Inf Retrieval J (2015) 18:73–94
DOI 10.1007/s10791-014-9248-5

A genetic programming framework to schedule webpage updates

Aécio S. R. Santos · Cristiano R. de Carvalho · Jussara M. Almeida · Edleno S. de Moura · Altigran S. da Silva · Nivio Ziviani

Received: 17 March 2014 / Accepted: 6 October 2014 / Published online: 28 October 2014

Abstract The quality of a Web search engine is influenced by several factors, including coverage and the freshness of the content gathered by the web crawler. Focusing particularly on freshness, one key challenge is to estimate the likelihood of a previously crawled webpage being modified. Such estimates are used to define the order in which those pages should be visited, and thus, can be exploited to reduce the cost of monitoring crawled webpages for keeping updated versions. We here present a Genetic Programming framework, called *GP4C—Genetic Programming for Crawling*, to generate score functions that produce accurate rankings of pages regarding their probabilities of having been modified. We compare *GP4C* with state-of-the-art methods using a large dataset of webpages crawled from the Brazilian Web. Our evaluation includes multiple performance metrics and several variations of our framework, built from exploring different sets of terminals and fitness functions. In particular, we evaluate *GP4C* using the ChangeRate and Normalized Discounted Cumulative Gain (NDCG) metrics as both objective function and

A. S. R. Santos · C. R. de Carvalho · J. M. Almeida (✉) · N. Ziviani
Department of Computer Science, Federal University of Minas Gerais, Belo Horizonte, MG, Brazil
e-mail: jussara@dcc.ufmg.br

A. S. R. Santos
e-mail: aeciosantos@dcc.ufmg.br; aecio.solando@gmail.com; aecio@zunnit.com

C. R. de Carvalho
e-mail: cristiano.dcc@gmail.com

N. Ziviani
e-mail: nivio@dcc.ufmg.br; nivio@zunnit.com

A. S. R. Santos · N. Ziviani
Zunnit Technologies, Belo Horizonte, MG, Brazil

E. S. de Moura · A. S. da Silva
Institute of Computing, Federal University of Amazonas, Manaus, AM, Brazil
e-mail: edleno@icomp.ufam.edu.br

A. S. da Silva
e-mail: alti@dcc.ufam.edu.br

evaluation metric. We show that, in comparison with ChangeRate, NDCG has the ability of better evaluating the effectiveness of scheduling strategies, since it is able to take the *ranking* produced by the scheduling into account.

Keywords Web crawling · Scheduling functions · Genetic Programming

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. The success of the crawling process of a web search engine, by its turn, depends on factors such as the coverage of the crawl, the policy used to select pages to collect, and the freshness of the pages. In this work we focus on freshness, that is, 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., VE-UNI Henrique et al. 2011) 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.

In this context, a key challenge faced when designing scheduling policy regarding freshness is to estimate the likelihood that a previously crawled webpage has been modified on the web, so that the scheduler may use this estimation to determine the order in which those pages should be visited. A good estimation of which pages are more likely of having been modified allows the system to reduce the overall cost of monitoring its crawled webpages for keeping updated versions. We note that the final scheduling should also take other information into account, such as estimates of page importance to the users, and the cost to download a page. However, the system can benefit from good estimations of the likelihood that a page has been modified when determining its final scheduling.

We here focus on the problem of estimating the likelihood that a webpage has been modified. To tackle this problem, we propose a novel machine learning based approach to generate score functions that allow schedulers to produce accurate rankings of pages regarding their probabilities of having been modified on the web when compared to the previously crawled version. Prior work has applied machine learning techniques to related tasks (e.g., grouping pages with similar change behavior (Tan and Mitra 2010), and predicting a page's change behavior (Radinsky and Bennett 2013), but none has used them to build those score functions. Specifically, we investigate the potential of using a Genetic Programming (GP) framework, called *GP4C—Genetic Programming for Crawling*, to learn these score functions.

We evaluate our solution on a large webpage dataset collected from the Brazilian Web (.br domain), using the ChangeRate (Douglis et al. 1997) and Normalized Discounted Cumulative Gain (NDCG) (Järvelin and Kekäläinen 2002) metrics as both objective function to be maximized in the learning to scheduling process and evaluation metric. Whereas ChangeRate has been previously used to evaluate the ability of a scheduler to detect webpage updates, the use of NDCG in this context is new. We argue that the NDCG metric has the ability of better evaluating the effectiveness of scheduling strategies, since it is able to take the ranking produced by the scheduling into account. To that end, it considers as relevant the pages that were modified, and measures the distance of the scheduling derived from the learned score function from an ideal oracle that correctly guesses all the pages that should (and should not) be visited. In contrast, ChangeRate considers any permutation of the resulting ranking set as equally good. Since the selected metric guides the learning process to generating new crawling strategies, the use of a better metric allows us to better evaluate final results and derive better ranking functions when used as objective function.

Our experimental results show that *GP*4*C* outperforms existing score functions (Cho and Garcia-Molina 2003; Tan and Mitra 2010), rendering it a viable alternative to solve the addressed problem, and opening opportunities for future work. Moreover, our results also show the superiority of the final score functions produced when NDCG is used as objective function, compared to ChangeRate: the results of the former are better in terms of both evaluation metrics.

In sum, our main contributions are:

- a novel machine learning based approach to generate score functions to estimate the likelihood that a webpage has been modified, a key component for designing effective freshness driven scheduling functions;
- a flexible GP framework to evolve effective score functions that optimize an objective function and considers features related to the behavior of page changes;
- a thorough evaluation of the benefits of using our framework over state-of-the-art strategies;
- the adoption of the NDCG metric as both evaluation metric and objective function to be maximized in the learning to scheduling process.

A preliminary version of this work was presented in Santos et al. (2013). We here greatly extend this prior work into three directions: (1) we add to the GP framework new features from pages that help in characterizing their updating behavior, (2) we propose the use of NDCG as an objective function to be maximized, and (3) we extend the experimental evaluation to include one new metric as well as new scenarios. Specifically, whereas in Santos et al. (2013), we explore only three basic features and use only ChangeRate as both objective function and evaluation metric, we here also explore five previously proposed estimators of page change probability as features, and use NDCG as both objective function and evaluation metric. Our experimental evaluation shows that both extensions lead to improvements over our previous solution.

The remainder of this paper is organized as follows. In Sect. 2, we review the related literature on crawling. We present a high-level description of the crawler architecture considered in our study in Sect. 3. In Sect. 4, we introduce the GP framework and discuss individuals, terminals, functions in the inner nodes as well as the fitness functions used in our study. In Sect. 5, we present the experimental evaluation of our approach. Finally, we present our concluding remarks as well as directions for future research in Sect. 6.

2 Related work

As previously mentioned, page freshness is a key factor in the success of a crawling process. Regarding page freshness, one goal of previous studies is to maximize the weighted freshness $WF(t)$ of the local repository of pages at time t, defined as:

$$WF(t) = \sum_{p \in C(t)} w(p) \cdot f(p,t),$$

where $C(t)$ denotes the set of pages crawled up to time t, $w(p)$ denotes a numeric weight associated with page p, and $f(p,t)$ is the freshness of page p at time t (Olston and Najork 2010). We here assume that C is static (i.e., there is no dependence on t), and, like Olston and Najork (2010), that each page $p \in C$ has a stationary stochastic pattern of content changes over time. We discuss freshness maximization focusing on the problem of scheduling, which is to produce a crawl order that adheres to the target re-visitation frequencies as closely as possible.

The *freshness* of a set C of webpages can be estimated by the average number of fresh pages in C at time t. Like Cho and Garcia-Molina (2000, 2003), Tan and Mitra (2010), we 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.

Probabilistic models have been proposed to approximate the history, and predict webpage changes. For example, Coffman et al. (1998) 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 (2003) also investigated estimators for the change frequency of elements that are updated autonomously, in various scenarios. In particular, 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. This estimator, which is used as baseline in this work, is defined in Sect. 5.1.

Cho and Ntoulas (2002) 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. However, the sample may be too coarse to represent all pages of the site. Tan and Mitra (2010) solved this problem by grouping pages into k clusters with similar change behavior, and sorting the clusters based on the mean change frequency of a representative cluster's sample. They explored features extracted from the webpage's content, web link structure, and search logs to effectively predict webpage change patterns. They proposed four strategies to compute the weights associated with a change in each download cycle. These strategies are used as baselines in this work, and are further described in Sect. 5.1. Our work differs from Tan and Mitra (2010) as our approach is not based on sampling, but rather relies on 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 effectively and efficiently, thus allowing large scale crawling using the architecture presented in Sect. 3.

Radinsky and Bennett (2013) proposed a webpage change prediction framework that uses, in addition to 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 do not explore such features as our goal is to assess the potential benefits of using GP to build the score functions, which, to our knowledge, has not been done yet. Thus, we 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.

Fetterly et al. (2009) introduced an evaluation framework measuring the maximum potential NDCG that is achievable using a particular crawl. The framework was based on relevance judgments pooled from multiple search engines, allowing the evaluation of different crawl policies. We here also adopt NDCG. However, unlike in Fetterly et al. (2009), where NDCG was used only to evaluate the search engine results using different scheduling functions, we use NDCG both as objective function to be maximized in the GP process and as evaluation metric of the derived scheduling functions.

In this article we choose GP as a tool for learning scheduling functions. Our choice of GP, over various alternatives, is motivated by many prior studies Fan et al. (2004a, b), Trotman (2005), de Almeida et al. (2007), Silva et al. (2009) that applied GP to discover ranking functions for search engines. After all, scheduling functions can be ultimately seen as ranking functions. For instance, Fan et al. (2004c) successfully applied GP to find ranking functions optimised to specific queries in the information routing task. Recent research comparing several alternative machine learning methods for learning ranking functions has indicated that GP produces results that are quite competitive when compared to other state-of-art methods (Carvalho et al. 2012). In particular, GP has the advantage of producing ranking formulas that can be easily integrated to the target system (the scheduling system, in our case). As far as we know, the usage of GP for learning scheduling functions was not previously proposed nor investigated.

3 Crawler architecture

The incremental crawler architecture considered in our study has four components: fetcher, extractor of URLs, uniqueness verifier, and scheduler (Henrique et al. 2011). While our results can be applied to other crawler architectures, we here adopt this architecture to give the reader a context about the problem addressed.

Figure 1 illustrates the crawl cycle. In step 1, the fetcher, which is the component that sees the Web, receives from the scheduler a set of URLs to download. In step 2, the extractor of URLs parses each downloaded page and obtains a set of new URLs. In step 3, the uniqueness verifier checks each URL against the repository of unique URLs. In step 4, the scheduler chooses a new set of URLs to be sent to the fetcher, thus finishing one cycle. Considering cycle i, the *fetcher* locates and downloads webpages. It receives from the *scheduler* a set of candidate URLs to be crawled and returns a set of URLs actually downloaded. The size of the set of candidate URLs is defined by the amount of memory space available to the uniqueness verifier.

We here focus on the algorithm for scheduling webpage updates. We already mentioned that there are many different policies in the literature to select the set of candidates to be crawled from a given set of servers at each cycle. To select a good crawling order there are two main issues: *coverage*, the fraction of desired pages that the crawler downloads successfully; and *freshness*, the degree to which the downloaded pages remain up-to-date

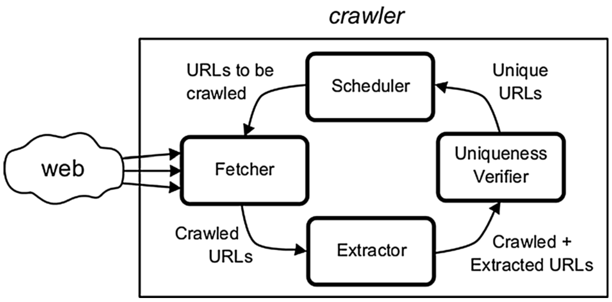


Fig. 1 Web page crawling cycle

relative to the current live web copies. As the amount of crawling resources is finite, there is a trade-off between coverage and freshness, and no consensus on how to balance them. Olston and Najork (2010) argue that balancing the two objectives should be left as a business decision, and most prior work focuses either on coverage or on freshness. This work is focused on freshness.

4 Genetic Programming for incremental crawling

Genetic Programming is a problem-solving technique based on principles of biological inheritance and evolution of individuals (Koza 1992). Given an optimization problem with a large space of solutions, it searches for a near-optimal solution by combining evolutionary selection and genetic operations to create better performing individuals in subsequent generations.

This section first discusses the basic concepts related to GP (Sect. 4.1), and then describes how this technique was applied to our target problem (Sect. 4.2).

4.1 Basic concepts of Genetic Programming

GP evolves a number of candidate solutions called *individuals*, which are represented in memory as binary trees with pre-defined maximum depth d. In a tree, each internal node is a function, and each leaf (terminal) is either a variable or a constant. The maximum number of nodes is determined by the depth of the tree. An example of an individual represented by a tree structure is provided in Figure 2. In this example, the tree represents the change probability function presented in Eq. 5 (Sect. 5.1).

The GP process starts with an *initial population* of N_p randomly generated individuals. Each individual is evaluated by a fitness function and receives a fitness value. This fitness function, whose definition depends on the problem specifics, is used to guide the evolutionary process (e.g., to select only individuals that achieve better fitness results). The individuals will evolve generation by generation through *reproduction*, *crossover*, and *mutation* operations.

The reproduction operation consists in reproducing an individual of a generation into the next. The mutation operation has the role of ensuring diversity in the population, and can be of two types: swap mutations, where randomly chosen subtrees of the individual are swapped, and replacement mutations, where subtrees of the individuals are completely replaced. In the latter, a random node in the tree representing an individual of the current generation is selected and replaced by a new randomly created subtree, which is included in the new generation. The crossover operation allows genetic content exchange between two individuals, the *parents*, selected among the best individuals of the current generation. A random subtree is selected from each parent. The two subtrees are swapped to build a new individual, which is included in the next generation.

These operations are parameterized by: the *reproduction rate*, which is the percentage of elements that are copied to the next generation, chosen among the best individuals according to the fitness function; the *crossover rate*, which is the percentage of elements that are used by the crossover operation; the *replacement* and *swap mutation rates*, which are the percentages of elements that can be affected by replacement and swap mutations; and the *maximum crossover depth*, which is the maximum depth of trees given as input to the crossover operation.

At the end of the evolutionary process, a new population is created to replace the current one. The process is repeated over many generations until the termination criterion (e.g, a predefined maximum number of generations N_g or a problem-specific success measure, such as an intended fitness value for a specific individual) is met.

4.2 Our framework

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 been modified. Pages with higher likelihood should receive higher scores, and thus higher priority in the scheduling process. The GP process we use is adapted from the one applied in (Costa Carvalho et al. 2012) to learn how to mix a set of sources of relevance evidence in a search engine at indexing time. Our framework, called *GP4C—Genetic Programming for Crawling*, is presented in Listing 1. It is an iterative process with two phases: *training* (lines 5–13) and *validation* (lines 14–16).

Listing 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         $\mathcal{F}_t \leftarrow \mathcal{F}_t \cup \{g, i, fitness(i, \mathcal{T})\}$;
11         $\mathcal{B}_t \leftarrow getBestIndividuals(N_b, \mathcal{B}_t \cup \mathcal{F}_t)$;
12         $\mathcal{P} \leftarrow applyGeneticOperations(\mathcal{P}, \mathcal{F}_t, \mathcal{B}_t, g)$;
13     }
14 }
15 $\mathcal{B}_v \leftarrow \emptyset$;
16 For each individual $i \in \mathcal{B}_t$ do
17     $\mathcal{B}_v \leftarrow \mathcal{B}_v \cup \{i, fitness(i, \mathcal{V})\}$;
18 BestIndividual $\leftarrow applySelectionMethod(\mathcal{B}_t, \mathcal{B}_v)$;
```

We build our training and validation sets considering a scenario where we train with an initial set of pages and validate the results with a distinct set of pages. This scenario is closer to the one found in large crawling tasks, such as when performing a crawling to a world wide search engine. In this case, an initial set of pages to build the training set is crawled first, and then a second set of validation pages is crawled. The experimental tests are performed by adopting the resulting function in a third set of pages.

As shown in Listing 1, *GP4C* starts with the creation of an initial random population of individuals (line 5) that evolves generation by generation using genetic operations (line 12). The process continues until the number of generations of the evolutionary process reaches a maximum value given as input. Recall that, 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 created, to avoid selecting individuals that work well in the training set but do not generalize for different pages (a problem known as *over-fitting*), the validation phase is applied. In this phase, the fitness function is also used, but

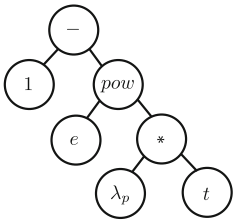


Fig. 2 Tree representing the individual (i.e., change probability function) $1 - e^{\lambda_p t}$

this time over the validation set (lines 16–17). Individuals that perform the best in this phase are selected as the final scheduling solutions (line 18).

4.2.1 Individuals, terminals and functions

In *GP*4*C*, each *individual* represents a function that assigns a score to each page when composing the schedule from the training set. Such score combines information useful for estimating the likelihood of a given page being updated in a period of time, taking into account its behavior in previous crawls. The training is performed in a period of time we selected, and each individual is evaluated as being the function to create the scheduling in the whole training period. Thus, in an individual (tree), *terminals* contain features obtained from the pages that may help in characterizing their updating behavior and thus can be useful as parameters of the score function.

In *GP*4*C*, we consider the values of the following features and constant values as terminals:[1]

- n, the number of times that the page was visited;
- X, the number of times that the page changed in n visits;
- t, the number of cycles since the page was last visited;
- estimators of page change probability previously proposed in the literature and further explained in Sect. 5.1: CG; NAD; SAD; AAD; GAD;
- constant values: 0.001; 0.01; 0.1; 0.5; 1; 10; 100; 1000.

As *functions* in the inner nodes, we use addition ($+$), subtraction ($-$), multiplication ($*$), division ($/$), logarithm (log), exponentiation (*pow*), and the exponential function (*exp*). To ensure the closure property, we implement protected division and logarithm, such that these operators return the default value 0 when their inputs are out of their domains. We also use the genetic operators of reproduction, crossover and (swap/replacement) mutation. In particular, for the crossover operation, the parents are randomly selected among the top best individuals of the current generation.

One possible individual—score function—is shown in Fig. 2. This individual, which represents the score function $1 - e^{\lambda_p t}$ and is here referred to as i, might have been generated after applying some genetic operation in previously generated individuals (line 12 of Listing 1) or might belong to the initial random population (line 5). Let's say i belongs to generation g of the GP process ($g = 1\ldots N_g$). Each individual of this generation, including i, will first be evaluated according to its fitness (line 10). Next, the N_b fittest individuals of all generations up to g will be selected (line 11) to evolve into new individuals after genetic operations are applied on them. These operations are performed over the corresponding

[1] We note that in our preliminary version of this work (Santos et al. 2013), only n, X and t were used as terminals.

trees, as described in Sect. 4. Individual i might or might not be selected, depending on how its fitness compare to the others, computed in the training set. For all selected individual (inserted into $\mathcal{B}_t$ in line 11), a last step consists of computing their fitness values in the validation set (line 17), and the final solution (score function) is selected considering the fitness in both training and validation sets (line 18), as will be described in Sect. 4.2.3.

4.2.2 Fitness function

In *GP*4*C*, 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 experiment with two fitness functions: ChangeRate (Douglis et al. 1997) and NDCG, (Järvelin and Kekäläinen 2002).

ChangeRate was proposed specifically to assess the ability of a scheduling policy to detect updates, thus being previously used with this goal in the literature. It is defined as:

$$C_i = \frac{D_i^c}{D_i}, \tag{1}$$

where D_i^c is the number of webpages downloaded in the ith download cycle that have changed, and D_i is the total number of webpages downloaded in the cycle. The intuition is that the higher the concentration of changed pages in a scheduled set, the better the scheduling. Notice that the ChangeRate definition relies on determining the number of pages to be scheduled and crawled.

Besides ChangeRate, we also propose the use of NDCG as fitness function. NDCG was previously adopted as a function to measure the final quality of the ranking provided by search systems. To our knowledge, this is the first time it is used as a fitness function to assert the quality of a scheduling.

To compute NDCG we need first to define DCG—Discounted Cumulative Gain, which is based on the premise that highly relevant pages appearing lower in the result list should be penalized as the graded relevance value is reduced logarithmically proportional to the position of the result. The DCG accumulated at a particular position k is given by:

$$DCG@k = \sum_{i=1}^{k} \frac{rel_i}{\log_e(i)}, \tag{2}$$

where $rel_i \in \{0, 1\}$.

As the resulting lists vary in length, the cumulative gain at each position for a chosen value k should be normalized. This is done by sorting all the pages from the dataset by putting the changed pages at the top of the ranking and computing the *DCG* of this sorted list, producing the maximum possible *DCG* until position k, called ideal *DCG* until that position (or *IDCG*@k). The NDCG at position k is then defined as:

$$NDCG@k = \frac{DCG@k}{IDCG@k}. \tag{3}$$

We argue that NDCG provides a better fitness function than ChangeRate, since it takes into account not only the number of pages scheduled that were effectively changed, but also the positions where those pages occur in the scheduling. Thus, scheduling functions that place changed pages in the top of the schedule achieve higher NDCG scores. Another advantage of NDCG is that the best functions are selected regardless of the amount of pages effectively crawled. While a good function according to ChangeRate may be bad if the crawler takes only a partial list of the scheduling pages, this is not likely to occur when using functions selected according to NDCG. This latter property is important in practice, since each scheduling may not be completely executed by the crawler before a new scheduling starts. Thus a function that concentrates important pages closer to the top of the scheduling should be scored better than others. While NDCG gives higher scores to such functions, ChangeRate ignores the ranking produced by the scheduler.

Note that both ChangeRate and NDCG are here used not only as fitness functions but also as evaluation metrics, as further discussed in Sect. 5.2.

4.2.3 Selection of the best individuals

We perform the validation step as in (Costa Carvalho et al. 2012). The best individuals are chosen by running the GP process with a set of distinct randomly selected seeds. The whole GP process depends on the selection of the initial seed to produce its results. To reduce the possible risks of finding a low performance local best individual, we run N processes with distinct random seeds, and pick the best individual among those generated by these N runs. We refer to this approach as $GP4C_{Best}$.

As in (de Almeida et al. 2007), we also consider two other selection strategies that are based on the average and the sum of the performances of each individual in both training and validation sets, minus the standard deviation of such performance when selecting best individuals. In (de Almeida et al. 2007), the authors referred to these measures as Avg_σ and Sum_σ. The individual with the highest value of Sum_σ (or Avg_σ) is selected. We here refer to $GP4C$ using these selection strategies as $GP4C_{Sum}$ and $GP4C_{Avg}$.

These specific selection strategies are useful to produce stronger and more stable results when running a GP process.

5 Experimental evaluation

Because of the very dynamic nature of the Web, a crawl simulation is the only way to ensure that all policies are compared under the same conditions. To carry the simulations, we first built a dataset of webpage changes using data gathered from the Web. Our evaluation is performed on a webpage dataset collected from the Brazilian Web (.br domain) using the crawler presented in (Henrique et al. 2011), whose architecture is described in Sect. 3. Table 1 summarizes our dataset, referred to as BRDC'12:[2] it consists of a fixed set of webpages, which were crawled on a daily basis during approximately 2 months (between September and November 2012).

To build BRDC'12, we used as seeds around 15,000 URLs of the most popular Brazilian sites (under the .br domain) according to Alexa.[3] A breadth-first crawl from these seeds gathered around 200 million URLs. We then selected a set of 10,000 websites using stratified random sampling, thus keeping the same distribution of the number of webpages per site of the complete dataset. Next, for each selected site, we chose the largest number of webpages that could be crawled in 1 day without violating politeness constraints. In total, we selected 3,059,698 webpages, which were then daily monitored. The complete BRDC'12 data has about 1 Tb of data.

[2] The BRDC'12 dataset is publicly available at http://www.latin.dcc.ufmg.br/brdc12.html.

[3] http://www.alexa.com/topsites/countries/BR.

Table 1 Overview of our dataset (after filtering errors)

BRDC'12	
Monitoring period	57 days
Number of webpages	417,048
Number of websites	7,171
Minimum number of webpages/site	1
Maximum number of webpages/site	2,336
Average number of webpages/site	58.15
Percentage of downloads with errors	2.92

During the monitoring periods, our crawler ran from 0 A.M. to 11 P.M., recollecting each selected webpage every day, which allowed us to determine when each page was modified. Accesses to the same site were equally spaced to avoid hitting a website too often. We noted some download errors during monitoring periods, which might be due to, for example, the page being removed, the access permissions of the website (robots.txt) being changed, or the download time reaching a limit (30 s). We removed from BRDC'12 all webpages with more than 2 errors.

Note that in the remaining cases of errors we cannot tell whether the page changed on that particular day. To handle such cases, we guess this information by analyzing the history of changes of that page in the days preceding the error. Let us say that the download of page p failed on day d. We use the most frequent period without changes on p in the first $d-1$ days to determine whether we should consider that p changed on day d.

As shown in Table 1, our filtered dataset contains over 400 thousand webpages, which is a much larger number of pages than used by previous work (Tan and Mitra 2010). Note also that the errors that remain after filtering represent only 2.92 % of all downloads performed. Although these errors might somewhat impact the quantitative results of each method, they should not significantly impact our conclusions as they might affect all considered approaches.

5.1 Baselines

We compare $GP4C_{Best}, GP4C_{Sum}$ and $GP4C_{Avg}$ with five estimators of page change probability proposed in the literature. We refer to these baselines as CG, NAD, SAD, AAD and GAD.

Given the number of visits n and the number of times X that a page p changed in those n visits, the CG baseline, proposed by Cho and Garcia-Molina (2003), is defined as :

$$\mathrm{CG} = -\log\left(\frac{n - X + 0.5}{n + 0.5}\right)\cdot \tag{4}$$

The other four baselines were proposed by Tan and Mitra (2010). In order to compute the change frequency of the pages, they assume that the changes in each page p follow a Poisson process with parameter λ_p. Considering T the time that the next change will happen, the probability φ that page p will change in the interval $(0, t]$ is calculated as:

$$\varphi = Pr\{T \leq t\} = \int_0^t f_p(t)dt = \int_0^t \lambda_p e^{-\lambda_p t}dt = 1 - e^{\lambda_p t}. \tag{5}$$

Since φ depends on λ_p and time 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 page:

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

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 ith 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 ith download, or 0 otherwise.

The four baselines vary depending on how weights w_i are computed. Tan and Mitra (2010) proposed the following schemes:

- NAD (*Nonadaptive*): all changes are equally important ($w_1 = \cdots = w_n = \frac{1}{n}$).
- 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 ($w_i = \frac{i}{\sum_{i=1}^{n} i} = \frac{2i}{n(n+1)}$).
- GAD (*Geometrically adaptive*): as the previous scheme, but weights decrease more quickly, following a geometric progression ($w_i = \frac{2^{i-1}}{\sum_{i=1}^{n} 2^{i-1}} = \frac{2^{i-1}}{2^n - 1}$).

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).

5.2 Experimental methodology

We adopted a 5-fold cross validation. Four folds were equally divided into *training set* and *validation set*, and the last fold was used as *test set*. The training set was used to evolve the population in the GP process, and the *validation set* was used to choose the best individuals (as discussed in Sect. 4.2.3), particularly to compute Avg_σ and Sum_σ. The best individuals were evaluated using the *test set*.

Our experimental design differs from traditional 5-fold cross validation though as it also respects *temporal constraints*. In other words, we first divide the pages into 5 folds, say f_i for $i = 1, 2, 3, 4, 5$. We then further divide each fold into three subfolds, namely $f_{i,1}$, $f_{i,2}$ and $f_{i,3}$, where $f_{i,1}$ contains the first 19 days of pages in fold f_i, $f_{i,2}$ contains the next 19 days, and $f_{i,3}$ contains the final 19 days of the same set of pages. We then use pages in $f_{i,1}$ $(i = 1, 2, 3)$ as training set, pages in $f_{4,2}$ as validation set, and pages in $f_{5,3}$ as test set. By doing so we guarantee that validation is always performed using information collected *after* training, and testing is performed using information collected *after* validation.

We repeat this process 5 times, by shifting the folds f_i such that each fold is used as source of the test set once. After each shifting, we still respect temporal constraints as discussed above, that is, we use subfolds $f_{*,1}$ as training set, $f_{*,2}$ as validation set, and $f_{*,3}$ as test set. Thus, 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 (i.e., most likely of having been modified) is then simulated by updating statistics of the page such as number of visits (i.e., downloads), number of changes, etc.

Specifically, we use collected data up to day $d-1$ to predict if a page change should be detected on day d, and to produce the ranking (i.e., scheduling) according to the particular method being evaluated. Given the ranking, we simulate the download of the top-k pages by incrementing their numbers of visits by 1. We also update the number of changes of those pages depending on whether the page actually changed on that day. Thus, in a sense, we are "simulating" exactly what would happen in a real experiment.[4]

We evaluate the score functions generated by our method as well as the baselines using both ChangeRate and NDCG, defined in Sect. 4.2.2. When evaluating a scheduling, one must determine the number of pages that can be crawled on each day (k in our experiments). We set k equal to 5 % of the total number of webpages in the dataset. For each day, we compute ChangeRate and NDCG using the top-k pages in the sorted list produced by each method. Similarly, during the learning process, the fitness of the generated score functions are computed over the top-k pages. Whenever the actual number of changed pages on a day is smaller than k, no evaluated algorithm can reach a maximum ChangeRate. This particular detail may cause variations in the ChangeRate obtained by a function when comparing results in distinct days. This variation however does not affect our conclusions about the relative performance of the methods.

Parameterization of the GP framework was performed by using parameter values similar to those adopted in the related literature. Specifically, as in de Almeida et al. (2007), Costa Carvalho et al. (2012), we adopted the ramped half-and-half method (Koza 1992), a number of generations N_g equal to 50 as termination criterion, a tournament selection of size 2 to select individuals to evolve, and a number of random seeds equal to 5. We also set the crossover, reproduction, replacement mutation and swap mutation rates equal to 90, 15, 5 and 5 %, respectively, 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. These values are similar to those adopted in (de Almeida et al. 2007; Costa Carvalho et al. 2012). The main differences regard the number of individuals in each generation N_p and the maximum tree depth d. Compared to de Almeida et al. (2007), Costa Carvalho et al. (2012), we reduced N_p from 1,000 to 300 and d from 17 to 10. These two changes allowed a reduction in the evolving time while still producing results that are at least as good as (and often superior to) the baselines, as shown in the next section. Further optimization of the GP parameters could be carried out in a future work, which might lead to even better results.

5.3 Experimental results

We now discuss the results produced by our *GP*4*C* framework and the baselines using the BRDC'12 dataset.

[4] One interesting experiment consists of using inaccurate statistics about the pages to produce the schedulings. We note that such inaccuracies would affect all methods, including the baselines. Thus, we conjecture that our main conclusions remain the same, although a careful investigation must be conducted to support this claim. Such study is left for future work.

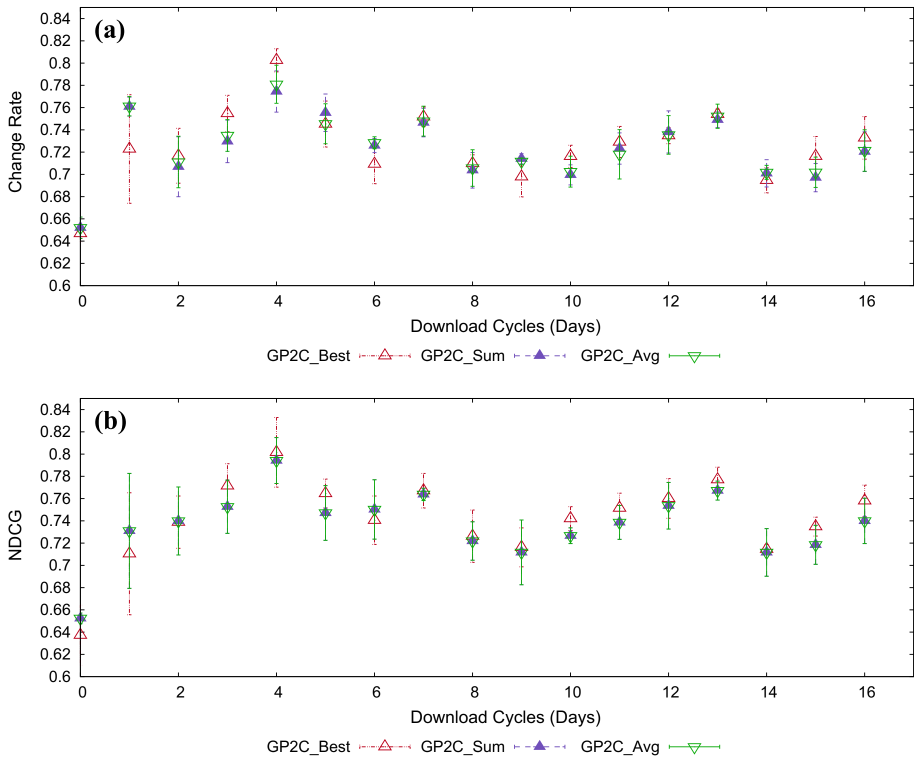


Fig. 3 (Color online) Performance of three variations of *GP4C* in each download cycle (averages across folds and 95 % confidence intervals). **a** Average ChangeRate, **b** average NDCG

We start by comparing our three strategies to select the best individuals, namely, $GP4C_{Best}, GP4C_{Avg}$ and $GP4C_{Sum}$. Figure 3a, b show average ChangeRate and average NDCG (along with corresponding 95 % confidence intervals) for each download cycle (day) of the test set, computed across all 5 test sets (folds). We here fix the evaluation metric and objective function (fitness) as the same, that is, ChangeRate results are obtained for score functions produced by optimizing ChangeRate, whereas NDCG results are shown for score functions developed by optimizing NDCG. We compare the use of different metrics as objective function at the end of this section.

Figure 3 shows that, for both metrics, all three strategies are very close to each other. A pairwise *t* test (Jain 1991) indicated that all three methods are statistically tied, with 95 % confidence, for almost all days. One exception is $GP4C_{Best}$, which, in terms of ChangeRate, outperforms $GP4C_{Avg}$ in 1 day and $GP4C_{Sum}$ in 2 days (with statistically significant differences).

We also note that the functions generated by *GP4C* are quite stable when changing the set of pages where they are applied. Indeed, the results obtained in the test set are pretty close to the ones obtained in the training and validation sets (differences under 1 %) in all folds, whereas differences across folds are also very small (below 3 %). These results indicate that *GP4C* has produced quite stable and generic functions, which is one of the properties desired when applying machine learning solutions to any problem. Considering

Table 2 Overall performance of all methods across all download cycles (averages and 95 % confidence intervals; best results in bold)

Rand	*Age*	NAD	SAD	AAD	GAD	CG	$GP4C_{Basic}$	$GP4C_{All}$
Average ChangeRate								
0.1566 ± 0.0019	0.1753 ± 0.0016	0.7162 ± 0.0033	0.5451 ± 0.0107	0.6778 ± 0.0071	0.6535 ± 0.0059	0.6672 ± 0.0036	0.7164 ± 0.0068	**0.7256 ± 0.0061**
Average NDCG								
0.1565 ± 0.0018	0.1754 ± 0.0015	0.7288 ± 0.0033	0.5674 ± 0.0098	0.6986 ± 0.0062	0.6769 ± 0.0054	0.6888 ± 0.0037	0.7287 ± 0.0042	**0.7419** ± 0.0077

Table 3 Number of victories and losses, in terms of both metrics, of our *GP*4*C* framework over each baseline (relative differences in parentheses)

Baseline	$GP4C_{Basic}$		$GP4C_{All}$	
	# victories (% diff)	# losses (% diff)	# victories (% diff)	# losses (% diff)
ChangeRate				
NAD	3 (3.97)	2 (−3.02)	9 (3.54)	1 (−6.36)
AAD	11 (8.20)	1 (−1.60)	14 (9.04)	1 (−1.19)
GAD	15 (11.80)	1 (−1.60)	15 (13.18)	1 (−1.19)
CG	12 (8.50)	0	16 (9.32)	0
NDCG				
NAD	0	2 (−0.83)	9 (4.21)	0
AAD	13 (6.21)	3 (−2.10)	14 (8.14)	0
GAD	13 (10.23)	2 (−1.67)	15 (11.71)	0
CG	15 (6.48)	0	15 (8.35)	0

overall results in both metrics, $GP4C_{Best}$ is the best performer, and thus we focus on it in the rest of the paper.

One of the advantages of the GP framework is its ability to include new terminals and features to improve the quality of the scheduling. In a preliminary version of this work (Santos et al. 2013), we experimented with only three terminals, namely the number of times that the page was visited (n), the number of times that the page changed in those n visits (X), and the number of cycles since the last visit (t). We here demonstrate the importance of the aforementioned property by extending this list to include also the CG, NAD, SAD, AAD, and GAD estimators of the page change probability as terminals.

Table 2 shows average ChangeRate and NDCG results, along with corresponding 95 % confidence intervals, for our GP scheduling framework using only the basic terminals and using all terminals available, referred to as $GP4C_{Basic}$ and $GP4C_{All}$, respectively. These averages were computed across all days in the test sets. Once again, the results of each evaluation metric were computed for score functions developed by optimizing the same metric. The table also shows results for the seven considered baselines: *Rand*, *Age*, NAD, SAD, AAD, GAD, and CG.

As it can be seen, $GP4C_{All}$ produces better results than $GP4C_{Basic}$ in both evaluation metrics. More importantly, our solution, even if using only the basic terminals, outperforms all baselines (but NAD). We note that these improvements are statistically significant, with 95 % confidence. Our solution is statistically tied with NAD when using only the basic terminals ($GP4C_{Basic}$), outperforming it when all terminals are used ($GP4C_{All}$), with gains of 1.3 and 1.8 % in average ChangeRate and average NDCG, respectively. Note also that, even though our $GP4C_{Basic}$ approach uses the exact set of parameters used by the *CG* baseline (Cho and Garcia-Molina 2003) (i.e., n, X, t), our method produces much better results, increasing the average ChangeRate from 0.667 to around 0.716, an improvement of around 7.4 %, and the average NDCG from 0.689 to 0.729, a 5.8 % improvement.

The best baseline is NAD, followed by CG and AAD, whereas *Rand* and *Age* produce much worse results. NAD uses a different set of parameters compared to CG, which may provide more useful information about a page's updating behavior. Nevertheless, our approach, using the complete set of terminals, provides statistically significant improvements over NAD, as discussed above.

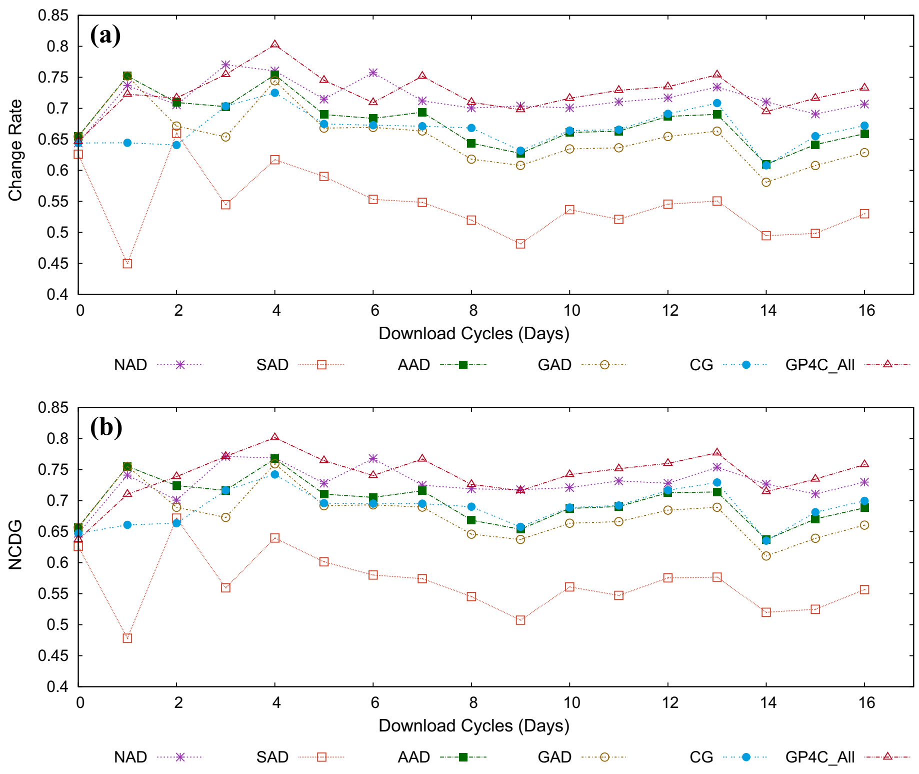


Fig. 4 (Color online) Performance of various methods on each download cycle (averages across folds and 95 % confidence intervals). **a** Average ChangeRate, **b** average NDCG

We note that, although it would be surprising if our solution, when using all terminals, performed worse than the baselines (since it explores all features used by them), it is not necessarily expected that it outperforms all baselines with statistically significant gains. It might have been just as good as the best baseline (e.g., the best score solution could have been one that explores mostly the features used by the best baseline). Instead, our results show the benefits from applying a learning technique to combine the terminals into a single score function.

We also compare the performance of the methods on each download cycle. Figure 4a, b show the average ChangeRate and the average NDCG of each method on each cycle. To improve the readability of the figure, we omit the curves for the *Rand* and *Age* baselines, which are much worse than the other methods, as well as confidence intervals. We also show results for *GP*4*C* only when using all terminals ($GP4C_{All}$).

Note that all curves exhibit similar shapes, with peaks coinciding mostly on the same days.[5] Those peaks are most probably due to a larger number of pages that changed on those particular days in our dataset. Indeed, the Pearson correlation coefficient between the number of changed pages and the NDCG obtained with $GP4C_{All}$ on each day is 0.42. The correlation between the number of changed pages and ChangeRate is even higher (0.56). Similar values were obtained for all other methods. Such reasonably strong (positive)

[5] Those peaks can also be noted in Fig. 3.

Table 4 Best score functions produced by both $GP4C_{Basic}$ and $GP4C_{All}$ in the 5 test sets, for both fitness function

Test set	ChangeRate	NDCG
$GP4C_{Basic}$		
1	$1 + X/n - 2.73/(t + 2.72) - 10$	$(t * X)^{100^{0.1}}$
2	$X + log(t * logX) - n$	$t * (2.72 * e^{t})^{X}$
3	$e^{tX} * log(1000)$	$(log(100) + 0.01) * t * X$
4	$(t * X)^{100.5-t}$	$t * X * log(e)$
5	$t * X * (X/n + 10^{0.01}) * e^{0.1}$	$2.72 * t * X/100$
$GP4C_{All}$		
1	$e^{CG*(1+log(t))}$	$(e^{GAD-0.01} - e^{GAD/100-1000} + NAD) * t * CG$
2	$pow(99.5^{e^{GAD}}, (t + GAD)^{CG})$	$1000 * t * CG^{2.72}$
3	$CG * t * log(1000 * AAD)$	$(CG - AAD) * (t/2.72 + 1)$
4	$1000 * t * CG^{2.72}$	$t * CG^{2.72}$
5	$GAD * e^{NAD+t}$	$GAD * e^{NAD+t}$

correlations between number of changed pages and scheduling effectiveness can be explained as follows: as the number of changed pages increases, all scheduling functions tend to improve their results since the chance of downloading a page that actually changed also increases.

We summarize the results in Fig. 4 by counting the number of download cycles when each method provides the best scheduling, according to one of the evaluation metrics. Table 3 shows the number of cycles (out of a total of 17) in which our solution outperforms each baseline (referred to as *victories*) as well as the number of cycles in which our solution is outperformed by a baseline (referred to as *losses*). It shows results for both evaluation metrics as well as for both $GP4C_{All}$ and $GP4C_{Basic}$. The number of statistical ties (with 95 % confidence) are omitted as they can be easily inferred. We focus on the most competitive baselines, omitting results for *Rand* and *Age*, as both $GP4C_{Basic}$ and $GP4C_{All}$ greatly outperforms them in all download cycles. Similarly, Table 3 also shows, in parentheses, the relative differences (improvements and losses) between our strategies and each baseline for the cases of victories and losses.

Once again, we can see that *GP4C* is superior to all baselines in most of the days. Focusing on the best baseline, NAD, we notice that *GP4C* achieves better average ChangeRate results in 3 download cycles when using only the basic terminals, while the use of the additional terminals raises the number of victories (in terms of this metric) to 9. In terms of NDCG, the use of the additional terminals improves the number of victories from 0 to 9.

The gains can also be assessed in terms of the performance gap between our approaches and NAD. Table 3 shows that the use of only the basic terminals produces improvements in ChangeRate that, when statistically significant, reach 3.97 % on average. Moreover, the use of all terminals produces average gains in both ChangeRate and NDCG that, when statistically significant, reach 3.54 and 4.21 %, respectively. Overall, despite a performance loss in at most one download cycle, our solution provides statistically significant gains over NAD, if all terminals are used. These results illustrate the ability of our framework to take advantage of previous proposals and to include new features into the scheduling task.

Considering the other baselines, specifically the more competitive CG, AAD and GAD, *GP*4*C*, even if using only the basic terminals, produces statistically significant improvements in the vast majority of the days. For example, considering ChangeRate, $GP4C_{Basic}$ outperforms AAD, GAD, CG in 11, 15 and 12 download cycles, respectively. The results in terms of NDCG are somewhat similar (13, 13 and 15 download cycles, respectively). When using all terminals, the number of victories in both metrics as well as corresponding performance differences increase slightly. We note that the only few download cycles in which *GP*4*C* is statistically inferior to one of the baselines (at most 3 cycles) occur in the beginning of the simulation: after the 6th cycle, our solution remains the best one in all successive download cycles. This result corroborates the flexibility of our framework as it is able to produce results as good as, and often better than, all five baselines in the vast majority of the cycles.

We also note that our *GP*4*C* framework can be used for better understanding the scheduling problem. To illustrate this point, Table 4 presents the best score functions produced by $GP4C_{Basic}$ and $GP4C_{All}$ with both fitness functions, in each test set. Note that the best results produced by $GP4C_{Basic}$ were obtained with score functions that, in general, tend to give more importance to the number of cycles since the page was last visited (t) and to the number of times it changed in the last visits (X), particularly if NDCG is used as fitness function. Overall these functions are reasonably simple, and they are able to produce results that are at least as good (if not significantly better) than all baselines, as discussed above. The small impact of n on the final scores of such functions might be due to the period of crawling used for training being only 17 days (ignoring the initial $W = 2$ days of warm-up). For longer periods, n may become more important. Even though experiments with longer periods are now unfeasible to us (as they require continuous crawling), this result shows the ability of our method to adapt its functions to the dataset given for training.

As another example, a simple but effective function generated by our method is $t * X$. This function resulted in final performance superior to most of the baselines, with average ChangeRate above 0.70. It was not the best function found by *GP*4*C*, 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.

Table 4 also shows the best score functions produced by $GP4C_{All}$. Note that most functions combine multiple baselines (NAD and GAD, or CG and GAD). Once again, this shows the benefits of using a learning technique—GP in particular—to combine multiple features into a single score function.

Next, we assess the possible advantages of using NDCG instead of ChangeRate as the fitness function. The choice of NDCG as fitness function and also as evaluation function is based on the observation that crawlers commonly may not follow a scheduler up to its end, requesting and adopting sometimes a new scheduling before the current one is completely executed. In such situations, we are interested in schedulings that still perform well even if just a portion of them is executed. In that case, optimizing NDCG might lead to more effective solutions as the metric prioritizes functions that place changed pages in higher positions of the scheduling.

Recall that, up to this point, we have evaluated all methods in the top-k % of the pages in the sorted list produced by each method (with k equal to 5 %), simulating a scenario in which the scheduling is interrupted after downloading those pages. To assess the benefits of using NDCG as fitness function, we compare the performance of the best score function obtained when using either metric as fitness, considering smaller values of k. Note that, even though we evaluate both functions assuming smaller values of k, they were calibrated

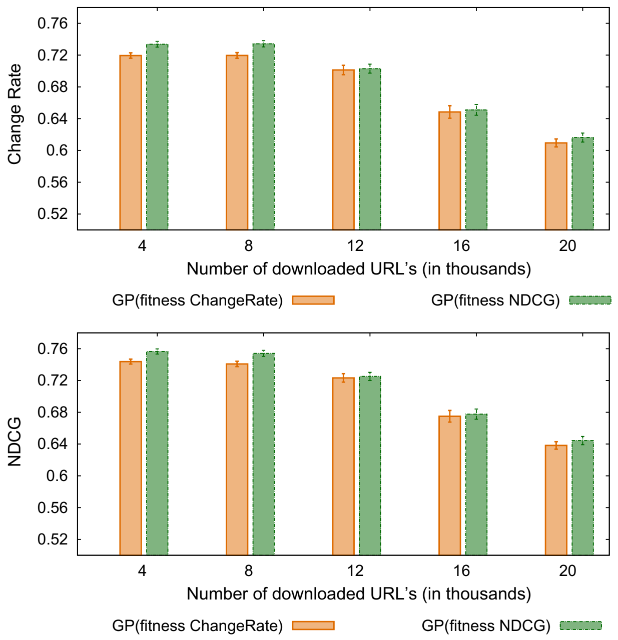
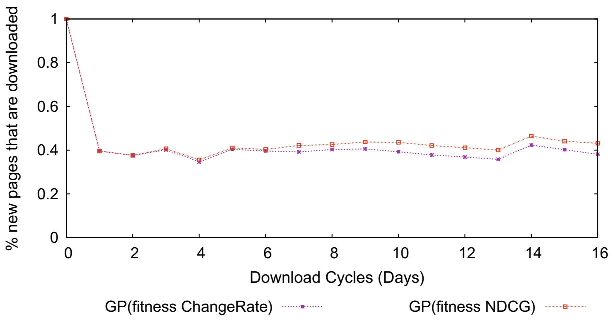


Fig. 5 Average ChangeRate and NDCG for 4 to 20 thousand downloaded pages across all download cycles in the test sets, which corresponds to k varying from 1 to 5 % in the top-k pages in the sorted list produced by each method

Fig. 6 Fraction of new pages downloaded on each day when $GP4C_{All}$ is used with NDCG as fitness function



assuming a fixed $k = 5$ %. In other words, the score functions were learned assuming that 5 % is the maximum fraction of pages that can be downloaded in one cycle. Figure 5 presents average ChangeRate and NDCG results, along with corresponding 95 % confidence intervals, for a total number of downloaded pages varying from 4 to 20 thousand, which corresponds to values of k varying from 1 to 5 %.

As we can see, the use of NDCG as fitness function produces statistically better results (up to 2 % improvements), compared to those produced when ChangeRate is optimized. These improvements in the scheduling are observed in terms of both evaluation metrics. Thus, the use of NDCG as fitness leads to more robust solutions, particularly when the amount of resources dedicated to scheduling is too constrained, leading to early interruptions in the execution.

One final issue we address regards whether the ranking produced by our solution consists mostly of the same pages, which are often updated, or whether it is able to bring new pages to the top of the ranking. To address this issue, we measured the fraction of *new* pages that are downloaded on each day when $GP4C_{All}$ is used to produce the schedulings. A page is considered *new* if it has not been downloaded in any of the previous cycles, i.e., it did not appear in the top-k% positions of the rankings produced on any of the previous days. Figure 6 shows the results produced with both ChangeRate and NDCG as fitness functions for one test set. These results are representative of those produced for all test sets. Note that, after the first day, when all downloaded pages are new, the fraction of new pages remains roughly stable, around 40 %. This holds for both fitness functions. In other words, a large fraction of the pages that are downloaded on each day had not been downloaded on any of the previous days, and this fraction remains stable throughout the period covered by the test set. Thus, these results illustrate that $GP4C_{All}$ is able to discover a large fraction of new pages that are very likely of having being modified (and thus should be downloaded).

6 Conclusions and future work

In this article, 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 having been modified since they were last crawled. We have extensively evaluated our framework, called *GP4C*, comparing it against various baselines, including state-of-the-art page change probability estimators, using a webpage dataset collected from the Brazilian Web. Our evaluation, which considered both ChangeRate and NDCG as performance metrics, included several variations of our framework built from different sets of terminals as well as fitness functions.

Our experimental results indicate that our best function, $GP4C_{Best}$, is statistically superior to most baselines in most of the simulated download cycles, even when a basic set of terminals is used. Further improvements, in both ChangeRate and NDCG, are achieved when the set of terminals is extended to include the previously proposed page change probability estimators. Our results also show that the use of NDCG as fitness function, as opposed to ChangeRate, leads to statistically superior results, which makes it a more robust approach, particularly when the schedulings are only partially executed.

As future work, we plan to investigate the use of our *GP4C* framework to derive functions that increase the chance of finding novel pages, thus giving priority to coverage, as well as functions that balance the two main objectives of a scheduler, freshness and coverage.

Acknowledgments We thank the partial support given by 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' individual grants and scholarships from CNPq and FAPEMIG.

References

Carvalho, A., Rossi, C., de Moura, E. S., Fernandes, D., & da Silva, A. S. (2012). LePrEF: Learn to pre-compute evidence fusion for efficient query evaluation. *Journal of the American Society for Information Science and Technology*, *55*(92), 1–28.

Cho, J., & Garcia-Molina, H. (2000). Synchronizing a database to improve freshness. *SIGMOD Record*, *29*(2), 117–128.

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

Cho, J., & Ntoulas, A. (2002). Effective change detection using sampling. In *28th international conference on very large data bases*, pp. 514–525.

Coffman, E. G., Liu, Z., & Weber, R. R. (1998). Optimal robot scheduling for web search engines. *Journal of Scheduling*, *1*(1), 15–29.

da Costa Carvalho, A. L., Rossi, C., de Moura, E. S., da Silva, A. S., & Fernandes, D. (2012). Lepref: Learn to precompute evidence fusion for efficient query evaluation. *Journal of the American Society for Information Science and Technology*, *63*(7), 1383–1397.

de Almeida, H. M., Gonçalves, M. A., Cristo, M., & Calado, P. (2007). A combined component approach for finding collection-adapted ranking functions based on genetic programming. In *30rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. pp. 399–406.

Douglis, F., Feldmann, A., Krishnamurthy, B., & Mogul, J. (1997). Rate of change and other metrics: A live study of the world wide web. In *USENIX symposium on internet technologies and systems*, pp. 14–14.

Fan, W., Fox, E. A., Pathak, P., & Wu, H. (2004a). The effects of fitness functions on genetic programming-based ranking discovery for web search. *Journal of the American Society for Information Science and Technology*, *55*(7), 628–636.

Fan, W., Gordon, M., Pathak, P., Xi, W., & Fox, E. (2004b). Ranking function optimization for effective web search by genetic programming: An empirical study. In *37th Hawaii International Conference on System Sciences*, pp.105–112.

Fan, W., Gordon, M. D., & Pathak, P. (2004c). Discovery of context-specific ranking functions for effective information retrieval using genetic programming. *IEEE Transactions on Knowledge and Data Engineering*, *16*(4), 523–527.

Fetterly, D., Craswell, N., & Vinay, V. (2009). The impact of crawl policy on web search effectiveness. In *32nd international ACM SIGIR conference on research and development in information retrieval*, pp. 580–587.

Henrique, W. F., Ziviani, N., de Cristo, M. A. P., de Moura, E. S., da Silva, A. S., & Carvalho, C. (2011). A new approach for verifying url uniqueness in web crawlers. In *18th international symposium on string processing and information retrieval*, pp. 237–248.

Jain, R. (1991). *The art of computer systems performance analysis: Techniques for experimental design, measurement, simulation, and modeling*. London: Wiley-Interscience.

Järvelin, K., & Kekäläinen, J. (2002). Cumulated gain-based evaluation of ir techniques. *ACM Transactions on Information Systems*, *20*(4), 422–446.

Koza, J. R. (1992). *Genetic programming: On the programming of computers by means of natural selection*. Cambridge: MIT Press.

Olston, C., & Najork, M. (2010). Web crawling. *Foundations and Trends in Information Retrieval*, *4*(3), 175–246.

Radinsky, K., & Bennett, P. (2013). Predicting content change on the web. In *6th ACM international conference on web search and data mining,* pp. 415–424.

Santos, A. S. R., Ziviani, N., Almeida, J. M., Carvalho, C., de Moura, E. S., & da Silva, A. S. (2013). Learning to schedule webpage updates using genetic programming. In *20th international symposium on string processing and information retrieval*, pp. 271–278.

Silva, T. P. C., de Moura, E. S., Cavalcanti, J. M. B., da Silva, A. S., de Carvalho, M. G., & Gonçalves, M. A. (2009). An evolutionary approach for combining different sources of evidence in search engines. *Information Systems*, *34*, 276–289.

Tan, Q., & Mitra, P. (2010). Clustering-based incremental web crawling. *ACM Transactions on Information Systems*, *28*, 17:1–17:27.

Trotman, A. (2005). Learning to rank. *Information Retrieval*, *8*(3), 359–381.