Available online at www.sciencedirect.com

SCIENCE DIRECT®

LINEAR ALGEBRA AND ITS APPLICATIONS

ELSEVIER

Linear Algebra and its Applications 386 (2004) 51–65

www.elsevier.com/locate/laa

Adaptive methods for the computation of PageRank

Sepandar Kamvar [a,*], Taher Haveliwala [b], Gene Golub [a]

[a] *Scientific Computing and Computational Mathematics, Stanford University, P.O. Box 18544, Stanford, CA 94305, USA*

[b] *Department of Computer Science, Stanford University, Stanford, CA 94305, USA*

Received 12 August 2003; accepted 13 December 2003

Submitted by C. Meyer

Abstract

We observe that the convergence patterns of pages in the PageRank algorithm have a non-uniform distribution. Specifically, many pages converge to their true PageRank quickly, while relatively few pages take a much longer time to converge. Furthermore, we observe that these slow-converging pages are generally those pages with high PageRank. We use this observation to devise a simple algorithm to speed up the computation of PageRank, in which the PageRank of pages that have converged are not recomputed at each iteration after convergence. This algorithm, which we call Adaptive PageRank, speeds up the computation of PageRank by nearly 30%.

Keywords: PageRank; Eigenvalue problem; Web matrix

1. Introduction

One of the best-known algorithms in web search is Google's PageRank algorithm [8]. PageRank computes the principal eigenvector of the matrix describing the hyperlinks in the web using the famous Power Method [1]. Due to the sheer size of the web (over 3 billion pages), this computation can take several days. Speeding up this

* Corresponding author.

E-mail addresses: sdkamvar@stanford.edu (S. Kamvar), taherh@stanford.edu (T. Haveliwala), golub@stanford.edu (G. Golub).

0024-3795/$ - see front matter © 2004 Published by Elsevier Inc.
doi:10.1016/j.laa.2003.12.008

computation is important for two reasons. First, computing PageRank quickly is necessary to reduce the lag time from when a new crawl is completed to when that crawl can be made available for searching. Secondly, recent approaches to personalized and topic-sensitive PageRank schemes [3,6,9] require computing *many* PageRank vectors, each biased towards certain types of pages. These approaches intensify the need for faster methods for computing PageRank.

Accelerating the PageRank algorithm poses many challenges. First, the convergence rate of the Power Method is very fast (generally, $|\lambda_2|/|\lambda_1| = 0.85$ [4]) due to the structure of the Google matrix [4]. The Power Method on a web data set of over 80 million pages converges in about 50 iterations. Improving on this already fast convergence rate is a difficult problem. Further, many other fast eigensolvers (e.g. inverse iteration) are not feasible for this problem because the size and sparsity of the web matrix makes inversion or factorization prohibitively expensive.

In this paper, we make the following simple observation: the convergence rates of the PageRank values of individual pages during application of the Power Method is nonuniform. That is, many pages converge quickly, with a few pages taking much longer to converge. Furthermore, the pages that converge slowly are generally those pages with high PageRank.

We devise a simple algorithm that exploits this observation to speed up the computation of PageRank, called Adaptive PageRank. In this algorithm, the PageRank of pages that have converged are not recomputed at each iteration after convergence. In large-scale empirical studies, this algorithm speeds up the computation of PageRank by nearly 30%.

2. Preliminaries

In this section we summarize the definition of PageRank [8] and review some of the mathematical tools we will use in analyzing and improving the standard iterative algorithm for computing PageRank.

Underlying the definition of PageRank is the following basic assumption. A link from a page $u \in Web$ to a page $v \in Web$ can be viewed as evidence that v is an "important" page. In particular, the amount of importance conferred on v by u is proportional to the importance of u and inversely proportional to the number of pages u points to. Since the importance of u is itself not known, determining the importance for every page $i \in Web$ requires an iterative fixed-point computation.

To allow for a more rigorous analysis of the necessary computation, we next describe an equivalent formulation in terms of a random walk on the directed Web graph G. Let $u \to v$ denote the existence of an edge from u to v in G. Let $\deg(u)$ be the outdegree of page u in G. Consider a random surfer visiting page u at time k. In the next time step, the surfer chooses a node v_i from among u's out-neighbors $\{v|u \to v\}$ uniformly at random. In other words, at time $k+1$, the surfer lands at node $v_i \in \{v|u \to v\}$ with probability $1/\deg(u)$.

The PageRank of a page i is defined as the probability that at some particular time step $k > K$, the surfer is at page i. For sufficiently large K, and with minor modifications to the random walk, this probability is unique, illustrated as follows. Consider the Markov chain induced by the random walk on G, where the states are given by the nodes in G, and the stochastic transition matrix describing the transition from i to j is given by P with $P_{ij} = 1/\deg(i)$.

For P to be a valid transition probability matrix, every node must have at least 1 outgoing transition; i.e., P should have no rows consisting of all zeros. This holds if G does not have any pages with outdegree 0, which does not hold for the Web graph. P can be converted into a valid transition matrix by adding a complete set of outgoing transitions to pages with outdegree 0. In other words, we can define the new matrix P' where all states have at least one outgoing transition in the following way. Let n be the number of nodes (pages) in the Web graph. Let $\vec{v}$ be the n-dimensional column vector representing a uniform probability distribution over all nodes

$$\vec{v} = \left[\frac{1}{n}\right]_{n\times 1}. \tag{1}$$

Let $\vec{e}$ be the n-dimensional column vector where every element $e_i = 1$

$$\vec{e} = [1]_{n\times 1}. \tag{2}$$

Let $\vec{d}$ be the n-dimensional column vector identifying the nodes with outdegree 0

$$d_i = \begin{cases} 1 & \text{if } \deg(i) = 0, \\ 0 & \text{otherwise.} \end{cases}$$

Then we construct P' as follows:

$$\begin{aligned} &D = \vec{d} \cdot \vec{v}^{\mathrm{T}} \\ &P' = P + D. \end{aligned}$$

In terms of the random walk, the effect of D is to modify the transition probabilities so that a surfer visiting a dangling page (i.e., a page with no outlinks) randomly jumps to another page in the next time step, using the distribution given by $\vec{v}$.

By the Ergodic Theorem for Markov chains [2], the Markov chain defined by P' has a unique stationary probability distribution if P' is aperiodic and irreducible; the former holds for the Markov chain induced by the Web graph. The latter holds iff G is strongly connected, which is generally *not* the case for the Web graph. In the context of computing PageRank, the standard way of ensuring this property is to add a new set of complete outgoing transitions, with small transition probabilities, to *all* nodes, creating a complete (and thus strongly connected) transition graph. In matrix notation, we construct the irreducible Markov matrix P'' as follows:

$$\begin{aligned} &E = \vec{e} \cdot \vec{v}^{\mathrm{T}}, \\ &P'' = cP' + (1 - c)E. \end{aligned}$$

In terms of the random walk, the effect of E is as follows. At each time step, with probability $(1 - c)$, a surfer visiting any node will jump to a random Web page (rather than following an outlink). The destination of the random jump is chosen according to the probability distribution given in $\vec{v}$. Artificial jumps taken because of E are referred to as *teleportation*.

By redefining the vector $\vec{v}$ given in Eq. (1) to be nonuniform, so that D and E add artificial transitions with nonuniform probabilities, the resultant PageRank vector can be biased to prefer certain kinds of pages. For this reason, we refer to $\vec{v}$ as the *personalization* vector.

For simplicity and consistency with prior work, the remainder of the discussion will be in terms of the transpose matrix, $A = (P'')^{\mathrm{T}}$; i.e., the transition probability distribution for a surfer at node i is given by row i of P'', and column i of A.

Note that the edges artificially introduced by D and E never need to be explicitly materialized, so this construction has no impact on efficiency or the sparsity of the matrices used in the computations. In particular, the matrix–vector multiplication $\vec{y} = A\vec{x}$ can be implemented efficiently using Algorithm 1. In the algorithms presented in this paper, all matrix multiplications are assumed to use Algorithm 1.

Algorithm 1. Computing $\vec{y} = A\vec{x}$

```
    $\vec{y} = cP^{\mathrm{T}}\vec{x}$;
    $w = \|\vec{x}\|_1 - \|\vec{y}\|_1$;
    $\vec{y} = \vec{y} + w\vec{v}$;
```

Assuming that the probability distribution over the surfer's location at time 0 is given by $\vec{x}^{(0)}$, the probability distribution for the surfer's location at time k is given by $\vec{x}^{(k)} = A^k\vec{x}^{(0)}$. The unique stationary distribution of the Markov chain is defined as $\lim_{k\to\infty}\vec{x}^{(k)}$, which is equivalent to $\lim_{k\to\infty}A^k\vec{x}^{(0)}$, and is independent of the initial distribution $\vec{x}^{(0)}$. This is simply the principal eigenvector of the matrix $A = (P'')^{\mathrm{T}}$, which is exactly the PageRank vector we would like to compute.

The standard PageRank algorithm computes the principal eigenvector using the Power Method (Algorithm 2). That is, it begins with the uniform distribution $\vec{x}^{(0)} = \vec{v}$ and computes successive iterates $\vec{x}^{(k)} = A\vec{x}^{(k-1)}$ until convergence. Haveliwala and Kamvar show in [4] that the convergence rate of the Power Method, in terms of number of iterations, is fast for this problem (generally, $|\lambda_2|/|\lambda_1| = 0.85$). However, it is still important to accelerate the computation, since each matrix multiplication is so expensive (on the order of 10 billion flops).

Algorithm 2. PageRank

```
    function **pageRank**($A$, $\vec{x}^{(0)}$, $\vec{v}$){
    **repeat**
        $\vec{x}^{(k+1)} = A\vec{x}^{(k)}$;
        $\delta = \|\vec{x}^{(k+1)} - \vec{x}^{(k)}\|_1$;
    **until** $\delta < \epsilon$;
    **return** $\vec{x}^{(k+1)}$;
    }
```

While many algorithms have been developed for fast eigenvector computations, many of them are unsuitable for this problem because of the size and sparsity of the Web matrix (see Section 1 for a discussion of this).

3. Experimental setup

In the following sections, we will be describing experiments run on the following data sets. The STANFORD.EDU link graph was generated from a crawl of the `stanford.edu` domain created in September 2002 by the Stanford WebBase project. This link graph contains roughly 280,000 nodes, with 3 million links, and requires 12 MB of storage. We used STANFORD.EDU while developing the Adaptive PageRank algorithm, to get a sense for its performance. For real-world, Web-scale performance measurements, we used the LARGEWEB link graph, generated from a large crawl of the Web that had been created by the Stanford WebBase project in January 2001 [5]. LARGEWEB contains roughly 80M nodes, with close to a billion links, and requires 3.6 GB of storage. Both link graphs had dangling nodes removed as described in [8]. The graphs are stored using an adjacency list representation, with pages represented by 4-byte integer identifiers. On an AMD Athlon 1533 MHz machine with a 6-way RAID-5 disk volume and 2 GB of main memory, each application of Algorithm 1 on the 80M page LARGEWEB dataset takes roughly 10 min. Given that computing PageRank generally requires anywhere from 30 to 100 applications of Algorithm 1, depending on the desired error, the need for fast methods for graphs with billions of nodes is clear.

We measured the rates of convergence of the PageRank and Adaptive PageRank using the L_1 norm of the residual vector; i.e.,

$$\|Ax^{(k)} - x^{(k)}\|_1.$$

We describe why the L_1 residual is an appropriate measure in [7].

4. Distribution of convergence rates

Table 1 and Fig. 1 show convergence statistics for the pages in the STANFORD.EDU dataset. We say that the PageRank x_i of page i has converged when

$$\left|x_i^{(k+1)} - x_i^{(k)}\right| \Big/ \left|x_i^{(k)}\right| < 10^{-3}.$$

Table 1 shows the number of pages and average PageRanks of those pages that converge in less than 15 iterations, and those pages that converge in more than 15 iterations. Notice that most pages converge in less than 15 iterations, and their average PageRank is far lower than those pages that converge in more than 15 iterations.

Fig. 1(a) shows the profile of the bar graph, where each bar represents a page i and the height of the bar is the convergence time t_i of that page i. The pages are sorted

Table 1
Statistics about pages in the STANFORD.EDU dataset whose convergence times are quick ($t_i \leqslant 15$) and pages whose convergence times are long ($t_i > 15$)

	Number of pages	Average PageRank
$t_i \leqslant 15$	227597	2.6642e−06
$t_i > 15$	54306	7.2487e−06
Total	281903	3.5473e−06

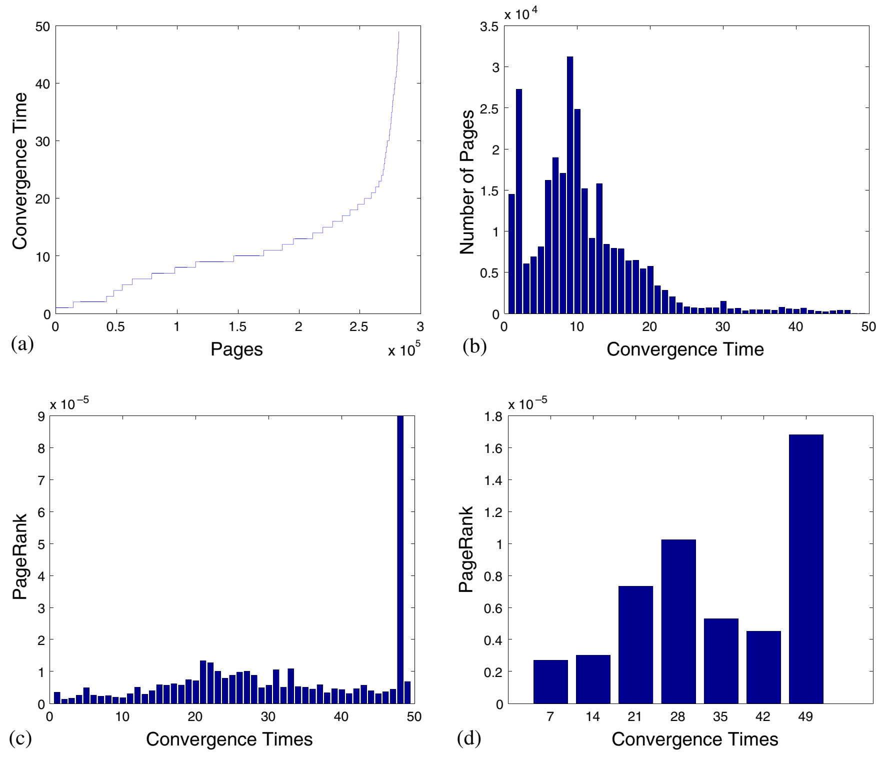


Fig. 1. Experiments on STANFORD.EDU dataset. (a) Profile of bar graph where each bar represents a page i, and its height represents its convergence time t_i. (b) Bar graph where x-axis represents the discrete convergence time t, and the height of t_i represents the number of pages that have convergence time t. (c) Bar graph where the height of each bar represents the average PageRank of the pages that converge in a given convergence time. (d) Bar graph where the height of each bar represents the average PageRank of the pages that converge in a given interval.

from left to right in order of convergence times. Notice that most pages converge in under 15 iterations, but there are some pages that require over 40 iterations to converge.

Fig. 1(b) shows a bar graph where the height of each bar represents the number of pages that converge at a given convergence time. Again, notice that most pages converge in under 15 iterations, but there are some pages that over 40 iterations to converge.

Fig. 1(c) shows a bar graph where the height of each bar represents the average PageRank of the pages that converge in a given convergence time. Notice that those pages which converge in less than 15 iterations generally have a lower PageRank than those pages who converge in over 40 iterations. This is illustrated in Fig. 1(d) as well, where the height of each bar represents the average PageRank of those pages that converge within a certain interval (i.e., the bar labeled "7" represents the pages that converge in anywhere from 1 to 7 iterations, and the bar labeled "42" represents the pages that converge in anywhere from 36 to 42 iterations.)

Figs. 2 and 3 show some statistics for the LargeWeb dataset. Fig. 2(a) shows the proportion of pages whose ranks converge to a relative tolerance of 0.001 in

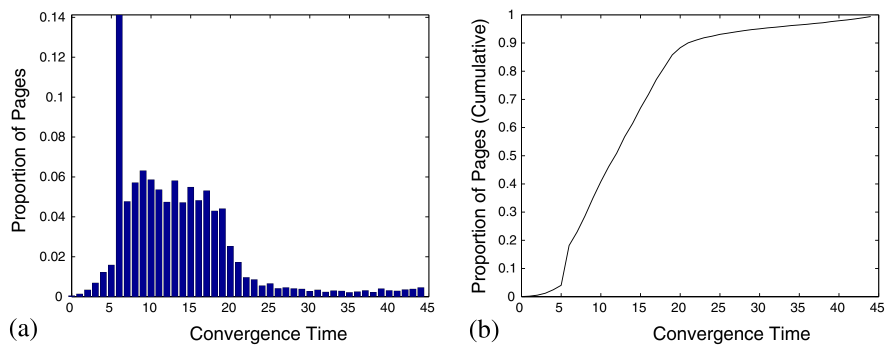


Fig. 2. Experiments on the LargeWeb dataset. (a) Bar graph where x-axis represents the convergence time t in number of iterations, and the height of bar t_i represents the proportion of pages that have convergence time t. (b) Cumulative plot of convergence times. The x-axis gives the time t in number of iterations, and the y-axis gives the proportion of pages that have a convergence time $\leqslant t$.

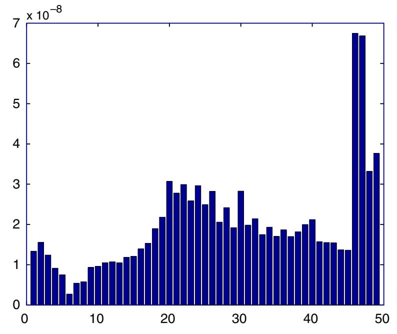


Fig. 3. Average PageRank vs. Convergence time (in number of iterations) for the LargeWeb dataset. Note that pages that are slower to converge to a relative tolerance of 0.001 tend to have high PageRank.

each iteration. Fig. 2(b) shows the cumulative version of the same data; i.e., it shows the percentage of pages that have converged up through a particular iteration. We see that in 16 iterations, the ranks for over two-thirds of pages have converged. Fig. 3 shows the average PageRanks of pages that converge in various iterations. Notice that those pages that are slow to converge tend to have higher PageRank.

5. Adaptive PageRank algorithm

The skewed distribution of convergence times shown in the previous section suggests that the running time of the PageRank algorithm can be significantly reduced by eliminating redundant computation. In particular, we do not need to recompute the PageRanks of the pages that have already converged, and we do not need to recompute the contribution of PageRank from pages that have converged to other pages. We discuss in this section how each of these redundancies can be eliminated.

5.1. Algorithm intuition

We begin by describing the intuition behind the Adaptive PageRank algorithm. We consider next a single iteration of the Power Method, and show how we can reduce the cost.

Consider that we have completed k iterations of the Power Method. Using the iterate $\vec{x}^{(k)}$, we now wish to generate the iterate $\vec{x}^{(k+1)}$. Let C be the set of pages that have converged to a given tolerance, and N be the set of pages that have not yet converged.

We can split the matrix A defined in Section 2 into two submatrices. Let A_N be the $m \times n$ submatrix corresponding to the inlinks of those m pages whose PageRanks have not yet converged, and A_C be the $(n-m) \times n$ submatrix corresponding to the inlinks of those pages that have already converged.

Let us likewise split the current iterate of the PageRank vector $\vec{x}^{(k)}$ into the m-vector $\vec{x}_N^{(k)}$ corresponding to the components of $\vec{x}^{(k)}$ that have not yet converged, and the $(n-m)$-vector $\vec{x}_C^{(k)}$ corresponding to the components of $\vec{x}^{(k)}$ that have already converged.

We may order A and $\vec{x}^{(k)}$ as follows:

$$\vec{x}^{(k)} = \begin{pmatrix} \vec{x}_N^{(k)} \\ \vec{x}_C^{(k)} \end{pmatrix} \tag{3}$$

and

$$A = \begin{pmatrix} A_N \\ A_C \end{pmatrix}. \tag{4}$$

We may now write the next iteration of the Power Method as

$$\begin{pmatrix} \vec{x}_N^{(k+1)} \\ \vec{x}_C^{(k+1)} \end{pmatrix} = \begin{pmatrix} A_N \\ A_C \end{pmatrix} \cdot \begin{pmatrix} \vec{x}_N^{(k)} \\ \vec{x}_C^{(k)} \end{pmatrix}.$$

However, since the elements of $\vec{x}_C^{(k)}$ have already converged, we do not need to recompute $\vec{x}_C^{(k+1)}$. Therefore, we may simplify each iteration of the computation to be

$$\vec{x}_N^{(k+1)} = A_N \vec{x}^{(k)}, \tag{5}$$

$$\vec{x}_C^{(k+1)} = \vec{x}_C^{(k)}. \tag{6}$$

The basic Adaptive PageRank algorithm is given in Algorithm 3.

Algorithm 3. Adaptive PageRank

```
function adaptivePR($A, \vec{x}^{(0)}, \vec{v}$) {
repeat
    $\vec{x}_N^{(k+1)} = A_N \vec{x}^{(k)}$;
    $\vec{x}_C^{(k+1)} = \vec{x}_C^{(k)}$;
    $[N, C]$ = detectConverged($\vec{x}^{(k)}, \vec{x}^{(k+1)}, \epsilon$);
    periodically, $\delta = \|A\vec{x}^{(k)} - \vec{x}^k\|_1$;
until $\delta < \epsilon$;
return $\vec{x}^{(k+1)}$;
}
```

Identifying pages in each iteration that have converged is inexpensive. However, reordering the matrix A at each iteration is expensive. Therefore, we exploit the idea given above by periodically identifying converged pages and constructing A_N without explicitly reordering identifiers. Since A_N is smaller than A, the iteration cost for future iterations is reduced. We describe the details of the algorithm in the next section.

5.2. Filter-based adaptive PageRank

Since the web matrix A is several gigabytes in size, forming the submatrix A_N needed in Eq. (5) will not be practical to do in each iteration. Furthermore, there is in general no efficient way to simply "ignore" the unnecessary entries (e.g., edges pointing to converged pages) in A if they are scattered throughout A. We describe in this section an efficient implementation of the Adaptive PageRank scheme.

Consider the following reformulation of the algorithm that was described in the previous section. Consider the matrix A as described in Eq. (4). Note that the submatrix A_C is never actually used in computing $\vec{x}^{(k+1)}$. Let us define the matrix A' as

$$A' = \begin{pmatrix} A_N \\ 0 \end{pmatrix}, \tag{7}$$

where we have replaced A_C with an all-zero matrix of the same dimensions as A_C. Similarly, let us define $\vec{x}'^{(k)}_C$ as

$$\vec{x}'^{(k)}_C = \begin{pmatrix} \vec{0} \\ \vec{x}^{(k)}_C \end{pmatrix}. \tag{8}$$

Now note that we can express an iteration of Adaptive PageRank as

$$\vec{x}^{(k+1)} = A'\vec{x}^{(k)} + \vec{x}'^{(k)}_C. \tag{9}$$

Since A' has the same dimensions as A, it seems we have not reduced the iteration cost; however, note that the cost of the matrix–vector multiplication is essentially given by the number of nonzero entries in the matrix, *not* the matrix dimensions.[1]

The above reformulation gives rise to the filter-based Adaptive PageRank scheme: if we can periodically increase the sparsity of the matrix A, we can lower the average iteration cost. Consider the set of indices C of pages that have been identified as having converged. We define the matrix A'' as follows:

$$A''_{ij} = \begin{cases} 0 & \text{if } i \in C, \\ A_{ij} & \text{otherwise.} \end{cases} \tag{10}$$

In other words, when constructing A'', we replace the row i in A with zeros if $i \in C$. Similarly, define $\vec{x}''^{(k)}_C$ as follows:

$$(x''^{(k)}_C)_i = \begin{cases} (x^{(k)})_i & \text{if } i \in C, \\ 0 & \text{otherwise.} \end{cases} \tag{11}$$

Note that A'' is much sparser than A, so that the cost of the multiplication $A''\vec{x}$ is much cheaper than the cost of the multiplication $A\vec{x}$. In fact, the cost is the same as if we had an ordered matrix, and performed the multiplication $A_N\vec{x}$. Now note that

$$\vec{x}^{(k+1)} = A''\vec{x}^{(k)} + \vec{x}''^{(k)}_C \tag{12}$$

[1] More precisely, since the multiplication $A\vec{x}$ is performed using Algorithm 1 using the matrix P and the vector $\vec{v}$, the number of nonzero entries in P determines the iteration cost. Note that subsequently, when we discuss zeroing out rows of A, this corresponds implementationally to zeroing out rows of the sparse matrix P.

represents an iteration of the Adaptive PageRank algorithm. No expensive reordering of page identifiers is needed. The filter-based implementation of Adaptive PageRank is given in Algorithm 4.

Algorithm 4. Filter-Based Adaptive PageRank

```
function **filterAPR**($A, \vec{x}^{(0)}, \vec{v}$) {
**repeat**
    $\vec{x}^{(k+1)} = A''\vec{x}^{(k)} + \vec{x}''_C$;
    periodically,
        $[N, C]$ = detectConverged($\vec{x}^{(k)}, \vec{x}^{(k+1)}, \epsilon$);
        $[A'']$ = filter($A'', N, C$);
        $[\vec{x}''_C]$ = filter($\vec{x}^{(k)}, C$);
    periodically, $\delta = \|A\vec{x}^{(k)} - \vec{x}^k\|_1$;
**until** $\delta < \epsilon$
**return** $\vec{x}^{(k+1)}$;
}
```

5.3. Modified adaptive PageRank

The core of the Adaptive PageRank algorithm is in replacing the matrix multiplication $A\vec{x}^{(k)}$ with Eqs. (5) and (6), reducing redundant computation by not recomputing the PageRanks of those pages in C (i.e., those pages that have converged).

In this section, we show how to further reduce redundant computation by not recomputing the components of the PageRanks of those pages in N due to links from those pages in C.

More specifically, we can write the matrix A in Eq. (4) as follows:

$$A = \begin{pmatrix} A_{NN} & A_{NC} \\ A_{CN} & A_{CC} \end{pmatrix},$$

where A_{NN} are the links from pages that have not converged to pages that have not converged, A_{CN} are links from pages that have converged to pages that have not converged, and so on.

We may now rewrite Eq. (5) as follows:

$$\vec{x}_N^{(k+1)} = A_{NN}\vec{x}_N^{(k)} + A_{CN}\vec{x}_C^{(k)}.$$

Since the $\vec{x}_C$ does not change at each iteration, the component $A_{CN}\vec{x}_C^{(k)}$ does not change at each iteration. Therefore, we only need to recompute $A_{CN}\vec{x}_C^{(k)}$ each time the matrix A is reordered. This variant of Adaptive PageRank is summarized in Algorithm 5.

Algorithm 5. Modified Adaptive PageRank

```
function **modifiedAPR**($A, \vec{x}^{(0)}, \vec{v}$) {
repeat
    $\vec{x}_N^{(k+1)} = A_{NN}\vec{x}_N^{(k)} + \vec{y}$;
    $\vec{x}_C^{(k+1)} = \vec{x}_C^{(k)}$;
    periodically,
        $[N, C] = \text{detectConverged}(\vec{x}^{(k)}, \vec{x}^{(k+1)}, \epsilon)$;
        $\vec{y} = A_{CN}\vec{x}_C^{(k)}$;
    periodically, $\delta = \|A\vec{x}^{(k)} - \vec{x}^k\|_1$;
until $\delta < \epsilon$
return $\vec{x}^{(k+1)}$;
}
```

As with the standard Adaptive PageRank scheme, explicit reordering of identifiers is not necessary in the implementation. As shown in Algorithm 6, we can simply form two matrices A_{CN} and A_{NN} that have their "deleted" columns and rows zeroed out, increasing their sparsity and thereby reducing their effective size. We expect that this algorithm should speed up the computation of PageRank even further as the partial sum denoted as $\vec{y}$ in Algorithm 6 is not recomputed in every iteration.

Algorithm 6. Filter-Based Modified Adaptive PageRank

```
function **filterMAPR**($A, \vec{x}^{(0)}, \vec{v}$) {
repeat
    $\vec{x}^{(k+1)} = A_{NN}\vec{x}^{(k)} + \vec{y} + \vec{x}''_C$;
    periodically,
        $N' = N, C' = C$;        /* Keep track of prev. values */
        $[N, C] = \text{detectConverged}(\vec{x}^{(k)}, \vec{x}^{(k+1)}, \epsilon)$;
        $[A''_{NN}, A''_{CN}] = \text{filter}(A''_{N'N'}, A''_{C'N'}, N, C)$;
        $[\vec{x}''_C] = \text{filter}(\vec{x}^{(k)}, C)$;
        $\vec{y} = A_{CN}\vec{x}^{(k)}$;
    periodically, $\delta = \|A\vec{x}^{(k)} - \vec{x}^k\|_1$;
until $\delta < \epsilon$
return $\vec{x}^{(k+1)}$;
}
```

5.4. Advantages

We now discuss how the Adaptive PageRank scheme speeds up the computation of PageRank. The key parameter in the algorithm is how often to identify converged pages and construct the "compacted" matrix A'' (or in the case of Modified Adaptive-PageRank, A''_{CN} and A''_{NN}); since the cost of constructing A'' from A is on the order of the cost of the multiply $A\vec{x}$, we do not want to apply it too often. However, looking at the convergence statistics given in Section 4, it is clear that even periodically filtering out the "converged edges" from A will be effective in reducing the cost of future iterations for three reasons:

1. Reduced i/o for reading in the link structure.
2. Fewer memory accesses when executing Algorithm 1.
3. Fewer flops when executing Algorithm 1.

We expect the number of iterations required for convergence to stay roughly constant, although the average iteration *cost* will be lowered.

5.5. Misconvergence

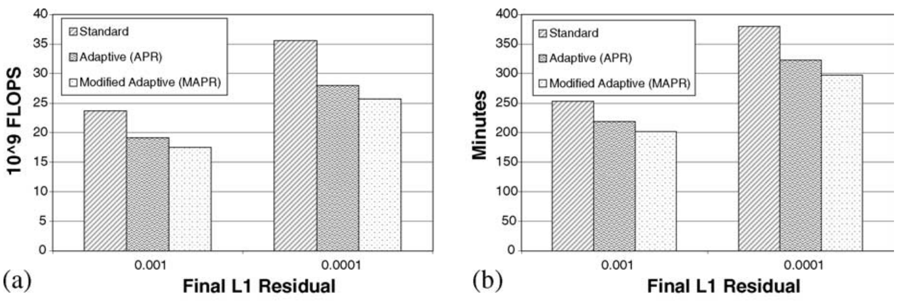
It is possible that the convergence test determines incorrectly that the PageRank of a page has converged, when in fact, it has not. For example, a page i may have the same value for several iterations, and later on change significantly. In this case, the adaptive algorithms proposed will decide that the page i has converged, and fix the ith component.

To combat this, we run the adaptive PageRank algorithm in phases where in each phase, we begin with the original version of the link structure, iterate a certain number of times (in our case 8), prune the link structure, and iterate some additional number of times (again, 8). In successive phases, we reduce the tolerance threshold used when pruning. In each phase, pruning using the current threshold is done once, during the 8th iteration.[2] This strategy tries to keep all pages at roughly the same level of error while computing successive iterates to achieve some specified final tolerance.

6. Experimental results

A comparison of the total cost of the standard PageRank algorithm and the two variants of the Adaptive PageRank algorithm follow. Fig. 4(a) depicts the total number of FLOPS needed to compute the PageRank vector to an L_1 residual threshold of

[2] For slightly better performance, our implementations of Algorithms 4 and 6 fold the filter() operation into the previous matrix multiply step.



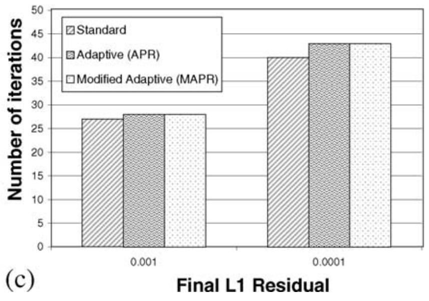


Fig. 4. Experiments on LargeWeb dataset depicting total cost for computing the PageRank vector to an L_1 residual threshold of 10^{-3} and 10^{-4}: (a) FLOPS, (b) wallclock time, (c) number of iterations.

10^{-3} and 10^{-4} using the Power Method and the two variants of the Adaptive Power Method. The Adaptive algorithms operated in phases as described above using 10^{-2}, 10^{-3}, and 10^{-4} as the successive tolerances. As shown in Fig. 4(a), the Modified Adaptive PageRank (MAPR) algorithm decreases the number of FLOPS needed by 26.2% and 27.8% in reaching final L_1 residuals of 10^{-3} and 10^{-4}, respectively, compared with the standard Power Method. Fig. 4(b) depicts the total wallclock time needed for the same scenarios. The MAPR algorithm reduces the wallclock time needed to compute the PageRank vectors by 20.3% and 21.6% in reaching final L_1 residuals of 10^{-3} and 10^{-4}, respectively. Note that the adaptive methods took a few more iterations for reaching the desired tolerances than the standard Power Method, as shown in Fig. 4(c); however, as the average iteration cost was much lower, the overall speedup is still significant.

7. Conclusion

In this work, we present two contributions. First, we show that most pages in the web converge to their true PageRank quickly, while relatively few pages take much longer to converge. We further show that those slow-converging pages generally have high PageRank, and those pages that converge quickly generally have low PageRank.

Second, we develop two algorithms, called Adaptive PageRank and Modified Adaptive PageRank, that exploit this observation to speed up the computation of PageRank by 18% and 28%, respectively, by avoiding redundant computation.

Acknowledgements

We would like to thank Chris Manning for useful conversations.

This paper is based on work supported in part by the National Science Foundation under Grant No. IIS-0085896 and Grant No. CCR-9971010, and in part by the Research Collaboration between NTT Communication Science Laboratories, Nippon Telegraph and Telephone Corporation and CSLI, Stanford University (research project on Concept Bases for Lexical Acquisition and Intelligently Reasoning with Meaning).

References

[1] G.H. Golub, C.F.V. Loan, Matrix Computations, The Johns Hopkins University Press, Baltimore, 1996.
[2] G. Grimmett, D. Stirzaker, Probability and Random Processes, Oxford University Press, 1989.
[3] T.H. Haveliwala, Topic-sensitive PageRank, in: Proceedings of the 11th International World Wide Web Conference, 2002.
[4] T.H. Haveliwala, S.D. Kamvar, The second eigenvalue of the Google matrix, Stanford University Technical Report, 2003.
[5] J. Hirai, S. Raghavan, H. Garcia-Molina, A. Paepcke, WebBase: a repository of web pages, in: Proceedings of the Ninth International World Wide Web Conference, 2000.
[6] G. Jeh, J. Widom, Scaling personalized web search, in: Proceedings of the 12th International World Wide Web Conference, 2003.
[7] S.D. Kamvar, T.H. Haveliwala, C.D. Manning, G.H. Golub, Extrapolation methods for accelerating PageRank computations, in: Proceedings of the 12th International World Wide Web Conference, 2003.
[8] L. Page, S. Brin, R. Motwani, T. Winograd, The PageRank citation ranking: bringing order to the web, Stanford Digital Libraries Working Paper, 1998.
[9] M. Richardson, P. Domingos, The intelligent surfer: probabilistic combination of link and content information in PageRank, in: Advances in Neural Information Processing Systems, vol. 14, MIT Press, Cambridge, MA, 2002.