Computing web page importance without storing the graph of the web (extended abstract)

S. Abiteboul
INRIA and Xyleme

M. Preda
Xyleme

G. Cobena
INRIA

Abstract

The computation of page importance in a huge dynamic graph has recently attracted a lot of attention because of the web. Page importance or page rank is defined as the fixpoint of a matrix equation. Previous algorithms compute it off-line and require the use of a lot of extra CPU as well as disk resources in particular to store and maintain the link matrix of the web. We briefly discuss a new algorithm that works on-line, and uses much less resources. In particular, it does not require storing the link matrix. It is on-line in that it continuously refines its estimate of page importance while the web/graph is visited. When the web changes, page importance changes as well. We modify the algorithm so that it adapts dynamically to changes of the web. We report on experiments on web data and on synthetic data.

1 Introduction

All the pages on the web do not have the same "importance". For example, Le Louvre homepage is more important that an unknown person's homepage. Page importance information is very valuable. It is used by search engines to display results in the order of page importance [4]. It is also useful for guiding the refreshing and discovery of pages: important pages should be refreshed more often[1] and when crawling for new pages, important pages have to be fetched first [2]. Following some previous ideas of [7], Page and Brin proposed a notion of page importance based on the link structure of the web [11]. This was then used by Google with a remarkable success. Intuitively, a page is important if there are many important pages pointing to it. This leads to a fixpoint computation by repeatedly multiplying the matrix of links between pages with the vector of the current estimate of page importance until the estimate is stable, i.e., until a fixpoint is reached.

The main issue in this context is the size of the web, billions of pages [6, 8]. Techniques have been developed to compute page importance efficiently, e.g., [5]. The web is crawled and the link matrix computed and stored. A version of the matrix is then frozen and one separate process computes page importance, which may take hours or days for a very large graph. So, the core of the technology for the off-line algorithms is fast sparse matrix multiplication (in particular by extensive use of parallelism). This is a classical area, e.g., [12]. Our algorithm computes the importance of pages on-line while crawling the web.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

[1] Google seems to use such a policy for refreshing pages. Xyleme [13] does.

Intuitively speaking, some "cash" is initially distributed to each page and each page when it is crawled distributes its current cash equally to all pages it points to. This fact is recorded in the history of the page. The importance of a page is then obtained from the "credit history" of the page. The intuition is that the flow of cash through a page is proportional to its importance. It is essential to note that the importance we compute does not assume anything about the selection of pages to visit. If a page "waits" for a while before being visited, it accumulates cash and has more to distribute at the next visit.

The web changes all the time. With the off-line algorithm, we need to restart a computation. Although techniques can be used to take into account previous computations, several costly iterations over the entire graph have to be performed by the off-line algorithm. We modify our on-line algorithm to adapt to changes. Intuitively, this is achieved by taking into account only a recent window of the history.

We report on experiments with variants of our algorithms, synthetic data and real data obtained from real crawls of the web. One of the algorithms presented here is used in production by the Xyleme company [13].

The present paper is just an extended abstract. The full paper may be obtained from [1]. The paper is organized as follows. In Section 2, we introduce the algorithm focusing on static graphs. In Section 3, we move to dynamic graphs, i.e., graphs that are continuously updated like the web. The following section deals with implementation and discusses some experiments. The last section is a conclusion.

To conclude this introduction, we recall the standard notion of page importance that we use here.

Page importance or page rank is defined as the fixpoint of a matrix equation [11, 10]. Let G be a graph (e.g., the web) represented as a link matrix $L[1..n, 1..n]$. If there is an edge for i to j, $L[i, j] = 1$, otherwise it is 0. Let $out[1..n]$ be the vector of out-degrees. Let L' be the matrix defined by, for each i, j, $L'[i, j] = \frac{L[i,j]}{out[i]}$. The importance Imp of nodes as defined by [11] is given by:

$$Imp[j] = \sum_i (L'[i, j] * Imp[i])$$

This definition and the computation as a fixpoint are mathematically founded. One can show that the importance of a page is the probability to be on that page after an infinite random walk on the graph. So it is the steady vector of a Markov chain with a transition matrix L'. The importance can be computed as the limit of X_n for $X_n = L' * X_{n-1}$ starting, for instance, from a vector that gives equal importance to all pages. To be more precise, this holds under some hypothesis: (i) the graph has to be [9] aperiodic (a reasonable assumption on the web) and (ii) strongly connected. The mathematical model is slightly modified to take into account the fact that the web is not strongly connected. Details may be found in [1].

2 The algorithm for static graphs

We present in this section the algorithm for the case of a static graph (no update).

To start, we distribute some initial cash to each node, e.g., if there are n nodes, we distribute $1/n$ to each node. For every page, we keep two values. We call the first *cash*. In some sense, it records the recent information discovered about the page, more precisely, the sum of the cash obtained by the page since the last time it was crawled. We also record the *(credit) history* of the page, the sum of the cash obtained by the page until the last time it was crawled. The cash is stored in main memory whereas the history may be stored on disk. When a page is retrieved by the web agent, we know the pages it points to. In other words, we have at no cost the link information for the retrieved page. We record its cash in the history, i.e., we add it to the history. We also distribute this cash equally between all pages it points to. We reset the cash of the page to 0. This happens each time we read a page. We will see that this provides enough information to compute the importance of the page as used in standard methods. We will consider in a further section how this may be adapted to handle dynamic graphs.

More precisely, we use two vectors $C[1..n]$ (the cash) and $H[1..n]$ (the history). The initialization of C has no impact on the result. A more detailed history will be needed only when we move to an adaptive version of the algorithm. For now, $H[i]$ is simply a number that accumulates all the cash page i received. As history H is stored on disk, $H[i]$ is only updated when page i is crawled. The algorithm works as follows:

```
On-line Page Importance Computation (OPIC) Algorithm

for each i let C[i] := 1/n ;
for each i let H[i] := 0 ;
let t:=0 ;
let G:=0 ;
do forever
    begin
    t := t+1;
    choose some node i ;
    %% assuming each node is selected infinitely often
    H[i] += C[i];
    for each child j of i, C[j] += C[i]/out[i];
    G += C[i];
    C[i] := 0 ;
    end
```

Note that the algorithm does not impose any requirement on the order we visit the nodes of the graph as long as each node is visited infinitely often, so it can be applied "online" i.e. when pages are crawled. This dramatically reduces the system complexity and the usage of resources. At the time we visit a node (we crawl it), the list of its children is available on the document itself and does not require disk access. As long as the cash of children is stored in main memory, no disk access is necessary to update it.

Let C_t and H_t be the values of C and H at the end of the t-th step of the algorithm. The vector C_0 denotes the value of C at initialization (all entries are $1/n$). Let X_t be defined by:

$$X_t = \frac{H_t}{(\sum_i H_t[i])}, \text{i.e., } \forall j, X_t[j] = \frac{H_t[j]}{(\sum_i H_t[i])}$$

One can prove that, when t goes to infinity, $|H_t|$ goes to infinity and for each j,

$$\left|(L' * X_t)[j] - X_t[j]\right| < \frac{1}{\sum_i H_t[i]}$$

and $|X_t| = 1$. Thus the vector X_t converges to the vector of importance, i.e., to Imp.

This shows that

$$\frac{H_t[j] + C_t[j]}{(\sum_i H_t[i]) + 1}$$

provides an estimate for the importance of a page, and that it converges to it.

In other words, at each step of the computation, an estimate of the importance of page k is given by $(H[j] + C[j])/(G+1)$.

The OPIC algorithm uses only local information, i.e. the outgoing links of the page that is being crawled that can be found in the page as URLs. Thus our algorithm presents the following advantages: (i) it requires less storage resources than standard algorithms; (ii) it also requires less CPU, memory and disk access than standard algorithms; (iii) it is less complex because the web agent and the evaluator of page importance can run in a single process.

As mentioned earlier, OPIC converges independently on how pages are selected (assuming each page is selected infinitely often). However, the speed of convergence depends on this selection. For instance, consider the following two refresh strategies:

1. RANDOM: We choose the next page to crawl randomly with equal probability.
2. GREEDY: We read next the page with highest cash. This is a greedy way to increase G for each page selected. (The exact policy used in Xyleme is more complex than GREEDY but comes very close to it.)

It turns out that, as we shall see in Section 4, the convergence is much faster with GREEDY.

Note that the GREEDY selection strategy suggests a totally different way of computing importance. It turns out that if GREEDY is used, the importance of a page coincides with the frequency with which the page is read. If GREEDY is used, it thus suffices to store the times of the visits of the page to obtain that frequency and thus an estimate of the importance of the page.

3 Dynamic graph

In this section, we consider the case of a changing graph and modify the algorithm to adapt to changes.

Consider a dynamic graph (the case of the web). Pages come and disappear and edges too. To simplify, we will ignore here the changes of the set of nodes and focus on the changes of edges. So, we assume that the set of nodes is fixed. Note that the importance of a page is now a moving target and that we should not expect to obtain a perfect estimate. We may only hope to stay close to it if this importance does not change too rapidly, which is typically the case on the web. The problem with OPIC on a changing graph is that its estimate of page importance would typically change slowly and even more slowly when the graph gets older because the past history would simply have too much weight. This suggests using only some recent history, e.g., say the history that has been gathered during the last month, or two months.

To fix the ideas, we have to use some time basis. If we use a clock, results would be influenced by the speed of crawling. So, instead, a more appropriate time basis is the current value of G. So, now we think of the variable G as the current time.

Consider some time instants $t, t+T$. Let $H_{t,t+T}[i]$ be the total of cash added to the history of page i between time t and $t+T$, i.e., $H_{t+T}[i] - H_t[i]$. Let

$$\forall j, X_{t,t+T}[j] = \frac{H_{t,t+T}[j]}{(\sum_i H_{t,t+T}[i])}$$

Because the theorem did not impose any condition on the initial state of X_t, we know that $X_{t,t+T}$ converges to the vector of importance when T goes to infinity[2]. This comes to ignoring the history before time t and start with the state of the cash at time t for initial state.

To adapt to changes on the web, we therefore use time windows. We estimate the importance of a page by looking at the history between $G-T$ and G. We call the interval $[G-T, G]$ the (time) *window*. If we choose a very large window, we get a good estimate of importance (a perfect one with an infinite window: $T \to \infty$). On the other hand, a too large window means that we take into account very old information that may have changed, so our reaction to change may be too slow. Now, a very small window would be very adaptive to changes but may yield a too important error in the estimate. Thus there is a trade-off between precision and adaptability to changes and a critical parameter of the technique is the choice of the size of the window.

To compute page importance for a changing graph, we use the Adaptive OPIC algorithm based on time windows. In Adaptive OPIC, we use the cash vector in the same way as in the case of a static graph. The global variable G records the total sum of cash that has been distributed to the nodes of the graph since the initialization

[2]On the other hand, when t goes to infinity, $X_{t,t+T}$. does not converge to the vector of importance.

of the process (the current time). To avoid having a value that grows infinitely, we keep this value modulo some very large number G_0. (G_0 should be large enough so that no confusion may arise.) For the history, we keep a certain number of measures. We studied two variants of the algorithm based on the following policies:

1. FIXED WINDOW: For every page i, we store the value of cash $C_t[i]$ and the global value G_t for all times it was crawled in a fixed time window, say the last m months.

2. VARIABLE WINDOW: For every page i, we store the value of cash $C_t[i]$ and the global value G_t for the last n times it was crawled.

The Xyleme system is actually using a variant of FIXED WINDOW called BAO (for Basic Adaptive OPIC). Let T be the size of the window. The idea is that the history of a page consists of a single value, an *estimate* of what the page obtained in an interval T before the last crawl. The next time the page is crawled a new estimate is computed by interpolation based on the history and the cash it gathered between the two crawls. Details are omitted.

4 Experiments

We implemented and tested first the standard off-line algorithm for computing Page Importance, then OPIC and several variants of Adaptive OPIC, and BAO. We performed some tests on synthetic data and some on a much larger collection of URLs from the web.

Synthetic data *The graph model:* We performed most of the experiments with (reasonably small) graphs of about 100 000 nodes with a rather simple random distribution of edges. This was to be able to experiment with many variants of the algorithm, many sizes of windows and many patterns of changes (from very small changes to very intense ones). We also performed tests with much larger graphs and with much more complex edge distribution. We found that the results were not depending so much on these two criteria. For instance, to study the impact of the distribution of incoming/outgoing links and the correlations between them, we tried several graph models in the spirit of [3]. Even with significant changes in parameters describing the graphs, the patterns of the results did not change substantially from the simple graph model.

Convergence: We studied the convergence of OPIC for various page selection policies. In particular, we considered RANDOM and GREEDY. We found OPIC converges much faster with GREEDY than with RANDOM. Typically, a “reasonable error” (less than one percent) is obtained with GREEDY when each page has been read about 5 times on average. RANDOM is roughly twice slower. We also compared these, somewhat artificially, to the off-line algorithm. In the off-line, we count N steps for each iteration of the off-line algorithm. With appropriate tuning, OPIC with the GREEDY page selection policy presents the following two advantages over the off-line algorithm:

- it obtains very fast (after reading each page once or twice on average) a reasonable estimate even if the off-line algorithm eventually (after a few iterations) converges faster.

- it is tailored to provide good estimates for important pages which is what it useful in practice.

The fact that the off-line algorithm eventually converges faster (roughly twice faster) is not surprising since more information is available to it at each step.

Impact of the size of the window: As already mentioned, a small window means more reactivity to changes but at the cost of some lack of precision. A series of experiments was conducted to determine how much. Consider for instance the GREEDY policy and VARIABLE WINDOW (we keep the history for the last n crawls of the page). When n is too small (e.g., $n = 2$), the error is too large and we do not converge to a reasonable estimate. Typically, a value of n around 4 seems a reasonable compromise. The error is limited and our reactivity to change is good. Depending on some parameters such as the update rate of the graph, different window sizes perform better.

Variant of the Adaptive algorithm: We compared FIXED WINDOW and VARIABLE WINDOW. We found that VARIABLE WINDOW performs in general better. This is because FIXED WINDOW has a tendency to accumulate more data than needed for important pages and too little for unimportant ones. On the other hand, when the graph changes very rapidly, FIXED WINDOW adapts better to changes. We also found that a a yet better strategy is to use different policies depending on the importance of the page.

The BAO algorithm (interpolation on FIXED WINDOW) turned out to yield surprisingly good results, often even better than VARIABLE WINDOW, while it uses less resources. We believe that the reason for this is that it avoids some “noise” at each crawl of a page resulting from the introduction of a new measure and the deletion of an old one. The interpolation acts, in some sense, as a filter. We selected BAO for the intensive experiment with web data.

Web data We tested BAO with a real crawl of the web. We used a cluster of Linux PCs, between 4 and 8 PCs at various times of the experiment. The experiment lasted several months with the PCs active only about 50% of the time. Each PC can read about 3 million pages per day. The selection of pages was such that pages estimated as important were read first and refreshed more often. (As mentioned earlier, the refresh strategy gives results close to those of GREEDY so a page twice more important is read twice more often.)

We discovered close to one billion URLs. Some pages were never read: only 400 million of them were actually read, so we computed Page Importance over 400 million pages, to a certain extent the 400 million most important pages of the web. The top most important pages were read almost daily. We used BAO, i.e., FIXED WINDOW with interpolation. This experiment was sufficient (with limited human checking of the results) to conclude that the algorithm can be used in a production environment. Indeed, this is the algorithm currently used by Xyleme. During the test, we experimented with various sizes of the times windows. After a trial/error phase, the time window was fixed to three weeks.

More experiments are reported in the full paper.

5 Conclusion

We proposed a simple algorithm to implement with limited resources a realistic computation of page importance over a graph as large as the web. We demonstrated both the correctness and usability of the technique.

To understand more deeply the algorithms, more experiments are being conducted. Also, we are working on a precise mathematical analysis of the convergence speed of the various algorithms. One goal is to obtain variants of Adaptive OPIC that provide better estimates of page importance and also give an estimate of the error and the change rate of this importance. This includes critical aspects such as the selection of the appropriate size for the time window. Other aspects of our algorithm may also be improved such as the management of newly discovered pages that we ignored here.

Acknowledgments We want to thank Laurent Mignet and Tova Milo for discussions on the present work.

References

[1] S. Abiteboul, M. Preda, and G. Cobena. Adaptive on-line page importance computation. *Technical report*, 2002. http://www-rocq.inria.fr/verso.

[2] Junghoo Cho, Hector García-Molina, and Lawrence Page. Efficient crawling through URL ordering. *Computer Networks and ISDN Systems*, 30(1–7):161–172, 1998.

[3] Colin Cooper and Alan M. Frieze. A general model of undirected web graphs. In *European Symposium on Algorithms*, pages 500–511, 2001.

[4] Google. www.google.com/.

[5] T. Haveliwala. Efficient computation of pagerank. *Technical report, Stanford University*, 1999.

[6] A. Broder K. Bharat. Estimating the relative size and overlap of public web search engines. *7th International World Wide Web Conference (WWW7)*, 1998.

[7] Jon M. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5):604–632, 1999.

[8] Steve Lawrence and C. Lee Giles. Accessibility of information on the web. *Nature*, 400:107–109, 1999.

[9] Rajeev Motwani and Prabhakar Raghavan. Randomized algorithms. *ACM Computing Surveys*, 28(1):33–37, 1996.

[10] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web, 1998.

[11] L. Page S. Brin. The anatomy of a large-scale hypertextual web search engine. *WWW7 Conference, Computer Networks 30(1-7)*, 1998.

[12] S. Toledo. Improving the memory-system performance of sparse-matrix vector multiplication. *IBM Journal of Research and Development*, 41(6):711–??, 1997.

[13] Xyleme. www.xyleme.com/.