes the length of the shortest path between p_s and p in the number of hops. When $d_{LB} > 0$, the crawler follows only the out-links; when $d_{LB} < 0$ the crawler follows only the in-links. d_{LB} between two siblings (i.e. webpages that share a common parent) is defined as 0.

Figure 1 depicts the partial topology of a website. If there are more than one paths from p_s, take the shortest one (e.g., p_{10}). d_{LB} of p_s's siblings is always 0 although there are paths from p_s to them (e.g., p_5). When $d_{LB} = 1$, the crawler

in our algorithm downloads all the webpages that have $d_{LB} \leq 1$, i.e. p_s, p_4, p_5, p_6, p_7, and p_{10}.

4.2 Directory-Based Downloading Policy

Alternatively, we define the download granularity d based on the directory structure of the Web server, and refer to this approach as the *directory-based downloading policy* (DB). In this policy, we consider the directory structure as a hierarchical tree structure and webpages as the leaf nodes (illustrated in Figure 2). Formally, let p_s denote the seed webpage at $http://u_1/u_2/.../u_m$, p denote a webpage at $http://v_1/v_2/.../v_n$, and $u_c = v_c$ denote their nearest common ancestor, i.e. $\exists c$, $0 \leq c \leq min(m, n)$, $\forall i \leq c, u_i = v_i$ and $\forall i > c, u_i \neq v_i$. d_{DB} can then be defined as follows:

$$d_{DB} = [(m - c) + (n - c) - 2] * sign(n - m). \tag{2}$$

The constant 2 is a scaler used to leverage the value of d_{DB} so that webpages at the same level as p_s in the directory tree will have $d_{DB} = 0$. The function $sign(n - m)$ indicates whether the webpage is on the level upper or lower than the level of the seed webpage.

We use the following example to illustrate the definition of d_{DB}. Assume in Figure 2 the webpage at *http://www.abc.com/dir1/index1.htm* is chosen as the seed p_s for the downloading process. The values of d_{DB} from p_s to other webpages are also shown in the figure. Intuitively, the download granularity is a directory on the Web server. For example, $d_{DB} = 0$ indicates downloading all the webpages in the same directory as the seed; $d_{DB} > 0$ indicates downloading every webpage in the *upper* directories; and $d_{DB} < 0$ indicates downloading every webpage in the *lower* directories.

4.3 Cluster-Based Downloading Policy

We now cast the problem of discovering webpages that are updated concurrently with a seed page into a clustering problem, which we refer to as the *cluster-based downloading policy* (CB). There are two main steps in this policy. We first extract features that are relevant to the webpages' change pattern. Recent studies have found that some characteristics of webpages are strongly correlated with their change frequencies. For example, Douglis et al. [12] noted that actively-changing webpages are often larger in size and have more images. Fetterly et al. [14] observed that the top-level domains of the webpages are correlated with their change patterns. Based on previous work and our observations, the feature vector used for the clustering algorithm include the following features: content of the webpage and URLs (see details below), number of non-markup words, number of incoming and outgoing links, number of images, name of the top-level domain in the URL, depth of the URL (i.e. number of slashes), size of the file, etc. We construct a word-level vector to represent the content of the webpage. To avoid its dominance over other features, we first cluster all the webpages based on the

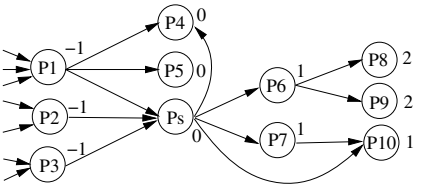


Fig. 1. d_{LB} from p_s to other webpages in the link topology. Webpages further away from p_s have larger $|d_{LB}|$s.

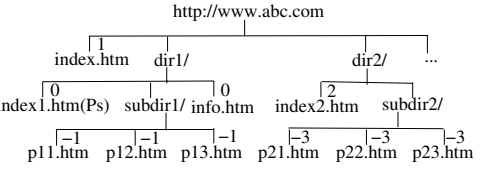


Fig. 2. d_{DB} from p_s to other webpages in the directory structure. Under DB, d_{DB} is defined at a directory level.

$tf \cdot idf$ vector of all the words. The IDs of the 10 largest clusters are chosen as the labels of 10 dimensions in the feature vector. Then for each webpage p, the values of these 10 dimentsions are computed as follows. Suppose the webpage belongs to the cluster C_p, the feature's cluster label is C_f. The value of the feature f_{C_f} is computed as follows: $f_{C_f} = 0$ for the webpages in $C_p = C_f$, $f_{C_f} = 1$ for the webpages in C_p which has the shortest distance to C_f, $f_{C_f} = 2$ for the webpages in C_p which has the second shortest distance to C_f, and so on. We use the distance of the centroids of each cluster to represent the distance of two clusters. A similar method is used to generate another 10 dimensions of the feature vector, using all the words in the URLs.

Second, with each webpage represented by a feature vector, we apply the Repeated Bisection Clustering algorithm [20] to construct hierarchical clusters, each of which contains webpages with a similar update pattern. A k-way clustering solution is obtained via a sequence of cluster bisections: first the entire set of webpages is bisected, then one of the two clusters is selected and further bisected; such process of selecting and bisecting a particular cluster continues until k clusters are obtained. Note that the two partitions generated by each bisection are not necessary equal in size, and each bisection tries to ensure a criterion function is locally optimized. Based on the clusters obtained from the algorithm, we use the distance between the centroids of two clusters to compute d_{CB}, similar to the method used to compute the values of the URL/webpage features.

5 Adaptive Download Probability

In Algorithm 1, a download probability $\varphi \in [0...1]$ is used. The larger the value of φ, the more likely it is that the sample is used as a seed to trigger the download process. We first consider the baseline case in which φ is a fixed binary variable $\{0, 1\}$ for all webpages. Note that the sample itself will always be downloaded regardless of its change status. Therefore, $\varphi = 0$ indicates that no webpages other than the sample is downloaded, which is equivalent to randomly download webpages from the Web. When $\varphi = 1$, all of the *neighbors* of the sample are downloaded. Both of these two extreme cases are intuitively not good downloading strategies to optimize performance. In the following discussions, we

propose adaptively assigning different φs for various sampled webpages based on three characteristics, current change status, change history, and webpage popularity.

5.1 Adapting to Current Change Status

A straightforward improvement over random download is to adapt the *download probability* φ to the sampled webpage's *current* change status as follows:

$$\varphi = \begin{cases} 1 & \text{if the sampled webpage has changed,} \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

With the above definition, whether or not a sample has changed in the current download cycle determines whether to crawl and download its neighbors. While this is an effective approach with relatively easy implementation, it considers only the most recent change status of the sample, with no regard to information such as the temporal patterns of change and the intrinsic quality of the sampled webpage.

5.2 Adapting to Change History

Here we discuss how to adapt the *download probability* φ toward the change history of a sampled webpage. Figure 3 shows three webpages with different change patterns. Assume that at time T_{i+1} these three pages are downloaded. By comparing their $(i+1)^{st}$ and i^{th} versions, the crawler observes that all of them have changed and a naive crawler concludes that they are equally likely to change again at time T_{i+2}. However, by looking at their change history between T_0 and T_{i+1}, we observe that $P1$ has only one change, while $P2$ and $P3$ both have three changes. Moreover, $P2$ has two of its three changes at the beginning of $[T_0, T_{i+1}]$. On the contrary, $P3$ has no change at the beginning but three changes toward the end. Through these historical change patterns we could determine the ascending order of their change probabilities as $P1 < P2 < P3$.

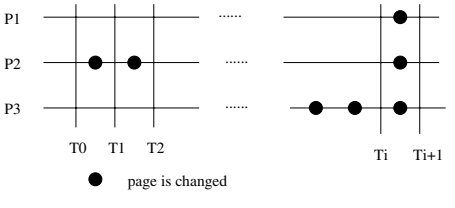


Fig. 3. Three webpages with different change patterns over time. Considering only their latest updates will lead to a misconception that they have a similar change pattern.

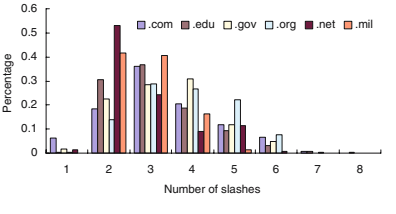


Fig. 4. Distribution of webpages by the number of slashes in the URLs for each top-level domain. Majority of URLs have depth smaller than 7.

We model the change of the webpages using a *Poisson process* [18], which has been shown effective in experiments with real webpages [4,10]. The *Poisson process* is often used to model a sequence of events that happen randomly and independently at a fixed rate over time. In the *Poisson process*, the time to the next event is exponentially distributed. In our scenario it is reasonable to assume the updates of a webpage p follows a Poisson process with its own change rate λ_p. This means a webpage changes on its own, independent of other pages. This assumption may not strictly be true but is a workable first approximation. Note that the change rate may differ from page to page. For each webpage p, let T denote the time when the next event occurs as a *Poisson process* with change rate λ_p. Then, we obtain the probability that p changes in the interval $(o, t]$ by integrating the probability density function:

$$Pr\{T \leq t\} = \int_0^t f_p(t)dt = \int_0^t \lambda_p e^{-\lambda_p t} dt = 1 - e^{-\lambda_p t}. \tag{4}$$

We set the parameter *download probability* φ to be $Pr\{T \leq t\}$ where $t = 1$, which means one download cycle. Therefore,

$$\varphi = Pr\{T \leq 1\} = 1 - e^{-\lambda_p}. \tag{5}$$

Obviously, φ depends on the parameter λ_p. We compute λ_p based on the change history of the webpage p within n download cycles:

$$\lambda_p = \frac{\sum_{i=1}^{n} \mathbf{I}(L_i[p])}{n}. \tag{6}$$

where $\mathbf{I}(p_i)$ is an indicator function defined as follows:

$$\mathbf{I}(L_i[p]) = \begin{cases} 1 & \text{if } L_i[p] \neq L_{i-1}[p], \\ 0 & \text{otherwise.} \end{cases} \tag{7}$$

Next, in order to take into account the distribution of change events, we attach different importance to changes occurred in different download cycles. To formalize this, we extend the definition of λ_p in Equation (6) by assigning a weight w_i to each download cycle and define:

$$\lambda_p = \frac{\sum_{i=1}^{n} w_i \cdot \mathbf{I}(L_i[p])}{n}, \sum_{1}^{n} w_i = 1. \tag{8}$$

Typically, $w_a < w_b$ is satisfied when $a < b$, which indicates that changes occurred in the more recent download cycles are more important. When all w_i are equal, Equation (8) reduces to Equation (6). We refer to Equation (6) as the *History-Adaptive strategy* (HA), and refer to Equation (8) as the *Weighted-History–Adaptive strategy* (WHA).

5.3 Adapting to Webpage Popularity

PageRank [15] is a probability distribution to indicate the likelihood that a person randomly traversing the Web will eventually arrive at any particular

page. The PageRank of a webpage, p_i, is calculated as follows:

$$PR(p_i) = \frac{1-d}{N} + d \sum_{p_j \in IL(p_i)} \frac{PR(p_j)}{OL(p_j)}. \quad (9)$$

where $p_1, p_2, ..., p_N$ are the webpages on the Web, $IL(p_i)$ is the set of pages that link to p_i, $OL(p_j)$ is the number of out-going links on page p_j, and N is the total number of webpages in the local repository.

Intuitively, PageRank is a good estimator for the *popularity* of a webpage. As the third adaptive strategy, we propose to set φ for each sampled webpage in proportion to its *popularity* as reflected in the PageRank. Note that under this definition, whether to crawl and download the neighbors of a sample is independent of whether the sample has changed. We refer to this as the *PageRank-Adaptive strategy* (PRA).

6 Evaluation and Discussion

We carried out extensive experiments on a large dataset to evaluate the sampling-based update detection algorithm and the various parameter settings we proposed.

6.1 Data Collection and Evaluation Metrics

All of our experiments were carried out on a collection of real webpages from the WebArchive project[1]. To obtain the historical snapshots of these webpages, we implemented a special spider to crawl the Internet Archive[2], which has archived more than 55 billion webpages since 1996. Excluding the webpages that were not available in the Internet Archive, we eventually constructed a dataset containing approximately 300,000 distinct webpages that belong to more than 210 websites, with their historical snapshots dated between Oct. 2002 and Oct. 2003. We have the largest number of websites in the .com domain and the largest number of webpages in the .edu domain (see Table 1 for the distribution by different top-level domains). Overall, our dataset is diverse enough to evaluate our proposed algorithms.

Figure 4 shows the *depth* distribution of the crawled webpages, which is measured in the number of slashes in the URLs. We can see from this figure that the majority of URLs have depth of 2, 3, and 4, while only a few URLs have depth of 7 and 8. Based on this observation, we tuned the download depth d from -6 to 8 for DB in our experiments.

The following two metrics were used in the evaluation of our proposals.

- *ChangeRatio*: The fraction of downloaded and changed webpages D_i^c over the total number of downloaded webpages D_i in the ith download cycle [12]. In our experiments we measure the per-download-cycle *ChangeRatio* C_i as well as the *average ChangeRatio* $\overline{C}$ which is the mean C_i over all download cycles.

[1] http://webarchive.cs.ucla.edu/
[2] http://www.archive.org/

Table 1. Distribution of top-level domains in the collected data; .com and .edu together account for more than 70% of the webpages

domain	total	unique websites	avg urls per site
.edu	107204 (37.7%)	68	1576.5
.com	100725 (35.4%)	92	1094.8
.gov	38696 (13.6%)	23	1682.4
.org	22391 (7.9%)	16	1399.4
.net	12972 (4.6%)	12	1081.0
.mil	1998 (0.7%)	1	1998.0
Sum	284692 (including 706 misc. URLs)		

– *Weighted ChangeRatio*: The above general *ChangeRatio* metrics treat every webpage as equally important. However, in practice some webpages may be more *popular* than others so that once they have changed, such change has a higher priority to be detected by the crawler. Motivated by this observation, we also introduce a *Weighted ChangeRatio* [7] by giving different weights w_p to the changed pages, formally defined as

$$C_i^w = \sum_{p \in D_i} w_p \cdot \frac{\mathbf{I_1}(p)}{|D_i|} \tag{10}$$

in which $\mathbf{I_1}(p)$ is an indicator function:

$$\mathbf{I_1}(p) = \begin{cases} 1 & \text{if } p \in D_i^c, \\ 0 & \text{otherwise.} \end{cases} \tag{11}$$

In our experiments, we use a webpage's PageRank [15] as an indicator for the popularity of the webpage.

6.2 Results and Discussion

We now present our findings in applying the sampling-based algorithm on the aforementioned data collection. For all of the experiments, the download cycle is set incrementally from 2 weeks, 4 weeks, to 8 weeks, and the download resources R is set to be 25K, 50K, and 100K. Note that at the end of each download cycle, our crawler chooses some webpages to download, each of which is downloaded only once during one cycle. When a webpage is crawled, the corresponding historical snapshot with the same time-stamp from the Wayback Machine is checked whether has changed or not; if there is no exact match in the archive, the version with the closest time-stamp is checked instead.

Comparison of downloading policies. To compare the performance under different definitions of *download granularity* d, we implemented our proposals, *LB*, *DB*, and *CB*, with the download probability φ set to 1 for the changed samples and 0 for the unchanged ones. We tuned d from -6 to 8. We present here

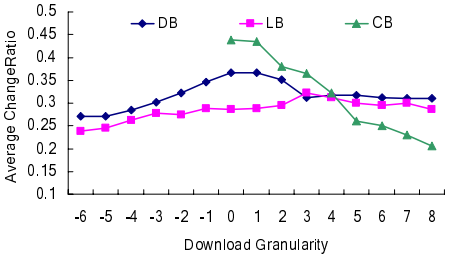


Fig. 5. Comparison of three downloading policies in $\overline{C}$

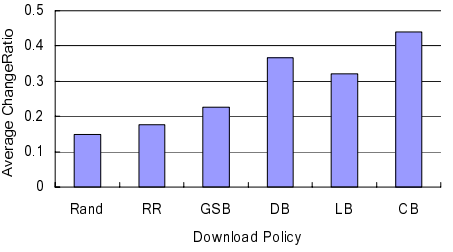


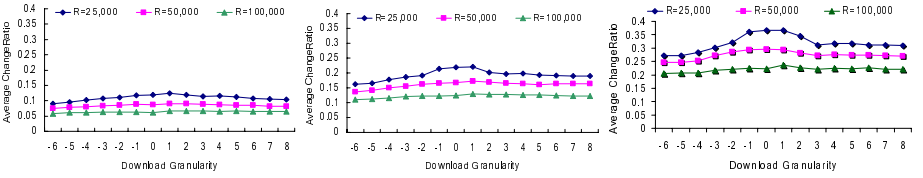
Fig. 6. Comparison of existing downloading policies with ours

only the setting for the download cycle to be 8 weeks and the download resource limit to be 25,000 webpages, because the results under other settings are similar. Later on, we will show more details with the different settings of download cycle and resources.

As illustrated in Figure 5, overall DB outperforms LB in terms of $\overline{C}$. Based on the empirical results, it is interesting to see that following the hyperlinks might not be a good strategy for the crawler to discover newly updated webpages, as webpages with similar change patterns are put in the same or a nearby directory rather than linked with each other. Compared with CB, however, DB is less effective when $0 \leq d \leq 3$. As DB is much more efficient than CB, in the following experiments, we fixed thef downloading policy as DB to further investigate in detail the other facets of the parameter space.

We also compared our proposed algorithm with three other existing downloading policies. First, we used the Random Node Sampling (RNS) method proposed in [21] as a baseline in our comparison. With the RNS method, the crawler uniformly re-downloads random webpages in each download cycle. Second, we included the round-robin (RR) method for comparison, which is currently adopted by many systems [5,19] because it is simple to implement. In RR, the webpages are downloaded in a round-robin fashion in each download cycle. The advantage of this method is that every page is guaranteed to be downloaded within a certain period of time. Finally, we implemented the greedy sampling-based technique (GSB) over site level, setting the sample size to be $\sqrt{Nr}$ as proposed in [7]. Here N is the average number of pages in all sites, and r is the ratio of download resources to the total number of webpages in the local repository. For our approaches, we use the optimal setting of d, i.e., $d_{DB} = 1$, $d_{LB} = 3$, and $d_{CB} = 0$.

We present the results in Figure 6. From this figure we can see that our approaches all perform better compared to RNS, RR, and GSB. More specifically, RR has similar performance as the baseline aglorithm, RNS. The performance of GSB is better than RNS and RR. Our sampling-based algorithm can detect about three times as many changed webpages as RNS/RR, and about twice as many changed webpages as GSB, which is a sampling-based approach at a site level.



(a) Cycle = 2 weeks (b) Cycle = 4 weeks (c) Cycle = 8 weeks

Fig. 7. *ChangeRatio* as a function of download depth d under different settings of download cycles and download resources R. Overall, a smaller download granularity and fewer download resources perform the best in a longer download cycle.

Effect of download granularity. Figure 5 has already shown $\overline{C}$ is strongly influenced by the download granularity d. For all the downloading policies, $\overline{C}$ goes up when d increases, and it drops after reaching its peak at $d = d_{opt}$. Empirically, $d_{opt} = [0, 1]$ for DB, $d_{opt} = 3$ for LB, and $d_{opt} = 0$ for CB. This indicates that given a changed sample, the most likely changed webpages are usually stored in the same or the upper directory of the sample, or are within three hops away following the out-links of the changed sample. CB has the highest $\overline{C}$ when $d_{CB} = 0$ because the webpages within the same cluster have similar change pattern. Then the $\overline{C}$ largely decreases when $d_{CB} > 0$. This is because d_{CB} is defined at a cluster level, which contains average 3,000 webpages in our experiments. Thus, the change of d_{CB} leads to significant descrease of $\overline{C}$ when $d_{CB} > 0$. Moreover, for both LB and DB, $\overline{C}$ with positive d is higher than that of negative d. This means that given a changed sample, changed pages can be found in the sample's upper directories more than lower ones, and its out-links more than in-links.

The aforementioned results were obtained by setting the download cycle to be 8 weeks and the download resources to be 25,000 webpages. Next, we further evaluate the influence of d by varying the other two parameters: download cycle and download resources. We still assign the download probability φ as in Equation (3) ($\varphi = 1$ for the changed samples and 0 for the unchanged ones). Figure 7 shows $\overline{C}$ as a function of d under different settings of the download cycle and the download resource R. From the graph, we can confirm the trend that we discussed in Figure 5. Furthermore, there are three observations. First, the longer the download cycle, the better performance in terms of $\overline{C_i}$. This can be explained by the fact that more webpages are likely to have changed in a longer timeframe. Second, the shapes of the three $\overline{C_i}$ curves under the same settings are roughly the same, indicating that our proposal could be potentially applied toward crawlers with different download abilities and scalabilities. Third, it appears that a smaller R produces better results. This shows that our sampling-based algorithm works fine even if R is limited.

Impact of adaptive download probability. We present our findings on the impact of adaptively tuning the download probability φ. We tuned the download

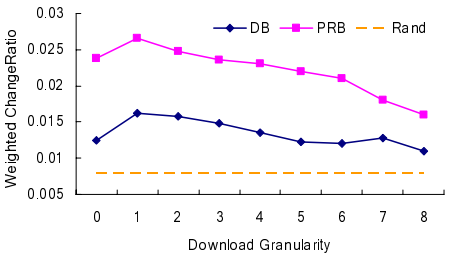


Fig. 8. Comparison of the adaptive PageRank-adaptive strategy and the non-adaptive DB in $\overline{C^w}$

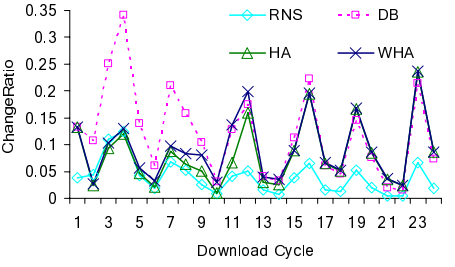


Fig. 9. Comparison of the long-term performance over 24 cycles. C_i of the two history-adaptive strategies eventually outperformed that of the DB.

probability φ with the following three adaptive strategies: *PRA*, *HA*, and *WHA*. For PRA, we retrieved from CPAN Google PageRank API[3] to obtain each page's PageRank, which is an integer between 0 and 10. We then set φ to be 1/11, 2/11, ..., and 11/11, for webpages with *PageRank* of 0, 1, ..., 10, respectively. For HA and WHA, we assign φ according to Equation (5). Specifically for *WHA*, to calculate the weighted λ_p according to Equation (8), we assign the weight w_i for the i^{th} download cycle as $w_i = i/(1 + ... + n)$ where n denotes the total number of cycles the crawler has processed thus far.

We compared these adaptive strategies with DB. Here we present the results of a typical setting, i.e., the download cycle is set to 2 weeks, the download resources are set to 25,000 webpages, and d is positive. Figure 8 shows the comparison results measured in C_i^w. We see that PRA clearly outperforms DB by almost 100% in C_i^w.

From Figure 9, we see that at the beginning, the two *history-adaptive* strategies that take into account the webpages' change history performed worse than DB. Specifically, from cycle 2 to cycle 5, their *ChangeRatios* were similar or even slightly worse than the random sampling policy. However, as time went by and more information about the change history could be gathered, the *ChangeRatios* for the two history-adaptive strategies gradually became higher in the following download cycles, and eventually outperformed DB. This reconfirms that in the long-run, algorithms that take into account the past changes of the webpages will have a better prediction about future changes to these webpages. Between the two history-adaptive strategies, WHA performed better than HA. Moreover, WHA outperformed DB in download cycle 16, earlier than HA did in cycle 18. These results indicate that a crawler which takes into account the change history of the webpages should pay more attention to the more recent changes than the older ones. Toward the right-hand side of the figure, the difference in *ChangeRatio* among the three investigated adaptive strategies became less significant. We believe this is because the 24 download cycles were still not long enough for the two history-adaptive strategies to demonstrate their advantages over DB.

[3] http://search.cpan.org/

We plan to explore the history-adaptive strategy over a longer timeframe in our future work.

7 Conclusion

In this paper, we studied a challenging problem in automatic Web content archiving: how to make the local repository as up-to-date as possible under a fixed amount of available download resources? Motivated by the observation that *relevant* webpages tend to have similar change patterns, we proposed a sampling-based algorithm which uses the changed samples as the *seeds* to discover more changed webpages. We investigated in detail the parameter space and conducted extensive experiments to evaluate our proposals. We found that given a changed sample, the most likely changed webpages are usually stored in the same cluster as that of the changed sample. And the PageRank adaptive strategy is good at discovering changes to the popular webpages. Finally, for long-term performance, the weighted-history–adaptive strategy will outperform the others once the crawler has acquired sufficient information about the historical change patterns.

References

1. Barbosa, L., Salgado, A.C., de Carvalho, F., Robin, J., Freire, J.: Looking at both the present and the past to efficiently update replicas of web content. In: WIDM '05, pp. 75–80 (2005)
2. Brandman, O., Cho, J., Garcia-Molina, H., Shivakumar, N.: Crawler-friendly web servers. In: PAWS '00 (2000)
3. Brewington, B.E., Cybenko, G.: How dynamic is the Web? In: WWW '00 (2000)
4. Brewington, B.E., Cybenko, G.: Keeping up with the changing web. Computer 33, 52–58 (2000)
5. Brin, S., Page, L.: The anatomy of a large-scale hypertextual web search engine. In: WWW'98, pp. 107–117 (1998)
6. Castillo, C.: Effective web crawling. ACM SIGIR Forum 39, 55–56 (2005)
7. Cho, J., Ntoulas, A.: Effective change detection using sampling. In: Bressan, S., Chaudhri, A.B., Lee, M.L., Yu, J.X., Lacroix, Z. (eds.) CAiSE 2002 and VLDB 2002. LNCS, vol. 2590, Springer, Heidelberg (2003)
8. Cho, J., Garcia-Molina, H.: Synchronizing a database to improve freshness. In: SIGMOD '00, pp. 117–128 (2000)
9. Cho, J., Garcia-Molina, H.: The evolution of the web and implications for an incremental crawler. In: VLDB '00, pp. 200–209 (2000)
10. Cho, J., Garcia-Molina, H.: Effective page refresh policies for web crawlers. ACM TODS 28, 390–426 (2003)
11. Davison, B.: Topical locality in the web. In: SIGIR '00 (2000)
12. Douglis, F., Feldmann, A., Krishnamurthy, B., Mogul, J.C.: Rate of change and other metrics: a live study of the world wide web. In: USENIX Symposium on Internet Tech. and Syst. (1997)
13. Edwards, J., McCurley, K., Tomlin, J.: An adaptive model for optimizing performance of an incremental web crawler. In: WWW '01, pp. 106–113 (2001)

14. Fetterly, D., Manasse, M., Najork, M., Wiener, J.L.: A large-scale study of the evolution of web pages. In: WWW '04, Budapest, Hungary (2004)
15. Page, L., Brin, S., Motwani, R., Winograd, T.: The pagerank citation ranking: Bringing order to the web. Technical report, Stanford Digital Library Technologies Project (1998)
16. Ntoulas, A., Cho, J., Olston, C.: What's new on the web?: the evolution of the web from a search engine perspective. In: WWW '04, pp. 1–12 (2004)
17. Selberg, E., Etzioni, O.: On the instability of web search engines. In: 6th Recherche dInformations Assistee par Ordinateur (RIAO) Conference (2000)
18. Grimmett, G., Stirzaker, D.: Probability and Random Processes, 2nd edn. Oxford University Press, Oxford, England (1992)
19. Heydon, A., Najork, M.: Mercator: A scalable, extensible web crawler. World Wide Web 2, 219–229 (1999)
20. Karypis, G., Han, E.H.S.: Fast supervised dimensionality reduction algorithm with applications to document categorization & retrieval. In: CIKM '00, pp. 12–19 (2000)
21. Leskovec, J., Faloutsos, C.: Sampling from large graphs. In: KDD '06, Philadelphia, USA, pp. 631–636 (2006)
22. Pandey, S., Olston, C.: User-centric web crawling. In: WWW '05, pp. 401–411 (2005)
23. Wolf, J.L., Squillante, M.S., Yu, P.S., Sethuraman, J., Ozsen, L.: Optimal crawling strategies for web search engines. In: WWW '02. pp. 136–147 (2002)