Clustering-based Incremental Web Crawling

QINGZHAO TAN and PRASENJIT MITRA
The Pennsylvania State University

When crawling resources, e.g., number of machines, crawl-time, etc., are limited, a crawler has to decide an optimal order in which to crawl and re-crawl webpages. Ideally, crawlers should request only those webpages that have changed since the last crawl; in practice, a crawler may not know whether a webpage has changed before downloading it. In this paper, we identify features of webpages that are correlated to their change frequency. We design a crawling algorithm that clusters webpages based on features that correlate to their change frequencies obtained by examining past history. The crawler downloads a sample of webpages from each cluster and depending upon whether a significant number of these webpages have changed in the last crawl cycle, it decides to recrawl the entire cluster. To evaluate the performance of our incremental crawler, we develop an evaluation framework that measures which crawling policy results in the best search results for the end-user. We run experiments on a real Web data set of about 300,000 distinct URLs distributed among 210 websites. The results demonstrate that the clustering-based sampling algorithm effectively clusters the pages with similar change patterns, and our clustering-based crawling algorithm outperforms existing algorithms in that it can improve the quality of the user experience for those who query the search engine.

Categories and Subject Descriptors: H.4.0 [**Information Systems Applications**]: General

General Terms: Algorithms, Performance, Design, Measurement, Experimentation

Additional Key Words and Phrases: Clustering, refresh policy, sampling, search engine, Web crawler

1. INTRODUCTION

Before webpages can be indexed, they need to be crawled [Brin and Page 1998]. Because webpages change, local repositories are never completely up-to-date, i.e., they may not reflect the latest changes to all the webpages at all times. However, a crawler must attempt to keep the local repository as up-to-date as possible. Given the limited resources that a crawler has, it must decide on a crawling policy that will keep its repository the freshest. Because different webpages have different change frequencies, the crawler must decide which webpages it must crawl, in which order and at what frequency.

Crawling webpages that have not changed is a waste of resources. Crawlers request for webpages using an *if-modified-since* HTTP (Hypertext Transfer Protocol) request. If the webpage has not changed since the last time it was crawled, a reply without the webpage is sent back. However, when a crawler has to crawl billions of webpages, even if most of them have not been modified, the round-trip-times taken by the web servers to check

Author's address: Q. Tan, The Pennsylvania State University, University Park, PA 16802. Email: qtan@cse.psu.edu

whether these billions of webpages have changed could be weeks when using a standard desktop machine. Therefore, a crawler that could predict whether a webpage has changed or not could save resources wasted in checking for webpages that have not changed. That is, we need a good prediction model to predict whether a webpage is likely to have changed since the last time it was crawled. Building a prediction model for updates to webpages is difficult. Also, if a webpage is never retrieved in response to any user query, ideally, a crawler should not crawl the webpage. Or if the change to the webpage is cosmetic, e.g., to the visual presentation of a webpage, the updated webpage should not be crawled because only the main words in a webpage are indexed. Even if the crawler has not crawled the webpage with cosmetic changes, if the webpage is returned as a result of the query, upon clicking the link to the webpage, an end-user can see the latest page.

In this work, we investigate a typical scenario in which due to resource constraints, a crawler is only allowed to periodically download a fixed number of webpages. We define this fixed number of pages as the *download resources* and the periodical interval as the *download cycle*. We count the number of webpages downloaded because in several cases the webpages do not change and an *if-modified-since* HTTP request results in no webpages returned. Previous research has shown that the round-trip-times dominate the transfer time for each changed webpage [Liu et al. 1998]. Admittedly, a more accurate model where we count both the number of pages and the size of the webpages sent to the crawler measures resource requirements could be designed. However, because the round-trip-times dominate, we believe counting resources using the number of pages is a reasonable approximation.

Cho and Ntoulas [Cho and Ntoulas 2002] proposed a sampling-based algorithm. Their algorithm draws samples of webpages from a website, detects their changes, and then decides if the entire website should be crawled. Their sampling is at the level of a website. The level of a website may not be a good granularity for sampling because webpages with similar update patterns may distribute across different websites [Fetterly et al. 2003].

We investigate and determine which features of webpages can predict the *change frequency* of the webpage and predict the *importance* of a webpage. Webpage features have already been utilized Webpage features have already been utilized to "predict high level concepts such as page dynamism"[1] in some existing work [Pant and Srinivasan 2009]. Pant and Srinivasan "capture page dynamism through content inertia measure and explore its association with link-based page importance". Following this trend, our algorithm clusters webpages based on features like the existence of certain content words, etc., (see Section 3 for the full set of features we considered). Our experiments show that webpages in a cluster change at similar frequencies. We identify the change frequency of each cluster and rank the clusters based on their change frequency. Our crawling algorithm downloads a sample set of webpages from the highest-ranked cluster, and checks whether the downloaded pages in the cluster have changed. If a significant number of pages in a cluster have changed, the whole cluster is re-crawled, otherwise, it proceeds to process the next highest ranked cluster.

We propose a crawling policy evaluation model that measures the improvement of the bottom-line, namely, the quality of users' search results. We show that the clustering-based crawling policy outperforms other methods with respect to keeping the local repository fresh. Our results can be easily incorporated in a commercial crawler and can improve the

[1]Communicated by anonymous reviewer.

quality of local repositories and consequently improve the search experience of users.

In this paper, our major contributions are as follows:

—We propose a new clustering-based crawling algorithm to detect webpage updates. We investigate the feature space for clustering webpages according to their likelihood of change, by studying the static features (e.g. size of a webpage), as well as the dynamic features (e.g. the observed changes to a webpage over time). By partitioning the feature vector into different categories, we study which set of features are the most effective in change prediction.

—We compare four different settings, (1) non-adaptive, (2) shortsighted-adaptive, (3) arithmetically-adaptive, and (4) geometrically-adaptive policy, to compute the change frequency of the samples. We conclude that the prediction made by the adaptive policies is more accurate than that of the non-adaptive one.

—Two evaluation metrics, *top-k freshness* and *top-k Mean Precision* are used to measure the ranking quality. We show empirically that the cluster-based downloading policy performs the best on our datasets.

The rest of the paper is organized as follows. In Section 2 we describe existing work related to our problem. We propose a clustering-based sampling process and the features used for clustering in Section 3. In Section 4 we investigate the sampling process on the cluster level. In Section 5 we present our evaluation model for incremental crawlers. In Section 6 we discuss the empirical results. Finally, we outline our conclusion and scope for future work in Section 7.

2. RELATED WORK

In this section, we survey related work on incremental crawlers. Incremental crawlers detect webpages' updates that may affect the performance of search engines.

2.1 Incremental Crawlers

2.1.1 *Web Dynamics.* Studies have shown that webpages change at varying rates from a few hours, a few weeks, to a year [Brewington and Cybenko 2000a; 2000a; Cho and Garcia-Molina 2000a; Ntoulas et al. 2004]. It is suggested that by following some refresh policies, the crawler should be able to maximize some pre-defined evaluation metrics by incrementally downloading and updating the webpages in the local repository [Cho and Garcia-Molina 2000a; Bouras et al. 2005] . In this section, we will give a systematic overview of the existing refresh policies and evaluation metrics.

2.1.2 *Refresh Policies.* Crawling techniques can be categorized into different groups. First, probabilistic models have been proposed to approximate the observed update history of a webpage and to predict its change in the future [Cho and Garcia-Molina 2003; Edwards et al. 2001]. Most of these change-frequency–based refresh policies assume that the webpages are modified by *Poisson processes* [Grimmett and Stirzaker 1992]. Edwards, et al., [Edwards et al. 2001] argue that the crawling frequency should be increased for frequently-changing pages. Several metrics have been proposed to guide the crawler to choose webpages for re-downloading, such as *freshness* and *age* [Cho and Garcia-Molina 2003]. Besides these metrics, relevance related metrics are also used, such as the *"embarrassment"* metric [Wolf et al. 2002], and the user-centric metric [Pandey and Olston 2005]. Olston and Pandey have proposed using *information longevity* [Olston and Pandey

2008] as another critical Web evolution factor. Updates on persistent content are worth detecting more than those on ephemeral content. However, a limitation common to all these approaches is that they need to gather enough accurate historical information related to each webpage before they can use this information to predict its future changes. In our approach, we do not assume the crawler has any existing historical information at the very beginning [Tan et al. 2007]. We made this choice because 1) it is not always realistic for a crawler to have a set of webpages with their change behaviors, which can be used as the training data set to predict the change behavior of other webpages, 2) new webpages are being created all the time; their change history is not known, yet, they must be crawled at some frequency.

Sampling-based approaches have been proposed to detect webpage changes by analyzing the update patterns of a small sample of webpages [Cho and Ntoulas 2002]. Their method samples and downloads data in the unit of a website. In our prior work [Tan et al. 2007], we have analyzed whether to fetch a set of pages within a distance of few hyperlinks, distance in the directory structure, and pages belonging to the same cluster having similar features and change patterns, can be more effective than the sampled-website-based crawling method [Cho and Ntoulas 2002].

Barbosa, et al., have proposed a classification-based algorithm [Barbosa et al. 2005] takes into account both the history and the content of pages to predict their behavior. Their experimental results based on real web data indicate that their solution had better performance than the change-frequency–based policy [Cho and Garcia-Molina 2000a]. Similar to our approach, they also extracted features from each webpage related to its update and used them to group webpages into different categories, each of which represents one specific change frequency. The key difference between their algorithm and ours is that they fix the number of classes and the change rates for all the classes in their classification process while we use a clustering approach with tunable number of clusters and associate clusters with dynamic change frequencies. Therefore, in our case each cluster can be recrawled every k days for different values of k, but with four classes they are limited to four values only. The value of k is dynamically set based on a set of features defined in our clustering algorithm. Moreover, we use a richer set of features which includes not only the static features but also the dynamic ones. Therefore, our clustering-based approach is more practical than the classification-based method. We will further compare our approach with Barbosa's with experiments in the future.

The basic idea of our clustering-based sampling method has been proposed in our preliminary work [Tan et al. 2007]. Here we provide evidence that there is a positive correlation between the contents and other features of a webpage and its change frequency and extend our preliminary approach by including ranking features (described in Section 3.3) in the clustering process. Also, our crawler assigns higher priority to re-download popular webpages. Experiment results in Section 6 show that the clustering-based sampling with ranking features outperforms that without ranking features in terms of the final ranked list quality.

Zheng, Dimitriev and Giles investigate how to select a set of seed pages for a crawler [Zheng et al. 2009]. Their technique deals with an orthogonal problem and can be incorporated in conjunction with our crawling policy.

2.2 Evaluation Metrics for Incremental Crawlers

Several evaluation metrics have been used to compare the performance of various refresh policies. They can be classified into three categories: staleness, change rate, and query-based metrics.

2.2.1 *Staleness.* The most straightforward way to measure the staleness of a set of webpages is to count the number of obsolete webpages in the local repository. Cho and Garcia-Molina [Cho and Garcia-Molina 2000b] define the freshness of webpage p_i at time t as

$$F(p_i;t) = \begin{cases} 1 & \text{if } p_i \text{ is up-to-date at time } t, \\ 0 & \text{otherwise.} \end{cases} \quad (1)$$

Here, up-to-date means that the locally stored image of the webpage is exactly the same as its counterpart at the source. Then the freshness of the entire local copy at time t is

$$F(U;t) = \frac{1}{|U|} \sum_{p_i \in U} F(p_i;t). \quad (2)$$

Here, U is the set of all locally stored webpages. For example, if we maintain 100 pages and if 80 pages are up-to-date at t, the freshness is 0.8.

Secondly, the age of a webpage p_i at time t is defined as

$$A(p_i;t) = \begin{cases} 0 \text{ if } p_i \text{ is up-to-date at time } t, \\ t - \text{modification time of } p_i, \text{ otherwise.} \end{cases} \quad (3)$$

And the age of the entire local copy is

$$A(U;t) = \frac{1}{|U|} \sum_{o_i \in U} A(o_i;t). \quad (4)$$

The age represents how old the local copy is. For example, if the source webpage changed one month ago, and if the crawler has not downloaded the page since then, the age of the local webpage is one month.

2.2.2 *Change Ratio.* To measure the refresh policy's ability to detect updates, Douglis et al. [Douglis et al. 1997] defined *ChangeRatio* as the fraction of downloaded and changed webpages D_i^c over the total number of downloaded webpages D_i in the ith download cycle. The *average ChangeRatio* $\overline{C}$ is the mean change-ratio over all download cycles.

The above general *ChangeRatio* metric treats every webpage as equally important. However, in practice some webpages may be more *popular* than others. Once their content have changed significantly, the crawler should detect such changes with a higher priority. Motivated by this observation, a *Weighted ChangeRatio* [Cho and Ntoulas 2002], C_i^w, is introduced by giving different weights w_p to the changed pages p. The weight w_ps can represent different types of importance of the pages, such as their PageRank. Formally C_i^w is defined as

$$C_i^w = \frac{1}{|D_i|} \sum_{p \in D_i} w_p \cdot \mathbf{I_1}(p) \quad (5)$$

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

$$\mathbf{I_1}(p) = \begin{cases} 1 & \text{if } p \in D_i^c, \\ 0 & \text{otherwise.} \end{cases} \quad (6)$$

2.2.3 *Query-based Metric.* Query-based metrics evaluate the quality of a search engine's local repository of webpages with respect to the quality of the users' experience when they use the search engine.

Wolf, et al., proposed a query-based metric to evaluate the *badness* of the search engine's local copies of the webpages, called *Embarrassment Level Metric* [Wolf et al. 2002]. It is defined as the probability that a client issues a query, clicks on a URL returned by the search engine, and then finds that the resulting page is irrelevant to the query. Under such a metric, the crawler's job is to minimize the *Embarrassment Level Metric*.

Another query-based metric is proposed by Pandey and Olston [Pandey and Olston 2005]. They argue that minimizing embarrassment cannot guarantee a good search experience. Consider that the search results returned by a search engine generate no embarrassment, while the low quality of the relevant documents can still substantially degrade the quality of the users' search experience. In this context, the average quality of search repository can be expressed as

$$Avg.\ Quality \propto \sum_{q}(freq_q \cdot \sum_{p_i \in U}(V(p_i)) \cdot (rel(q, p_i))).$$

Here, $V(p_i)$, the likelihood of viewing page p_i, is computed using a scoring function over the query q and the local repository, which could potentially be stale. The relevance $rel(q, p_i)$ is obtained using the same scoring function over q and the "live" copy of p_i. Both $freq_q$ and $V(p_i)$ can be obtained from the user logs from the search engine. The *user-centric metric* is the change in such an average quality before and after the crawler updates each p_i in the local repository. The proposed user-centric Web crawling is driven directly by this user-centric metric based on the search repository quality.

In Section 5, we propose several evaluation metrics that compare the search results of two different crawling policies. The crawling policy that generates the better search result should be preferred. How two ranked lists of search results can be compared in the absence of a gold standard that ranks and lists the ideal search results is an active area of research [Bar-Ilan et al. 2006]. In reality, because users seldom examine search results beyond the first page, the two top-10 search results resulting from the two policies should be compared. If needed, top-10 can be generalized to top-k.

3. IMPROVING CLUSTERING WEBPAGES USING FEATURES RELATED TO CHANGE PATTERNS

In this section, we outline how our algorithm clusters webpages. The clustering is based on three types of intrinsic features: static, dynamic, and user query log related. In our experiments (reported in the next section) we found that these features are correlated with the webpages' change patterns.

3.1 Learning Static Features

Recent studies have found that some characteristics of webpages are strongly correlated with their change frequencies. For example, Douglis, et al., [Douglis et al. 1997] noted that webpages, that are actively changing, are often larger in size and have more images than webpages that are not actively changing. Fetterly, et al., [Fetterly et al. 2003] observed that the top-level domains of the webpages are correlated with their change patterns. Based on the previous findings and our observations, the following static features are used in our clustering process.

Content features Our algorithm uses 17 content features in total. First, we construct a 10-dimensioned word-level vector to represent the content of the webpage. If we use all the words in all documents as features for clustering, this set of features is so large that it will dominate over other features. To avoid this, we shrink the feature set and construct a 10-dimensioned word-level feature vector. First, we cluster all the webpages based on the *TF.IDF* [Salton 1991] 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 dimensions are computed as follows. Suppose the webpage belongs to the cluster C_p, the feature's corresponding cluster label is C_f. Using the distance between the centroids of two clusters to represent the distance of two clusters, the value of the feature, $f_{C_p}^{C_f}$, is computed as follows. $f_{C_p}^{C_f} = 0$ for the webpages in $C_p = C_f$, $f_{C_p}^{C_f} = 1$ for the webpages in C_p which has the shortest distance to C_f, $f_{C_p}^{C_f} = 2$ for the webpages in C_p which has the second shortest distance to C_f, and so forth. To speed up the computation, we use the 1000 most frequently appearing words(after filtering the stop words) in all the webpages to calculate the distance between clusters.

In addition to the 10 content-related features, we consider seven other features: number of images, number of tables, number of non-markup words, file size in bytes (excluding the HTML tags), and file types: whether the file is an HTML file, a TXT file, or others (i.e. each file type is represented as one dimension in the feature vector and they are all in binary form, e.g. for an HTML file the corresponding dimension has a value of 1; otherwise, 0).

URL features Our algorithm uses 17 URL features in total. First, it processes the words in the URL in a manner similar to how the words on the webpage in order to keep the final feature vector at a reasonable size. Second, we compute the depth of the webpage in its domain, i.e. the number of "/" in the URL. We also consider the name of the top-level domain in the URL, and reserve six dimensions in the feature vector for the domains, .com, .edu, .gov, .org, .net., and .mil. Each of these six dimensions is a binary feature indicating whether the webpage belongs to the corresponding domain or not. We chose these six domains as the features because most webpages fall in these six domains. If a crawler is crawling a set of webpages where another domain, such as .biz, .uk, etc., is frequent, the feature set can be easily adapted to add or substitute the set of domain features.

Linkage features The algorithm uses four linkage features in total. It uses the PageRank [Page et al. 1999] of the webpage as an indicator for the *popularity* of a webpage. The assumption is that the more popular a webpage is, the more up-to-date it needs to be kept. Commercial search engines will have a more direct way to measure the popularity of a webpage by examining query logs and click logs. Our algorithm can then be easily adapted to use a measure based on the query/click logs instead of using PageRank as a surrogate. The algorithm also considers three other features related to linkage: number of incoming links within the local repository, number of outgoing links, and number of email addresses on the webpage.

3.2 Learning Dynamic Features

Ali and Williams [Ali and Williams 2003] have discussed that the significance of the past changes in document content is an effective measure for estimating whether the document will change again and should be re-crawled. We extend this idea to some of the aforementioned static features as follows. Note that it is impossible to calculate these dynamic features until after the second download cycle, due to the fact that they are based on the

comparison between two consecutive snapshots of a webpage.

Dynamic Content Features We use the following five features: Change in the webpage content, which is computed by the cosine similarity between the webpages' content in two consecutive download cycles; change in the number of images; change in the number of tables; change in the file size (in bytes, without the HTML tags); and change in the number of non-markup words on the webpage.

Dynamic Linkage Features We use the following four features: Change in PageRank, change in the number of incoming links within the local repository; change in the number of outgoing links, and change in the number of email addresses on the webpage.

We also include the webpage's change frequency as one dynamic feature for clustering. This feature can help the crawler adapt to the observed change history. When computing the change frequency for a webpage, our algorithm considers only the changes detected by the incremental crawler while ignoring those changes that are not detected.

3.3 Learning Ranking Features

The motivation of including the users' browsing behavior in the clustering process is that, the most popular webpages among users should be kept fresh [Pandey and Olston 2005]. To achieve this, we use 20 ranking features, which are generated as follows. First, based on the users' query log, the 20 most popular keywords are extracted. These keywords are denoted as $k_i, i \in [1,20]$. In our data set of 300,000 webpages, about 0.3% webpages contain at least one of these 20 popular keywords. Second, for each keyword, the similarities between it and the webpages are computed as *TFIDF* scores. The webpages are ranked according to their scores. Let *Ranking* be the webpage's position from the top in the ranked list. We define the ranking feature between webpage p_i and keyword k_j, $RF(p_i,k_j)$, as

$$RF(p_i,k_j) = \begin{cases} \frac{11-Ranking}{10} & \text{if } Ranking \leq 10, \\ 0 & \text{otherwise.} \end{cases} \quad (7)$$

Essentially, the documents that get higher ranking in terms of the popular keywords have higher scores for the ranking features.

Note that this idea is similar to that used in *user-centric crawling* [Pandey and Olston 2005]. They obtain from the user logs the probability of each webpage being viewed by users and use it to drive the incremental crawler. Also, similar technique has been used in *focused crawler*. A *focused crawler* [Chakrabarti et al. 1999] crawls webpages related to a particular topic quickly without having to explore every web page. Focused crawlers predict the importance of webpages and their relevance to a specific topic. The prediction is based on partial information, like, their backlinks, their anchor texts, and so on [de Bra et al. 1994; Herscovici et al. 1998; Mukherjea 2000; Menczer et al. 2001], because it is done before downloading the entire webpages. Popular queries are also used to bias the crawler towards pages on certain topics. We adopt the technique here for the incremental crawlers to detect webpages' updates that may affect the performance of search engines.

3.4 Clustering Webpages

With each webpage represented by a feature vector, we apply the *Repeated Bisection Clustering (RBC)* algorithm [Karypis and Han 2000] to construct hierarchical clusters. With the RBC algorithm, 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 that a criterion function (such as the one given below) is locally optimized. We do not choose the bottom-up clustering algorithm for our clustering process because with *RBC*, which is a top-down hierarchical clustering algorithm, we do not need initial seeds at the beginning.

An important question that needs to be answered before applying the clustering algorithm is how many clusters there are in the data set. As shown in Section 6.3.7, the coefficient of variation of the clusters decreases as we increase the number of clusters. As we increase the number of clusters, the webpages within the clusters are more similar to each other with respect to their change frequencies and thus the coefficient of variation decreases. Figure 8 shows that after about 100 clusters the coefficient of variation stops decreasing rapidly (in our dataset) and suggests that the number of clusters should be chosen carefully. Since clustering analysis is an unsupervised learning technique, we do not have prior knowledge about the number of clusters required. For different samples of the web, the number of clusters needed could be different. We can estimate this number by using the method of *v-fold cross-validation*. We divide the overall data set into a number of v folds. The clustering process is then applied to the data belonging to the $v-1$ folds (training samples) to get k clusters. The data in the vth fold (test samples) are put into k different clusters. We measure the quality of the clustering in the v^{th} fold using a measure which is based on how similar the data within the same cluster are. As is done in $v-fold$ cross validation, we repeat the experiment by selecting a different fold as the test fold and use the rest of the $v-1$ folds as the training folds until each fold has been used as a test fold once.

In general, we try a range of values for k while clustering. For each k, we apply the v-fold cross-validation method and observe a score function used in the clustering process. Specifically, we focus on the criterion function used by most vector-space based clustering algorithm. Let the k clusters denote as S_r, $r \in [1,k]$, and their centroids denote as C_r. If we use the cosine function to measure the similarity between a webpage p_i and a centroid C_r, the criterion function becomes the following:

$$\tau = \sum_{r=1}^{k} \sum_{p_i \in S_r} \cos(p_i, C_r). \tag{8}$$

where $cos(p_i, C_r)$ is obtained as

$$\cos(p_i, C_r) = \frac{p_i^T C_r}{\| p_i \| \| C_r \|}. \tag{9}$$

To maximize τ in (8) is to maximize the similarity between each webpage and the centroid of the cluster to which the webpage is assigned. Generally speaking, the larger k is, the higher τ is (see Figure 1). However, a very large k may impact the efficiency of the clustering algorithm. Thus we need to try different values of k and select the one that can balance both. The results of the v-fold cross-validation are best reviewed in a simple line graph. The line showing the score function should first quickly increase as the number of clusters increases, but then levels off. As such, the optimal number of clusters can be found at the point switching from increasing to leveling off.

4. SAMPLING IN CLUSTER LEVEL FOR UPDATE DETECTION

We present the cluster-based crawling algorithm in Algorithm 1. Let L_{i-1} denote the set of webpages in the local repository following $i-1$ download cycle.

Algorithm 1 Sampling and Downloading in the i^{th} Download Cycle

Input: L_{i-1}, number of clusters N, confidence level α, download resources R
Procedure:

```
1: Get N clusters Cs from webpages in L_{i-1}
   //Sampling process
2: for each cluster C do
3:    Compute C's centroid;
4:    Sort webpages in C based on their distance to centroid;
5:    repeat
6:       Select the webpage from the top of the sorted list for cluster C and remove the
         webpage from the list;
7:       Compute the samples' mean change frequency;
8:    until Mean change frequency is valid according to t-test at a confidence level α OR
      A fixed max-number of webpages have already been selected.
9: end for
   // Downloading process
10: Sort Cs based on their samples' mean change frequency;
11: repeat
12:    Download webpages in each C one by one from top to end in the sorted cluster list;
13: until R is used up
```

The algorithm first clusters the webpages based on features specified in the next section. Then, for each cluster, the algorithm incrementally adds webpages to a sample set. Next, the clusters are sorted based on their sample mean change frequency. Thereafter, the crawler downloads all webpages from each cluster visiting the clusters in their sorted order until all download resources for a cycle is exhausted.

4.1 Sampling webpages

While sampling webpages, our algorithm needs to address the following questions: which webpages should be selected from a cluster, and how many webpages from each cluster should be crawled. Webpages that are near a cluster's *centroid* are selected as *representative* webpages for the whole cluster. The distance between a webpage and the centroid is computed using the cosine function given by Equation 9.

A *sufficient* sample for a cluster is defined as a sample that produces a valid mean score for the cluster. This valid mean score should represent the population mean μ within $\mu \pm \delta$ at a confidence level (e.g. 80%) in a standard t-test for statistical significance. Here μ is the average change frequency of the samples (explained below). The sample's valid mean score is used to represent the mean score of the whole cluster. No sample of reasonable size may be able to produce a valid mean score. Therefore, we set an upper bound for the sample size in each cluster to make sure that sampling one cluster does not exhaust all

download resources. In our setting, we set the upper bound of the sample size as the ratio of the total number of download resources and the number of clusters.

4.2 Predicting Change Patterns

For each sampled webpage, we estimate its change frequency based on its update history (see below for details). Then for each cluster, we compute its *download probability* φ as the average change frequency of all sampled pages in that cluster. We model the update of the webpages using a *Poisson process* [Grimmett and Stirzaker 1992], which has been shown to be effective in experiments with real webpages [Brewington and Cybenko 2000b; Cho and Garcia-Molina 2003]. For our scenarios, it is reasonable to assume that the update of a webpage p follows a Poisson process with its own change rate λ_p. This means a webpage changes on its own, instead of depending on other pages. This assumption may not strictly hold but it has been proved to be a workable approximation for real applications. 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:

$$Pr\{T \leq t\} = \int_0^t f_p(t)dt = \int_0^t \lambda_p e^{-\lambda_p t} dt = 1 - e^{-\lambda_p t}. \tag{10}$$

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

$$\varphi = Pr\{T \leq 1\} = 1 - e^{-\lambda_p}. \tag{11}$$

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

$$\lambda_p = \frac{\sum_{i=1}^{n} w_i \cdot \mathbf{I}(L_i[p])}{n}, \sum_{1}^{n} w_i = 1, \tag{12}$$

where L_i denote the set of webpages in the local repository following i download cycle, and $\mathbf{I}(p_i)$ is an indicator function defined as follows:

$$\mathbf{I}(L_i[p]) = \begin{cases} 1 & \text{if } L_i[p] \neq L_{i-1}[p], \\ 0 & \text{otherwise.} \end{cases} \tag{13}$$

And w_i is the weight assigned to changes that occurred in different download cycles so that the distribution of change events is also taken into account.

We propose four settings for w_i that we evaluate empirically in Section 6:

(1) Non-adaptive (NAD) – $w_1 = w_2 = ... = w_n = \frac{1}{n}$. In this case, all change events have equal importance.

(2) Shortsighted adaptive (SAD) – $w_1 = w_2 = ... = w_{n-1} = 0, w_n = 1$. In this case, the crawler only considers the current change status of the webpage.

(3) Arithmetically adaptive (AAD) – $w_i = \frac{i}{\sum_{i=1}^{n} i}$.

(4) Geometrically adaptive (GAD) – $w_i = \frac{2^{i-1}}{\sum_{i=1}^{n} 2^{i-1}}$.

5. PROPOSED EVALUATION MODEL

Besides the existing evaluation methods in Section 2, we propose our evaluation model for a web search engine incremental crawler. We believe crawling policies should be evaluated based on which policy results in a better bottom-line, i.e., in better search results. Therefore, in our evaluation, we focus on the quality of the final rankings returned to the users of the search engines. The difference between our metric and the existing user-centric metrics is that we measure the quality of the returned webpages according to the users' queries directly, instead of evaluating the quality of all the webpages in the local repository. That means, we consider the subset of the local repository that can be used to answer user queries. For those webpages not of interest to the users, their updates have negligible importance to the search engine and its crawlers; our evaluation process takes this into account. There are several factors that govern the quality of the ranking generated from the search engine, such as coverage, spam detection, ranking function, and so on. Among them two essential factors are: (1) effectiveness of the indexing and ranking function, and (2) freshness of the local repository [Pandey and Olston 2005]. We fix the first factor, namely the indexing and the ranking function, and then evaluate the ranked results returned to the users. We use the following evaluation metrics to measure the quality of the returned ranking: the top-*k mean precision (MP)* and the top-*k freshness*.

5.1 Ranking Generation

To generate the ranking on the repositories, we apply two scoring functions: (1) the TF-IDF metric [Salton 1991] and (2) the PageRank [Page et al. 1999]; both are widely adopted in the information retrieval community. We also combine these two scoring functions to generate a final ranking score for each webpage as follows.

$$score = \alpha \cdot TF \cdot IDF + \beta \cdot PageRank \tag{14}$$

where $\alpha + \beta = 1$. We set $\alpha = \beta = 0.5$ in our experiments.

5.2 Evaluation Metrics

We evaluate our crawling policies using two metrics in our experiments:

(1) *Top-k Freshness*: This metric reflects the percentage of the results in the top-k ranking that are up-to-date with the real copy on the Web. This is based on the simplifying assumption that all the top-k ranked pages are important pages for the query. A higher freshness indicates that more of those webpages are fresh than in the case where the freshness is low. An user is less likely to view a lowly ranked result in the returned result list. Thus, we assign different weights to the freshness of each result according to its ranked order. The weight for each result, $W(r)$, is approximated as the likelihood that user will scan the result in a ranking. Analysis of AltaVista usage logs [Lempel and Moran 2003; Cho and Roy 2004] have revealed that $W(r)$ can be computed as:

$$W(r) = c \cdot r^{-3/2} \tag{15}$$

where c is a normalization constant and r is the rank of the result.
Formally, given a ranking R_k with k results, we define *top-k freshness*, $freshness@k$, as:

$$freshness@k = \frac{1}{k} \sum_{p_r \in R_k} W(r) \cdot F(p_r; t) \tag{16}$$

where $F(p_r;t)$ is computed as indicated in Formula (1).

(2) *Top-k MP (Mean Precision)*:

Given a test set of queries Q and the top-k ranking $Rank_k$ generated from each query, the top-k MP, $MP@k$, can be denoted as:

$$MP@k = \frac{1}{|Q|}\sum_{q\in Q} P_q \tag{17}$$

$$P_q = \sum_{r=1}^{k} rel(p_r)\cdot precision@r$$

$$precision@r = \frac{1}{r}\sum_{r=1}^{k} rel(p_r) \tag{18}$$

Typically, we consider only the top-10 results in the returned ranking because the probability that a user clicks on a result at rank 10.000 is virtually zero even if the result is relevant to the query [Silverstein et al. 1999]. We judged the relevance of each result manually in the returned ranking to create a gold standard. Note that there are two steps to simulate the users' browsing experience. That is, we need to check the relevance of 1) the results' title and summary – which are obtained from the local copies of the corresponding webpages in the search engine – in the returned ranking, and then 2) the current webpage to which the result is linking, which is the real copy on the Web. If the title and summary of the result in the first step are judged to be irrelevant, it is not necessary to execute the second step. By executing the above two steps, we simulate the users' search behavior and get a reasonably accurate judgment of the ranking quality. For example, when a user inputs a faculty member's name to the search engine to get his homepage, it still works if there are updates on the person's homepage while the crawler does not catch such an update. This is because, when the search engine returns the title of the webpage, a snippet and the URL of a person's stale homepage in its local repository, the user can decide whether this is the correct person the user is searching for by examining the URL, title, and the snippet. When the user clicks on the link of the result, the user will be directed to the updated version of the homepage. This is called the "lucky" case for the search engine. However, in another scenario, the change on the webpages could be so significant that when the user clicks on the link the user is given a totally irrelevant webpage or even a dead link. Wolf, et al. [Wolf et al. 2002] discussed this "embarrassing" case and a crawler should try to avoid this case. An effective evaluation process should be able to test whether the crawler can catch the changes that may lead to these cases. In our method, by including the top-*k freshness* and top-*k MP*, we can put these cases under evaluation.

6. EVALUATION AND DISCUSSION

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

6.1 Data Collection

We obtained a collection of real URLs whose change history we use from the WebArchive[2]. We implemented a spider to crawl the Internet Archive[3], which has archived more than 55 billion webpages since 1996. We got the historical snapshots of these URLs. Excluding those webpages from the WebArchive that were not available in the Internet Archive, we eventually constructed a dataset containing approximately 300,000 distinct URLs that belong to more than 210 websites. For each URL, we downloaded from the Internet Archive their historical snapshots and their change history for a period of one year. The distribution by different top-level domains is shown in Table I.

domain	total	sites	avg urls / 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)		

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

As far as we know the WebArchive does not have change frequency prediction. It uses hundreds of x86 machines to download and hold the data locally. For our experiments, we have more than 900M snapshots of 300k webpages in one year, which we think is enough to cover most of the changes.

6.2 Evaluation Metrics

We used the following three metrics discussed above to evaluate our proposal: *ChangeRatio*, *top-k freshness*, and *top-k MP*. For the first metric, the webpage's local copy is compared with the historical snapshot with the same time-stamp from the Internet Archive to check whether it has changed or not. For the i-th download cycle, we measure the per-download-cycle *ChangeRatio* C_i. The average *ChangeRatio* $\overline{C}$ is the mean C_i over all download cycles.

For the *top-k freshness*, and *top-k MP* metrics, we generate rankings of search results based on user queries. We randomly selected 100 queries from the query log of the AOL search engine[4]. We use Equation 14 to obtain the rankings. The popular Lucene software[5] is used to generate the TF-IDF value. MatLab codes for PageRank computation are downloaded from The MathWorks[6] for our evaluation process.

[2]http://webarchive.cs.ucla.edu/
[3]http://www.archive.org/
[4]Chronicle of AOL Search Query Log Release Incident, http://sifaka.cs.uiuc.edu/xshen/aol/aol_querylog.html.
[5]Apache Lucene, http://lucene.apache.org/.
[6]The MathWorks - Numerical Computing with MATLAB, http://www.mathworks.com/moler/ncmfilelist.html.

6.3 Results and Discussion

We let the crawler maintain the data set in Section 6.1 as its local repository. To verify the effectiveness of our sampling approach, we applied our sampling approaches with different download cycles (1 week, 2 weeks, 4 weeks, and 8 weeks) and download resources (25K, 50K, and 100K). Note that our results are dependent upon the sampling frequency of the Internet Archive. That is, if the pages change at a faster rate than their crawl frequency of the Internet Archive crawler, i.e., there is more than one change between two consecutive crawls, then only one change is detected. When a webpage is crawled, the corresponding historical snapshot with the same time-stamp from the Wayback Machine is checked to determine whether the webpage has changed or not; if there is no exact match with respect to the timestamp in the archive, the version with the closest time-stamp is checked instead.

6.3.1 *Estimating Number of Clusters.* For the cluster based sampling approach, we employ the *10-fold cross-validation* method to cluster the content of webpages. Figure 1 shows the values of τ under different numbers of clusters, k. The value of τ goes up as k increases. Its gradient is sharp when k is smaller. For URL clustering, there is a turning point for τ when k reaches 100: as k passes 100, τ increases at a much slower rate than before. This indicates that once $k > 100$, increasing k does not have a significant impact on the clustering output. A similar trend also appears in content-based clustering. Therefore, we use $k = 100$ for all the clustering processes in the following experiments.

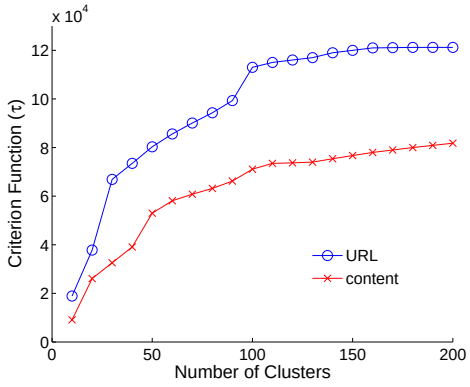


Fig. 1. Cluster quality for URL and content clustering over different numbers of clusters. $k = 100$ is a good setting because it can balance the quality of the clusters and the overhead for clustering.

We now take a close look at the number of webpages in each cluster. The comparison between our cluster-level sampling and the site-level sampling is shown in Table II and Figure 2. These statistics show that the distribution of cluster sizes is more even than that of website sizes. Most websites have less than 2k webpages, while most clusters have 1k–3k webpages. In the following experiments, we will show that our cluster-level sampling is more effective in grouping webpages with similar change patterns than the website-level sampling strategy.

sampling unit	number	min	max	mean
website	212	1	3313	1343
cluster	100	798	6604	2847

Table II. Some statistics of webpages in different sampling units.

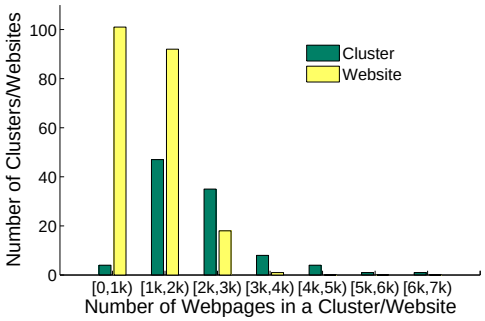


Fig. 2. The distribution of the sizes of clusters and websites.

6.3.2 *Estimating Number of Dimensions for Word-level Clustering.* Among the content features, we propose using 10 dimensions as a word-level vector to represent the content of the webpage. The choice of the number of dimensions, N_{dim}, for the word-level vector is critical here for the clustering process. If N_{dim} is too large, the content related features will dominate the other features. Otherwise, if N_{dim} is too small, the contribution of the content features is so little that the update detection performance is impacted. In this experiment, we tune the number of dimensions from 5 to 50 and test the impact of this tuning on the *ChangeRatio*. We set the number of clusters to 100, the download cycle to one month, the change frequency to the non-adaptive one, and confidence level at 80%.

Figure 3 shows that the average *ChangeRatio* is the lowest when N_{dim} equals to 5. This indicates that the webpages' content features are related to the webpages' change patterns. By using the content features, we can group together webpages with similar change patterns. The average *ChangeRatio* increases as N_{dim} does until it equals to 50. The average *ChangeRatio*s with N_{dim} equal to 10, 20, and 30 have no significant difference. Clustering in a higher dimensional space is less efficient than clustering in a lower dimensional space. Hence, we choose the number of dimensions as 10 in the following experiments.

6.3.3 *Evaluating Feature Effectiveness.* We study the effectiveness of the following feature sets: (1) URL: all words in the webpages' URLs; (2) Content: all words on the webpages; (3) URL + content; (4) Static: all the static features listed in Section 3.1 except the 10 features related to URL words and the 10 features related to webpage content; there are 28 features in total; and, (5) Dynamic: the 10 dynamic features listed in Section 3.2 (6) Static + Dynamic; (7) Ranking features: the 20 ranking features generated from the query log, and, (8) all the 68 features we have discussed thus far.

We show the comparison results in Figure 4. For this set of experiments, the download cycle is set to one month and the download resources R is set incrementally from 25K, 50K, to 100K. For the sampling process, we set the change frequency to be the non-adaptive one

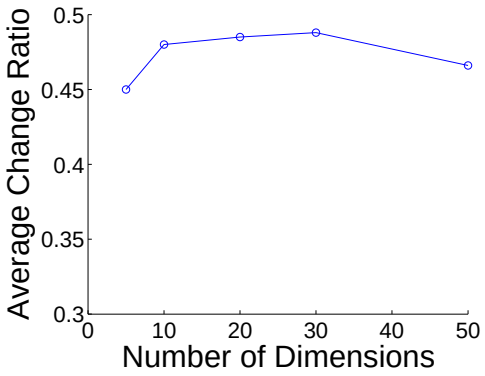


Fig. 3. Effectiveness to the ChangeRatio of different numbers of dimensions used in the word-level clustering.

and confidence level at 80%. The figure gives us the following illustrations: First, when download resources are scarce, the choice of feature sets for clustering has a huge impact on the average ChangeRatio of the clusters. While the static + dynamic features set and the combined feature set perform the best, using only the URL words for clustering has the lowest ChangeRatio because it ignores other important features. Second, when the download resource is abundant, the performance of all feature settings become similar.

Using words in both URLs and content for clustering has similar performance as using words in the content only. This is because, compared with the content of webpages, URLs are relative shorter and much less informative. The performance of using static and dynamic features for clustering is a little bit worse than using all features, which includes the ranking features. We use all features but the ranking features for clustering in the following experiments. In Section 6.3.6 we further study the effectiveness of the ranking features using other metrics besides average *ChangeRatio*, such as $freshness$@10 and *MP*@10.

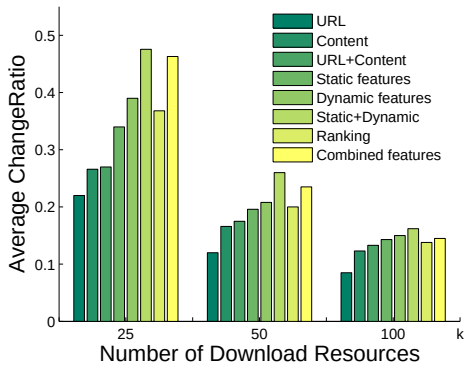


Fig. 4. Comparison of different clustering feature sets. Dynamic features can predict change patterns better than static features.

6.3.4 *Selecting w_i for Change History.* We experimented with the four different settings for w_i discussed in Section 4. For this set of experiments, the download cycle is set to be two weeks and the download resources R is set to be 25K. Figure 5 shows the results from this experiment. At the beginning, the policy with shortsighted-adaptive w_i, which focuses on only the current change status, outperforms all others because the others (except the non-adaptive policy) do not have any knowledge about the change history. On the contrary, using the non-adaptive w_i has the lowest *ChangeRatio* at the beginning and then it gradually outperforms the shortsighted-adaptive one. The other two adaptive policies have very similar performances, and eventually their *ChangeRatio*s are the highest among all. In this figure, there are variations across different download cycles because most webpages have similar change patterns wherein they change at the end of the week or at the end of the month, etc.

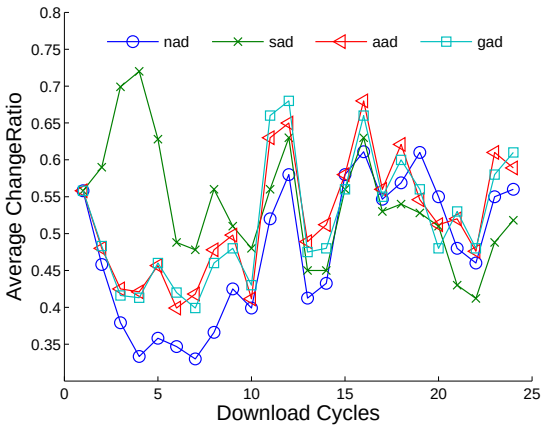


Fig. 5. Comparison of adaptive and non-adaptive w_i . Using adaptive w_i can improve the average *ChangeRatio* in the long term.

6.3.5 *Comparison with Existing Methods.* We compared our algorithm with the following downloading policies proposed in prior work.

—Random Node Sampling (Rand): The crawler uniformly re-downloads at random webpages in each download cycle [Leskovec and Faloutsos 2006]. In this case, every webpage has the equal probability to be downloaded in each download cycle.

—Round Robin (RR): The crawler downloads webpages in a round-robin fashion in each download cycle. This method is adopted by many systems [Brin and Page 1998; Heydon and Najork 1999] currently because it is simple to implement.

—Change-Frequency–Based policy (Freq): The crawler selects webpages to re-download based on their past change frequencies.

—Greedy Site-Level Sampling (SLS): We set the sample size for each site as the optimal one, $\sqrt{Nr}$ as proposed by Cho, et al. [Cho and Ntoulas 2002] 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.

—Directory-based Downloading Policy (DB): When the algorithm identifies that a sampled webpage has been updated, it crawls all webpages within the same directory of the sampled webpage [Tan et al. 2007].

—Link-based Downloading Policy (LB): When the algorithm identifies that a sampled webpage has been updated, it crawls all webpages within 3 hops of the sampled webpage [Tan et al. 2007].

We also compare all the methods with an "ideal" scenario (Ideal), in which the crawler is so powerful that it has no resource limitation and it can keep every webpage in its local repository completely synchronized with the real copy on the Web at all times.

In our cluster level sampling algorithm (CLS), we show the average *ChangeRatio* of four different settings for w_i (*nad*, *sad*, *aad*, and *gad*), and set the confidence level to 80%. The results shown in Figure 6 is averaged over 24 download cycles during 1 year. In our dataset, the Change-Frequency based policy never outperforms the clustering-based policies after 24 download cycles so we do not show the evolution of each metric here.

Figure 6(a) gives the *ChangeRatio* for all download policies. The performance of our cluster-level sampling is better than that of the site-level, directory-level, and link-level sampling. This is because CLS groups webpages based on the features most relevant to their update patterns, while in the other three sampling methods, webpages are grouped based only on some of the useful features, such as top-level domains (SLS), directory structures (DB), and linkage relations (LB). Thus, a crawler implementing the CLS scheme can easily capture webpages with similar change patterns. In this set of experiments, using *sad* has higher average *ChangeRatio* than *nad*, *aad*, and *gad*. However, as we observe in Figure 5, the two adaptive strategies *aad* and *gad* outperform *sad* in the long term.

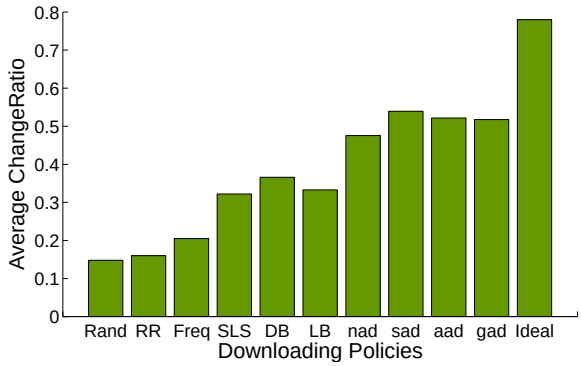
Figure 6(b) shows the $freshness@10$ for all download policies. The performance of CLS is better than that of the SLS, DB, and LB. CLS uses both static and dynamic features for clustering webpages. Our approach is good at catching the significant changes that impact the search engine's final ranking. In terms of $freshness@10$, there is no significant difference between the three adaptive download policies, *sad*, *aad*, and *gad*.

Figure 6(c) gives the $MP@10$ for all download policies. Keeping the local repository fresh can lead to a significantly higher *MP* than others. When the search engine's index is built on top of the most up-to-date webpages, users can potentially get more accurate answers to their queries than when the search engine index has stale webpages. For the other download policies, their *MP*s are very similar as shown in the figure.

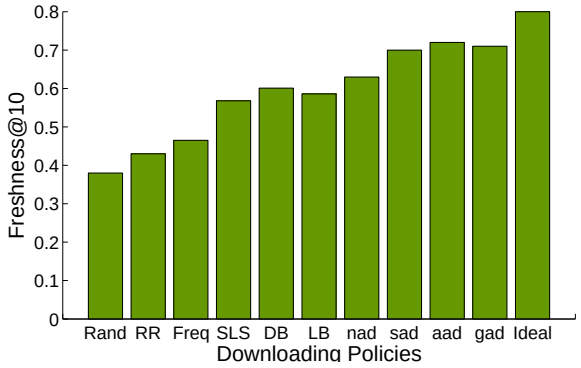
To check the statistical significance of the effectiveness of the download policies, we apply a *two-sample hypothesis testing about means* [Schelfler 1988]. Specifically, we check whether there is a significant *statistical difference* between the means values of the download policies.

From all the t-values for the comparisons between pairs of samples, we conclude that all the t-values are larger than the critical value of 1.960 ($\alpha = 0.05$). This result implies that we are able to reject the null hypothesis that the two methods have the same values at the 0.05 significance level. That is, with 97.5% confidence, *CLS-nad*, *CLS-sad*, *CLS-aad*, and *CLS-gad* achieve significantly better *ChangeRatio*, $freshness@10$, and *MP*s than the other existing approaches, such as *Rand*, *RR*, *Freq*, and *SLS*.

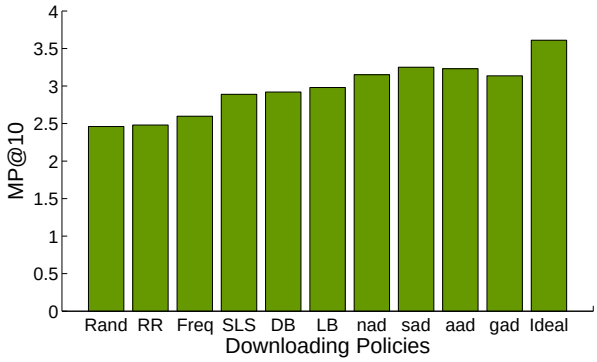
We finally analyze the comparison between the *Ideal* case with other methods. Figure 6 shows that the *Ideal* case has the best performance in terms of the three metrics. More specifically, we compare the difference between *Ideal* and the other methods in each



(a) *ChangeRatio*



(b) *Freshness*@10



(c) *MP*@10

Fig. 6. Comparison with four published update-detection methods averaging 24 download cycles over one year. Our cluster-level sampling algorithm significantly outperforms the site-level sampling method in terms of *ChangeRatio*, *Freshness*@10, and *MP*@10.

metric. Clearly, the differences in terms of *freshness*@10 and *MP*@10 are much less significant than that in average *ChangeRatio*. Our experimental results indicate that the CLS method is more effective than other crawling methods in maintaining the quality of final ranking, and satisfying the search engine users' queries.

6.3.6 *Investigating the Effectiveness of Ranking Features.* We have shown in Section 6.3.3 that including ranking features (RFs) in the clustering process does not necessarily improve the performance of the update detection in terms of average *ChangeRatio*. Now, we study the effectiveness of ranking features in terms of other two metrics, *freshness*-@10 and *MP*@10. We use the AOL query logs that contain queries posed over one year. First, we clean the noisy queries, i.e., random queries without any meaning. For example, queries like "f" and "asdf" are removed, while meaningful queries like "classic american homes" are included in our study. We then separate the queries according to the download cycle. After that, we compute the frequency of each keyword, and get the 20 keywords that are most popular in the queries. Note that the queries used here are different from those 100 queries used for the evaluation.

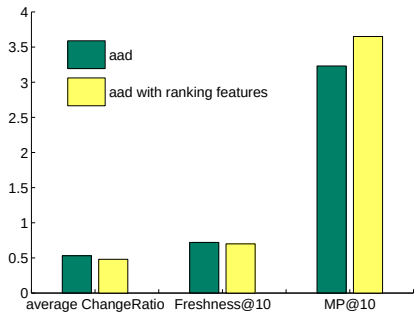


Fig. 7. Comparison between the clustering-based sampling approach with and without ranking features in terms of average *ChangeRatio*, *Freshness*@10, and *MP*@10.

Figure 7 shows that including the RFs has similar performance in terms of average *ChangeRatio* and *Freshness*@10 but has improvement in terms of *MP*@10. Hypothesis tests are applied and results show that the mean values of *MP*@10 for CLS with and without RFs are significantly different. This is because by including RFs, the CLS is biased towards webpages that are popular among the users. Thus the freshness and quality of those popular webpages are improved.

6.3.7 *Investigating Webpages' Average Change Intervals in Clusters.* To further illustrate that our clustering method effectively groups webpages based on similar change behaviors, in this experiment, we tested the change of webpages within one cluster. Based on the change history we crawled from the WebArchive, we calculated the average change interval for each URL. For example, if a particular URL changed 10 times in a one year period (365 days), its average change interval is 365/10=36.5 days. Then for each cluster we calculated the mean change interval and its standard deviation. The coefficient of

variation is as follows:

$$Coefficient\ of\ variation = \frac{standard\ deviation}{mean}. \tag{19}$$

Given a number of clusters, we got the coefficient of variation for each cluster and then computed the average coefficient of variation for the whole set of clusters. Note that a low average coefficient of variation means that on average, each cluster contains webpages with similar change intervals. That is, the clustering results actually align with the real change intervals of the webpages. We compared our cluster level method with the change frequency based method (Freq) and the site level change detection method (SLS). The experiment results are shown in Figure 8. The number of clusters is plotted on the horizontal axis. In this figure, the vertical axis shows the average coefficient of variation corresponding to different number of clusters. For the site level method, the number of groups is fixed to be 213 so its average coefficient of variation is fixed. For the change frequency based method, the webpages are grouped based on their historical change frequencies.

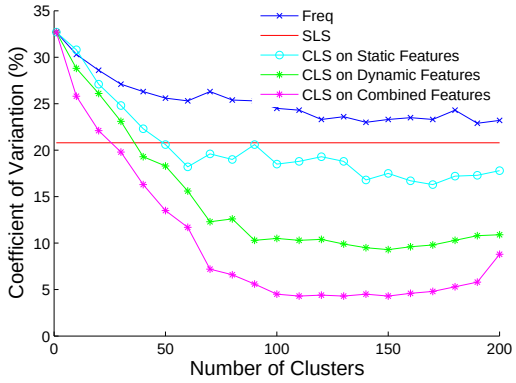


Fig. 8. Comparison of average change interval distributions of different crawling policies.

From Figure 8 we have the following observations. First, our experiment demonstrates that, as the number of clusters increases, the average coefficient of variation decreases. At the beginning when there is only one cluster, the average coefficient of variation is the highest because webpages with very diverse change history are contained within the same group. After that, as the set of webpages are separated into different groups based on the change relative features we defined, webpages with similar change history are grouped together. This shows the effectiveness of our clustering method. Second, our method outperforms both Freq and SLS as evidenced by its lower average coefficient of variation. In the figure, we can see that when there are more than 25 clusters, we can obtain a lower average coefficient of variation than SLS. Third, comparing different cases of our own cluster level method using different clustering features, we can see from the figure that the one with combined features outperforms the one with static or dynamic features, because by using static and dynamic features, we can better predict changes in webpages than just using a subset of features. We can also see that the one with dynamic features outperforms the one with static features, because dynamic features are directly related to webpage changes and

thus do better than the static features. Therefore, based on the above experimental results, we can conclude that the combined feature set does a better job in detecting changes. This is consistent with our previous finding in Section 6.3.3.

7. CONCLUSION

We propose an algorithm for crawling webpages with limited resources. First, existing webpages are divided into 100 clusters using both static and dynamic features. Static features are extracted from the content of a webpage, the webpage's URL and hyperlinks. Dynamic features are extracted from changes to webpage content, hyperlinks, pagerank, etc. The crawler fetches a sample of webpages from a cluster to check if the pages have changed since their last download. If a significant number of pages in a cluster have changed, the other webpages in the cluster are also downloaded. Clusters of webpages have different change frequencies and based on its change history, different clusters are crawled at different frequencies by our crawler. We demonstrated the superiority of our algorithm over various existing sampling-based webpage update detection algorithms.

This paper provides preliminary evidence that a clustering-based approach may improve web crawlers significantly. Further investigation is necessary to show the effectiveness of our methods in diverse environments. For example, we used a combination of PageRank and TF-IDF as our ranking function. Will the results hold if we use different ranking functions? How do the results vary if we use different clustering algorithms? Among different clustering algorithms, which ones are best for use with crawlers? Also, in a real system, we expect a clustering-based approach to be used in conjunction with other effective heuristics and techniques. While we focused on the effectiveness of our algorithms in this work, we intend to explore the runtime efficiency and the scalability of these clustering-based algorithms in the future. Finally, experiments will be done to directly compare our method with the classification-based method proposed by Barbosa et al. [Barbosa et al. 2005].

ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grant Nos. 0535656, and 0845487. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

ALI, H. AND WILLIAMS, H. E. 2003. What's changed? measuring document change in Web crawling for search engines. In *SPIRE '03: Proceedings of the 10th International Symposium on String Processing and Information Retrieval*. 28–42.

BAR-ILAN, J., MAT-HASSAN, M., AND LEVENE, M. 2006. Methods for comparing rankings of search engine results. *Computer Networks 50*, 10, 1448–1463.

BARBOSA, L., SALGADO, A. C., DE CARVALHO, F., ROBIN, J., AND FREIRE, J. 2005. Looking at both the present and the past to efficiently update replicas of Web content. In *WIDM '05: Proceedings of the 7th annual ACM international workshop on Web information and data management*. ACM, New York, NY, USA, 75–80.

BOURAS, C., POULOPOULOS, V., AND THANOU, A. 2005. Creating a polite adaptive and selective incremental crawler. In *IADIS International Conference WWW/INTERNET 2005*. 307 – 314.

BREWINGTON, B. E. AND CYBENKO, G. 2000a. How dynamic is the Web? *Computer Network 33*, 1-6, 257–276.

BREWINGTON, B. E. AND CYBENKO, G. 2000b. Keeping up with the changing Web. *Computer 33*, 5, 52–58.

BRIN, S. AND PAGE, L. 1998. The anatomy of a large-scale hypertextual Web search engine. *Computer Network ISDN System 30*, 1-7, 107–117.

CHAKRABARTI, S., VAN DEN BERG, M., AND DOM, B. 1999. Focused crawling: A new approach to topic-specific Web resource discovery. *Computer Networks 31*, 1623.

CHO, J. AND GARCIA-MOLINA, H. 2000a. The evolution of the Web and implications for an incremental crawler. In *VLDB '00: Proceedings of the 26th International Conference on Very Large Data Bases*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 200–209.

CHO, J. AND GARCIA-MOLINA, H. 2000b. Synchronizing a database to improve freshness. *SIGMOD Record 29*, 2, 117–128.

CHO, J. AND GARCIA-MOLINA, H. 2003. Effective page refresh policies for Web crawlers. *ACM Transactions on Database Systems 28*, 4, 390–426.

CHO, J. AND NTOULAS, A. 2002. Effective change detection using sampling. In *VLDB '02: Proceedings of the 28th international conference on Very Large Data Bases*. VLDB Endowment, 514–525.

CHO, J. AND ROY, S. 2004. Impact of search engines on page popularity. In *WWW '04: Proceedings of the 13th international conference on World Wide Web*. ACM, New York, NY, USA, 20–29.

DE BRA, P., JAN HOUBEN, G., KORNATZKY, Y., AND POST, R. 1994. Information retrieval in distributed hypertexts. In *RIAO '94: Proceedings of the 1st Recherche dInformations Assistee par Ordinateur Conference*. 481–491.

DOUGLIS, F., FELDMANN, A., KRISHNAMURTHY, B., AND MOGUL, J. 1997. Rate of change and other metrics: a live study of the World Wide Web. In *USITS'97: Proceedings of the USENIX Symposium on Internet Technologies and Systems on USENIX Symposium on Internet Technologies and Systems*. USENIX Association, Berkeley, CA, USA, 147–158.

EDWARDS, J., MCCURLEY, K., AND TOMLIN, J. 2001. An adaptive model for optimizing performance of an incremental Web crawler. In *WWW '01: Proceedings of the 10th international conference on World Wide Web*. ACM, New York, NY, USA, 106–113.

FETTERLY, D., MANASSE, M., NAJORK, M., AND WIENER, J. 2003. A large-scale study of the evolution of Web pages. In *WWW '03: Proceedings of the 12th international conference on World Wide Web*. ACM, New York, NY, USA, 669–678.

GRIMMETT, G. AND STIRZAKER, D. 1992. *Probability and Random Processes, 2nd ed*. Oxford University Press, Oxford, England.

HERSCOVICI, M., JACOVI, M., MAAREK, Y. S., PELLEG, D., SHTALHAIM, M., AND UR, S. 1998. The shark-search algorithm. an application: Tailored web site mapping. *Computer Networks 30*, 317.

HEYDON, A. AND NAJORK, M. 1999. Mercator: A scalable, extensible Web crawler. *World Wide Web 2*, 4, 219–229.

KARYPIS, G. AND HAN, E.-H. S. 2000. Fast supervised dimensionality reduction algorithm with applications to document categorization & retrieval. In *CIKM '00: Proceedings of the ninth international conference on Information and knowledge management*. ACM, New York, NY, USA, 12–19.

LEMPEL, R. AND MORAN, S. 2003. Predictive caching and prefetching of query results in search engines. In *WWW '03: Proceedings of the 12th international conference on World Wide Web*. ACM, New York, NY, USA, 19–28.

LESKOVEC, J. AND FALOUTSOS, C. 2006. Sampling from large graphs. In *KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, New York, NY, USA, 631–636.

LIU, B., ABDULLA, G., JOHNSON, T., AND FOX, E. A. 1998. Web response time and proxy caching. In *WebNet '98: Proceedings of The 3rd World Conference of the WWW, Internet, and Intranet*. 92–97.

MENCZER, F., PANT, G., SRINIVASAN, P., AND RUIZ, M. E. 2001. Evaluating topic-driven Web crawlers. In *SIGIR '01: Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, New York, NY, USA, 241–249.

MUKHERJEA, S. 2000. WTMS: a system for collecting and analyzing topic-specific Web information. *Computer Networks 33*, 1-6, 457–471.

NTOULAS, A., CHO, J., AND OLSTON, C. 2004. What's new on the Web?: the evolution of the web from a search engine perspective. In *WWW '04: Proceedings of the 13th international conference on World Wide Web*. ACM, New York, NY, USA, 1–12.

OLSTON, C. AND PANDEY, S. 2008. Recrawl scheduling based on information longevity. In *WWW '08: Proceeding of the 17th international conference on World Wide Web*. ACM, New York, NY, USA, 437–446.

PAGE, L., BRIN, S., MOTWANI, R., AND WINOGRAD, T. 1999. The pagerank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford InfoLab. November. Previous number = SIDL-WP-1999-0120.

PANDEY, S. AND OLSTON, C. 2005. User-centric Web crawling. In *WWW '05: Proceedings of the 14th international conference on World Wide Web*. ACM, New York, NY, USA, 401–411.

PANT, G. AND SRINIVASAN, P. 2009. Predicting web page status. *Information Systems Research*.

SALTON, G. 1991. Developments in automatic text retrieval. *Science 253*, 974–979.

SCHELFLER, W. 1988. *Statistics: Concepts and Applications*. Benjamin/Cummings Publishing Company.

SILVERSTEIN, C., MARAIS, H., HENZINGER, M., AND MORICZ, M. 1999. Analysis of a very large Web search engine query log. *SIGIR Forum 33*, 1, 6–12.

TAN, Q., MITRA, P., AND GILES, C. L. 2007. Designing clustering-based web crawling policies for search engine crawlers. In *CIKM '07: Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*. ACM, New York, NY, USA, 535–544.

TAN, Q., ZHUANG, Z., MITRA, P., AND GILES, C. L. 2007. Efficiently detecting webpage updates using samples. In *ICWE '07: Proceedings of International Conference of Web Engineering*. 285–300.

WOLF, J. L., SQUILLANTE, M. S., YU, P. S., SETHURAMAN, J., AND OZSEN, L. 2002. Optimal crawling strategies for web search engines. In *WWW '02: Proceedings of the 11th international conference on World Wide Web*. ACM, New York, NY, USA, 136–147.

ZHENG, S., DMITRIEV, P., AND GILES, C. L. 2009. Graph based crawler seed selection. In *WWW '09: Proceedings of the 18th international conference on World wide web*. ACM, New York, NY, USA, 1089–1090.

Received October 2008; revised October 2009; revised January 2010; accepted March 2010