Looking at both the Present and the Past to Efficiently Update Replicas of Web Content

Luciano Barbosa[†] Ana Carolina Salgado[‡] Francisco de Carvalho[‡]
Jacques Robin[‡] Juliana Freire[†]
[†]School of Computing
University of Utah
{lbarbosa,juliana}@cs.utah.edu
[‡]Centro de Informatica
Universidade Federal de Pernambuco
{acs,fatc,jr}@cin.ufpe.br

ABSTRACT

Since Web sites are autonomous and independently updated, applications that keep replicas of Web data, such as Web warehouses and search engines, must periodically poll the sites and check for changes. Since this is a resource-intensive task, in order to keep the copies up-to-date, it is important to devise efficient update schedules that adapt to the change rate of the pages and avoid visiting pages not modified since the last visit. In this paper, we propose a new approach that learns to predict the change behavior of Web pages based both on the static features and change history of pages, and refreshes the copies accordingly. Experiments using real-world data show that our technique leads to substantial performance improvements compared to previously proposed approaches.

Categories and Subject Descriptors

H.3.5 [**Online Information Services**]: Web-based services

General Terms

Algorithms, Design, Experimentation

Keywords

indexing update, machine learning, update policy

1. INTRODUCTION

Several applications, including search engines and cache servers, keep replicas or summaries (e.g., indexes) of Web content [17, 11]. Since Web sites are autonomous and independently updated, these applications need to periodically poll the sites and check for changes. A common (and simple) mechanism for refreshing the replicas is to revisit all documents stored at the same frequency, in a round-robin fashion. This is a very costly operation, and for applications that keep a large number of replicas, it may not be possible to refresh all pages at a reasonable interval so as to guarantee freshness. Take for example search engines: studies show that not only do search engines return a high-percentage of broken (obsolete) links, but it may also take them several months to index new pages [13, 14].

Since different kinds of content are modified at different rates, it is wasteful to revisit all pages at uniform intervals. A more efficient strategy is to adopt a non-uniform refresh policy and revisit Web pages according to their change rate, i.e., pages that are modified more often should be visited more frequently than those with smaller change frequency. Techniques have been proposed that use the observed change history of a page to predict its change rate [4, 9]. Although the historical behavior of a page is a good indicator of its future behavior, when a page is first visited, no information is available information about its change pattern. Thus, these approaches have an important limitation: since it takes time to *learn* the change pattern of a page, they fall back to the uniform update policy while the history is constructed. As a result, newly acquired pages can quickly become stale.

In this paper we address this problem by taking into account both the history and the content of pages in order to predict their behavior. When a page is first accessed, the prediction is based on its content. For subsequent accesses, we devise a novel strategy to estimate the change rate based on historical information which derives accurate predictions by quickly adapting to the observed change history.

We evaluate our approach using a real Web data. We actively monitored84,699 pages from the most popular sites in the Brazilian Web for 100 days. Our experimental results indicate that our technique it makes better use of resources and keeps the replicas fresher than strategies previously proposed in the literature.

2. RELATED WORK

Several works have addressed the problem of Web-replica maintenance. Brandman et al. [1] proposed a server-side approach, where the Web server keeps a file with a list of URLs and their respective modification dates. 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 and crawler resources, but it has a serious drawback in that it requires modifications to the Web server implementation. As a result, its effectiveness depends on the adoption of this mechanism by Web sites.

Client-side techniques (see e.g., [4, 9, 7]) make no assumptions with respect to server functionality. Compared to server-side solutions, they incur substantially higher overheads, as they need retrieve all pages and compare them against the replicas, as opposed to just retrieving the pages that are marked as modified since the previous visit. Different techniques have been proposed which aim

[*]Work done while the author was a master student at the Centro de Informática, Universidade Federal de Pernambuco.

WIDM'05, November 5, 2005, Bremen, Germany.

to predict the change rate of pages based on their history. Cho and Molina [4] proposed an incremental Web crawler that uses statistical estimators to adjust revisit frequency of pages based on how often the page changes. Edwards et al. [9] used linear programming to improve replica freshness. Cho and Ntoulas [7] extract samples from data sources (e.g., a Web site) to estimate their change rate and update the replicas of these sources based on this sampling, i.e., more resources are allocated to more dynamic sources. Their experimental results showed that, in the long term, the performance of the sampling approach is worse than the non-uniform policy [7].

Cho and Molina [5] performed an in-depth study on how to maintain local copies of remote data sources fresh when the source data is updated autonomously and independently. They proposed several update policies and studied their effectiveness. To maximize the overall freshness of the data in the replicated repository, they showed that the uniform policy is always superior to the proportional (non-uniform) approach. Although overall freshness is maximized, their measure penalizes the most dynamic pages which may not be updated as frequently as they change. Since very dynamic pages have been found to be accessed more often by users [8], even if the overall freshness is maximized, the the perceived quality of the repository may be low if the uniform refresh policy is used. To address this limitation, Pandey and Olston [16] proposed a user-centric approach to guide the update process. During the update process, they prioritize pages that if updated, maximize the expected improvement in repository quality. This expectation takes into account the likelihood that this page is viewed in search results.

A limitation that is common to all these approaches is that they base their predictions solely on the page change history. As we show in Section 4, this leads to resource waste, since during the learning process many pages are visited unnecessarily. By estimating the changing behavior using static features of Web pages, our technique achieves substantially improved performance.

Another factor that contributes to the improved performance of our approach is a novel strategy for history-based prediction. Unlike the Bayesian estimator proposed by Cho and Molina [5], our historic classifier quickly adapts to variations in the change rates and consequently, it is able to predict the change rate with higher accuracy (see Section 4).

3. OUR SOLUTION

A natural solution to the problem of updating Web-page replicas is trying to predict the change rate of pages based on their actual change history. Unlike previous works, in addition to learning the dynamic behavior of pages, our approach also learns how *static* features of pages influence their associated change frequency. As illustrated in Figure 1, our solution to the Web-replica update problem consists of two phases:

- (Phase I) When a page *P* is first visited, there is no a priori knowledge about its behavior. The only available information are the attributes of *P*, e.g., file size or number of images. The *Static Classifier* relies on these attributes to predict how fast *P* changes. The predictions are stored in the *Change Predictions* repository.

- (Phase II) The *Historic Classifier* learns to predict how often the page *P* changes based on its change history. During each subsequent visit to *P*, historical information is accumulated in the *Change History* repository, and based on this information, the historic classifier continuously updates the predicted change rate for *P*.

Both the static and historic classifier try to estimate the average interval of time at which a given page is modified, which is a real number. To simplify the learning task, we turn this regression task (i.e., the approximation of a real-valued function) into a classification task by discretizing the target attribute. We create a finite number of change rate groups, and each page retrieved is assigned to one of these groups. The non-uniform refresh policy will then update a page based on the average change rate of the group it belongs to.

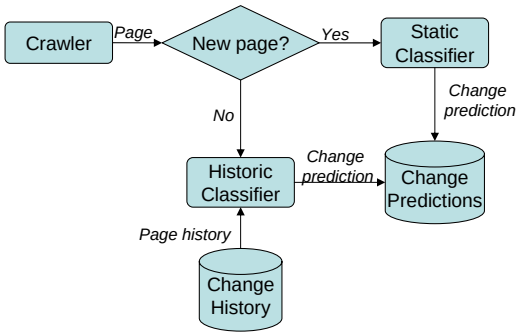

Figure 1: Two-phase solution to the Web-replica maintenance problem.

3.1 Building the Training Data Set

In order to train the static classifier and generate the history-based predictions, we need to obtain a set of pages and their change history. For our experiments, we selected the 100 most accessed sites of the Brazilian Web[1] We performed a breadth-first search on these sites down to depth 9 (depth 1 is the root), and by limiting the maximum number of retrieved pages to 500 per level (to avoid overloading these sites), we gathered 84,699 pages.

For 100 days, we visited each of these pages once a day. To detect changes between visits, the checksum of current page (without html tags) was generated by MD5 algorithm [15] and compared against the checksum of the replica stored in our local repository. Note that our history information is *incomplete*: we have no information about the behavior of these pages before the monitoring started; and since the pages were visited once a day, if they changed more than once daily, this information was not captured.

3.2 Creating Change Rate Groups

As discussed above, we turn the regression task of estimating the change rate of pages into classification task by defining a finite number of change rate groups. As we are discretizing the attribute that represents the groups of classification, we performed this task using an unsupervised discretization. We selected 56,466 pages from our repository, and estimated the change rate of each page using the following estimator:

$$-ln(n - X + 0.5/n + 0.5) \quad (1)$$

Here, *n* is the number of visits to the page and *X* is the number of modifications in these *n* visits. This estimator was found to be most appropriate when pages are actively monitored at regular intervals and the change history is incomplete [6].

[1]This information was obtained from the Ibope E-Ratings Institute – http://www.ibope.com.br.

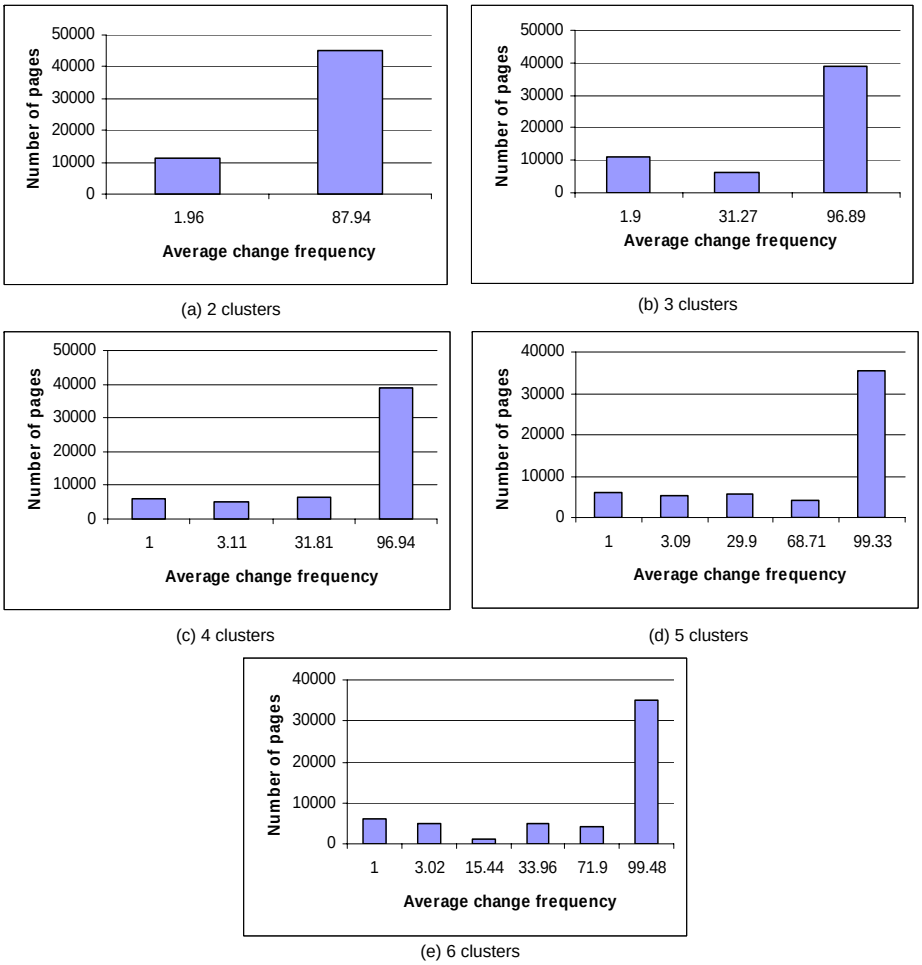

Figure 2: Outputs of the k-means algorithm.

For the unsupervised discretization, we used the k-means clustering algorithm [10]. We ran the k-means algorithm with 2, 3, 4, 5 and 6 clusters. The Figure 2 shows, for different values of k, the cluster sizes and associated change frequencies. The configuration with 4 groups was selected to be used by both the static and historic classifiers. One of the reasons we selected the output with 4 clusters was that it promotes the existence of groups with very dynamic pages. Since pages that are modified more often are also pages that are accessed more often [8], it makes sense to prioritize dynamic content that is most interesting to the users. The output with 4 clusters (Figure 2(c)) is better than 2 (Figure 2(a)) and 3 (Figure 2(b)) because, in these, the most dynamic pages are in only one group (average change frequency of 1.9), whereas, in the 4, these pages are distributed in two different clusters with average change frequencies of 1 and 3.11. We did not select the configuration with 5 (Figure 2(d)) and 6 (Figure 2(e)) clusters because only groups with few dynamic pages were generated compared to the configuration with 4 clusters. Besides, the 4-cluster configuration avoids groups with proportionally few elements. The intuition behind this choice is that with a larger number of clusters the classifier is not only more likely to misclassify pages but also the classification task becomes more expensive.

3.3 Learning Static Features

The static classifier learns static features of a page and their relationship to the page's change behavior. Its goal is to predict, after seeing a page for the first time, to which change group it belongs. Although it may be counter-intuitive that such a relationship exists, recent studies have found that some characteristics of Web pages are related to their dynamism. For instance, Douglis et al [8] noted that dynamic pages are larger and have more images. Brewington and Cybenko [2] observed that the absence of the `LAST-MODIFIED` information in the HTTP header indicates that a page is more volatile than pages that contain this information.[2] Page features that can be learned by the static classifier include: number of links; number of e-mail addresses; existence of the HTTP header LAST-MODIFIED; file size in bytes (without html tags); number of images; depth of a page in its domain (a domain represents, for instance, for the site www.yahoo.com every page in *.yahoo.com) and the directory level of the page URL in relation to the URL root from the Web server (e.g., www.yahoo.com is level 1, www.yahoo.com/mail/ level 2, and so on).

[2]This header indicates the last time a page was modified. It is mainly used by proxy cache servers to verify if a page was changed since the last time that it was stored in the cache. The absence of this header can be interpreted to mean that the page is so dynamic that it is not worthwhile to cache it.

Algorithms	Error test rate	Classification time
J48 without pruning	11.9	2.41s
J48 postpruning	10.7	1.63s
NaivesBayes	40.5	120.5s
IBk with k=1	11.27	4393.15s
IBk with k=2	11.88	6335.49s

Table 1: Results of the execution of the learning algorithms.

Class	Visitation rate	Window size	Min. threshold	Max. threshold
one_day	1 day	10	0.3	0.7
one_week	7 days	8	0.3	0.7
one_month	30 days	6	0.3	0.7
greater_month	100 days	2	0.3	0.7

Table 2: Example of configuration for the historic classifier.

We performed feature selection [10] to derive the feature set for the static classifier. The feature selection technique used was the wrapper method, which uses induction algorithms (e.g., decision tree) to evaluate relevant feature subsets [12], i.e., it conducts a search in a space of possible subset of features and chooses the subset that results in the lowest error rate. As a result of this task, from the features listed above, the only one that was found to have little or no corrrelation with the change rate was "depth of a page in its domain". The relevant features were used to construct the static classifier.

The discretization step used two-thirds of the monitored pages to create four change rate classes. Since these groups have different cardinalities, we set the maximum number of pages in each group to be the cardinality of the smallest group – 5,000 pages. This prevents classes with more elements to be over-represented in the training and test data set. If, for example, 90% of these data belonged to a single class in the test set, the classifier could assign every sample as belonging to this class and its error test rate would be 0.1, what does not make sense. Thus, the input data for constructing the static classifier consists of 20,000 pages.

We used WEKA [18] to generate different classifiers for this input set. We built the classifiers using two thirds of this corpus, and with the remaining one-third, we verified their effectiveness. For our experiments, we tested the following classification algorithms [10]:

- J48 (decision tree): a decision tree is a flow graph in a tree structure, in which each internal node represents a test on an attribute; each branch is the result of this test; and leaf-nodes represent the target classes. To classify a new observation, the values are checked against the decision tree. A trace is performed from the root to the leaf-node, where the sample is classified;
- NaiveBayes (naïve bayes): the naïve bayes algorithm is based on Bayes' theorem. This algorithm calculates the likelihood of an instance belonging to a given class based on the distribution of the data in the training set;
- IBk (k-nearest neighbor): the k-nearest neighbor algorithm is based on analogy learning. The training samples are represented by n-dimensional numeric attributes. Each sample is a point in a n-dimensional space, where each dimension is an attribute. Thus, all the training set samples are plotted in this space. When a new observation is presented, the classifier searches in this space the k closest examples. In the case of k=1, for instance, the new observation is assigned to the class of the nearest sample.

These algorithms have been shown to be effective for various learning tasks; their parameters are easily configured; and they are scalable and capable of handling large volumes of data such as required in our classification task.

In order to compare these classifiers, we take into account both the error rate and classification time. The error rate on the testing set measures the overall performance of a classifier. Small errors rates lead to improved resource usage and repository freshness. Only when a page is misclassified, will the crawler visit it in a different rate than the actual change frequency. The classification time consists of the time it takes to classify all the samples on the testing set. Especially for repositories that maintain a large number of replicas (e.g., a Web search engine), if it takes too long to classify a page, the solution may become infeasible. Thus, it is crucial to select an efficient classifier.

We tested five distinct configurations for the classification algorithms. The different configurations and their results are shown in Table 1. The NaiveBayes classifier led to the highest error rate. As this method assumes that attributes are independent, a possible reason for this result could be the existence of dependencies among the attributes. The error rate for the two configurations of the IBk classifier was low, but the classification time was very high. This is due to the fact that the k-nearest neighbor method stores the whole training set and builds the classifier model when a new sample is received, and for this experiment the training set is very large. The J48 configurations resulted in the best overall performance. Both the error rate and the classification time were low. Note that the J48 configuration using postpruning obtained better results than J48 without pruning. This makes sense because the pruning technique tries to improve the quality of the classifier by removing branches of the tree that might reflect noise in the training set [10]. As the J48 classifier with postpruning obtained the best results, we used it as our static classifier.

3.4 Learning from History

In order to more reliably predict the change rate of pages, it is necessary to continuously monitor the pages' change behavior. Our historic classifier, described in Algorithm 1, infers the behavior of a given page based on its change history. It works as follows. Once the static classifier categorizes the pages into the change rate groups, these pages are visited using the non-uniform refresh policy, i.e., according to the average modification rate of their respective groups. A window size is selected for each group which determines how often the page will be re-classified – when the window size is reached for a group (line 2), the page is re-classified. The classifier verifies how many times the page was changed (number of changes) and calculates the average number of changes in these visits (number of changes/window size) (line 4). Finally, based on the average number of changes, the minimum and maximum change averages threshold for this class, the historic classifier checks whether a given page really belongs to its current class or whether it needs to move to lower (lines 5-6) or higher (lines 7-8) change groups.

Consider, for example, a configuration with 4 change rate groups. The parameters for each group are given in Table 2. Suppose that a page belonging to the one_week group changed 6 times after 8 visits (average of changes = 0.75). As the average of changes was larger than the maximum threshold for this class (0.7), this page is moved to a higher frequency class, the one_day class. If, on the

Algorithm 1 Historic Classifier

1: **Input:** $Page\ \mathcal{P}$, $CurrentClass\ \mathcal{CC}$, $NumberOfChanges\ \mathcal{NC}$, $NumberOfVisits\ \mathcal{NV}$
{$\mathcal{P}$ is the page to be re-classified; $\mathcal{CC}$ is the current class of $\mathcal{P}$; $\mathcal{NC}$ is the number of changes of $\mathcal{P}$; and $\mathcal{NV}$ is the number of visits to $\mathcal{P}$}
{Verify if the page needs to be re-classified}
2: **if** $\mathcal{NV}$ == getNumberOfVisits($\mathcal{CC}$) **then**
3: WS = getWindowSize($\mathcal{CC}$){Get the window size of the current class}
4: AC = $\mathcal{NC}$ / WS {Calculate the average of changes}
{Test if AC is lower than the minimum class threshold}
5: **if** $AC <$ getMinimumThreshold($\mathcal{CC}$) **then**
6: moveToLowerClass($\mathcal{P}$) {Move $\mathcal{P}$ to a lower change rate class}
{Test if AC is higher than the maximum class threshold}
7: **else if** $AC >$ getMaxThreshold($\mathcal{CC}$) **then**
8: moveToHigherClass($\mathcal{P}$) {Move the page to a higher change rate class}
9: **end if**
10: **end if**

other hand, it had been modified only once (average of changes = 0.125), it would be moved to the one_month class.

As we discuss in Section 4, a key feature of this algorithm is that it adapts quickly to the observed change history, preventing misclassified pages to remain in the incorrect group for too long.

4. EXPERIMENTAL EVALUATION

To verify the effectiveness of our approach (J48+historic), we compare it against the strategy proposed by Cho and Molina [4]. We selected this as our baseline because it has been shown to lead to better performance than other strategies [5]. It works as follows. Initially the *pages are classified in a random fashion*; in subsequent visits they use a *bayesian estimator* to decide in which class a page belongs based on its change history. A page is assigned to the class that matches its change history with the highest probability. These probabilities are updated as the pages are visited. If, for instance, a page is modified according to the frequency of its class, its probability of belonging to this class increases, while the probability of belonging the other classes decreases. Otherwise, the probability of the original class decreases and the probability of the class which has a frequency more similar with the change rate of the page increases.

As discussed in Section 3, we actively monitored 84,699 pages from the 100 most popular sites in the Brazilian web for 100 days. Two-thirds of these pages were used to train our static classifier and to obtain the change history for the historic classifier. The remaining one-third of the pages was used to measure the error test rate of each configuration. Recall that a low error rate means the refresh scheme will visiting the pages in a frequency very similar to their actual change rates – thus, only a very small percentage of pages is visited unnecessarily, and the freshness of the repository is maximized, since it will take less time to update stale pages.

In order to assess the effectiveness of the static classifier, we compared it against the random classifier. The results are shown in Table 3. The random classifier leads to a substantially higher error rate – almost 3 times the error rate of our static classifier. These numbers indicate that the static classifier leads to a much improved resource usage than the random classifier.

To better study the effects of the different phases in our approach in isolation, besides J48+historic and Random+bayesian, we examined two additional configurations: our static classifier followed by Cho and Molina's bayesian estimator (J48+bayesian); and the random classifier followed by our historic classifier (Random+historic).

Classifier	Error rate
Random	75.22
J48	25.64

Table 3: Error rate of the initial classifiers.

Class	Change frequency	Window size	Min. threshold	Max. threshold
Group 0	1 day	3	0.2	0.8
Group 1	3 days	2	0.2	0.8
Group 2	31 days	2	0.2	0.8
Group 3	96 days	1	0.2	0.8

Table 4: Values used by the historic classifier.

The parameters used by the historic classifier are shown in Table 4. The values of change frequency represent the average relative change frequency of the elements that belong to each of the 4 groups generated in the discretizing task (see Figure 2). The values for the window size determine the visitation frequency for the pages in a group. For instance, the change frequency of Group 3 is 96 days; since the window size of Group 3 is 1, pages in this group will be visited every 96 days (96 times 1). If the value of the window were 2, the page would be re-visited in 182 days. We ran our historic classifier with different combinations of window sizes – the values shown in Table 4 were selected because they led to the best performance. The bayesian estimator has a single parameter to be configured: the visitation frequency. The same values of visitation frequency used by the historic classifier were used by the bayesian estimator.

Approach	Error rate
Random + Bayesian	34.73
Random + Historic	28.33
J48 + Bayesian	37.87
J48 + Historic	14.95

Table 5: Error rates for the composition of the classifiers.

Table 5 shows the error rates for the configurations we experimented with. Our solution (J48+historic) has the lowest error rate, roughly half of that of the Random+historic configuration. The table also indicates that the historic classifier is very effective: using either the static or the random classifier with our historic classifier has a smaller error rate than the configurations that use the Bayesian estimator. One possible reason for the lower error rate achieved by the historic classifier is that it adapts faster to the variations in the page change rates. Using the bayesian estimator, it might take several iterations for a page to migrate between classes. Suppose, for instance, that there are two change frequency groups: one week, and one month. Suppose too that, at time t_0, a given page P belongs to the "one week" group with probability of 0.9 and to the "one month" group with probability 0.1. After one week (t_0 + 7 days), P is visited but is not modified. The probabilities are then re-computed based on this new observation. Using the bayesian estimator, the new probabilities are: 0.89 to "one week" and 0.11 to "one month". In contrast, using the historic classifier, with the window size of "one week" equals to 1, P would be immediately moved to a lower change rate group, in this case, to the "one month" group.

5. CONCLUSION AND FUTURE WORK

In this paper, we propose a new approach to the problem of updating replicas of Web content. Our solution consists of two phases: when a page is first visited, a decision-tree-based classifier predicts it change rate based on the page's (static) features; during subsequent visits, an estimator uses the page's change history to predict its future behavior. Experiments using actual snapshots of the Web showed that our solution obtained a substantially improved performance compared to other approaches to updating Web data.

There are several directions we plan to pursue in future work. Notably, we would like to investigate how to obtain additional page features that can potentially improve the performance of the static classifier. For example, backlinks and the page rank [3] provide some indication of a page's popularity, and are thus intuitively related to the page dynamism.

6. REFERENCES

[1] O. Brandman, J. Cho, H. Garcia-Molina, and N. Shivakumar. Crawler-friendly web servers. *ACM SIGMETRICS Performance Evaluation Review*, 28:9–14, 2000.

[2] B. E. Brewington and G. Cybenko. How Dynamic is the Web? In *Proc. of WWW*, pages 257–276, 2000.

[3] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks*, 30(1-7):107–117, 1998.

[4] J. Cho and H. Garcia-Molina. The Evolution of the Web and Implications for an Incremental Crawler. In *Proc. of VLDB*, pages 200–209, 2000.

[5] J. Cho and H. Garcia-Molina. Effective page refresh policies for web crawlers. *ACM Transactions on Database Systems*, 28(4):390–426, 2003.

[6] J. Cho and H. Garcia-Molina. Estimating frequency of change. *ACM Transactions on Internet Technology*, 3(3):256–290, 2003.

[7] J. Cho and A. Ntoulas. Effective Change Detection Using Sampling. In *Proc. of VLDB*, pages 514–525, 2002.

[8] F. Douglis, A. Feldmann, and B. Krishnamurthy. Rate of Change and other Metrics: a Live Study of the World Wide Web. In *Proc. of the USENIX Symposium on Internetworking Technologies and Systems*, pages 147–158, 1999.

[9] J. Edwards, K. McCurley, and J. Tomlin. An Adaptive Model for Optimizing Performance of an Incremental Web Crawler. In *Proc. of WWW*, pages 106–113, 2001.

[10] J. Han and M. Kambe. *Data Mining: Concepts e Techniques*. Morgan Kaufmann Publishers, 2001.

[11] Internet archive. http://www.archive.org.

[12] R. Kohavi and G. H. John. Wrappers for feature subset selection. *Artifical Intelligence*, 97(1-2):273–324, 1997.

[13] S. Lawrence and C. L. Giles. Searching the world wide web. *Science*, 280(5360):98–100, 1998.

[14] S. Lawrence and C. L. Giles. Accessibility of information on the web. *Nature*, 400(6740):107–109, 1999.

[15] The MD5 Message-Digest Algorithm. http://www.rfc-editor.org/rfc/rfc1321.txt.

[16] S. Pandey and C. Olston. User-Centric Web Crawling. In *Proc. of WWW*, pages 401–411, 2005.

[17] Webarchive project. http://webarchive.cs.ucla.edu.

[18] Weka 3: Data Mining Software in Java. http://www.cs.waikato.ac.nz/ ml/weka.