Learning to Schedule Webpage Updates Using Genetic Programming

Aécio S.R. Santos[1], Nivio Ziviani[1], Jussara Almeida[1], Cristiano R. Carvalho[1], Edleno Silva de Moura[2], and Altigran Soares da Silva[2]

[1] Universidade Federal de Minas Gerais, Department of Computer Science, Belo Horizonte, Brazil
[2] Universidade Federal do Amazonas, Institute of Computing, Manaus, Brazil

Abstract. A key challenge endured when designing a scheduling policy regarding freshness is to estimate the likelihood of a previously crawled webpage being modified on the web. This estimate is used to define the order in which those pages should be visited, and can be explored to reduce the cost of monitoring crawled webpages for keeping updated versions. We here present a novel approach to generate score functions that produce accurate rankings of pages regarding their probability of being modified when compared to their previously crawled versions. We propose a flexible framework that uses genetic programming to evolve score functions to estimate the likelihood that a webpage has been modified. We present a thorough experimental evaluation of the benefits of our framework over five state-of-the-art baselines.

1 Introduction

The quality of a Web search engine depends on several factors, such as the content gathered by the web crawler, the ranking function that produces the document ordering, and the user interface. By its turn, the success of the crawling process of a web search engine depends the coverage of the crawl, the policy used to select pages to collect, and the freshness of the pages. The focus of this work is on freshness, i.e., on the design of policies for scheduling webpage updates.

Web crawlers usually have access to limited bandwidth and their scheduler should periodically sort a large list of known URLs to define the order in which they should be visited. In this scenario, performing a full scan of all priorly crawled webpages to assure database freshness is unfeasible. To avoid that, crawling architectures (e.g., VEUNI [8]) use a score function to assign a weight to each known webpage (URL). Only the top k pages, k being a parameter, are taken to be visited. After crawling the k pages, the scheduler starts a new crawling cycle, using the score function to rank the known pages to be visited.

We here focus on the problem of estimating the likelihood that a webpage has been modified. Prior work has used machine learning techniques to related tasks (e.g., grouping pages with similar change behaviour [11], and predicting a page's change behaviour [10]), but none has applied them to build score functions. We

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-3-319-02432-5_33 investigate the potential of using a genetic programming (GP) framework to learn these score functions. Our experimental evaluation shows that our solution outperforms existing score functions [3,11], being it a viable alternative to solve the addressed problem and opening opportunities for future work.

2 Background and Related Work

Like [2,3,11], we here consider a binary freshness model where the freshness of page p at time t is 1 if the copy of p is identical to the live copy, or 0, otherwise. The *freshness* of a set C of webpages can then be estimated by the average number of fresh pages in C at time t.

Probabilistic models have been proposed to approximate the history and predict webpage changes. For example, Coffman et al. [5] proposed to model the occurrences of changes on each page p by a Poisson process with parameter λ_p changes per time unit. Cho and Garcia-Molina [3] also investigated estimators for the change frequency of elements that are updated autonomously, in various scenarios. They showed that a web crawler can achieve improvements in freshness by setting its refresh policy to visit pages proportionally more often based on their proposed estimator, which is defined in Section 5.1.

Cho and Ntoulas [4] proposed a sampling-based method to detect webpage changes based on the number of pages that changed in a sample downloaded from the web site, which may be too coarse to represent all of its pages. Tan and Mitra [11] proposed to solve this problem by grouping the pages into k clusters with similar change behavior, and then sorting the clusters based on the mean change frequency of a representative cluster's sample. They proposed four strategies to compute the weights associated with a change in each of the downloaded cycles, which are further described in Section 5.1. Our work differs from [11] as our approach is not sampling based, but uses machine learning to build a score function that allows the scheduling of webpage updates. Once the score function has been learned, which is done off-line, it can be applied quickly, thus allowing large scale crawling using the architecture presented in Section 3.

Radinsky and Bennett [10] proposed a webpage change prediction framework that uses content features, the degree and relationship among the prediction page's observed changes, the relatedness to other pages, and the similarity in the kinds of changes they experienced. We here only use features related to whether the page changed or not during each cycle. However, given the flexibility of GP, our approach can be easily extended to include other features in the future.

3 Crawler Architecture

The incremental crawler architecture considered here has four main components: fetcher, URL extractor, uniqueness verifier, and scheduler [8]. Considering cycle i, the *fetcher* receives from the *scheduler* a set of candidate URLs to be crawled, locates them, and returns a set of URLs actually downloaded. The URL extractor parses each downloaded page and obtains a set of new URLs. The uniqueness

Listing 1.1. Genetic Programming for Crawling (GP4C)

```
1  Let $\mathcal{T}$ be a training set of pages crawled in a given period;
2  Let $\mathcal{V}$ be a validation set of pages crawled in a given period;
3  Let $N_g$ be the number of generations;
4  Let $N_b$ be the number of best individuals;
5  $\mathcal{P} \leftarrow$ Initial random population of individuals;
6  $\mathcal{B}_t \leftarrow \emptyset$;
7  For each generation $g$ of $N_g$ generations do {
8      $\mathcal{F}_t \leftarrow \emptyset$;
9   For each individual $i \in \mathcal{P}$ do
10         $\mathcal{F}_t \leftarrow \mathcal{F}_t \cup \{g, i, \mathit{fitness}(i, \mathcal{T})\}$;
11     $\mathcal{B}_t \leftarrow \mathit{getBestIndividuals}(N_b, \mathcal{B}_t \cup \mathcal{F}_t)$;
12     $\mathcal{P} \leftarrow \mathit{applyGeneticOperations}(\mathcal{P}, \mathcal{F}_t, \mathcal{B}_t, g)$;
13 }
14 $\mathcal{B}_v \leftarrow \emptyset$;
15 For each individual $i \in \mathcal{B}_t$ do
16     $\mathcal{B}_v \leftarrow \mathcal{B}_v \cup \{i, \mathit{fitness}(i, \mathcal{V})\}$;
17 BestIndividual $\leftarrow$ $\mathit{applySelectionMethod}(\mathcal{B}_t, \mathcal{B}_v)$;
```

verifier checks each URL against the repository of unique URLs[1]. The scheduler chooses a new set of URLs to be sent to the fetcher, thus starting a new cycle.

We here focus on the algorithm for scheduling webpage updates, which is driven by two main goals: *coverage*, the fraction of desired pages that the crawler downloads successfully; and *freshness*, the degree to which the downloaded pages remain up-to-date, relative to the current live web copies. Most prior work focuses on only one of them. This work is focused on freshness.

4 Genetic Programming for Incremental Crawling

We here apply GP to the problem of scheduling webpage updates, using it to derive score functions that capture the likelihood that a page has changed. Pages with higher likelihood should receive higher scores, and thus higher priority in the scheduling process. Our method, called *GP4C – Genetic Programming for Crawling*, uses a GP process adapted from [1], and is presented in Listing 1.1.

As shown in Listing 1.1, GP4C is an iterative process with two phases: *training* (lines 5–13) and *validation* (lines 14–16). Our training and validation sets are built as follows: we train with an initial set of pages and validate the results with a distinct set of pages. This scenario is closer to that of large crawling tasks (e.g., crawling to a world wide search engine), where an initial set of pages to build the training set is crawled first, and then a set of validation pages is crawled. Experimental tests apply the resulting function in a third set of pages.

GP4C starts with the creation of an initial random population of N_p individuals (line 5) that evolves generation by generation using genetic operators (line 12) until a maximum number of generations (N_g). We apply the genetic

[1] Note that the size of the set of candidate URLs passed to the fetcher is defined by the amount of memory space available to the uniqueness verifier.

operators of reproduction, crossover and (swap/replacement) mutation at predefined rates. In particular, for the crossover operation, the selection of the parents is performed randomly among the top best individuals of the current generation. In the training phase, a fitness function is applied to evaluate all individuals of each generation (lines 9–10), so that only the N_b fittest individuals, across all previous generations, are selected to continue evolving (line 11). After the last generation is built, to avoid *over-fitting*, the validation phase is applied: the fitness function is used over the validation set (lines 15–16), and individuals that perform the best are selected as the final scheduling solutions (line 17).

Each *individual* represents a function that assigns a score to each page when composing the scheduling at the training set. Such score combines information useful for estimating the likelihood of a given page being updated in a period of time, exploring, for instance, its behavior in previous crawls. The training is performed in a period of time considered by us, and each individual is evaluated as being the function to create the scheduling in the whole training period.

An individual is represented by a binary tree with a maximum depth d, where terminals are features that help characterizing a page's updating behavior. We here consider three features: (1) n, the number of times that the page was visited; (2) X, the number of times that the page changed in n visits; and t, the number of cycles since the page was last visited. We also use the following *constant values* as terminals: $0.001; 0.01; 0.1; 0.5; 1; 10; 100; 1000$. As inner nodes of the tree, we use the functions addition ($+$), subtraction ($-$), multiplication ($*$), division ($/$), logarithm (log), exponentiation (*pow*), and the exponential function (*exp*).

The *fitness function* measures the quality of the ranking produced using a given individual for the whole training period. To compute the fitness of an individual, we take the score it produces for each page in the training set of each day and generate a schedule for the crawling to be performed on the next day. We here use as fitness function the ChangeRate metric, defined in Section 5.1.

As in [1], we select the *best individuals* in the validation step by running the GP process N times with distinct random seeds, so as to reduce the risk of finding a low performance local best individual. We pick the best individual among those generated by these N runs, referring to this approach as $GP4C_{Best}$. As in [6], we also consider two other strategies that are based on the average Avg_σ and the sum Sum_σ of the performances of each individual in both training and validation sets, minus the standard deviation of such performance when selecting best individuals. The individual with the highest Sum_σ (or Avg_σ) is selected. We refer to GP4C using these selection strategies as $GP4C_{Sum}$ and $GP4C_{Avg}$.

5 Experimental Evaluation

We used a crawl simulation to ensure that all policies are compared under the same conditions. We built a webpage dataset collected from the Brazilian Web (.br domain) using the crawler presented in [8], whose architecture is described in Section 3. Table 1 summarizes the dataset, referred to as BRDC'12[2], which

[2] Available at `http://homepages.dcc.ufmg.br/~aeciosantos/datasets/brdc12/`

consists of a fixed set of webpages crawled on between September and November 2012. From a repository of around 200 million URLs we selected 3,059,698 webpages, which were then daily monitored. During the monitoring periods, our crawler ran from 0AM to 11PM, recollecting each selected webpage every day, which allowed us to determine when each page was modified.

Table 1. Overview of our BRDC'12 dataset

Monitoring period	Number of webpages	Number of websites	Number of webpages/site		
			Min	Max	Average
57 days	417,048	7,171	1	2,336	58.15

5.1 Baselines and Evaluation Metric

We compare $GP4C_{Best}$, $GP4C_{Sum}$ and $GP4C_{Avg}$ with five baselines, referred to here as CG, NAD, SAD, AAD and GAD. Given n the number of visits and X the number of times that a page p changed in those n visits, the CG baseline [3] estimates the change frequency of p as:

$$\mathrm{CG} = -\log(\frac{n - X + 0.5}{n + 0.5}). \tag{1}$$

The other four baselines were proposed by Tan and Mitra [11]. In order to compute the change frequency of the pages, they assume that each page p follows a Poisson process with parameter λ_p. That is, the probability that a page p will change in the interval $(0, t]$ is given by $1 - e^{\lambda_p t}$. We set t to be the number of cycles since the page was last downloaded and compute λ_p using the change history of the pages:

$$\lambda_p = \sum_{i=1}^{n} w_i \cdot \mathrm{I}_i(p),$$

where n is the number of times the page was downloaded so far, w_i is a weight associated with a change occurred in the i^{th} download of the page ($\sum_{i=1}^{n} w_i = 1$), and $\mathrm{I}_i(p)$ is either 1 if page p changed in the i^{th} download, or 0 otherwise.

The weights w_i are computed according to one of the following schemes:

– NAD (*Nonadaptive*): all changes are equally important ($w_i{=}\frac{1}{n}$, $\forall i = 1..n$).
– SAD (*Shortsighted adaptive*): only the last change is important ($w_1{=}\cdots = w_{n-1} = 0$, $w_n = 1$).
– AAD (*Arithmetically adaptive*): more recent changes are more important, and weights decrease according to an arithmetic progression ($w_i = \frac{i}{\sum_{i=1}^{n} i}$).
– GAD (*Geometrically adaptive*): as the previous scheme, but weights decrease more quickly, following a geometric progression ($w_i = \frac{2^{i-1}}{\sum_{i=1}^{n} 2^{i-1}}$).

We also consider two simpler approaches to build score functions, referred to as *Rand* and *Age*. In *Rand*, the scores are randomly chosen, whereas in *Age*, they are equal to the time t since the page was last visited (i.e., downloaded).

Our main evaluation metric is the ChangeRate, defined in [7] to assess the ability of a scheduling policy to detect updates. The ChangeRate at cycle i is the fraction of pages that were downloaded during i that had changed. The intuition is that the higher the concentration of changed pages, the better the scheduling. We use ChangeRate both as evaluation metric and fitness function, leaving the use of alternative metrics (e.g., weighted ChangeRate [4]) for the future.

5.2 Experimental Methodology

We adopted a 5-fold cross validation: 4 folds were equally divided into *training set* and *validation set*, and the last fold was used as *test set*. We report average results for the 5 test sets, along with corresponding 95% confidence intervals.

In order to evaluate the score functions and compute fitness values we simulate a crawl using our dataset. Our simulation starts with a warm-up period W=2 days, during which collected data is used to build basic statistics about each page. For each day following warm-up, we apply our proposed score function and each baseline to assign scores to each page. The download of the top-k pages with highest scores produced by each method is then simulated by updating statistics of the page such as number of visits (i.e., downloads), number of changes, etc. We set k equal to 5% of the total number of webpages in the dataset. Whenever the actual number of changed pages on a day is smaller than k, no evaluated algorithm can reach a maximum ChangeRate.

Regarding parametrization of the GP framework, we set N_p equal to 300 individuals, created using the ramped half-and-half method [9]. Due to the stability of results, we set N_g equal to 50 generations as termination criterion. We adopted tournament selection of size 2 to select individuals to evolve and set the crossover, reproduction, replacement mutation and swap mutation rates equal to 90%, 15%, 5% and 5%, respectively. We set the maximum tree depth d to 10 and the maximum depth for crossover to 9. During the evolution process we kept the $N_b = 50$ best individuals discovered through all generations to the validation phase. We ran the GP process using N=5 random seed values.

5.3 Results

We now discuss the results produced by our GP4C framework and the baselines using the BRDC'12 dataset. We consider only a basic set of terminals - n, X and t (see Section 4) - to show that our solution can derive functions that perform as good or better than the baselines.

Figure 1 shows the average ChangeRate for each day, for $GP4C_{Best}$ and all baselines. We omit the results for the other GP4C variations as they are either statistically tied or inferior to $GP4C_{Best}$. With 95% confidence, $GP4C_{Best}$ is statistically superior to all baselines in most days, being tied to NAD, AAD, GAD and CG, the most competitive baselines, in only a few days. Specifically, $GP4C_{Best}$ is statistically superior to NAD, AAD, GAD and CG in 22, 47, 49 and 50 of the simulated download cycles, respectively, being statistically tied

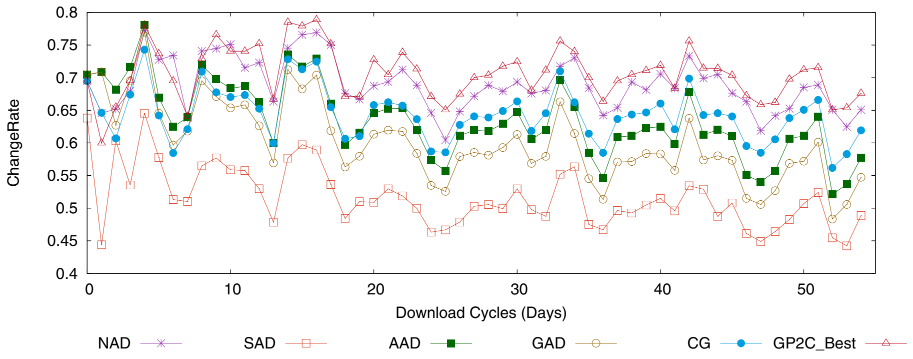

Fig. 1. (Color online) Average ChangeRate on each download cycle

with them in the other days. The only exceptions occur in the three initial days: *GP4C*$_{Best}$ is statistically inferior to AAD in days 1 and 3 and to GAD in day 1. This result corroborates the flexibility of our framework as it is able to produce results at least as good, if not better, than all five baselines.

Table 2 summarizes these results, showing average ChangeRate along with 95% confidence intervals for all methods, including the Rand and Age baselines (omitted in Figure 1). Once again, our *GP4C* solutions produce score functions superior to all baselines. Note that the results of Rand and Age are much worse than all other methods. Moreover, even though our *GP4C* approaches use the exact set of parameters used by the *CG* baseline [3] (i.e., n, X, t), our methods produce much better results, increasing the average ChangeRate by around 10%. The best baseline is NAD, which uses a different set of parameters that may provide more useful information about a page's updating behavior. Nevertheless, our approaches are still slightly better than NAD and can easily derive other functions if more parameters are given as input.

Table 2. Average ChangeRate for all days along with 95% confidence intervals

Rand	Age	NAD	SAD	AAD	GAD	CG	*GP4C*$_{Best}$	*GP4C*$_{Sum}$	*GP4C*$_{Avg}$
0.1857 ± 0.0007	0.2130 ± 0.0009	0.6892 ± 0.0056	0.5166 ± 0.0066	0.6344 ± 0.0095	0.6016 ± 0.0059	0.6439 ± 0.0067	0.7058 ± 0.0096	0.7008 ± 0.0176	0.7034 ± 0.0107

Finally, we note that our GP4C framework can be used for better understanding the scheduling problem. As example, an extremely simple, but also effective, function generated by our method is $t * X$, which yields a final performance superior to most of the baselines, with average ChangeRate above 0.690. It was not the best function found by GP4C, but illustrates how the framework can be applied not only to derive good score functions, but also to give insights about the most important parameters.

6 Conclusions and Future Work

We have presented a GP framework to automatically generate score functions to be used by schedulers of web crawlers to rank webpages according to their likelihood of being modified since they were last crawled. We compared three variations of our framework against seven state-of-the-art baselines, using a webpage dataset collected from the Brazilian Web. Our results show that our best function, $GP4C_{Best}$, is statistically superior to all baselines in most of the simulated download cycles. Moreover, our framework is quite flexible and can derive new score functions by exploiting new features (e.g., Pagerank of the pages, cost for crawling) or alternative fitness functions that balance the objectives of freshness and coverage. This is a direction we intend to pursue in the future.

Acknowledgements. We thank the Brazilian National Institute of Science and Technology for the Web (grant MCT-CNPq 573871/2008-6), Project MinGroup (grant CNPq-CT-Amazônia 575553/2008-1) and authors' grants and scholarships from CNPq.

References

1. Carvalho, A.L., Rossi, C., de Moura, E.S., da Silva, A.S., Fernandes, D.: Lepref: Learn to precompute evidence fusion for efficient query evaluation. Journal of the American Society for Information Science and Technology 63(7), 1383–1397 (2012)
2. Cho, J., Garcia-Molina, H.: Synchronizing a database to improve freshness. In: SIGMOD Record, pp. 117–128 (2000)
3. Cho, J., Garcia-Molina, H.: Estimating frequency of change. ACM Transactions on Internet Technology 3, 256–290 (2003)
4. Cho, J., Ntoulas, A.: Effective change detection using sampling. In: VLDB, pp. 514–525 (2002)
5. Coffman, E.G., Liu, Z., Weber, R.R.: Optimal robot scheduling for web search engines. Journal of Scheduling 1(1) (1998)
6. de Almeida, H.M., Gonçalves, M.A., Cristo, M., Calado, P.: A combined component approach for finding collection-adapted ranking functions based on genetic programming. In: SIGIR, pp. 399–406 (2007)
7. Douglis, F., Feldmann, A., Krishnamurthy, B., Mogul, J.: Rate of change and other metrics: a live study of the world wide web. In: USENIX Symposium on Internet Technologies and Systems, p. 14 (1997)
8. Henrique, W.F., Ziviani, N., Cristo, M.A., de Moura, E.S., da Silva, A.S., Carvalho, C.: A new approach for verifying URL uniqueness in web crawlers. In: Grossi, R., Sebastiani, F., Silvestri, F. (eds.) SPIRE 2011. LNCS, vol. 7024, pp. 237–248. Springer, Heidelberg (2011)
9. Koza, J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press (1992)
10. Radinsky, K., Bennett, P.: Predicting content change on the web. In: WSDM (2013)
11. Tan, Q., Mitra, P.: Clustering-based incremental web crawling. ACM Transactions on Information Systems 28, 17:1–17:27 (2010)