Inf Retrieval (2012) 15:503–526
DOI 10.1007/s10791-012-9183-2

Lumping algorithms for computing Google's PageRank and its derivative, with attention to unreferenced nodes

Qing Yu · Zhengke Miao · Gang Wu · Yimin Wei

Received: 4 July 2011 / Accepted: 6 January 2012 / Published online: 1 February 2012

Abstract In this paper, we introduce five type nodes for lumping the Web matrix, and give a unified presentation of some popular lumping methods for PageRank. We show that the PageRank problem can be reduced to solving the PageRank corresponding to the strongly non-dangling and referenced nodes, and the full PageRank vector can be easily derived by some recursion formulations. Our new lumping strategy can reduce the original PageRank problem to a much smaller one, and it is much cheaper than the recursively reordering scheme. Furthermore, we discuss sensitivity of the PageRank vector, and present a lumping algorithm for computing its first order derivative. Numerical experiments show that the new algorithms are favorable when the matrix is large and the damping factor is high.

Keywords Google · PageRank · Web information retrieval · Dangling nodes · Unreferenced nodes

Q. Yu
Department of Arts and Sciences, Xuzhou Higher Normal School, Xuzhou 221116, Jiangsu, People's Republic of China
e-mail: qingyuyu@yahoo.cn

Z. Miao · G. Wu
School of Mathematical Sciences, Xuzhou Normal University, Xuzhou 221116, Jiangsu, People's Republic of China
e-mail: zkmiao@xznu.edu.cn

G. Wu
e-mail: gangwu76@yahoo.com.cn; wugangzy@gmail.com

Y. Wei (✉)
School of Mathematical Sciences and Shanghai Key Laboratory of Contemporary Applied Mathematics, Fudan University, Shanghai 200433, People's Republic of China
e-mail: ymwei@fudan.edu.cn; yimin.wei@gmail.com

1 Introduction

With the booming development of the Internet, Web search engines have become one of the most important tools to Web information retrieval. A query to a Web search engine often produces a very long list of answer because of the enormous number of pages. These pages have to be listed starting from the most relevant ones (Brin et al. 1998). One important measure of a page is PageRank which plays an important role in Google's search engine. Google's key idea is that a page is considered to be important if many other important pages are pointing to it (Brin and Page 1998; Page et al. 1998).

Let us introduce the mathematical background of the PageRank problem, for more details, refer to (Langville and Meyer 2006b). The hyper-link structure of the Web can be represented as a directed graph, whose nodes are Web pages and the edges are links. The graph can be expressed as a nonnegative matrix $\hat{P}$ whose (i, j)-th element is nonzero if page i contains a link to page j. However, if page i has no out-links, then the i-th row of the matrix will be zero, and the page is called a *dangling node* (Ipsen and Selee 2007). For instance, it can be any page whose links have not yet been crawled. So as to transform the Web matrix $\hat{P}$ into a stochastic matrix, one can add artificial links to the dangling nodes. Then we obtain a row stochastic matrix $\widetilde{P} = \hat{P} + d\hat{w}^T$, where $\hat{w}$ is a non-negative vector satisfying $\mathbf{e}^T\hat{w} = 1, \mathbf{e}$ is the vector of all ones, and $d = \mathbf{e} - \hat{P}\mathbf{e}$ is the dangling page indicator.

However, the surfer can still be trapped by a cyclic path in the Web graph. So another adjustment is required, which yields *the Google matrix*

$$G = \alpha\widetilde{P} + (1 - \alpha)\mathbf{e}\hat{v}^T. \tag{1.1}$$

Here $\hat{v}$ is *the personalization vector* which is a non-negative vector satisfying $\mathbf{e}^T\hat{v} = 1$, and α $(0 < \alpha < 1)$ is called *the damping factor*. A popular choice for $\hat{v}$ and $\hat{w}$ is $\hat{v} = \hat{w} = \mathbf{e}/n$ (Ipsen and Selee 2007).

The PageRank vector $\hat{x}$ is the stationary distribution of G, i.e.,

$$\hat{x}^T G = \hat{x}^T, \quad \hat{x}^T\mathbf{e} = 1, \quad \hat{x} \geqslant \mathbf{O}. \tag{1.2}$$

A standard way to compute PageRank is the classical power method (Golub and Van 1996). Unfortunately, when the largest eigenvalue of the Google matrix is not separated well from the second one, the convergence rate of the power method will be very slow (Golub and Van 1996). Many technologies have been proposed to speed up the computation, such as the Monte Carlo methods (Avrachenkov et al. 2007), the inner-outer method (Gleich et al. 2010), the extrapolation methods (Brezinski et al. 2005; Haveliwala et al. 2003; Kamvar et al. 2003a), the adaptive method (Kamvar et al. 2004), and the Krylov subspace methods (Gleich et al. 2005; Golub and Greif 2006; Wu and Wei 2007, 2010a, b), etc.

The PageRank problem can be also viewed as a large sparse linear system (Gleich et al. 2005, 2010; Langville and Meyer 2006a; Wu and Wei 2010a). Indeed, if we set $\hat{v} = \hat{w}$, by (1.1) and (1.2), the PageRank problem can be rewritten as the following linear system (Gleich et al. 2005; Langville and Meyer 2006a)

$$\hat{x}^T(I - \alpha\hat{P}) = \hat{v}^T. \tag{1.3}$$

It can be solved by the classical Jacobi iteration (or more precisely, the Richardson iteration) (Arasu 2002; Saad 2003)

$$x^T_{(\text{iter})} = \alpha \cdot x^T_{(\text{iter}-1)}\hat{P} + \hat{v}^T, \qquad \text{iter} = 1, 2, \ldots, \tag{1.4}$$

where $x^T_{(\text{iter})}$ stands for the approximation obtained from the iter-th iteration. This algorithm is equivalent to the power method, however, Gleich et al. (2005) demonstrated that iterations of linear system may converge faster than the simple power method, and are less sensitive to the changes in teleportation.

The World Wide Web exhibits a well-defined structure, characterized by several interesting properties. For instance, this structure was clearly revealed by Broder et al. (2000) who presented the evocative bow-tie picture of the Web. Donato et al. 2008 presented a series of measurements on the Web, which offer a better understanding of the individual components of the bow-tie. Therefore, it is interesting to investigate efficient algorithms that rely on the structure of the Web. Lumping Google's PageRank via reordering the Web matrix have been investigated in (Arasu 2002; Del et al. 2004; Eiron et al. 2004; Ipsen and Selee 2007; Kamvar et al. 2003b; Langville and Meyer 2006a; Lee et al. 2007; Lin et al. 2009). In essence, lumping is a special permutation of the hyperlink matrix underlying the PageRank problem. So it can be viewed as a unitary similarity transformation to the Google matrix. This permutation induces a special structure on the solution, which follows from the block form of the permuted hyperlink matrix. From this structure, computing a PageRank vector involves solving a core problem on a smaller matrix, and then filling in the rest of the details.

One advantage of the lumping algorithms is that they are often easier to compute than a strongly connected decomposition of a matrix. There are at least two advantages in reducing the matrix dimension: faster computation and smaller round-off error (Wills and Ipsen 2009). For instance, in (2002), Arasu proposed a strong component decomposition. Del Corso et al. (2004) explored permutations to actually make iterative computations faster. Lee et al. (2007) presented a fast two-stage algorithm for computing the PageRank vector. The algorithm exploits the observation that the homogeneous discrete-time Markov chain associated with PageRank is lumpable, with the lumpable subset of nodes being the dangling nodes. Motivated by Lee's strategy, Langville and Meyer (2006a) gave a recursively reordering algorithm for computing PageRank. In the recursively reordering process, locating zero rows can be repeated recursively on smaller and smaller sub-matrices of the Web matrix $\hat{P}$, continuing until a sub-matrix is created that has no zero rows. As a result, the recursively reordering algorithm reduces the computation of the PageRank vector to that of solving a much smaller linear system, and then using forward substitution to obtain the full solution vector. Unfortunately, this algorithm may suffer from the overhead of the recursively reordering procedure itself. Ipsen and Selee (2007) expressed lumping as a similar transformation of the Google matrix, and showed that the PageRank of the non-dangling nodes can be computed separately from that of the dangling nodes. Lin et al. (2009) extended the results of (Ipsen and Selee 2007), and proved that the reduced matrix obtained by lumping the dangling nodes can be further reduced by lumping weakly non-dangling nodes to another single node, and the further reduced matrix is also stochastic with the same nonzero eigenvalues as the Google matrix. In essence, all the results in (Ipsen and Selee 2007; Lin et al. 2009) considered the PageRank problem from a large scale eigenvalue problem point of view, and the algorithms are based on the power method. These algorithms are efficient for the computation of PageRank, especially when the number of dangling nodes exceed that of non-dangling nodes. However, when the number of the dangling nodes is not overwhelming, they may be unsatisfactory in practice.

To our best knowledge, none of the above-mentioned lumping algorithms explores the effect of the *unreferenced pages*, i.e., the pages without inlinks, to the PageRank computation. We refer to the nodes corresponding to the unreferenced pages as *the unreferenced nodes*, and *referenced nodes* otherwise. Therefore, a node is either referenced or unreferenced. Notice that the columns of the Web matrix associated with the unreferenced nodes are zero. In this paper, we lump the Web matrix with respect to five type nodes. We show that the size of the PageRank problem can be further reduced to solve PageRank linear system corresponding to *the strongly non-dangling & referenced nodes*, and the full PageRank can be obtained by some recursion formulations. Compared with the lumping strategies due to Lee et al. (2007) and Lin et al. (2009), our new lumping strategy can reduce the original PageRank problem to a (much) smaller one, and the overhead of the three reordering schemes is comparable. Furthermore, the new lumping strategy is much cheaper than the recursively reordering strategy due to Langville and Meyer (2006). Numerical experiments show that the new algorithm is favorable when the matrix is large and the damping factor is high.

This paper is organized as follows. In Sect. 2, we briefly review some known lumping strategies for the Web matrices. In Sect. 3, we propose a new strategy that is based on lumping five type nodes for computing PageRank, and apply it to compute the first order derivative of the PageRank vector. Sensitivity of the PageRank problem is also discussed. In Sect. 4, we make numerical experiments on some real-world problems, which illustrate numerical behavior of our new algorithms. Concluding remarks are given in Sect. 5.

2 Some lumping algorithms for the PageRank problem

In this section, we briefly introduce the ideas behind lumping the Web matrix with respect to two and three type nodes (Ipsen and Selee 2007; Lee et al. 2007; Lin et al. 2009), as well as the recursively reordering algorithm due to Langville and Meyer (2006a, b).

2.1 Lumping the Web matrix with respect to two and three type nodes

The nodes in the Web can be classified into two classes, i.e., *the non-dangling nodes* (*ND*) and *the dangling nodes* (*D*) (Langville and Meyer 2006b). It is interesting to exclude the dangling nodes with their artificial links from the PageRank computation (Ipsen and Selee 2007; Langville and Meyer 2006a, b; Lee et al. 2007; Lin et al. 2009). This can be done by lumping all the dangling nodes into a single node, and the PageRank of the non-dangling nodes can be computed separately from that of the dangling nodes (Lee et al. 2007). Consequently, a large amount of operations may be saved.

If the rows and columns of $\hat{P}$ are permuted (i.e., the indices are reordered), so that the rows corresponding to dangling nodes are at the bottom of the hyper-link matrix, i.e.,

$$P = Q\hat{P}Q^T = \begin{array}{c} \\ ND \\ D \end{array} \begin{array}{c} \begin{array}{cc} ND & D \end{array} \\ \begin{bmatrix} P_{11} & P_{12} \\ \mathbf{O} & \mathbf{O} \end{bmatrix} \end{array}, \tag{2.1}$$

where Q is a permutation matrix with each row and column has exactly one 1 and all other entries are 0. Note that P_{11} represents the links among the non-dangling nodes, and P_{12} represents the links from non-dangling to dangling nodes, and the zero rows in P are associated with the dangling nodes.

It follows from (1.3) and (2.1) that

$$\hat{x}^T Q^T \cdot Q(I - \alpha \hat{P}) Q^T = \hat{v}^T Q^T,$$

that is,

$$x^T (I - \alpha P) = v^T, \tag{2.2}$$

where $x^T = \hat{x}^T Q^T$, and $v^T = \hat{v}^T Q^T$, and the PageRank vector is

$$\hat{x}^T = x^T Q. \tag{2.3}$$

Partition consistently with (2.1) that $x = [x_1^T, x_2^T]^T$, and $v = [v_1^T, v_2^T]^T$, we have

$$x_1^T = v_1^T (I - \alpha P_{11})^{-1}, \tag{2.4}$$

$$x_2^T = \alpha x_1^T P_{12} + v_2^T. \tag{2.5}$$

Therefore, if we divide the nodes into non-dangling and dangling nodes, then it is sufficient to apply the Jacobi iteration on a smaller matrix P_{11} to compute the PageRank vector. Indeed, this algorithm is mathematically equivalent to the specialized iterative algorithm proposed by Lee et al. (2007).

Algorithm 1 (Langville and Meyer 2006) A lumping algorithm with respect to two type nodes (Lump 2)

1. *Reorder the hyper-link matrix (as well as the personality vector $\hat{v}$) so that the reordered matrix has the structure of* (2.1).
2. *Start: Given a prescribed tolerance tol, and the damping factor α.*
3. *Solve $x_1^T(I - \alpha P_{11}) = v_1^T$, say, by using the Jacobi iteration.*
4. *Compute $x_2^T = \alpha x_1^T P_{12} + v_2^T$.*
5. *Set $x = [x_1^T, x_2^T]^T$, compute the PageRank vector $x^T := x^T Q$, and normalize $x^T := x^T / \|x^T\|_1$.*

Motivated by the idea proposed in (Lee et al. 2007; Lin et al. (2009) further classified the non-dangling nodes into two classes. Consider the nodes that are not dangling, but only pointing to the dangling nodes. They refer to these non-dangling nodes as *the weakly non-dangling nodes* [notice that these nodes are called *the weakly dangling nodes* in (Lee et al. 2007)]. The other non-dangling nodes are called *the strongly non-dangling nodes*. Thus, a node is either dangling (D), weakly non-dangling (W), or strongly non-dangling (S). Moreover, Lin et al. (2009) showed that one of two classes of non-dangling nodes can also be lumped to a single node, and the PageRank of the other class of non-dangling nodes can be computed separately.

Therefore, if the rows and columns of $\hat{P}$ are permuted (i.e., the indices are reordered), so that the rows corresponding to dangling nodes are at the bottom of the hyper-link matrix, and the rows corresponding to strongly non-dangling nodes are at the top of the hyper-link matrix, i.e.,

$$P = Q\hat{P}Q^T = \begin{array}{c} \\ S \\ W \\ D \end{array} \begin{array}{c} \begin{array}{ccc} S & W & D \end{array} \\ \begin{bmatrix} P_{11} & P_{12} & P_{13} \\ \mathbf{O} & \mathbf{O} & P_{23} \\ \mathbf{O} & \mathbf{O} & \mathbf{O} \end{bmatrix} \end{array}, \tag{2.6}$$

where Q is a permutation matrix. Then P_{11} represents the links among strongly non-dangling nodes, P_{12} denotes the links from strongly non-dangling to weakly non-dangling nodes, P_{13} stands for the links from strongly non-dangling to dangling nodes, and P_{23} represents the links from weakly non-dangling to dangling nodes.

Partition consistently with (2.6) that $x = [x_1^T, x_2^T, x_3^T]^T$, and $v = [v_1^T, v_2^T, v_3^T]^T$. It is easy to verify that

$$x_1^T(I - \alpha P_{11}) = v_1^T, \tag{2.7}$$

and

$$x_2^T = \alpha x_1^T P_{12} + v_2^T, \tag{2.8}$$

$$x_3^T = \alpha x_1^T P_{13} + \alpha x_2^T P_{23} + v_3^T. \tag{2.9}$$

We can present the following algorithm with respect to three types of nodes for computing PageRank. This algorithm is simpler and cleaner than, but is mathematically equivalent to, the iterative method proposed by Lin et al. (2009, Algorithm 2.1).

Algorithm 2 A lumping algorithm with respect to three type nodes (Lump 3)

1. *Reorder the hyper-link matrix (as well as the personality vector $\hat{v}$) so that the reordered matrix has the structure of* (2.6).
2. *Start: Given a prescribed tolerance tol, and the damping factor α.*
3. *Solve the linear system $x_1^T(I - \alpha P_{11}) = v_1^T$, say, by using the Jacobi iteration.*
4. *Compute $x_2^T = \alpha x_1^T P_{12} + v_2^T$, and $x_3^T = \alpha x_1^T P_{13} + \alpha x_2^T P_{23} + v_3^T$.*
5. *Set $x = [x_1^T, x_2^T, x_3^T]^T$, compute $x^T := x^T Q$, and normalize $x^T := x^T / \|x^T\|_1$.*

2.2 The recursively reordering algorithm for PageRank

In (2006a), Langville and Meyer proposed a recursively reordering PageRank algorithm, which is a more general version of the dangling node method due to Lee et al. (2007). The key idea is that the process of locating zeros rows can be repeated recursively on smaller and smaller sub-matrices of $\hat{P}$, continuing until a sub-matrix is created that has no zero rows. In general, after this symmetric reordering, the hyper-link matrix has the following structure (Langville and Meyer 2006)

$$P = Q\hat{P}Q^T = \begin{bmatrix} P_{11} & P_{12} & P_{13} & \cdots & P_{1p} \\ & \mathbf{O} & P_{23} & \cdots & P_{2p} \\ & & \mathbf{O} & \cdots & P_{3p} \\ & & & \ddots & \vdots \\ & & & & \mathbf{O} \end{bmatrix}, \tag{2.10}$$

where Q is a permutation matrix, and p (≥ 2) is the number of square diagonal blocks in the reordered matrix. The steps of the recursively reordering PageRank algorithm are enumerated as follows.

Algorithm 3 (Langville and Meyer 2006) The recursively reordering algorithm for PageRank (Recursive)

1. *Reorder the Web matrix (as well as the personality vector $\hat{v}$) according to the recursive dangling node idea.*
2. *Start: Choose a prescribed tolerance tol, and the damping factor α.*
3. *Solve for x_1^T in $x_1^T(I - \alpha P_{11}) = v_1^T$.*
4. *For $i = 2, 3, \ldots, p$, form $x_i^T = \alpha \sum_{j=1}^{i-1} x_j^T P_{ji} + v_i^T$.*
5. *Set $x = [x_1^T, x_2^T, \ldots, x_p^T]^T$, compute $x^T := x^T Q$, and normalize $x^T := x^T / \|x^T\|_1$.*

The recursively reordering algorithm reduces the computation of the PageRank vector to that of solving a much smaller linear system, and then using forward substitution to get the full solution vector. However, as is shown in Sect. 5, this algorithm may suffer from the overhead of the recursively reordering procedure when p, the number of square diagonal blocks in the reordered matrix, is relatively large.

3 Lumping five type nodes for computing PageRank and its derivative vector

We call the nodes corresponding to the unreferenced pages *the unreferenced nodes*, and *referenced nodes* otherwise. Therefore, a node is either referenced or unreferenced. As an aside, the proper way of looking at dangling node algorithms is in terms of cores in a graph (Batagelj and Zaveršnik 2012). Dangling nodes are vertices with 0 out-core numbers, and unreferenced nodes are vertices with 0 in-core numbers. In this section, we reorder the Web matrix with respect to five type nodes, and consider the nodes that are *strongly non-dangling and referenced*.

3.1 A lumping algorithm with respect to five type nodes for PageRank

In this paper, we point out that the Web matrix can be lumped according to five type nodes: the strongly non-dangling & referenced nodes ($S\&R$), the strongly non-dangling & unreferenced nodes ($S\&UR$), the weakly non-dangling nodes (W), the dangling & referenced nodes ($D\&R$), and the dangling & unreferenced nodes ($D\&UR$). After such a reduction, we will see that the computation time for PageRank mainly depends on the number of the strongly non-dangling & referenced nodes, and the round-off error mainly depends on the maximal number of inlinks to the strongly non-dangling & referenced nodes.

Figure 1 depicts a picture explaining connections between different components. It is seen that node 1 is a strongly non-dangling & unreferenced node ($S\&UR$), node 2 is a strongly non-dangling & referenced node ($S\&R$), node 3 is a dangling & referenced node ($D\&R$), node 4 is a dangling & unreferenced node ($D\&UR$), and node 5 is a weakly non-dangling node (W).

Assume that we have reordered the original Web matrix $\hat{P}$ so that the unreferenced nodes are numbered lastly, then the resulting matrix is of the form

$$P = Q\hat{P}Q^T = \begin{array}{c} \\ S\&R \\ S\&UR \\ W \\ D\&R \\ D\&UR \end{array} \begin{array}{c} \begin{array}{ccccc} S\&R & S\&UR & W & D\&R & D\&UR \end{array} \\ \begin{bmatrix} P_{11} & \mathbf{O} & P_{13} & P_{14} & \mathbf{O} \\ P_{21} & \mathbf{O} & P_{23} & P_{24} & \mathbf{O} \\ \mathbf{O} & \mathbf{O} & \mathbf{O} & P_{34} & \mathbf{O} \\ \mathbf{O} & \mathbf{O} & \mathbf{O} & \mathbf{O} & \mathbf{O} \\ \mathbf{O} & \mathbf{O} & \mathbf{O} & \mathbf{O} & \mathbf{O} \end{bmatrix} \end{array}, \tag{3.1}$$

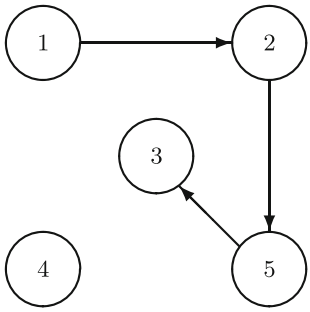

Fig. 1 Directed graph representing Web of 5 pages

where Q is a permutation matrix, and P_{11} represents the links among the strongly non-dangling & referenced nodes (*S&R*). It is obvious to see that the size of P_{11} obtained from our new lumping strategy is at least as small as those from (Lee et al. 2007; Lin et al. 2009). However, we stress that one can not tell which strategy, the recursively reordering strategy or the Lumping 5 strategy, can reduce the original linear system to a smaller one, and the answer is problem-dependent.

We see from (3.1) that the eigenvalues of P are union of the eigenvalues of P_{11} and some zero. Thus, P_{11} and P share the same non-zero eigenvalues, and $I - \alpha P_{11}$ is nonsingular. Indeed, all eigenvalues of the matrix αP_{11} are less than 1, and $(I - \alpha P_{11})^{-1}$ can be represented as a infinite sum of positive matrices (Saad 2003). As a result, both $(I - \alpha P_{11})^{-1}$ and $(I - \alpha P_{11})^{-1} P_{13}$ are nonnegative matrices. If we define

$$||z||_{P_1} \equiv z^T (I - \alpha P_{11})^{-1} \mathbf{e}, \qquad ||z||_{P_3} \equiv z^T (I - \alpha P_{11})^{-1} P_{13} \mathbf{e},$$

then $||z||_{P_1}$ and $||z||_{P_3}$ are vector norms for $z \geq \mathbf{O}$ (Ipsen and Selee 2007). The following theorem provides an expression for x^T. It also gives the weights of PageRanks corresponding to the five type nodes.

Theorem 3.1 *Partition consistently with* (3.1) *that* $x = [x_1^T, x_2^T, x_3^T, x_4^T, x_5^T]^T$, *and* $v = [v_1^T, v_2^T, v_3^T, v_4^T, v_5^T]^T$. *Under the above notation, there hold*

$$\begin{aligned}
x_1^T &= (v_1^T + \alpha v_2^T P_{21})(I - \alpha P_{11})^{-1}, \\
x_2^T &= v_2^T, \\
x_3^T &= \alpha(x_1^T P_{13} + x_2^T P_{23}) + v_3^T, \\
x_4^T &= \alpha(x_1^T P_{14} + x_2^T P_{24} + x_3^T P_{34}) + v_4^T, \\
x_5^T &= v_5^T.
\end{aligned}$$

Moreover, we have

$$\begin{aligned}
||x_1||_1 &= ||v_1||_{P_1} + \alpha ||P_{21}^T v_2||_{P_1}, \\
||x_2||_1 &= ||v_2||_1, \\
||x_3||_1 &= \alpha ||v_1||_{P_3} + \alpha^2 ||P_{21}^T v_2||_{P_3} + \alpha ||P_{23}^T v_2||_1 + ||v_3||_1, \\
||x_4||_1 &= 1 - ||v_1||_{P_1} - \alpha ||P_{21}^T v_2||_{P_1} - \alpha ||v_1||_{P_3} - \alpha^2 ||P_{21}^T v_2||_{P_3} - \alpha ||P_{23}^T v_2||_1 \\
&\quad - ||v_3||_1 - ||v_2||_1 - ||v_5||_1, \\
||x_5||_1 &= ||v_5||_1.
\end{aligned}$$

Proof Notice that

$$I - \alpha P = \begin{bmatrix}
I - \alpha P_{11} & \mathbf{O} & -\alpha P_{13} & -\alpha P_{14} & \mathbf{O} \\
-\alpha P_{21} & I & -\alpha P_{23} & -\alpha P_{24} & \mathbf{O} \\
\mathbf{O} & \mathbf{O} & I & -\alpha P_{34} & \mathbf{O} \\
\mathbf{O} & \mathbf{O} & \mathbf{O} & I & \mathbf{O} \\
\mathbf{O} & \mathbf{O} & \mathbf{O} & \mathbf{O} & I
\end{bmatrix}.$$

Furthermore, it follows from $x^T(I - \alpha P) = v^T$ that

$$[x_1^T, x_2^T, x_3^T, x_4^T, x_5^T] \begin{bmatrix} I - \alpha P_{11} & \mathbf{O} & -\alpha P_{13} & -\alpha P_{14} & \mathbf{O} \\ -\alpha P_{21} & I & -\alpha P_{23} & -\alpha P_{24} & \mathbf{O} \\ \mathbf{O} & \mathbf{O} & I & -\alpha P_{34} & \mathbf{O} \\ \mathbf{O} & \mathbf{O} & \mathbf{O} & I & \mathbf{O} \\ \mathbf{O} & \mathbf{O} & \mathbf{O} & \mathbf{O} & I \end{bmatrix} = [v_1^T, v_2^T, v_3^T, v_4^T, v_5^T].$$

That is,

$$\begin{aligned} & x_1^T(I - \alpha P_{11}) - \alpha x_2^T P_{21} = v_1^T, \\ & x_2^T = v_2^T, \\ & \quad - \alpha x_1^T P_{13} - \alpha x_2^T P_{23} + x_3^T = v_3^T, \\ & \quad - \alpha x_1^T P_{14} - \alpha x_2^T P_{24} - \alpha x_3^T P_{34} + x_4^T = v_4^T, \\ & x_5^T = v_5^T. \end{aligned}$$

Or equivalently,

$$\begin{aligned} x_1^T &= (v_1^T + \alpha v_2^T P_{21})(I - \alpha P_{11})^{-1}, \\ x_2^T &= v_2^T, \\ x_3^T &= \alpha(x_1^T P_{13} + x_2^T P_{23}) + v_3^T, \\ x_4^T &= \alpha(x_1^T P_{14} + x_2^T P_{24} + x_3^T P_{34}) + v_4^T, \\ x_5^T &= v_5^T. \end{aligned}$$

Next we consider the weights of the PageRanks corresponding to the five types nodes. Taking 1-norm of x_1, x_2, x_3, x_4 and x_5 yields

$$\begin{aligned} ||x_1||_1 = x_1^T \mathbf{e} &= (v_1^T + \alpha v_2^T P_{21})(I - \alpha P_{11})^{-1} \mathbf{e} \\ &= v_1^T (I - \alpha P_{11})^{-1} \mathbf{e} + \alpha v_2^T P_{21} (I - \alpha P_{11})^{-1} \mathbf{e} \\ &= ||v_1||_{P_1} + \alpha ||P_{21}^T v_2||_{P_1}, \end{aligned}$$

and

$$||x_2||_1 = x_2^T \mathbf{e} = v_2^T \mathbf{e} = ||v_2||_1.$$

Moreover,

$$\begin{aligned} ||x_3||_1 = x_3^T \mathbf{e} &= [\alpha(x_1^T P_{13} + x_2^T P_{23}) + v_3^T]\mathbf{e} \\ &= \alpha x_1^T P_{13} \mathbf{e} + \alpha x_2^T P_{23} \mathbf{e} + v_3^T \mathbf{e} \\ &= \alpha(v_1^T + \alpha v_2^T P_{21})(I - \alpha P_{11})^{-1} P_{13} \mathbf{e} + \alpha v_2^T P_{23} \mathbf{e} + v_3^T \mathbf{e} \\ &= \alpha ||v_1||_{P_3} + \alpha^2 ||P_{21}^T v_2||_{P_3} + \alpha ||P_{23}^T v_2||_1 + ||v_3||_1, \end{aligned}$$

and

$$\begin{aligned} ||x_5||_1 &= x_5^T \mathbf{e} = v_5^T \mathbf{e} = ||v_5||_1, \\ ||x_4||_1 &= 1 - ||x_1||_1 - ||x_2||_1 - ||x_3||_1 - ||x_5||_1 \\ &= 1 - ||v_1||_{P_1} - \alpha ||P_{21}^T v_2||_{P_1} - \alpha ||v_1||_{P_3} - \alpha^2 ||P_{21}^T v_2||_{P_3} \\ &\quad - \alpha ||P_{23}^T v_2||_1 - ||v_3||_1 - ||v_2||_1 - ||v_5||_1. \end{aligned}$$

□

Theorem 3.1 indicates that, if the nodes can be divided into five types, then the main overhead is to iteratively solve a (possibly much) smaller linear system on x_1, i.e., the PageRank of the *strongly non-dangling & referenced nodes*, while x_3 and x_4 can be easily derived from the recursion formulations. In summary, the main algorithm of this paper is outlined as follows.

Algorithm 4 A lumping algorithm with respect to five type nodes (Lump 5)

1. *Reorder the hyper-link matrix (as well as the personality vector $\hat{v}$) so that the reordered matrix has the structure of* (3.1).
2. *Start: Given a prescribed tolerance tol, and the damping factor α.*
3. *Solve the linear system $x_1^T(I - \alpha P_{11}) = v_1^T + \alpha v_2^T P_{21}$, say, by using the Jacobi iteration.*
4. *Compute $x_3^T = \alpha(x_1^T P_{13} + x_2^T P_{23}) + v_3^T$, and $x_4^T = \alpha(x_1^T P_{14} + x_2^T P_{24} + x_3^T P_{34}) + v_4^T$.*
5. *Set $x = [x_1^T, v_2^T, x_3^T, x_4^T, v_5^T]^T$, compute $x^T := x^T Q$, and normalize $x^T := x^T / \|x^T\|_1$.*

Remark 3.1 Since Q is a permutation matrix whose each row and column has exactly one 1 and all other entries are 0, we can use an n-dimensional vector to store it, and compute $x^T Q$ in $\mathcal{O}(n)$ operations.

Finally, we point out that one can also take *the referenced and unreferenced weakly-dangling nodes* into consideration. In this case, the reordered Web matrix is of the following form

$$P = Q\hat{P}Q^T = \begin{array}{c} \\ S\&R \\ S\&UR \\ W\&R \\ W\&UR \\ D\&R \\ D\&UR \end{array} \begin{array}{c} \begin{array}{cccccc} S\&R & S\&UR & W\&R & W\&UR & D\&R & D\&UR \end{array} \\ \begin{bmatrix} P_{11} & \mathbf{O} & P_{13} & \mathbf{O} & P_{15} & \mathbf{O} \\ P_{21} & \mathbf{O} & P_{23} & \mathbf{O} & P_{25} & \mathbf{O} \\ \mathbf{O} & \mathbf{O} & \mathbf{O} & \mathbf{O} & P_{35} & \mathbf{O} \\ \mathbf{O} & \mathbf{O} & \mathbf{O} & \mathbf{O} & P_{45} & \mathbf{O} \\ \mathbf{O} & \mathbf{O} & \mathbf{O} & \mathbf{O} & \mathbf{O} & \mathbf{O} \\ \mathbf{O} & \mathbf{O} & \mathbf{O} & \mathbf{O} & \mathbf{O} & \mathbf{O} \end{bmatrix} \end{array}. \tag{3.2}$$

Based on this "Lumping 6" strategy, it is easy to check that the expression for x is a little more complicated than the one from Lump 5, and the derivation of a lumping algorithm with respect to six type nodes is straightforward. However, similar to the Lumping 5 strategy, the computation time for PageRank still mainly depends on the number of *the strongly non-dangling & referenced nodes*, and the round-off error still mainly depends on the maximal number of inlinks to *the strongly non-dangling & referenced nodes*.

Example 3.1 In this example, we try to illustrate the reordering strategies in terms of two types (Ipsen and Selee 2007), three types (Lee et al. 2007), five type nodes, as well as the recursively reordering strategy (Langville and Meyer 2006a). To this end, we plot the structure of the original and the reordered Web matrices. The test matrix is the 4772 × 4772 *Epa* Web matrix available from http://www.cs.cornell.edu/Courses/cs685/2002fa. It contains 1,146 dangling nodes, and 1,474 unreferenced nodes. Figures 2 and 3 depict the structures of the original as well as the reordered adjacency matrices, respectively.

From this example, the advantage of lumping five type nodes is obvious. Indeed, one has to solve a linear system of size 4772 × 4772 for the original Jacobi iteration, a linear system of size 3626 × 3626 for Lump 2, and a 1363 × 1363 linear system for Lump 3. There are 9 square diagonal blocks along the recursively reordered matrix, and the size of the resulting iteration matrix is 590 × 590. As a comparison, we only need to solve a 390 × 390 linear system for Lump 5.

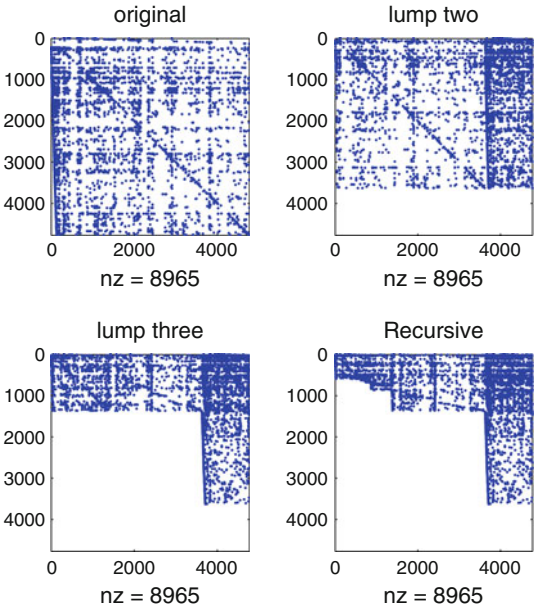

Fig. 2 Structures of the original and the three reordered *Epa* Web matrix, where the *dots* represent nonzero elements, and *white* stands for zero elements

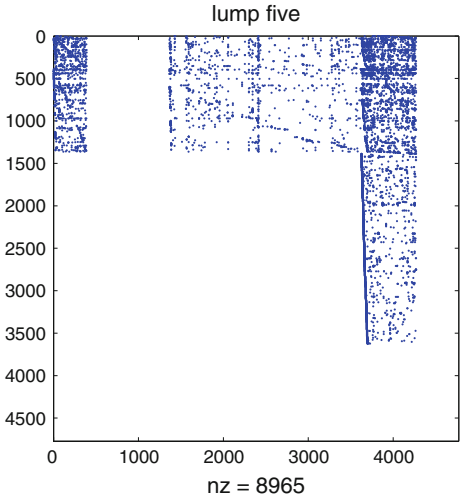

Fig. 3 Structure of the reordered *Epa* Web matrix with respect to lumping five type of nodes, where the *dots* represent nonzero elements, and *white* stands for zero elements

3.2 Sensitivity of the PageRank vector and a lumping algorithm for computing its first order derivative

In this subsection, we firstly consider the *structured perturbation* of the Google matrix. That is, we focus on sensitivity of the PageRank vector when the Google matrix G is perturbed by some perturbation F, while $\widetilde{G} = G + F$ is still a Google matrix. It is shown that some results of (Ipsen and Wills 2006) are just special cases of our theorem. Secondly, we propose an algorithm for computing the first order derivative of the PageRank vector with respect to α, using the new lumping strategy proposed in Sect. 3.1.

Theorem 3.2 *Let G be a Google matrix and $\hat{x}$ be the PageRank vector. Suppose that $\widetilde{G} = G + F$ is the perturbed Google matrix and $\tilde{x}$ is the associated PageRank vector, then*

$$||\tilde{x} - \hat{x}||_1 \leqslant \frac{1}{1-\alpha}||F||_\infty, \tag{3.3}$$

where $||\cdot||_1$ and $||\cdot||_\infty$ denote the 1-norm and ∞-norm of a matrix or vector, respectively.

Proof Notice that

$$\hat{x}^T G = \hat{x}^T, \quad \mathbf{e}^T\hat{x} = 1, \quad \hat{x} \geqslant \mathbf{O},$$

and

$$\tilde{x}^T \widetilde{G} = \tilde{x}^T, \quad \mathbf{e}^T\tilde{x} = 1, \quad \tilde{x} \geqslant \mathbf{O}.$$

Therefore,

$$\begin{aligned}\tilde{x}^T - \hat{x}^T &= \tilde{x}^T\widetilde{G} - \hat{x}^T G = (\tilde{x}^T - \hat{x}^T)G + \tilde{x}^T F \\ &= (\tilde{x}^T - \hat{x}^T)[\alpha\widetilde{P} + (1-\alpha)\mathbf{e}v^T] + \tilde{x}^T F \\ &= \alpha(\tilde{x}^T - x^T)\widetilde{P} + \tilde{x}^T F,\end{aligned}$$

where we use the fact that $\tilde{x}^T\mathbf{e} = \hat{x}^T\mathbf{e} = 1$. Thus,

$$(\tilde{x}^T - \hat{x}^T)(I - \alpha\widetilde{P}) = \tilde{x}^T F.$$

Recall that $I - \alpha\widetilde{P}$ is nonsingular in that $\rho(\alpha\widetilde{P}) \leqslant ||\alpha\widetilde{P}||_\infty = \alpha < 1$, and $(I - \alpha\widetilde{P})\mathbf{e} = (1-\alpha)\mathbf{e}$. So we obtain

$$(I - \alpha\widetilde{P})^{-1}\mathbf{e} = \frac{1}{1-\alpha}\mathbf{e}.$$

Since $(I - \alpha\widetilde{P})^{-1} \geqslant \mathbf{O}$ (Langville and Meyer 2006), we have

$$||(I - \alpha\widetilde{P})^{-1}||_\infty = \left\|\frac{1}{1-\alpha}\mathbf{e}\right\|_\infty = \frac{1}{1-\alpha}.$$

Consequently,

$$\begin{aligned}||\tilde{x} - \hat{x}||_1 &= ||\tilde{x}^T - \hat{x}^T||_\infty = ||\tilde{x}^T F(I - \alpha\widetilde{P})^{-1}||_\infty \leqslant ||\tilde{x}^T||_\infty ||F||_\infty ||(I - \alpha\widetilde{P})^{-1}||_\infty \\ &= \frac{1}{1-\alpha}||F||_\infty,\end{aligned}$$

which completes the proof. □

Remark 3.2 We mention that some results of (Ipsen and Wills 2006) are special cases of Theorem 3.2, and our result is more general.

- If we choose $F = \alpha E$, then (3.3) reduces to (Ipsen and Wills 2006)

$$||\tilde{x} - x||_1 \leqslant \frac{1}{1-\alpha}||F||_\infty = \frac{\alpha}{1-\alpha}||E||_\infty.$$

- If we select $F = \mu(\widetilde{P} - \mathbf{e}v^T)$, then (3.3) becomes (Ipsen and Wills 2006)

$$||\tilde{x} - x||_1 \leqslant \frac{1}{1-\alpha}||F||_\infty \leqslant \frac{1}{1-\alpha}|\mu|\left[||\widetilde{P}||_\infty + ||\mathbf{e}v^T||_\infty\right] = \frac{2}{1-\alpha}|\mu|.$$

- If we take $F = (1-\alpha)\mathbf{e}f^T$, then (3.3) turns out to be (Ipsen and Wills 2006)

$$||\tilde{x} - x||_1 \leqslant \frac{1}{1-\alpha}||F||_\infty \leqslant \frac{1-\alpha}{1-\alpha}||\mathbf{e}f^T||_\infty \leqslant ||f||_1.$$

It is necessary to consider how the ordering of the PageRank vector changes related to the damping factor. One way is to compute the first order derivative of the PageRank vector with respect to α (Boldi et al. 2009; Gleich et al. 2007). In (2009), Boldi et al. presented two formulations for computing derivatives of PageRank, and Gleich et al. (2007) considered how to compute the first order derivative vector via solving PageRank problems.

Next we show that a combination of the Lumping 5 scheme with Gleich et al.'s strategy can be used to compute the first order derivative of the PageRank vector efficiently. Since the PageRank vector is a function on α, we rewrite $\hat{x}^T$ as $\hat{x}^T(\alpha)$ if there is no ambiguity. The following theorem due to Boldi *et al.* established a relationship between the k-th derivative $[\hat{x}^T(\alpha)]^{(k)}$ and the $(k+1)$-th derivative $[\hat{x}^T(\alpha)]^{(k+1)}$.

Theorem 3.3 (Boldi et al. 2009) *The following identities hold*

$$\begin{aligned} &1.\ [\hat{x}^T(\alpha)]'(I - \alpha\widetilde{P}) = \hat{x}^T(\alpha)\widetilde{P} - \hat{v}^T, \\ &2.\ [\hat{x}^T(\alpha)]^{(k+1)}(I - \alpha\widetilde{P}) = (k+1)[\hat{x}^T(\alpha)]^{(k)}\widetilde{P}, \text{ for any } k > 0. \end{aligned} \tag{3.4}$$

In terms of Theorem 3.3, we can present the following algorithm for computing the first order derivative of the PageRank vector. Indeed, it is the Jacobi iteration applies to (3.4). We see that the algorithm converges unconditionally, since the spectrum radius $\rho(\alpha\widetilde{P}) \leq \alpha < 1$.

Algorithm 5 An iterative algorithm for computing the first order derivative of PageRank (Dev-Boldi)

1. *Start: Choose the initial vector* $y_{(0)} = \mathbf{O}$, *the value of* α, *the personality vector* $\hat{v}$, *and a prescribed tolerance tol, set iter* $= 1$ *and* $r = 1$.
2. *Iterate: Compute* $\hat{x}(\alpha)$ *as the solution to the original PageRank problem* (1.3).
3. *Set* $b = \frac{1}{\alpha}(\hat{x}^T(\alpha) - \hat{v}^T)$, *refer to* (3.7).
4. *Iterate:*

while $r > tol$

$$y^T_{(\text{iter})} = \alpha\left[y^T_{(\text{iter}-1)}\hat{P} + \left(y^T_{(\text{iter}-1)}d\right)\hat{v}^T\right] + b;$$

$$r = \left\|y_{(\text{iter})} - y_{(\text{iter}-1)}\right\|_1;$$

$$\text{iter} = \text{iter} + 1;$$

end

Now we briefly introduce Gleich et al.'s strategy (2007) for computing the first order derivative of the PageRank vector. Notice that $e^T[\hat{x}(\alpha)]' = 0$, which follows directly from the fact that $\mathbf{e}^T\hat{x}(\alpha) = 1$. We can rewrite (3.4) as

$$\left[\hat{x}^T(\alpha)\right]'(I-\alpha\widetilde{P}) = \frac{1}{\alpha(1-\alpha)}\left[(1-\alpha)\hat{x}^T(\alpha) - (1-\alpha)\hat{v}^T\right]. \tag{3.5}$$

Indeed, it follows from (1.1) and (1.2) that

$$\hat{x}^T(\alpha)(I-\alpha\widetilde{P}) = (1-\alpha)\hat{v}^T, \tag{3.6}$$

where we use $\hat{x}(\alpha)^T\mathbf{e} = 1$. Therefore,

$$\hat{x}^T(\alpha)\widetilde{P} - \hat{v}^T = \frac{1}{\alpha}\left(\hat{x}^T(\alpha) - \hat{v}^T\right), \tag{3.7}$$

and (3.5) follows from (3.4) and (3.7). Let $y(\alpha)$ satisfy $y^T(\alpha)(I-\alpha\widetilde{P}) = (1-\alpha)\hat{x}^T(\alpha)$, then

$$\left[\hat{x}^T(\alpha)\right]'(I-\alpha\widetilde{P}) = \frac{1}{\alpha(1-\alpha)}\left[y^T(\alpha)(I-\alpha\widetilde{P}) - \hat{x}^T(\alpha)(I-\alpha\widetilde{P})\right],$$

that is,

$$\left[\hat{x}^T(\alpha)\right]' = \frac{1}{\alpha(1-\alpha)}\left(y^T(\alpha) - \hat{x}^T(\alpha)\right).$$

In summary, this strategy reduces the computation of the derivative vector to solving two PageRank problems. We can present the following algorithm for computing the first order derivative of PageRank.

Algorithm 6 An iterative algorithm for computing the first order derivative of PageRank (Dev-Gleich)

1. *Start: Given a prescribed tolerance tol, choose the damping factor* α *and the personality vector* $\hat{v}$.
2. *Iterate: Compute* $\hat{x}(\alpha)$ *as the solution to the original PageRank problem* (1.3).
3. *Iterate: Compute* $y(\alpha)$ *as the solution to the PageRank problem* $y^T(\alpha)(I-\alpha\widetilde{P}) = (1-\alpha)\hat{x}^T(\alpha)$.
4. *Form* $[\hat{x}(\alpha)]' = \frac{1}{\alpha(1-\alpha)}(y(\alpha) - \hat{x}(\alpha))$.

Remark 3.3 We mention that the cost of Algorithm 5 is a little higher than that of Algorithm 6 per iteration. Indeed, the former needs to perform one matrix-vector product, one dot product for computing $y_{(\text{iter}-1)}^T d$, in addition to $\mathcal{O}(n)$ flops for the addition of three n-dimensional vectors. As a comparison, the latter only requires one matrix-vector product, as well as $\mathcal{O}(n)$ flops for the addition of two n-dimensional vectors, refer to (1.4).

Indeed, the Lumping 5 strategy can be utilized in conjunction with Gleich et al.'s strategy to compute the first order derivative of PageRank. The resulting algorithm inherits from some attractive numerical properties of its two parents. In contrast to Algorithm 5, the new algorithm only needs to solve PageRank problems for PageRank's derivative. On the other hand, in contrast to Algorithm 6, one can apply the Jacobi iteration on a (possibly much) smaller linear system to the lumped matrix P_{11}, which may reduce the round-off error and CPU time significantly. This new algorithm is described as follows.

Algorithm 7 A lumping algorithm with respect to five type nodes for computing the first order derivative of PageRank (Dev-Lump 5)

1. *Reorder the Web matrix (as well as the personality vector $\hat{v}$) so that the reordered matrix has the structure of* (3.1).
2. *Start: Given a prescribed tolerance tol, and the damping factor α.*
3. *Iterate: Apply Lump 5 to compute the PageRank vector $\hat{x}(\alpha)$, using (the reordered) $\hat{v}$ as the teleportation distribution, set $\hat{x}(\alpha)^T := \hat{x}(\alpha)^T Q$, and normalize $\hat{x}(\alpha)^T := \hat{x}(\alpha)^T / \|\hat{x}(\alpha)^T\|_1$.*
4. *Iterate: Apply Lump 5 to compute $y(\alpha)$ with the teleportation distribution $\hat{x}(\alpha)$, let $y(\alpha)^T := y(\alpha)^T Q$, and normalize $y(\alpha)^T := y(\alpha)^T / \|y(\alpha)^T\|_1$.*
5. *Form* $[\hat{x}(\alpha)]' = \frac{1}{\alpha(1-\alpha)}(y(\alpha) - \hat{x}(\alpha))$.

4 Numerical experiments

In this section, we report numerical experiments on some real-world problems, and show numerical behavior of our new algorithms. For every Web matrix, we set the diagonal elements to be zero, and transform it into a (row) sub-stochastic matrix. The numerical experiments were run on a Dell Workstation with four core Intel(R) Pentium(R) processor with CPU 3.2 GHz and RAM 16 GB, under the Windows XP 64 bit operating system. All the experimental results were obtained from using a MATLAB 7.7 implementation with machine precision $\epsilon \approx 2.22 \times 10^{-16}$.

Similar to what was done in (Langville and Meyer 2006a), we choose the Jacobi iteration to solve the linear systems in this paper. Indeed, other efficient linear solvers, such as the Gauss-Sediel method (Arasu 2002; Golub and Van 1996), or the GMRES method (Gleich et al. 2005; Saad 2003), can also be applied. The difference is that all of the comparisons are made on a Gauss-Sediel or GMRES-based algorithm for computing the reduced PageRank problems. A numerical comparison of these methods are beyond the scope of this paper, but deserves further investigation.

Table 1 lists summary of the algorithms used in this section. To show efficiency of Algorithm 4 (denoted by "Lump 5"), we compare it with the Jacobi iteration on the original linear system (1.3) [denoted by "Original" in the tables below, which is mathematically equivalent to the original PageRank algorithm (Langville and Meyer 2006b)], Algorithm 1 (denoted by "Lump 2"), Algorithm 2 (denoted by "Lump 3"), as well as the recursively reordering algorithm due to Langville and Meyer (2006a) (denoted by

Table 1 Summary of the algorithms in the numerical experiments

Abbreviation	Algorithm
Original	The Jacobi iteration (1.4) without lumping strategy
Lump 2	The lumping algorithm with respect to 2 type nodes (Lee et al. 2007)
Lump 3	The lumping algorithm with respect to 3 type nodes (Lin et al. 2009)
Recursive	The recursively reordering algorithm for PageRank (Langville and Meyer 2006a)
Lump 5	The lumping algorithm with respect to 5 type nodes
Dev-Boldi	Boldi et al.'s strategy for computing the first order derivative (Boldi et al. 2009)
Dev-Gleich	Gleich et al.'s strategy for computing the first order derivative (Gleich et al. 2007)
Dev-Lump 5	Using Lumping 5 strategy for computing the first order derivative

"Recursive"). For the computation of the first order derivative of the PageRank vector, we compare Algorithm 7 (denoted by "Dev-Lump 5") with Algorithm 5 due to Boldi et al.'s strategy (2009) (denoted by "Dev-Boldi"), and Algorithm 6 due to Gleich et al.'s strategy (denoted by "Dev-Gleich") (2007).

In Tables 3, 4, 5, 6, 7, and 8, we denote by "mv" the number of matrix-vector products, by "size" the size of the iteration matrix in question, by "Lumping" the CPU time used in seconds for lumping (i.e., Step 1), by "Solving" the CPU time used in seconds for solving the corresponding linear system (i.e., Steps 2–5), and by "Total" the total CPU time used in seconds for PageRank computation. That is,

$$\text{Total} = \text{Lumping} + \text{Solving}. \tag{4.1}$$

For the sake of simplicity, in all the algorithms we pick $\hat{v} = \mathbf{e}/n$, and choose the same initial guess $x_{(0)} = [0, 0, \ldots, 0]^T$. All of the final solutions satisfy

$$r = ||x_{(\text{iter})} - x_{(\text{iter}-1)}||_1 \leqslant tol,$$

where $x_{(\text{iter})}$ represents the approximate solution obtained from the iter-th Jacobi iteration, and *tol* is a user-prescribed tolerance. As was done in (Gleich et al. 2005; Golub and Greif 2006; Kamvar et al. 2003a, 2004), in all the numerical experiments the damping factor α are set to be 0.85, 0.90, 0.95 and 0.99, respectively. The test matrices are summarized in Table 2, where "order" denotes the size of the Web matrix, and "nnz" stands for "the number of nonzero elements".

In (2009), Wills and Ipsen made a point of suggesting compensated summation for PageRank algorithms, and mentioned that explicit normalization during iterations is necessary. However, in this paper, we consider the PageRank problem from a linear system rather than from an eigenvalue problem point of view, so our codes do not use this type of summation.

Example 4.1 In this example, we compare Lump 5 with Lump 2 and Lump 3, and try to show superiority of the Lumping 5 strategy over the other two. As a by-product, we also list the numerical results obtained from running the Jacobi iteration on the original linear system. There are two test matrices in this example, whose data files are available from http://www.cise.ufl.edu/research/sparse/matrices/∼Gleich/index.html. The first one is the *Wikipedia-20060925* Web matrix which contains 2,983,494 nodes and 37,251,231 links. It contains 89,124 dangling nodes and 872,457 unreferenced nodes. The second one is the *Wikipedia-20070206* matrix, which contains 3,556,907 nodes and 45,013,315 links. It has 1,077,605 unreferenced nodes and 101,533 dangling nodes.

Figure 4 plots structures of the original and the reordered *Wikipedia-20070206* Web matrices, respectively. One observes that this matrix contains much more unreferenced

Table 2 Test matrices in the numerical experiments

Name	Order	nnz	avg nz per row
Wikipedia-20060925	2,983,494	37,251,231	12.5
Wikipedia-20070206	3,566,907	45,013,315	12.6
indochina-2004	7,414,866	191,606,827	25.8
Wb-edu	9,845,725	55,311,626	5.62
uk-2002	18,520,486	292,243,663	15.8

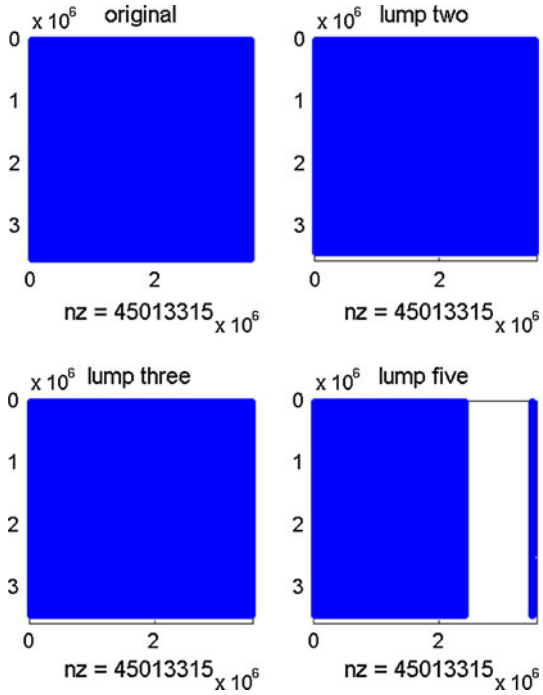

Fig. 4 Example 4.1: Structure of the original and the reordered *Wikipedia-20070206* matrix, where the *stripe* represents nonzero elements, and *white* stands for zero elements

nodes than the dangling ones. The size of the original adjacency matrix is 3,566,907. That is, the original PageRank algorithm has to solve a linear system of size 3,566,907 × 3,566,907. If the Web matrix is lumped into two type nodes according to Lee et al.'s strategy (2007), one needs to solve a 3,465,374 × 3,465,374 linear system for PageRank, while if the matrix is lumped into three type nodes according to Lin et al.'s strategy (2009), one can compute PageRank via solving a linear system of size 3,461,584 × 3,461,584. However, if the hyper-link matrix is lumped into five type nodes, we can compute PageRank via solving a much smaller linear system whose size reduces to 2,435,819 × 2,435,819, a great reduction. Therefore, we expect Lump 5 can do a better job than the other three algorithms. Tables 3 and 4 list the numerical results.

One observes from the numerical results that Lump 5 outperforms the other three algorithms in terms of CPU time, especially when the damping factor is close to 1. So we benefit from lumping five type nodes, and our new method will be an appropriate choice for the PageRank problem with high damping factors. In this experiment, it is seen that the numerical performance of Lump 2 and Lump 3 is comparable to that of the original PageRank problem. The reason is that the number of dangling nodes are relatively small compared with that of the non-dangling ones. We notice that for the same value of α, all the algorithms use about the same number of matrix-vector products. This is due to the fact that the spectral radius of the iteration matrix P_{11} is invariant under similar variant, which is the asymptotic convergence rate of the Jacobi iteration (Golub and Van 1996).

Example 4.2 The aim of this example is two-fold. First, we show that the Lumping 5 strategy is feasible for large matrices. Second, we aim to compare Lump 5 with the recursive reordering algorithm due to Langville and Meyer (2006a), and show superiority of our new algorithm. There are two test matrices in this example. The first one is the 7,414,866 ×

Table 3 Example 4.1, $tol = 10^{-8}$, Four algorithms on the Wikipedia-20060925 matrix, $n = 2{,}983{,}494$

	Algorithm	Original	Lump 2	Lump 3	Lump 5
α	Size	2,983,494	2,894,370	2,890,778	2,063,672
	Lumping	–	2.91	4.83	8.49
$\alpha = 0.85$	mv	109	109	109	109
	Solving	88.2	86.9	86.8	76.8
	Total	88.2	89.8	91.6	85.3
$\alpha = 0.90$	mv	165	165	165	165
	Solving	133.9	131.2	131.1	114.8
	Total	133.9	134.1	135.9	123.3
$\alpha = 0.95$	mv	331	331	331	331
	Solving	267.8	262.8	262.3	229.4
	Total	267.8	265.7	267.1	237.9
$\alpha = 0.99$	mv	1,678	1,678	1,678	1,677
	Solving	1,362.2	1,331.5	1,330.3	1,158.3
	Total	1,362.2	1,334.4	1,335.1	1,166.8

Table 4 Example 4.1, $tol = 10^{-8}$, Four algorithms on the Wikipedia-20070206 matrix, $n = 3{,}566{,}907$

	Algorithm	Original	Lump 2	Lump 3	Lump 5
α	Size	3,566,907	3,465,374	3,461,584	2,435,819
	Reordering	–	3.53	5.88	10.4
$\alpha = 0.85$	mv	109	108	108	108
	Solving	109.4	106.3	106.2	93.2
	Total	109.4	109.8	112.1	103.6
$\alpha = 0.90$	mv	162	162	162	162
	Solving	162.8	158.8	158.7	139.2
	Total	162.8	162.3	164.6	149.6
$\alpha = 0.95$	mv	306	306	306	305
	Solving	308.6	299.5	299.4	261.1
	Total	308.6	303.0	305.3	271.5
$\alpha = 0.99$	mv	1,139	1,139	1,139	1,139
	Solving	1,140.1	1,111.2	1,111.3	9,72.2
	Total	1,140.1	1,114.7	1,117.2	982.6

7,414,866 *indochina-2004* Web matrix, and the second one is the 18,520,486 $\times$ 18,520,486 *uk-2002* Web matrix. The data files are available from http://law.dsi.unimi.it/datasets.ph. We run Lump 5 and Recursive on the two large problems with $\alpha = 0.85,\ 0.90,\ 0.95$ and 0.99. As a comparison, we also list the numerical results obtained from running the Jacobi iteration on the original linear system. Tables 5 and 6 present the numerical results.

Firstly, we observe from Tables 5 and 6 that Lump 5 works much better than Recursive in terms of CPU time. On the other hand, when the damping factor is medium, say, 0.85, the numerical behavior of Lump 5 and that of the original PageRank algorithm is comparable.

Table 5 Example 4.2, $tol = 10^{-8}$, Three algorithms on the indochina-2004 matrix, $n = 7{,}414{,}866$

	Algorithm	Original	Recursive	Lump 5
α	Size	7,414,866	5,786,895	5,838,493
	Reordering	–	2,472.2	13.6
$\alpha = 0.85$	mv	106	106	106
	Solving	117.7	287.2	110.3
	Total	117.1	2,759.4	123.9
$\alpha = 0.90$	mv	162	162	162
	Solving	180.0	344.6	167.2
	Total	180.0	2816.8	180.8
$\alpha = 0.95$	mv	328	328	328
	Solving	367.6	513.2	335.5
	Total	367.6	2,985.4	349.1
$\alpha = 0.99$	mv	1,652	1,652	1,652
	Solving	1,833.7	1,852.9	1,678.7
	Total	1,833.7	4,325.1	1,692.3

Table 6 Example 4.2, $tol = 10^{-8}$, Three algorithms on the uk-2002 Web matrix, $n = 18{,}520{,}486$

	Algorithm	Original	Recursive	Lump 5
α	Size	18,520,486	15,093,204	15,124,174
	Reordering	–	898.0	29.0
$\alpha = 0.85$	mv	105	105	105
	Solving	214.8	282.7	198.3
	Total	214.8	1180.7	227.3
$\alpha = 0.90$	mv	159	159	159
	Solving	326.0	383.2	297.1
	Total	326.0	1281.2	326.1
$\alpha = 0.95$	mv	319	319	319
	Solving	653.8	680.2	591.0
	Total	653.8	1578.2	620.0
$\alpha = 0.99$	mv	1,587	1,587	1,587
	Solving	3,252.8	2,993.5	2,921.0
	Total	3,252.8	3,891.5	2,950.0

However, when the damping factor is close to 1, the new algorithm converges faster than the original PageRank algorithm, so our algorithm is promising for PageRank problems with high damping factors. Secondly, it is seen that the Lumping 5 reordering strategy is feasible for large problems. Furthermore, one observes from Tables 5 and 6 that the new lumping strategy can be of (much) less time-consuming than the recursively reordering strategy due to Langville and Meyer (2006a, b). For instance, for the *indochina-2004* Web matrix, the recursively reordering strategy requires 2472.2 seconds, while the Lumping 5 strategy only needs 13.6 s. For the *uk-2002* Web matrix, it takes us 29.0 s to lump the original data matrix with respect to five types nodes, while the recursively reordering strategy uses 898.0 seconds.

Thirdly, for this example, it seems that the size of the reduced matrix obtained from the recursively reordering scheme is smaller than that from the Lumping 5 strategy, e.g.,

Table 7 Example 4.3, $tol_1 = 10^{-12}$, Five algorithms on the Wb-edu Web matrix, $n = 9{,}845{,}725$

	Algorithm	Original	Lump 2	Lump 3	Recursive	Lump 5
α	Size	9,845,725	6,761,730	6,406,264	6,275,495	6,077,965
	Reordering	–	3.81	6.75	391.3	10.2
$\alpha = 0.85$	mv	160	160	160	160	160
	Solving	96.0	76.5	73.4	150.2	70.3
	Total	96.0	80.3	80.2	541.5	80.5
$\alpha = 0.90$	mv	245	245	245	245	245
	Solving	147.3	115.9	111.3	187.9	106.1
	Total	147.3	119.7	118.1	579.2	116.3
$\alpha = 0.95$	mv	501	501	501	501	501
	Solving	300.6	234.2	224.8	300.7	214.0
	Total	300.6	238.0	231.6	692.0	224.2
$\alpha = 0.99$	mv	2,535	2,535	2,535	2,535	2,535
	Solving	1,522.3	1,177.7	1,128.1	1,194.1	1,069.9
	Total	1,522.3	1,181.5	1,134.9	1,585.4	1,080.1

5,786,895 versus 5,838,493 for the *indochina-2004* Web matrix, and 15,093,204 versus 15,124,174 for the *uk-2002* Web matrix. Indeed, for the size of the reduced matrix with respect to the two reordering strategies, one can not tell which one is *definitely* smaller than the other, and the answer is problem-dependent, see Remark 3.1 and Table 7. Finally, compared with Lump 5, Recursive may cost more CPU time to solve the reduced linear system, even if they require the same number of matrix-vector products. Indeed, there are 477 and 95 square diagonal blocks along the diagonal of the recursively reordered *indochina-2004* and *uk-2002* Web matrices, respectively. Consequently, it will take Recursive more CPU time to form the solutions explicitly.

Example 4.3 The aim of this example is to illustrate that Dev-Lump 5 is superior to Dev-Boldi and Dev-Gleich for calculating the first order derivative vector of PageRank. The test matrix is the *Wb-edu* Web matrix available from http://www.cise.ufl.edu/research/sparse/matrices/~Gleich/index.html, which contains 9,845,725 pages and 55,311,626 links.

As is seen from Sect.3.2, the main overhead for the computation of the derivative vector includes two parts: First, the computation of the "accurate" PageRank vector. As was mentioned before, there are at least two advantages in reducing the matrix dimension: faster computation and smaller round-off error (Wills and Ipsen 2009). To show these two merits, we compare Lump 5 with Original, Lump 2, Lump 3, and Recursive, with a relatively high accuracy $tol_1 = 10^{-12}$. Second, the computation of the derivative vector, and the tolerance is set to be $tol_2 = 10^{-8}$. Therefore, the CPU time and the number of matrix-vector products for Algorithms 5, 6 and 7 consist of those for the "accurate" PageRank vector as well as those for the derivative vector. In this example, the "accurate" PageRank vector is obtained from running the Jacobi iteration on the original linear system (1.3) for Dev-Boldi and Dev-Gleich, and from running Lump 5 for Dev-Lump 5. Tables 7 and 8 list the numerical results.

So as to show superiority of Dev-Lump 5 over Dev-Boldi and Dev-Gleich for computing the first order derivative vector, we define

Table 8 Example 4.3, $tol_2 = 10^{-8}$, Computing the first derivative of the PageRank vector of the Wb-edu matrix

	Algorithm	Dev-Boldi	Dev-Gleich	Dev-Lump5
$\alpha = 0.85$	mv	254	267	267
	CPU	174.1	159.5	128.2
	Speedup	26.3%	19.6%	–
$\alpha = 0.90$	mv	389	410	410
	CPU	267.6	245.3	188.6
	Speedup	29.5%	23.1%	–
$\alpha = 0.95$	mv	792	840	840
	CPU	543.7	501.6	369.7
	Speedup	32.0%	26.3%	–
$\alpha = 0.99$	mv	3,957	4,295	4,295
	CPU	2,712.7	2,564.6	1,823.5
	Speedup	32.8%	28.9%	–

$$\textbf{Speedup} \equiv \frac{\text{CPU}_{\text{Alg.A}} - \text{CPU}_{\text{Dev-Lump5}}}{\text{CPU}_{\text{Alg.A}}},$$

the wall-clock speedups of Dev-Lump 5 with respective to "Alg.A", where "Alg.A" stands for Dev-Boldi or Dev-Gleich. Here "CPU" is the total CPU time for computing the "accurate" PageRank vector and the derivative vector.

It is seen from Table 8 that Dev-Lump 5 is superior to Dev-Boldi and Dev-Gleich, especially when α is close to 1. Indeed, we have to solve two linear systems of size 9,845,725 $\times$ 9,845,725 for Dev-Boldi and Dev-Gleich, while to solve two linear systems with order 6,077,965 $\times$ 6,077,965 for Dev-Lump 5. This explains why Dev-Lump 5 uses less CPU time than Dev-Gleich, even if they make use of the same number of matrix-vector products. On the other hand, we see that Dev-Boldi may need more CPU time than Dev-Gleich, even when the former uses fewer matrix-vector products than the latter. This is due to the fact that the cost of Dev-Boldi is a little higher than that of Dev-Gleich per iteration, refer to Remark 3.3.

On the other hand, we observe from Table 7 that the CPU time used for Lump 2, Lump 3 and Lump 5 are comparable. The reason is that the number of the unreferenced nodes is much less than that of the non-dangling one, refer to Fig. 5. However, we see that Lump 5 is more efficient than the original PageRank algorithm and the recursively reordering algorithm, especially when the damping factor is high. For instance, when $\alpha = 0.99$, Original uses 1,522.3 s, while Lump 5 requires 1,080.1 s to reach the desired accuracy of 10^{-12}. In other words, the new algorithm saves about 29% CPU time for computing PageRank relative to Original. Note that the recursively reordering algorithm needs 1,585.4 s to reach the same accuracy, which is no better than the original Jacobi iteration.

Indeed, Recursive suffers from the overhead of recursively reordering process, 391.3 s, versus 10.2 s for our Lumping 5 strategy. Moreover, there are $p = 163$ square diagonal blocks in the reordered matrix. This partially explains why Recursive often requires more CPU time than the other reordering algorithms, even if they use the same number of matrix-vector products for solving the linear systems, see also the numerical results given in Example 4.2.

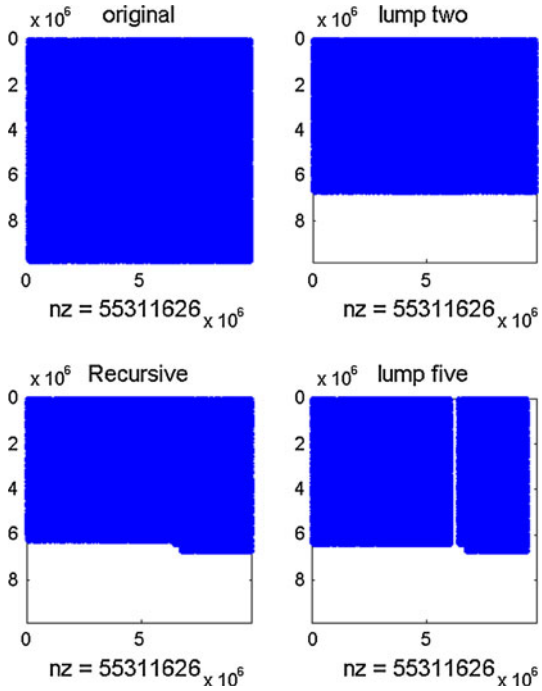

Fig. 5 Example 4.3: Structure of the original and the reordered *Wbedu* matrix, where the *stripe* represents nonzero elements, and *white* stands for zero elements

5 Conclusions and future work

In this paper, we present two new algorithms for computing PageRank and its derivative vector, respectively, based on lumping the Web matrix with respect to five type nodes. Compared with the lumping strategies proposed in (Lee et al. 2007; Lin et al. 2009), the new lumping strategy can reduce the original PageRank problem to a (much) smaller one, while the overhead of the three reordering schemes is comparable. Moreover, the new strategy is much cheaper than the recursively reordering strategy proposed by Langville and Meyer (2006a). Numerical examples show that the new algorithms are favorable when the matrix is large and the damping factor is high.

There are still lots of problems need to be further investigated. For instance, extending the techniques proposed in this paper to a MapReduce infrastructure (Dean and Ghemawat 2008) would be a very interesting research. Given that PageRank can typically only be computed in highly parallel environments, due to the scale of data involved, how to involve parallel computation, and even compute a decomposition as in (3.1) seems non-trivial in a parallel setting where a graph is already distributed across nodes. It would also require extensive evaluation against all the existing algorithms. How do the techniques compare against existing techniques on measures other than CPU time (e.g., I/O), especially in a setting where computation cannot be performed entirely in memory, or even on one computer? Another avenue would be to dive into the accuracy claims following (Wills and Ipsen 2009). These are interesting topics and will be our future research topics.

Acknowledgments We would like to express our sincere thanks to Professor Justin Zobel and two anonymous reviewers for their invaluable suggestions that make us greatly improve the representation of this paper. Meanwhile, we are grateful to Dr. David Gleich and Professor Tim Davis for data files of the

Web matrices, and to Dr. Amy N. Langville for providing us MATLAB codes of the recursively reordering algorithm (Langville and Meyer 2006a). Moreover, we thank Dr. Xing-hua Shi for MATLAB codes of Algorithms 1 and 2. Finally, Gang Wu and Qing Yu would like to thank School of Mathematical Sciences of Xuzhou Normal University for the use of their facilities during the development of this project. Zhengke Miao is supported by the National Natural Science Foundation of China under grants 10871166 and 11171288. Gang Wu is supported by the National Science Foundation of China under grants 10901132 and 11171289, the Qing-Lan Project of Jiangsu Province, and the 333 Project of Jiangsu Province. Yimin Wei is supported by the National Natural Science Foundation of China under grant 10871051, Doctoral Program of the Ministry of Education under grant 20090071110003, 973 Program Project under grant 2010CB327900, Shanghai Education Committee under Dawn Project 08SG01 and Shanghai Science & Technology Committee.

References

Arasu, A. (2002). *PageRank computation and the structure of the Web: Experiments and algorithms*. http://www2002.org/CDROM/poster/173.pdf.

Avrachenkov, K., Litvak, N., Nemirovsky, D., & Osipova, N. (2007). Monte Carlo methods in PageRank computation: When one iteration is sufficient. *SIAM Journal on Numerical Analysis, 45*(2), 890–904.

Broder, A., Kumar, R., Maghoul, F., Raghavan, P., Rajagopalan, S., Stata, R., Tomkins, A., & Wiener, J. (2000). Graph structure in the Web. *Computer Networks, 33*, 309–320.

Batagelj, V., & Zaveršnik, M. (2012). *Generalized cores*. http://arxiv.org/abs/cs.DS/0202039.

Boldi, P., Santini, M., & Vigna, S. (2009). PageRank: Functional dependencies. *ACM Transactions on Information Systems, 27*(4), Article 19.

Brezinski, C., Redivo-Zaglia, M., & Serra-Capizzano, S. (2005). Extrapolation method for PageRank computations. *C R Math Acad Sci Paris, 340*, 393–397.

Brin, S., & Page, L. (1998). The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems, 30*, 107–117.

Brin, S., Motwami, R., Page, L., & Winograd, T. (1998) What can you do with a Web in your pocket? *Data Engineering Bulletin, 21*, 37–47.

Del Corso, G. M., Gullí, A., & Romani, F. (2004). Fast PageRank computation via a sparse linear system. *Internet Mathematics, 2*(3), 251–273.

Dean, J., & Ghemawat, S. (2008). MapReduce: Simplified data processing on large clusters. *Communications of the ACM—50th Anniversary Issue: 1958–2008, 51*(1), 1–13.

Donato, D., Leonardi, S., Millozzi, S., & Tsaparas, P. (2008). Mining the inner structure of the Web graph. *Journal of Physics A: Mathematical and Theoretical, 41*(22).

Eiron, N., McCurley, K. S., & Tomlin, J. A. (2004). Ranking the web frontier. In *Proceedings of the 13th international conference on World Wide Web table of contents* (pp. 309–318), New York, NY, USA.

Gleich, D., Glynn, P., Golub, G. H., & Greif, C. (2007). Three results on the PageRank vector: Eigenstructure, sensitivity and the derivative. In A. Frommer, M. Mahoney, & D. Szyld (Eds.), *Proceedings of the Dagstuhl conference in Web retrieval and numerical linear algebra algorithms*.

Gleich, D., Gray, A., Greif, C., & Lau, T. (2010). An inner-outer iteration for computing PageRank. *SIAM Journal on Scientific Computing, 32*, 349–371.

Gleich, D., Zhukov, L., & Berkhin, P. (2005). *Fast parallel PageRank: A linear system approach*. WWW2005, Chiba and Japan.

Golub, G. H., & Greif, C. (2006). An Arnoldi-type algorithm for computing PageRank. *BIT, 46*, 759–771.

Golub, G. H., & Van Loan, C. F. (1996). *Matrix computations*, 3rd edn. Baltimore and London: The Johns Hopkins University Press

Haveliwala, T., Kamvar, S., Klein, D., Manning, C., & Golub, G. H. (2003). *Computing PageRank using power extrapolation*. Stanford University Technical Report.

Higham, N. J. (2002). *Accuracy and stability of numerical algorithms*, 2nd edn. Philadelphia: SIAM

Ipsen, I., & Selee, T. (2007). PageRank computation, with special attention to dangling nodes. *SIAM Journal on Matrix Analysis and Applications, 29*, 1281–1296.

Ipsen, I., & Wills, R. (2006). Mathematical properties and analysis of Google's PageRank. *Bol Soc Esp Mat Apl, 34*, 191–196.

Kamvar, S., Haveliwala, T., & Golub, G. H. (2004). Adaptive methods for the computation of PageRank. *Linear Algebra and its Applications, 386*, 51–65.

Kamvar, S., Haveliwala, T., Manning, C., & Golub, G. H. (2003). Extrapolation methods for accelerating PageRank computations. In *Twelfth international World Wide Web conference*.

Kamvar, S., Haveliwala, T., Manning, C., & Golub, G. H. (2003). *Exploiting the block structure of the Web for computing PageRank*. Stanford University Technical Report, SCCM-03-02.

Langville, A., & Meyer, C. (2006). A reordering for the PageRank problem. *SIAM Journal on Scientific Computing, 27*, 2112–2120.

Langville, A., & Meyer, C. (2006). *Google's PageRank and beyond: The science of search engine rankings*. Princeton, NJ: Princeton University Press

Lee, C., Golub, G. H., & Zenios, S. (2007). A two-stage algorithm for computing PageRank and multistage generalizations. *Internet Mathematics, 4*(4), 299–328.

Lin, Y., Shi, X., & Wei, Y. (2009). On computing PageRank via lumping the Google matrix. *Journal of Computational and Applied Mathematics, 224*, 702–708.

Page, L., Brin, S., Motwami, R., & Winograd, T. (1998). *The PageRank citation ranking: Bring order to the Web*. Technical Report, Computer Science Department, Stanford University.

Saad, Y. (2003). *Iterative methods for sparse linear systems*, 2nd edn. Philadelphia, PA: SIAM

Wills, R., & Ipsen, I. (2009). Ordinal ranking for Google's PageRank. *SIAM Journal on Matrix Analysis and Applications, 30*, 1677–1696.

Wu, G., & Wei, Y. (2007). A Power-Arnoldi algorithm for computing PageRank. *Numerical Linear Algebra with Applications, 14*(7), 521–546.

Wu, G., & Wei, Y. (2010). Arnoldi versus GMRES for computing PageRank: A theoretical contribution to Google's PageRank problem. *ACM Transactions on Information Systems, 28*(3), Article 11.

Wu, G., & Wei, Y. (2010). An Arnoldi-Extrapolation algorithm for computing PageRank. *Journal of Computational Applied Mathematics, 234*, 3196–3212.