Internet Mathematics Vol. 6, No. 2: 157–171

On Accelerating the PageRank Computation

Steve Osborne, Jorge Rebaza, and Elizabeth Wiggins

Abstract. In this paper we consider the problem of computing the PageRank vector in an efficient way. By combining some of the existing techniques and different approaches, including the power method, linear systems, iterative aggregation/disaggregation, and matrix reorderings, we propose algorithms that decrease the number of iterations to reach the desired solution, thus accelerating convergence to the vector that contains the importance of web pages.

1. Introduction

The success of Google's search engine can be partially accredited to Google's well-known PageRank algorithm, which orders the relevant pages returned by a search to provide users with accurate information. The PageRank algorithm uses a stochastic irreducible matrix that models the web and computes its dominant eigenvector. Because of the vast size of the web, this is not only complicated but also computationally expensive. The most basic way to find the dominant eigenvector for such a large matrix is the power method, which though very effective and simple, in general can exhibit very slow convergence. In this work, we explore ways to make the process of finding the eigenvector more efficient.

The basic idea is to exploit the best features of some of the existing techniques and combine them to achieve better rates of convergence. We also experiment with matrix reorderings that have the potential of reducing the magnitude of the subdominant eigenvalue of the link matrix and therefore speeding up the power method itself.

Recent developments in improving the computation of web-page ranking include [Jeh and Widom 02], in which personalized PageRank vectors (importance is redefined according to user preference) are computed using so-called partial vectors and hub skeletons. This excellent idea of decomposing the computation into two parts is similar to the idea later exploited in [Berkhin 05] to produce a faster and sparse version of this page-specific algorithm. In Berkhin's bookmark-coloring algorithm, a predefined threshold value induces sparsity, and his H-relative version of the bookmark-coloring algorithm runs even faster and produces sparser vectors. Based on the approaches from [Jeh and Widom 02] and [Berkhin 05], the algorithm for PageRank computation is further improved in [Andersen et al. 06, Andersen and Peres 09] and applied to developing PageRank-Nibble, a very efficient algorithm for local graph partitioning.

The original idea of [Brin et al. 99] is that given a set of n web pages, the rank P_j of a given page can be determined in an iterative fashion by adding the weighted ranks of the pages that have links to P_j. More precisely,

$$r_k\,(P_j) = \sum_{P_i \in B_{P_j}} \frac{r_{k-1}(P_i)}{|P_i|}, \quad i, j = 1, 2, \ldots, n,\ i \neq j, \tag{1.1}$$

where $|P|$ denotes the number of outlinks of page P.

If $v_k^T = [r_k(P_1)\ r_k(P_2)\ \cdots\ r_k(P_n)]$, then (1.1) can be written as

$$v_k^T = v_{k-1}^T H,$$

where the i, j entry of the matrix H is given by

$$H_{ij} = \begin{cases} 1/|P_i| & \text{if } P_j \text{ has a link from } P_i, \\ 0 & \text{otherwise.} \end{cases}$$

This equation defines the power method to find the dominant left eigenvector v of the link matrix H associated to the eigenvalue $\lambda = 1$. The eigenvector

$$v = \lim_{k \to \infty} v_k,$$

which contains the ranks of the n web pages, is the PageRank vector. To ensure convergence of the power method we need to require that the eigenvalues of H satisfy

$$|\lambda_1| > |\lambda_2| \geq \cdots \geq |\lambda_n|.$$

On the other hand, the matrix H is modified to ensure that it is stochastic (so that $\lambda_1 = 1$) and irreducible. This ensures that up to a rescaling, the matrix has a unique dominant eigenvector. To get a stochastic matrix, H is modified as

$$B = H + au^T,$$

where u is an arbitrary probabilistic vector, and a is defined as

$$a_i = \begin{cases} 1 & \text{if page } i \text{ is a dangling node,} \\ 0 & \text{otherwise.} \end{cases}$$

A dangling node is a web page with no outlinks, and therefore the corresponding row in H is a zero vector. To enforce irreducibility, B is further modified as

$$G = \alpha B + (1-\alpha)eu^T,$$

where $\alpha \in (0,1)$, and e is a vector of ones. The matrix G is known as the *Google matrix*. Not only is this matrix stochastic and irreducible, it is also primitive, so that all conditions for convergence to a unique positive dominant eigenvector are now satisfied.

Choosing α close to 1 makes G a good representation of the original web, but it has been shown to slow the convergence of the power method considerably. Similarly, a value of α close to zero speeds up the power method, but then G is no longer a good representation of the original web. A value around $\alpha = 0.85$ seems to be a reasonable choice.

Recall that G is a positive matrix, so that to recover sparsity, we write the power method iteration $v_{k+1}^T = v_k^T G$ as

$$v_k^T G = \alpha v_k^T H + \alpha v_k^T a u^T + (1-\alpha) v_k^T e u^T = \alpha v_k^T H + \alpha v_k^T a u^T + (1-\alpha) u^T. \quad (1.2)$$

This involves only one vector–matrix product per iteration, using the very sparse matrix H. This is what has made the power method the favorite algorithm for the PageRank problem.

2. Linear System Approach

It has been proved (see, for example, [Del Corso et al. 05, Langville and Meyer 06]) that the eigenvector problem can be formulated as a linear system. More precisely, one can show that

$$v^T = v^T G \Longleftrightarrow (I - \alpha H^T)x = u \quad \text{and} \quad v = \frac{x}{\|x\|_1}.$$

This provides the opportunity to apply a variety of available numerical methods and algorithms for sparse linear systems [Barrett et al. 94] and to look for efficient strategies to solve such systems as a better alternative to finding a dominant eigenvector through the power method. For instance, one can now reorder H by dangling nodes:

$$H = \begin{bmatrix} H_{11} & H_{12} \\ 0 & 0 \end{bmatrix},$$

where H_{11} is a square matrix that represents the links from nondangling nodes to nondangling nodes and H_{12} represents the links from nondangling nodes to dangling nodes. In this case the system $(I - \alpha H^T)x = u$ is [Langville and Meyer 04]

$$\begin{bmatrix} I - \alpha H_{11}^T & 0 \\ -\alpha H_{12}^T & I \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}, \tag{2.1}$$

so that we can solve the smaller system $(I - \alpha H_{11}^T)x_1 = u_1$ and then set $x_2 = u_2 + \alpha H_{12}^T x_1$, and finally normalize $v = x/\|x\|_1$.

3. Iterative Aggregation/Disaggregation

Another attempt at outperforming the power method in the computation of the PageRank vector comes from the theory of Markov chains. In this context, the PageRank vector v that satisfies $v^T = v^T G$ is called a stationary distribution of G. The main idea behind the iterative aggregation/disaggregation (IAD) approach to computing the PageRank vector v is to block or partition the irreducible stochastic matrix G so that the size of the problem is reduced to about the size of one of the diagonal blocks.

The precise aggregation/disaggregation theorem says (see, for example, [Ipsen and Kirkland 06]) that if u_2 is the stationary distribution of $S = G_{22} + G_{21}(I - G_{11})^{-1}G_{12}$ and $[v_1 \; c]^T$ is the stationary distribution of the aggregated matrix

$$A = \begin{bmatrix} G_{11} & G_{12}e \\ u_2^T G_{21} & 1 - u_2^T G_{21} e \end{bmatrix},$$

then the stationary distribution of

$$G = \begin{bmatrix} G_{11} & G_{12} \\ G_{21} & G_{22} \end{bmatrix}$$

is given by $v = [v_1 \; cu_2]^T$.

Since forming the matrix S and computing its stationary distribution u_2 would be very expensive and not efficient in practice, the idea is to use instead the approximate aggregation matrix

$$\tilde{A} = \begin{bmatrix} G_{11} & G_{12}e \\ \tilde{u}_2^T G_{21} & 1 - \tilde{u}_2^T G_{21} e \end{bmatrix},$$

and implement an iterative algorithm, where a power method step is included at every iteration. More precisely, we have Algorithm 1.

Algorithm 1. (**Iterative aggregation/disaggretagion (IAD).**)

```
1. Give an arbitrary probabilistic vector $\tilde{u}_2$ and a tolerance $\epsilon$.
2. For $k = 1, 2, \ldots,$
    (a) Find the stationary distribution $[w_1\ \tilde{c}]^T$ of $\tilde{A}$.
    (b) Set $v_k^T = [w_1\ \tilde{c}\tilde{u}_2]^T$.
    (c) Let $v_{k+1}^T = v_k^T G$.
    (d) If $\|v_{k+1}^T - v_k^T\| < \epsilon$, stop. Otherwise, set $\tilde{u}_2 = (v_{k+1})_y / \|(v_{k+1})_y\|_1$.
        Here $(v_{k+1})_y$ denotes the second block of the vector $v_{k+1}$.
```

4. Combining IAD with the Power Method and Linear Systems

On examining the IAD algorithm above, we see that we need to compute $[w_1\ \tilde{c}]^T$, the stationary distribution of $\tilde{A}$, that is,

$$\begin{bmatrix} w_1 \\ \tilde{c} \end{bmatrix}^T = \begin{bmatrix} w_1 \\ \tilde{c} \end{bmatrix}^T \tilde{A}. \tag{4.1}$$

We explore two ways to do this and compare their efficiencies.

4.1. IAD and the Power Method

The matrices G_{11}, G_{12}, and G_{21} that make up $\tilde{A}$ are full matrices, and applying the power method directly would not be efficient. The question is whether we will still be able to exploit the sparsity of the original link matrix H. By setting $z^T = \tilde{u}_2^T G_{21}$, $\tilde{A}$ becomes

$$\tilde{A} = \begin{bmatrix} G_{11} & G_{12}e \\ z^T & 1 - z^T e \end{bmatrix}.$$

From (4.1) we have

$$[w_1^T\ \tilde{c}] = [w_1^T\ \tilde{c}] \begin{bmatrix} G_{11} & G_{12}e \\ z^T & 1 - z^T e \end{bmatrix} = [w_1^T G_{11} + \tilde{c}z^T\ \ w_1^T G_{12}e + \tilde{c}(1 - z^T e)]. \tag{4.2}$$

To obtain some sparsity explicitly, first we write G in terms of H as before:

$$G = \alpha H + \alpha a u^T + (1 - \alpha) e u^T.$$

Then we block the matrices H, au^T, and eu^T according to the blocking of G:

$$G = \begin{bmatrix} G_{11} & G_{12} \\ G_{21} & G_{22} \end{bmatrix} = \alpha \begin{bmatrix} H_{11} & H_{12} \\ H_{21} & H_{22} \end{bmatrix} + \alpha \begin{bmatrix} A & B \\ C & D \end{bmatrix} + (1-\alpha) \begin{bmatrix} E & F \\ J & K \end{bmatrix}, \tag{4.3}$$

for some matrices A, B, C, D, E, F, J, K. Now we see that

$$\begin{aligned} G_{11} &= \alpha H_{11} + \alpha a_x u_x^T + (1-\alpha) e_x u_x^T, \\ G_{12} &= \alpha H_{12} + \alpha a_x u_y^T + (1-\alpha) e_x u_y^T, \\ G_{21} &= \alpha H_{21} + \alpha a_y u_x^T + (1-\alpha) e_y u_x^T, \end{aligned}$$

where the subscripts x and y denote blocking of the vectors a, u, and e according to the blocking of G. Thus, for example, a_x denotes the first block of entries of the vector a with size corresponding to the size of G_{11}.

Now we can use the sparsity of H_{11}, H_{12}, and H_{21} to make computations simpler and faster. The equations in (4.2) can now be written as

$$\begin{aligned} w_1^T &= \alpha w_1^T H_{11} + \alpha w_1^T a_x u_x^T + (1-\alpha) w_1^T e_x u_x^T + \tilde{c} z^T, \\ \tilde{c} &= \alpha w_1^T H_{12} e + \alpha w_1^T a_x u_y^T e + (1-\alpha) w_1^T e_x u_y^T e + \tilde{c}(1 - z^T e). \end{aligned} \tag{4.4}$$

For the iterative process of the power method within IAD, we give as usual an arbitrary initial guess $[w_1^T \ \tilde{c}]$ and iterate according to (4.4) for the next approximation $[w_1^T \ \tilde{c}]$ until a certain tolerance is reached.

4.2. Exploiting Dangling Nodes

Although not obvious in (4.4), it is possible to exploit not only the sparsity of the matrices H_{11}, H_{12}, and H_{21} but also some reorderings applied to H. For instance, if H is reordered by dangling nodes, bringing all the zero rows to the bottom of the matrix, then the matrix H_{21} that is used in $z = \tilde{u}_2^T G_{21}$ is a matrix of zeros. Also, by definition of the vector a, all the entries of a_x will be zero. All this reduces the power method computation in (4.4) to

$$\begin{aligned} w_1^T &= \alpha w_1^T H_{11} + (1-\alpha) w_1^T e_x u_x^T + \tilde{c} z^T, \\ \tilde{c} &= \alpha w_1^T H_{12} e + (1-\alpha) w_1^T e_x u_y^T e + \tilde{c}(1 - z^T e). \end{aligned} \tag{4.5}$$

4.3. IAD and Linear Systems

We start by rewriting equation (4.1) as

$$[w_1^T \ \tilde{c}] = [w_1^T \ \tilde{c}] \begin{bmatrix} G_{11} & G_{12} e \\ \tilde{u}_2^T G_{21} & 1 - \tilde{u}_2^T G_{21} e \end{bmatrix},$$

Algorithm 2. (Stationary distribution computation I (SDC1).)

```
1. Choose an initial guess $\tilde{c}$ and a tolerance $\epsilon$.
2. Repeat until $\|w_1 - w_1(\text{old})\| < \epsilon$:
    (a) Solve $(I - G_{11})^T w_1 = \tilde{c}\, G_{21}^T \tilde{u}_2$.
    (b) Adjust $\tilde{c} = \frac{w_1^T G_{12} e}{\tilde{u}_2^T G_{21} e}$.
```

which gives

$$\begin{aligned} w_1^T (I - G_{11}) &= \tilde{c}\tilde{u}_2^T G_{21}, \\ w_1^T G_{12} e &= \tilde{c}\tilde{u}_2^T G_{21} e. \end{aligned} \tag{4.6}$$

Observe that the first equation in (4.6) is a linear system of the order of G_{11}, and the second one is just a scalar equation. However, such a system with coefficient matrix $I - G_{11}$ cannot be solved, because the right-hand side contains $\tilde{c}$, which is unknown. Thus, the idea is to initially fix an arbitrary $\tilde{c}$ so that the system $w_1^T(I - G_{11}) = \tilde{c}\tilde{u}_2^T G_{21}$ can be solved, and once w_1 is computed, we can adjust $\tilde{c}$ using the second equation in (4.6).

More precisely, to compute the stationary distribution $[w_1\ \tilde{c}]^T$ of $\tilde{A}$ we apply Algorithm 2.

Typically, it takes just a few steps to reach a given tolerance ϵ, so that this computation is relatively fast in terms of the number of iterations. However, the matrices G_{11}, G_{12}, and G_{21} are full matrices, so that computations at each step are in general very expensive. Thus, similarly to what it was done in studying IAD with the power method, we return to the original matrix H to obtain some sparsity.

From equation (4.3), we get

$$I - G_{11} = I - \alpha H_{11} - \alpha A - (1-\alpha)E = I - \alpha H_{11} - \alpha a_x u_x^T - (1-\alpha) e_x u_x^T.$$

The last term, $(1-\alpha)e_x u_x^T$, can disturb the sparsity of $I - G_{11}$, but we will use the fact that $w_1^T e_x = 1 - \tilde{c}$ to simplify. We know this to be true, because

$$1 = [w_1^T\ \tilde{c}]e = [w_1^T\ \tilde{c}] \begin{bmatrix} e_x \\ 1 \end{bmatrix} = w_1^T e_x + \tilde{c}.$$

We also know that

$$G_{21} = \alpha H_{21} + \alpha a_y u_x^T + (1-\alpha) e_y u_x^T,$$

which can be used to keep the system sparse. With this knowledge, our linear system $w_1^T(I - G_{11}) = c\tilde{u}_2^T G_{21}$ becomes

$$w_1^T(I-\alpha H_{11}-\alpha a_x u_x^T) = \tilde{c}\tilde{u}_2^T\left(\alpha H_{21} + \alpha a_y u_x^T + (1-\alpha)e_y u_x^T\right)+(1-\alpha)(1-\tilde{c})u_x^T.$$

Thus to compute w_1, we solve the linear system

$$(I-\alpha H_{11}-\alpha a_x u_x^T)^T w_1 = \tilde{c}\left(\alpha H_{21} + \alpha a_y u_x^T + (1-\alpha)e_y u_x^T\right)^T \tilde{u}_2+(1-\alpha)(1-\tilde{c})u_x. \tag{4.7}$$

4.4. Dangling Nodes

As before, some reorderings of the matrix H can mean a reduction of the computational cost in the solution of the linear system. Thus, for example, assume that H has been reordered by dangling nodes, i.e., the zero rows have been moved to the bottom. In such a case we have

$$H = \begin{bmatrix} H_{11} & H_{12} \\ 0 & 0 \end{bmatrix},$$

so that H_{21} is a matrix of zeros. Then we have the following result.

Theorem 4.1. *Let the matrix H be reordered by dangling nodes. Then the system (4.7) reduces to*

$$(I-\alpha H_{11})^T w_1 = (1-\alpha(1-\tilde{c}))u_x.$$

Proof. Due to the reordering by dangling nodes, the vector a_x is a vector of zeros and the vector a_y is a vector of ones, of the same size as e_y, so that $a_y = e_y$. Since $\tilde{u}_2$ is a probabilistic vector, we also have $e_y^T\tilde{u}_2 = 1$. Then we can simplify (4.7) to

$$\begin{aligned}(I-\alpha H_{11})^T w_1 &= \tilde{c}(\alpha a_y u_x^T + (1-\alpha)e_y u_x^T)^T\tilde{u}_2 + (1-\alpha)(1-\tilde{c})u_x \\ &= \tilde{c}(e_y u_x^T)^T\tilde{u}_2 + (1-\alpha)(1-\tilde{c})u_x \\ &= \tilde{c}u_x e_y^T\tilde{u}_2 + (1-\alpha)(1-\tilde{c})u_x \\ &= \tilde{c}u_x + (1-\alpha)(1-\tilde{c})u_x \\ &= (\tilde{c} + (1-\alpha)(1-\tilde{c}))u_x \\ &= (1-\alpha(1-\tilde{c}))u_x,\end{aligned}$$

yielding the desired result. □

Following similar reasoning, one can prove that the computation of the scalar $\tilde{c}$ can also be reduced in the case of a link matrix H that has been reordered by

Algorithm 3. (Stationary distribution computation 2 (SDC2).)

```
1. Choose an initial guess $\tilde{c}$ and a tolerance $\epsilon$.
2. Repeat until $\|w_1 - w_1(\text{old})\| < \epsilon$:
    (a) Solve $(I - \alpha H_{11})^T w_1 = (1 - \alpha(1 - \tilde{c}))u_x$
    (b) Adjust $\tilde{c} = \frac{\alpha w_1^T H_{12} e + (1-\alpha)(1-\tilde{c}) u_y^T e}{u_x^T e}$.
```

dangling nodes. More precisely, since $G_{12} = \alpha H_{12} + \alpha a_x u_y^T + (1-\alpha) e_x u_y^T$, we have the following corollary.

Corollary 4.2. *The scalar*

$$\tilde{c} = \frac{w_1^T G_{12} e}{\tilde{u}_2^T G_{21} e}$$

can be computed as

$$\tilde{c} = \frac{\alpha w_1^T H_{12} e + (1-\alpha)(1-\tilde{c}) u_y^T e}{u_x^T e},$$

when H has been reordered by dangling nodes.

Thus, our Algorithm 3 computes the stationary distribution $[w_1 \ \tilde{c}]^T$ needed in Algorithm 1, using the original link matrix H arranged by dangling nodes, effectively combining IAD and linear systems.

5. Power Method and Reordering

We know that for stochastic matrices we have $\lambda_1 = 1$. Since the power method rate of convergence depends on the magnitude of the subdominant eigenvalue λ_2, it is desirable to be able to manipulate the matrix so that $|\lambda_2|$ is small enough to ensure fast convergence. In [Haveliwala et al. 03], the authors have considered improving the PageRank computation by exploiting the knowledge of some of its eigenvalues, in particular the subdominant eigenvalue. The best case is that in which all the entries of an $n \times n$ matrix are equal to $1/n$, because then $\lambda_2 = 0$, while the worst case is the identity matrix. Several cases in between are considered; for example, in [Dayar and Stewart 00] and [Molnar and Simonovits 98], the authors study the behavior of the subdominant eigenvalue. Although it has

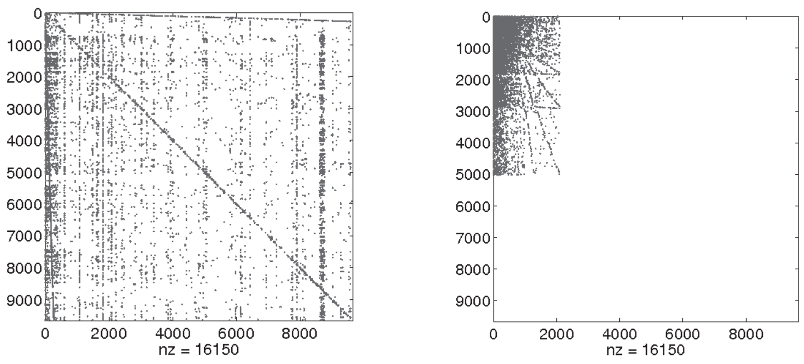

Figure 1. California data before and after a reordering by rows and columns.

been suggested that the power method may be independent of matrix reordering [Del Corso et al. 05], our experiments show that in the great majority of cases considered, a reordering of the matrix by decreasing row and column degree speeds up the power method considerably. This reordering tends to condense all the information into a relatively very small area. See Figure 1.

6. Testing the Algorithms

We have used Matlab to write programs to test the algorithms on a number of different data sets. The data sets are of different sizes, ranging from about ten thousand to millions of web pages. We list here seven of the basic programs developed:

1. A basic linear system, with $I - \alpha H$.
2. A linear system with $I - \alpha \hat{H}$, where $\hat{H}$ is reordered by decreasing row degree.
3. A linear system ordered by dangling nodes to solve the system (2.1) as indicated.
4. IAD with the power method to solve for w_1.
5. IAD and a linear system as described in Section 4.3.
6. The power method (1.2).
7. The power method with reordering by decreasing row and column degree.

	Calif.	**Stanford**	**CNR**	**Stan–Berk**	**EU**
Interval Size	100	5000	5000	10000	15000
Sample Size	87	57	66	69	58
Mean Diff. of (Time)	1.6334	2.2081	2.1136	1.4801	2.2410
S.D. Diff. of (Time)	0.6000	0.3210	0.1634	0.2397	0.2823
Iter. of RePM/PM	0.4789	0.2278	0.2280	0.2681	.2235
Iter. of PM/RePM	2.0880	4.3903	4.3856	3.7297	4.4752
S.D. of Ratio	0.9067	0.7636	0.7732	0.6795	0.6085
Favor. to Reorder	100.00%	98.25%	100.00%	100.00%	100.00%

Table 1. Comparison of how the power method works before and after reordering the data. Here "Stan–Berk" means "Stanford–Berkeley," "S.D. of Diff." refers to the standard deviation of the differences; "Iter." refers to number of iterations required; "PM" refers to the power method; "RePM" refers to the reordered power method; and "Favor." is short for "Favorability."

For linear systems, direct methods and SOR (with relaxation parameter $\omega = 1$) were used.

The reordered power method deserves some comment. Table 1 shows how reordering affects the power method. The test is run by forming a smaller sample matrix in the center of the whole matrix, where we measure the time and iterations required for the sample matrix, and then increase the matrix size by an interval, taking the time and iterations required for each partition of the matrix. The size of the starting sample matrix is the size of the interval, unless the interval is less than 1000, in which case the starting matrix is of size 1000. Samples are taken until the entire matrix is included.

The difference between the times is taken and recorded, as well at the standard deviation. The ratios between the power method and the reordered power method and the standard deviation of the ratio are also recorded. Observe the decrease in more than 50% in the number of iterations required for convergence.

Finally, the favorability is recorded. Given as a percentage, the favorability to reorder presents the percentage of the time that it is faster to reorder the matrix. As Table 1 shows, it is always favorable to reorder with four out of five tested matrices, whereas the fifth is favorable to reorder almost all of the time.

Figure 2 shows how reordering matrices will decrease the number of iterations required for the power method to converge. The first two graphs show two different data sets of different sizes, where reordering by row and column degree makes a drastic difference. The third graph contains the same data set as the first, except it is reordered only by row degree.

In studying the programs developed and how they compare to one another, we tested them on a number of matrices and recorded the data. Table 2 compares the time it takes to run the code successfully and the number of iterations for the

Matrix	Code	Time (sec)	GS-I	PM-I	w_1-I	Size G_{11}
California	$I - \alpha H$	0.119873	53			
	$I - \alpha H$ w/ reordering	0.060155	9			
9664 Nodes	Meyer's Algorithm	0.061451	18			
1.73E-4 Sparsity	IAD w/ Power Method	0.125680		13	46	1000
4637 Dang Nodes	IAD w/ system & SOR	0.690981		49	181	1000
	Power Method	0.084233		97		
	Power Method w/ reordering	**0.035091**		**22**		
Stanford	$I - \alpha H$	4.673486	56			
	$I - \alpha H$ w/ reordering	5.290621	55			
281903 Nodes	Meyer's Algorithm	5.584932	56			
2.91E-5 Sparsity	IAD w/ Power Method	4.485200		23	61	1500
172 Dang Nodes	IAD w/ system & SOR	3.684937		23	57	750
	Power Method	4.950945		90		
	Power Method w/ reordering	**2.276534**		**19**		
CNR (2000)	$I - \alpha H$	5.537661	50			
	$I - \alpha H$ w/ reordering	4.942164	24			
325557 Nodes	Meyer's Algorithm	4.380352	23			
3.03E-5 Sparsity	IAD w/ Power Method	5.328511		19	53	10000
78056 Dang Nodes	IAD w/ system & SOR	4.172643		19	48	1000
	Power Method	7.018947		87		
	Power Method w/ reordering	**3.102524**		**19**		
Stanford-Berkeley	$I - \alpha H$	9.320476	55			
	$I - \alpha H$ w/ reordering	11.704303	46			
685230 Nodes	Meyer's Algorithm	14.450852	62			
1.62E-5 Sparsity	IAD w/ Power Method	30.842767		90	237	4500
4744 Dang Nodes	IAD w/ system & SOR	24.503765		90	256	1000
	Power Method	8.958482		90		
	Power Method w/ reordering	**7.095738**		**28**		
EU(2005)	$I - \alpha H$	33.467238	50			
	$I - \alpha H$ w/ reordering	53.200779	37			
862664 Nodes	Meyer's Algorithm	45.817442	39			
2.58E-5 Sparsity	IAD w/ Power Method	25.945539		15	41	10000
4744 Dang Nodes	IAD w/ system & SOR	22.499486		16	48	10000
	Power Method	38.617480		82		
	Power Method w/ reordering	**16.727974**		**19**		
IN (2004)	$I - \alpha H$	38.29	52			
	$I - \alpha H$ w/ reordering	31.46	28			
1382908 Nodes	Meyer's Algorithm	29.22	25			
8.85E-6 Sparsity	**IAD w/ Power Method**	**26.69**		**18**	**44**	**10000**
282306 Dang Nodes	IAD w/ system & SOR	23.95		18	46	10000
	Power Method	46.09		88		
	Power Method w/ reordering	31.59		68		
Wikipedia	$I - \alpha H$	44.02	54			
	$I - \alpha H$ w/ reordering	52.95	22			
1634989 Nodes	Meyer's Algorithm	64.01	54			
7.39E-6 Sparsity	IAD w/ Power Method	46.32		19	54	10000
72556 Dang Nodes	IAD w/ system & SOR	119.94		101	400	10000
	Power Method	33.86		59		
	Power Method w/ reordering	**18.89**		**12**		

Table 2. This table lists each data set, the programs used, the time it took to run the program, and the number of iterations necessary for each code. Here "GS-I" indicates the number of Gauss–Seidel iterations, "PM-I" indicates the number of power method iterations, and "w_1-I" indicates the number of w_1 iterations. Sparsity is found by taking the number of nonzero entries in the matrix and dividing by the number of nodes in the matrix.

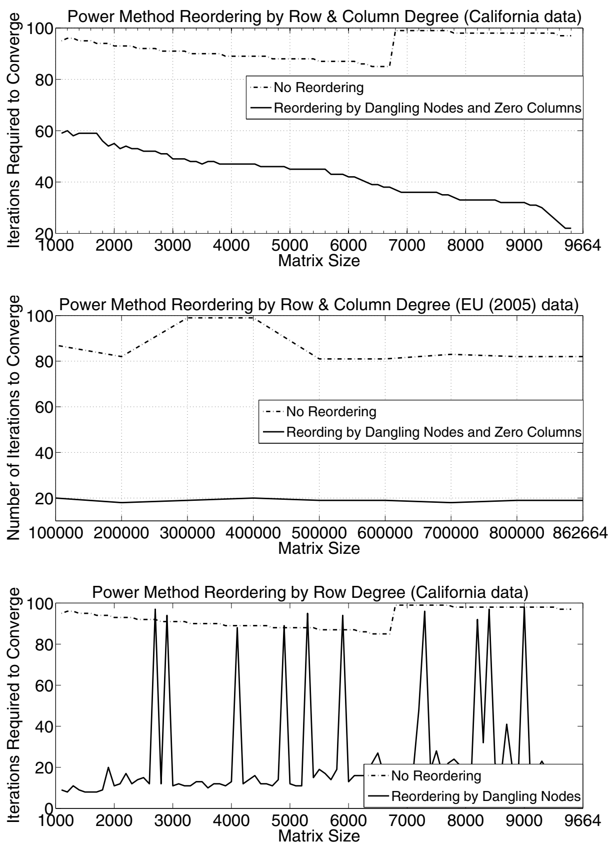

Figure 2. Comparison between the power method with original data and the power method with reordered data.

code's iterative method. The boldface lines name the programs that are optimal for the corresponding matrix.

7. Final Remarks

In exploiting some of the existing theory and techniques for computing the PageRank vector, we have developed and tested algorithms that combine basic eigenvector and linear systems approaches and reported on their efficiency. Of special interest is the faster convergence obtained when the power method is applied to a reordered matrix. We also observed that IAD combined with linear systems to solve for the stationary distribution of the approximate aggregated matrix gives in general better results than using IAD with the power method (see Algorithm 3). It remains to test a large set of algorithms for sparse linear systems [Barrett et al. 94] and to test with much larger matrices. Future research also includes attempting to achieve a rigorous theoretical explanation of how the subdominant eigenvalue may be affected by column–row reordering, and possible combination with recent theories developed in [Andersen et al. 06, Berkhin 05, Jeh and Widom 02].

Acknowledgments. This work was supported in part by NSF Grant DMS–0552573.

References

[Andersen et al. 06] R. Andersen, F. R. K. Chung, and K. J. Lang. "Local Graph Partitioning Using PageRank Vectors." In *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science*, pp. 475–486. Washington, DC: IEEE Press, 2006.

[Andersen and Peres 09] R. Andersen and Y. Peres. "Finding Sparse Cuts Locally Using Evolving Sets." In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing*, pp. 235–244. New York: ACM Press, 2009.

[Barrett et al. 94] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. M. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. Van der Vorst. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*. Philadelphia: SIAM, 1994.

[Berkhin 05] P. Berkhin. "Bookmark-Coloring Algorithm for Personalized PageRank Computing." *Internet Mathematics* 3 (2005), 41–62.

[Brin et al. 99] S. Brin, L. Page, R. Motwami, and T. Winograd. "The PageRank Citation Ranking: Bringing Order to the Web." Technical Report 1999-0120, Stanford University, 1999.

[Dayar and Stewart 00] T. Dayar and W. J. Stewart. "Comparison of Partitioning Techniques for Two-Level Iterative Solvers on Large, Sparse Markov Chains." *SIAM J. on Sci. Comput.* 21 (2000), 1691–1705 .

[Del Corso et al. 05] G. M. Del Corso, A. Gulli, and F. Romani. "Fast PageRank Computation via a Sparse Linear System." *Internet Mathematics* 2 (2005), 251–273.

[Haveliwala et al. 03] T. Haveliwala, S. Kamvar, D. Klein, C. Manning, and G. Golub, "Computing PageRank Using Extrapolation." Technical report, Stanford University, 2003.

[Ipsen and Kirkland 06] I. C. F. Ipsen and S. Kirkland. "Convergence Analysis of a PageRank Updating Algorithm by Langville and Meyer." *Siam J. Mat. Anal. and Appl.* 27 (2006), 952–967.

[Jeh and Widom 02] G. Jeh and J. Widom. "Scaling Personalized Web Search." Technical report, Stanford University, 2002.

[Langville and Meyer 04] A. N. Langville and C. D. Meyer. “Deeper inside PageRank.” *Internet Mathematics* 1 (2004), 335–380.

[Langville and Meyer 06] A. N. Langville and C. D. Meyer. “A Reordering for the PageRank Problem.” *SIAM J. on Scient. Comp.* 27 (2006), 2112–2120.

[Molnar and Simonovits 98] G. Molnar and A. Simonovits, “The Subdominant Eigenvalue of a Large Stochastic Matrix.” *Economic Systems Research* 10:1 (1998), 09535314.

Steve Osborne, Mathematics Department, Greenville College, Greenville, IL 62246-1145 (200506169@panthers.greenville.edu)

Jorge Rebaza, Department of Mathematics, Missouri State University, Springfield, MO 65897 (jrebaza@missouristate.edu)

Elizabeth Wiggins, Department of Mathematics, Physics, and Computer Science, Georgetown College, Georgetown, KY 40324 (Lwiggin0@georgetowncollege.edu)

Received January 13, 2009; accepted in revised form November 30, 2009.